

SALVADOR P. GIMENEZ

Microcontroladores 8051

- ✓ Teoria de Hardware e Software
- ✓ Aplicações em Controle Digital
- ✓ Laboratório/Simulação

Prentice
Hall

PÁGINA EM BRANCO

SALVADOR P. GIMENEZ

Microcontroladores 8051



São Paulo

Brasil Argentina Colômbia Costa Rica Chile Espanha
Guatemala México Peru Porto Rico Venezuela

© 2002 by Pearson Education do Brasil

Todos os direitos reservados

Editor: Roger Trimer

Gerente de Produção: Silas Camargo

Produtora Editorial: Sandra Cristina Pedri

Capa: Marcelo da Silva Françozo

Dados Internacionais de Catalogação na Publicação (CIP)
(Câmara Brasileira do Livro, SP, Brasil)

Gimenez, Salvador Pinillos

Microcontroladores 8051: Teoria do hardware e do software / Aplicações em controle digital / Laboratório e simulação.

2002 – São Paulo – Pearson Education do Brasil Ltda.

1. Intel 8051 (Microprocessadores)
2. Microcontroladores I. Título

ISBN: 85.87918-28-1

02-2834

CDD - 004.16

Índice para catálogo sistemático

1. Microcontroladores 8051 : Microcomputadores digitais : Ciências da computação. 004.16

2005

Direitos exclusivos para a língua portuguesa cedidos à
Pearson Education do Brasil,
uma empresa do grupo Pearson Education
Av. Hermano Marchetti, 1435
Cep: 05038-001 – São Paulo – SP
Tel: (11) 3613-1222 – Fax: (11) 3611-0444
e-mail: vendas@pearsoned.com.br

SUMÁRIO

Agradecimentos	vii
Prefácio	ix
Introdução	xiii
<i>Capítulo 1</i> Fundamentos de microcomputadores	1
<i>Capítulo 2</i> A família de microcontroladores MCS-51 da Intel	17
<i>Capítulo 3</i> O conjunto de instruções da família de microcontroladores MCS-51 da Intel	49
<i>Capítulo 4</i> Fluxograma e programação em Assembly, aplicados à família de microcontroladores MCS-51 da Intel	77
<i>Capítulo 5</i> Sub-rotinas e estruturação da linguagem de programação Assembly	101
<i>Capítulo 6</i> As portas (portes) de entrada e saída e suas aplicações no controle digital de máquina e de processo	125
<i>Capítulo 7</i> Os <i>timers/contadores</i>	147
<i>Capítulo 8</i> A interface de comunicação serial	167
<i>Apêndice A</i> Experiência 01: Implementação e execução de programas em Assembly por meio do simulador AVSIM51 ou do Kit de Desenvolvimento AES-51	189
<i>Apêndice B</i> Experiência 02: Modos de endereçamento da família de microcontroladores MCS-51 da Intel	205
<i>Apêndice C</i> Experiência 03: Fluxograma e programação em Assembly aplicados à família de microcontroladores MCS-51 da Intel	209
<i>Apêndice D</i> Experiência 04: Sub-rotina e estruturação da linguagem Assembly aplicados à família de microcontroladores MCS-51 da Intel	213
<i>Apêndice E</i> Experiência 05: As portas de entrada e saída da família de microcontroladores MCS-51 da Intel e suas aplicações no controle digital de máquina e de processo	219

<i>Apêndice F</i> Experiência 06: Os <i>timers</i> /contadores da família de microcontroladores MCS-51 da Intel	225
<i>Apêndice G</i> Experiência 07: A interface de comunicação serial da família de microcontroladores MCS-51 da Intel	233
Índice Remissivo	241

AGRADECIMENTOS

Dedico esta obra à minha família, razão de minha vida, à minha querida esposa Valéria e às minhas duas queridas, lindas e especiais filhas, Giovanna e Giorgia, pelos seus incansáveis e grandiosos incentivos em todos os momentos no decorrer da elaboração deste livro.

A meus pais, Salvador Gimenez e Maria Jesus P. Gimenez, minha eterna gratidão por tudo o que fizeram e continuam fazendo por mim.

Aos meus queridos sobrinhos Ana Carolina Gimenez de Godoy e Felipe Gimenez de Godoy, à minha querida irmã Maria Clara P. Gimenez de Godoy e ao seu marido, Antônio Cury de Godoy.

Aos meus eternos ‘anjos da guarda’ e colaboradores, Dona Leonice Dei Santi, Dona Leonor Dei Santi, Mirtes Ribeiro e Milton Ribeiro, pela formidável infra-estrutura.

Ao Prof. Dr. João Antônio Martino, pela confiança depositada em minha pessoa ao longo de todos esses anos.

Ao Prof. Aldo Belardi, pelo grande apoio e incentivo para a publicação desta obra.

Ao Prof. Devair Arrabassa, pelo imenso incentivo para a realização deste trabalho.

Aos professores Aderbal A. Penteado Jr., Fausto de Freitas Del Monte, Mario Cesar Ramos, Herald Barbuy, Renato Faria, Marco A. Fumagalli, Wilton Ney Amaral Pereira, Marcelo Pavanello, Marcelo Bellodi, Aparecido Nicollet, Vitor Sonnemberg, Luciano Camillo, Paulo H. Tetsuo, Jorge Nabarrete, M. A. Assis, Orlando Del Bianco, Orlando Onofre e Fabrício Leonardi, também pelos incessantes incentivos para a realização desta obra.

A todos os grandes amigos professores da FEI, da OMEC/UMC e do grupo SOI CMOS do LSI-EPUSP, que direta ou indiretamente contribuíram para a finalização desta obra.

Salvador Pinillos Gimenez

PÁGINA EM BRANCO

PREFÁCIO

É uma realidade que os microcomputadores são os grandes personagens de nossa sociedade. Eles visam principalmente facilitar e agilizar as atividades básicas do cotidiano. Parte desses microcomputadores integra a maioria dos equipamentos residenciais, comerciais, automotivos e industriais e são, em grande parte, implementados com microcontroladores. Isso ocorre principalmente por serem baratos e por reunirem características de hardware e software dedicadas a aplicações simples de automação, tais como:

- a) *As de uso pessoal*: relógios de pulso digitais, agendas eletrônicas, *pgers*, telefones celulares, *handhelds* etc.
- b) *As de uso residencial*: portões automáticos, alarmes residenciais, televisores, rádios digitais, equipamentos de áudio, vídeo e DVD, fornos de microondas, máquinas de lavar e secar roupas, máquinas de lavar pratos, rádio-relógios digitais etc.
- c) *As de uso industrial*: CLPs (Controladores Lógicos Programáveis), equipamentos digitais de medição de tensão, corrente, resistência ôhmica, pressão, temperatura e umidade, relógios de ponto, controladores de acesso restrito etc.
- d) *As de uso automotivo (eletrônica embarcada)*: computadores de bordo, alarmes de carros, rádios automotivos, injeções eletrônicas de automóveis e caminhões, controle de freios ABS etc.
- e) *As de uso geral*: caixas eletrônicos de bancos, catracas eletrônicas de ônibus urbanos e de metrô, impressoras, teclados de computadores, computadores pessoais etc.

Nota-se que a quantidade de aplicações é bastante grande, e visando disponibilizar as informações sobre essa tecnologia tão importante nos dias de hoje, este livro foi desenvolvido para capacitar os leitores para a tecnologia dos microcontroladores, principalmente os da família MCS-51 da Intel, em nível técnico-profissionalizante. Este livro pode ser usado também por estudantes de escolas técnicas profissionalizantes, alunos de escolas de engenharia elétrica (eletrônica, computação e telecomunicações) e mecatrônica.

Cabe aqui ressaltar que o conhecimento dessa tecnologia não é mais uma especialização adicional de poucos profissionais de sucesso no mercado

de trabalho, mas sim um requisito básico e fundamental para os profissionais que desejam fazer parte de um mercado de trabalho competitivo. Além disso, sabe-se que os profissionais especialistas nessa área obtêm grande penetração profissional em empresas de média e alta tecnologia, e em todos os países do mundo existem grandes oportunidades profissionais para os recursos humanos capacitados nessa área.

Após a leitura, o entendimento e a simulação prática sugerida neste livro, o leitor será capaz de desenvolver projetos de hardware e software de equipamentos inteligentes que utilizam microcontroladores, visando principalmente ao estudo da família de microcontroladores MCS-51 da Intel. A partir daí, o leitor também poderá executar sem muito esforço projetos com microcontroladores de fornecedores como Microchip, ATMEL, Texas Instruments e Motorola, pois os princípios eletrônicos básicos são exatamente os mesmos que os aqui apresentados. Para isso, o leitor só terá de conhecer a estrutura interna, a pinagem e as instruções de tais microcontroladores.

Assim, o objetivo principal deste livro é o estudo da família de microcontroladores MCS-51 da Intel. Após apresentar em detalhes sua estrutura interna relacionada ao hardware, sua pinagem e suas instruções, aborda a programação Assembly aplicada a essa família de microcontroladores (fluxograma, programa-fonte, programa-objeto e simulação) e, finalmente, ensina como implementar projetos de equipamentos inteligentes com a utilização dos microcontroladores MCS-51 da Intel.

A teoria sobre a família de microcontroladores MCS-51 é novamente abordada nos exercícios resolvidos e propostos. O livro traz também um conjunto de apêndices relacionados a cada capítulo que apresentam aplicações práticas por meio do uso do simulador AVSIM51 da AVOCET ou do kit de desenvolvimento AES-51. Essas aplicações práticas podem ser utilizadas como aulas de laboratório em cursos técnicos profissionalizantes ou em cursos de engenharia, para que o leitor possa entender, visualizar e praticar a utilização dos microcontroladores da família MCS-51 da Intel em projetos de controle digital de máquina, processo e automação.

Os exemplos foram implementados utilizando-se a linguagem de programação Assembly, e foram editados pelo editor de textos Edit, do DOS, compilados e *linkados* com o compilador e *linkador* produzidos pela AVOCET. Todos esses exemplos foram implementados e testados pelo autor, a fim de eliminar qualquer risco de este livro conter informações incorretas.

Por estar diretamente relacionado a cadeiras de cursos de graduação e pós-graduação de grande parte dos cursos técnicos e de engenharia elétrica e mecatrônica, este livro sugere um conjunto de oito apêndices com aulas práticas, para que o leitor desenvolva, de maneira gradativa, suas habilidades

práticas com o estudo da teoria apresentada em cada capítulo, como mostrado na Tabela 1, apresentada a seguir.

Tabela 1 Execução de aulas práticas de simulação após a leitura do(s) capítulo(s).

Após leitura e entendimento do(s) capítulo(s):	Aula prática seguindo o Apêndice:	Nome da experiência
1 e 2	A	Implementação e execução de programas em Assembly ou por meio do simulador AVSIM51 da AVOCET ou do kit didático AES-51.
3	B	Modos de endereçamento da família de microcontroladores MCS-51 da Intel.
4	C	Fluxograma e programação em Assembly, aplicados à família de microcontroladores MCS-51 da Intel.
5	D	Sub-rotinas e estruturação da linguagem Assembly.
6	E	As portas de entrada e saída da família de microcontroladores MCS-51 da Intel e suas aplicações no controle digital de máquina e de processo.
7	F	Os <i>timers</i> /contadores da família de microcontroladores MCS-51 da Intel.
8	G	A interface de comunicação serial da família de microcontroladores MCS-51 da Intel.

Após a leitura e o entendimento de cada capítulo é recomendado que o leitor pratique por meio da simulação, utilizando um dos simuladores disponibilizados na World Wide Web (basta procurar em uma das ferramentas de busca, utilizando as seguintes palavras-chave: *8051 simulator*) ou por meio de um kit didático ou de desenvolvimento, se existirem em sua instituição de ensino. A fim de facilitar a aplicação prática dessa tecnologia, serão fornecidas informações sobre o simulador AVSIM51 da AVOCET e sobre o kit de desenvolvimento AES-51.

Observação: não existe ligação comercial entre o autor e a editora deste livro com os fornecedores do simulador e do kit de desenvolvimento. Eles foram utilizados e citados aqui porque estavam disponíveis para o autor no momento da elaboração desta obra. Este livro não se propõe a recomendar a compra de nenhuma espécie de simulador ou de kit de desenvolvimento. Isso fica por conta do leitor.

Espera-se que o leitor possa, ao final da leitura deste livro, com o entendimento dos exercícios resolvidos, com a prática dos exercícios propostos e com a simulação prática apresentada nos apêndices, se tornar um especialista em projetos de produtos inteligentes, tanto no que se refere ao hardware como ao software. Lembre-se de que conhecimento nunca é demais.

PÁGINA EM BRANCO

INTRODUÇÃO

A expectativa da sociedade moderna é cada vez mais buscar e conquistar uma melhor qualidade de vida em todos os níveis relacionados ao ser humano e ao meio em que vive. Tal qualidade de vida, em grande parte, está associada à pesquisa e ao desenvolvimento das diferentes áreas da ciência e da tecnologia. Uma vez adquirida tal tecnologia pelos cientistas e disponibilizada para as pessoas por meio de equipamentos e máquinas, pode-se dizer que direta ou indiretamente ela é alcançada pelos diversos setores da sociedade (habitação, medicina, transporte, segurança, entretenimento etc.).

Uma das tecnologias que vêm crescendo em uma velocidade bastante impressionante é a tecnologia da engenharia eletrônica. Repare também que ela se tornou uma tecnologia-base, pois suporta, hoje em dia, as demais áreas de conhecimento. Isso se deve principalmente à descoberta e à melhoria contínua do que se pode chamar de *tecnologia do computador*.

Com o desenvolvimento muito rápido da tecnologia de integração de circuitos eletrônicos, que corresponde à área da engenharia elétrica chamada de microeletrônica, foi possível integrar as partes básicas de um computador (unidade central de processamento, unidades de armazenamento de informações e unidades de entrada e saída). Conseqüentemente, foi possível implementar computadores mais compactos, confiáveis e baratos, dando origem aos chamados *microcomputadores*.

Os primeiros microcomputadores foram implementados fisicamente com diferentes circuitos integrados, cada um com sua função, microprocessador, memória não-volátil (ROM/PROM/EPROM/EEPROM) e volátil (RAM) e portas de entrada e saída (E/S). Esses circuitos integrados formavam as partes básicas de um computador digital. Cada componente foi interligado entre si, formando os primeiros microcomputadores.

O primeiro microprocessador ou a primeira unidade de processamento central (CPU, *Central Processing Unit*) foi um microprocessador de 4 bits, o 4004 da Intel. Seu sucesso foi tal que os projetistas solicitavam unidades com maiores capacidades de processamento para aplicações mais sofisticadas. Depois, a Intel implementou os microprocessadores de 8 bits 8008, 8080 e finalmente o 8085. Outros fabricantes, como Motorola e Zilog, lançaram também microprocessadores de 8 bits, no caso o 6800 e o Z80, respectivamente, para concorrer com a Intel.

Depois vieram os microprocessadores de 16, 32 e 64 bits. Cada vez mais a microeletrônica foi sendo desenvolvida, e cada vez mais havia um número

maior de componentes integrados por unidade de área. Com isso foi possível integrar todos os blocos básicos de um microcomputador digital em uma única pastilha de silício, originando assim os *microcontroladores*.

O microcontrolador nada mais é que um microcomputador implementado em um único circuito integrado, no qual estão integradas todas as unidades básicas de um computador.

Esses microcontroladores são utilizados principalmente em equipamentos portáteis de baixo custo e são os grandes contribuidores para a automação do mundo atual, por isso o nosso interesse em estudá-los nesta obra.

Repare que isso não tem mais parada e nem tampouco volta. Hoje se pode dizer que o ser humano é totalmente dependente dessa tecnologia. Imagine o homem de hoje em dia viver sem as máquinas e os equipamentos disponíveis atualmente (carros, aviões, trens, barcos, telefones celulares, microcomputadores, equipamentos médicos etc.). Dessa maneira, pode-se perguntar como ficaria a qualidade de vida se não pudéssemos contar mais com essas máquinas e equipamentos.

Tais máquinas e equipamentos, em sua grande maioria, usam a tecnologia de computador. Quanto àquelas que ainda não foram implementadas com tal tecnologia, pode-se dizer que existe aí uma grande oportunidade de negócio, pois certamente, se implementadas, ficarão mais compactas, baratas e confiáveis, com grandes perspectivas de mercado.

Assim, os leitores, futuros técnicos em eletrônica ou futuros engenheiros nas mais diversas áreas de conhecimento da engenharia elétrica, principalmente no que diz respeito às áreas de eletrônica, computação, telecomunicações e mecatrônica, não podem e não devem ficar fora do contexto da tecnologia do computador, pois corresponderia a ficar fora do mercado de trabalho.

Assim, esta obra foi criada com o objetivo de suprir uma necessidade de mercado, sendo um material bibliográfico que apresenta teoria e exercícios resolvidos e propostos, além de aulas práticas em que o leitor pode praticar seus conhecimentos utilizando os simuladores disponíveis gratuitamente na Internet. Este livro também apresenta como características principais sua maneira simples, objetiva e gradativa de ensinar ao aluno de cursos técnico-profissionalizantes e de cursos de engenharia elétrica (eletrônica, computação, telecomunicações e mecatrônica) a *tecnologia do computador*.

Esta obra reúne o conhecimento acumulado pelo autor em quinze anos de experiência tanto em empresas de desenvolvimento de produtos inteligentes como em instituições de ensino de engenharia com significativo sucesso na formação de recursos humanos.

Assim, espera-se que o leitor, ao final do estudo e da prática da tecnologia aqui apresentada, se torne um projetista de hardware e software de equipamentos inteligentes que utilizam microcontroladores.

capítulo

1

FUNDAMENTOS DE MICROCOMPUTADORES

1.1 Objetivos

- ❖ Especificar as definições básicas da computação
- ❖ Apresentar e definir a estrutura básica dos microcomputadores
- ❖ Apresentar a arquitetura do sistema microcomputadorizado e como os blocos básicos se comunicam entre si

1.2 Introdução teórica

1.2.1 - Conceitos básicos da computação: antes de iniciarmos o estudo detalhado da teoria dos microcontroladores é necessário, em primeiro lugar, relembrar alguns conceitos básicos fundamentais para o entendimento dos assuntos que serão abordados nos diferentes capítulos e apêndices deste livro. Mesmo considerando tais definições como pré-requisitos para atingir as metas deste livro, o objetivo de descrevê-las é evitar que o leitor tenha de recorrer a outras obras durante a leitura deste livro.

Bit: abreviação de dígito binário (*binary digit*), que corresponde ao valor zero (0) lógico ou ao valor um (1) lógico. Zero lógico geralmente corresponde a zero volt (V) e um lógico, a 5 ou 3 volts (V).

Byte: representação numérica composta de 8 bits. Pode representar números de 00h (= 0₁₀) a FFh (= 255₁₀). Essa representação é vastamente utilizada em quase todos os capítulos deste livro, mas é principalmente empregada na representação das instruções que o microcontrolador é capaz de executar.

Registradores: conjunto de *flip-flops*, geralmente do tipo D (unidade básica de armazenamento de informação), que são interligados em paralelo entre si. Eles são responsáveis pelo armazenamento de uma informação que pode ter 8, 16, 32, 64 ou 128 bits, dependendo de quantos *flip-flops* do tipo D estão interligados entre si. Os registradores apresentam características voláteis e, quando eles são desenergizados, perdem suas informações. Embora bastante conhecido, esse conceito facilitará o entendimento do leitor à medida que este livro definir os registradores de funções especiais (SFRs, *Special Function Registers*) e também durante a descrição da memória RAM interna do microcontrolador.

Microcomputador: equipamento compacto cujo objetivo principal é *simular* (imitar) eletronicamente o sistema chamado *ser humano*. Suas principais vantagens são: grande velocidade do processamento das informações e alta confiabilidade nas atividades executadas. Ele é implementado por três blocos básicos: Unidade Central e de Processamento, Unidade de Memória e Unidade de Entrada e Saída de Informações. Essa definição é de primordial importância, pois a principal meta deste livro é transformar o leitor em um projetista de hardware e software de microcomputadores implementados com os membros da família de microcontroladores MCS-51 da Intel.

Instrução: define uma única ação (tarefa) que o microcomputador pode executar por vez. As ações das instruções podem corresponder a operações de leitura e escrita nos conteúdos dos registradores ou nas posições de memórias, a operações lógicas e aritméticas etc. Exemplo: copiar um byte do conteúdo de uma posição de memória para outra, escrever um byte de um registrador para outro, adicionar dois bytes etc. Uma instrução pode ser constituída ou representada por um ou mais bytes. Exemplo: a instrução MOV A,R0 é definida por um byte, E8h, enquanto a instrução MOV A,#25h é definida por dois bytes, 74h e 25h. Elas são definidas por seus fabricantes. Essa definição será utilizada para entender como um microcomputador funciona e o que vem a ser um conjunto de instruções que o microcontrolador é capaz de executar.

Equipamento inteligente: equipamento implementado com microprocessador ou com microcontrolador.

Programa ou software: definido por um *conjunto de instruções* arranjadas de maneira organizada por um programador (profissional especialista em computação), com o objetivo de informar ao microcomputador o que

ele deve executar, ou seja, o programa informa ao microcomputador quais as tarefas que devem ser executadas ao longo do tempo. Existem várias classificações de programas: os programas básicos (softwares básicos), os utilitários e os aplicativos. Exemplos de programas básicos: *DOS*, o programa de um forno de microondas etc. Exemplos de programas utilitários: *PCShell*, *Doctor Norton Utilities* etc. Exemplos de programas aplicativos: editores de texto, como o *MS-WORD*; as planilhas eletrônicas, como o *MS-EXCEL*, etc. O programa sempre deve estar armazenado em uma unidade de armazenamento de informações, ou seja, em uma memória. Essa definição é básica e fundamental para os leitores que se tornarão responsáveis pela implementação do software de equipamentos que utilizam microcontroladores (equipamentos inteligentes).

Firmware: programa (software) que está exclusivamente armazenado em uma memória não-volátil (ROM/PROM/EPROM/EEPROM) de um equipamento inteligente. No mínimo, o firmware tem a finalidade principal de programar a forma de operação do hardware (número de portas de entrada e saída, forma de comunicação serial, forma de operação dos *timers/contadores* etc.) e de definir suas condições iniciais de operação. Essa definição também é de fundamental importância, pois o leitor será orientado justamente para ser especialista no desenvolvimento e na implementação do firmware de equipamentos inteligentes.

Hardware: são as partes eletrônicas que compõem um microcomputador, como a *mother board* (placa mãe que contém pelo menos o microprocessador ou o microcontrolador de um microcomputador), a placa de vídeo de um microcomputador etc. Essa definição é básica e será utilizada em quase todos os capítulos e apêndices deste livro.

Microprocessador ou Unidade Central de Processamento (CPU): mais conhecido como CPU (*Central Processing Unit*). Fazendo analogia com o sistema *ser humano*, o microprocessador, ou CPU, de um microcomputador corresponde ao *cérebro* de um ser humano. Fisicamente é um dispositivo semicondutor (circuito integrado, CI ou IC, abreviação do nome em inglês: *Integrated Circuit*) constituído por milhões de transistores que implementam uma variedade de circuitos (registradores, máquinas seqüenciais, circuitos lógicos etc.). Ele é responsável pela busca de um programa na memória e por sua execução. Esse conceito será sempre empregado no decorrer desta obra, mas será enfatizado principalmente

durante a definição do princípio de funcionamento dos equipamentos inteligentes e na definição do ciclo de busca e execução das instruções do microcontrolador.

Microcontrolador: dispositivo semicondutor em forma de CI, que integra todas as partes básicas de um microcomputador – microprocessador (CPU), memórias não-voláteis (ROM/PROM/EPROM/EEPROM), memórias voláteis (RAM, SRAM, DRAM, *Flash RAM*), portas de entrada e saída (portas de comunicação paralela, portas de comunicação serial, conversores analógicos/digitais, conversores digitais/analógicos etc.). Ele é conhecido como um microcomputador implementado em um único CI. Geralmente, é limitado em termos de quantidade de memória, principalmente no que diz respeito à memória de dados, e é utilizado em aplicações específicas, ou seja, naquelas que não necessitam armazenar grandes quantidades de dados, como em automação residencial (fornos de microondas, máquinas de lavar louça, máquinas de lavar roupa, telefones, alarmes residenciais, automação de portões etc.), em automação predial (elevadores, controladores de energia elétrica etc.), em automação industrial (robótica, controladores lógicos programáveis, ou CLPs, controladores de acesso restrito, relógios de ponto) e na automação embarcada (computadores de bordo, alarmes etc.). Apresenta um custo bastante baixo, em torno de US\$ 4,00 (em 2001). São vários os fornecedores de microcontroladores. Os principais, atualmente, em termos de volume de vendas, são: Motorola, Microchip, Mitsubishi, NEC, Philips, SGS, Intel, Hitachi, Toshiba etc. Esse conceito é indispensável ao leitor com qualquer nível de conhecimento e é o objetivo primordial do estudo desta obra.

Com os conceitos descritos anteriormente, espera-se que o leitor tenha uma maior facilidade tanto na leitura como no entendimento dos assuntos relacionados a microcontroladores, que serão apresentados aqui.

1.2.2 - Blocos básicos de microcomputadores: todos os computadores, sejam eles microcomputadores, computadores de médio porte ou ainda computadores de grande porte, são constituídos por três blocos básicos: a unidade central de processamento (CPU ou microprocessador), a unidade de memória (na qual são armazenados os programas e as informações) e a unidade de entrada e saída (dispositivos que fazem a comunicação com o mundo externo), como está apresentado na Figura 1.1. A diferença entre esses equipa-

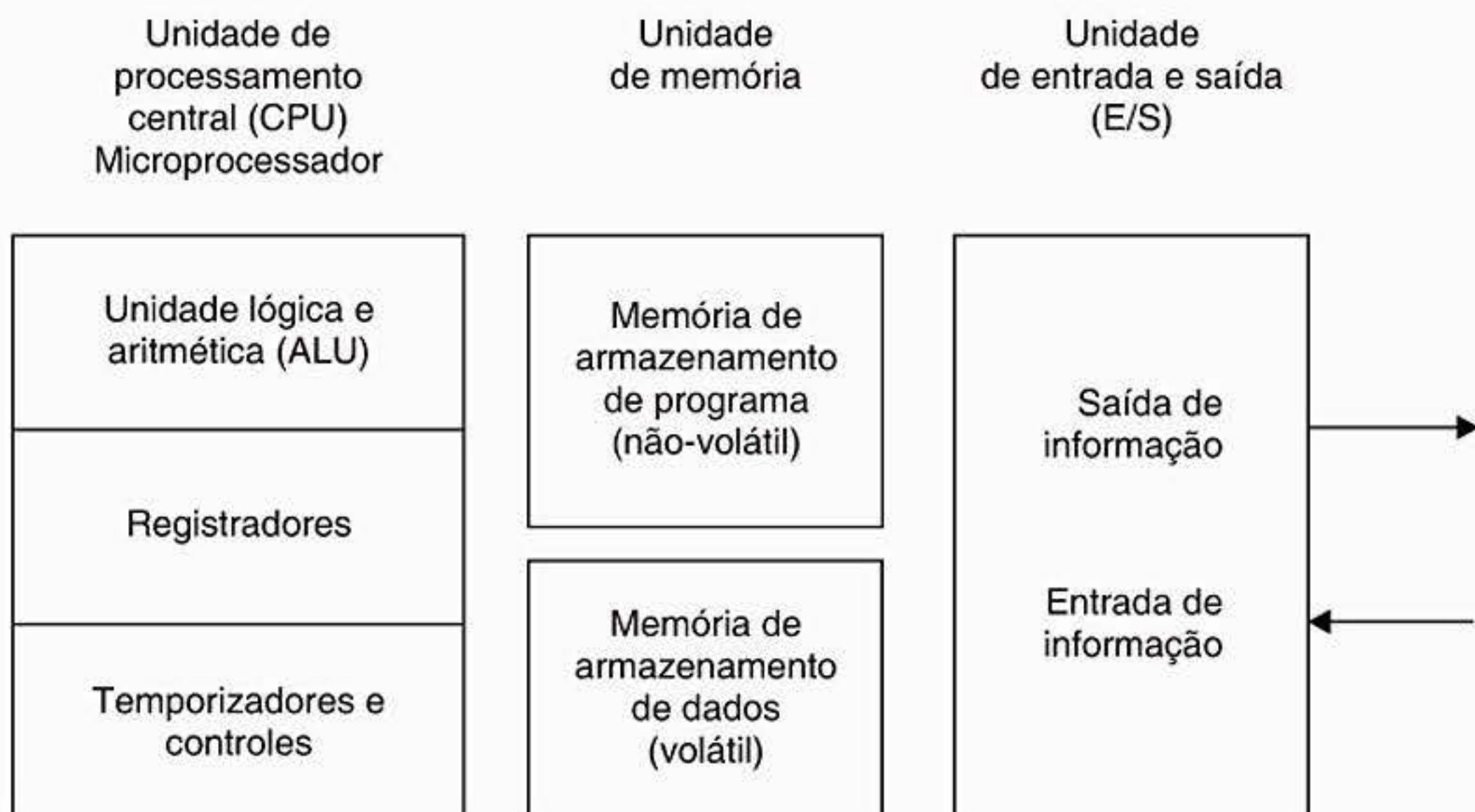


Figura 1.1 Blocos básicos de microcomputadores.

mentos está na variação das características de cada bloco básico que os compõem (p. ex.: o fornecedor do microprocessador/microcontrolador pode ser a Intel, a Motorola ou a Microchip; a velocidade de processamento pode ser a de um microcomputador de 10 MHz, 500 MHz etc., a capacidade de memória para o armazenamento de programas e informações pode ser de 8 Kbytes, 32 Mbytes etc., a quantidade de dispositivos de entrada e saída analógicas e digitais etc.). Repare que a arquitetura desse sistema é análoga ao sistema chamado *ser humano*, ou seja, ela tem um pseudocérebro, memórias e interfaces com o mundo externo. A seguir, será dada em detalhes a descrição de cada bloco básico.

Unidade Central de Processamento (CPU) ou Microprocessador: fazendo novamente a analogia com o sistema *ser humano*, esse bloco corresponde ao *cérebro* do microcomputador. A CPU é responsável pela inteligência dessa máquina, ou seja, é ela que tem a capacidade de tomar decisões (ações) no sistema microcomputadorizado, por meio da execução de um programa. É ela que executa o programa armazenado na memória do microcomputador, que foi projetado por um programador. Ao executar um programa, ela é responsável pela obtenção das informações a serem analisadas por meio de dispositivos de entrada (teclado, canal de comunicação serial etc.), pelo processamento (interpretação, manipulação, ordenação, modificação, cálculos lógicos ou matemáticos etc.) das informações e pela resposta (ação) do sistema microcomputadorizado a uma determinada situação de controle para a qual ele foi projetado a fim de

controlar, por meio de um dispositivo de saída (escrita na memória de vídeo, transmissão de dados para outro microcomputador, acionamento de um relé, acionamento de um bip etc.). Assim, por exemplo, um microcomputador projetado para funcionar como um caixa eletrônico de banco deve fazer as seguintes operações: leitura das informações contidas no cartão magnético (obtenção de informações), decodificação da senha digitada no teclado pelo usuário, busca de saldo bancário nos bancos de dados do banco (processamento), disponibilização do saldo bancário no monitor de vídeo e a impressão em papel do referido saldo bancário (resposta do sistema por meio da ação de gerar informações no monitor de vídeo e também pela impressão em papel das informações organizadas solicitadas pelo usuário).

De maneira simplificada, pode-se dizer que a unidade de processamento central (microprocessador) apresenta duas funções básicas:

- a) *Leitura (busca) e interpretação do programa alocado na memória, instrução por instrução*: como o programa fica alocado na memória do microcomputador, e não dentro do microprocessador/CPU, em primeiro lugar, o microprocessador precisa *buscar* as instruções, uma por uma, na memória. Isso é feito por meio da operação de leitura de um byte na memória que corresponde a uma determinada instrução e depois é realizada a *interpretação* da instrução lida (decodificação), pois o microprocessador tem uma determinada quantidade de instruções que ele é capaz de executar. Dessa maneira, ele precisa verificar qual dessas instruções deve ser executada. Assim, ao finalizar essa etapa, o microprocessador/CPU sabe o que deve fazer. O próximo passo é a execução da instrução.
- b) *Execução do programa, instrução por instrução*: a palavra execução de uma instrução pode significar a transferência de dados dos registradores internos do microprocessador/CPU para a memória ou para os dispositivos de entrada e saída, e vice-versa. Pode também significar a execução de operações lógicas e aritméticas etc.

Esse processo de buscar, interpretar e executar instrução por instrução é uma atividade contínua do microprocessador e, dessa forma, um programa é processado por inteiro.

A unidade de processamento central (microprocessador) é constituída por três partes principais: a *unidade lógica e aritmética*, os *registradores internos* e o *bloco de temporização e controle*.

- a) *Unidade Lógica e Aritmética (ULA)*: muitas vezes, também é chamada de *ALU (Arithmetic Logic Unit)*. Ela é responsável, principalmente, pela execução das instruções correspondentes às operações aritméticas e lógicas. Além disso, uma característica muitíssimo importante que deve ser destacada, referente à ULA, é que, ao executar uma instrução de uma operação aritmética ou lógica, ela também define a condição numérica do resultado por meio de alguns bits sinalizadores, chamados de *flags de sinalização*, tornando-os iguais a um (setando-os) ou tornando-os iguais a zero (resetando-os), e pela utilização de outras instruções que são capazes de testar esses *flags de sinalização*. Isso faz com que o sistema microcomputadorizado possa gerenciar uma determinada situação de projeto ou desempenhar uma determinada tarefa bem-definida.
- b) *Registradores internos*: são compostos por m registradores de n bits interligados em paralelo entre si. Os valores m e n dependem das características de cada microprocessador/microcontrolador (p. ex.: 32 registradores de 8 bits etc.). Como são formados por *flip-flops*, é possível realizar operações de leitura e escrita de informações. Quando a CPU é 'desenergizada' por apresentar características voláteis, essas informações são perdidas. Geralmente, eles são capazes de armazenar:
- ◆ um byte, se forem constituídos por registradores de 8 bits;
 - ◆ um duplo byte ou *address* ou *word*, se forem constituídos por registradores de 16 bits;
 - ◆ um duplo *address* ou duplo *word*, se forem constituídos por registradores de 32 bits etc.

Um microcomputador é chamado de *microcomputador de 8 bits* se ele for capaz de processar informações de oito em oito bits por vez (operações de movimentação de informações e operações aritméticas e lógicas). Analogamente, para os microcomputadores de 16 e 32 bits, quanto maior for a quantidade de bits processados em paralelo por vez, maior será sua capacidade de processamento. A unidade utilizada para medir a capacidade de processamento é o *Mips* (milhões de informações por segundo).

Cabe ressaltar que o microprocessador/CPU *não* é uma unidade de armazenamento de informações (memória), pois apresenta apenas alguns registradores internos que são utilizados, normalmente, para o armazenamento temporário de informações, ou seja, sua função não é o armazenamento de grandes quantidades de dados. Grandes quantidades de dados devem ser alocados nas memórias disponíveis no sistema microcomputadorizado.

O tempo de acesso às informações que estão alocadas nesses registradores internos é menor que o tempo de acesso às informações que estão alocadas na memória. Isso se justifica pelo fato de as informações já estarem dentro do próprio microprocessador/CPU. Com isso, em aplicações que necessitam de grandes velocidades de processamento, por exemplo, na leitura de uma tarja magnética de cartão de banco etc., recomenda-se a utilização desses registradores na composição do programa, para atingir altas velocidades de processamento, melhorando o desempenho da tarefa a ser realizada.

- c) *Bloco de temporização e controle:* responsável pelo controle do fluxo de informações do microprocessador para as unidades de memória e para as unidades de entrada e saída. É esse bloco que define os sinais de controle de temporização para o sincronismo do fluxo de informações no sistema microcomputadorizado (p. ex.: liberação de informações da ULA para os registradores, memória e unidades de entrada e saída etc.) por meio da definição dos sinais de leitura (*read*), de escrita (*write*), de inicialização do sistema (*reset*), de liberação dos barramentos de dados e endereços, entre outros.

Unidades de memória: são divididas em duas partes: *memória de armazenamento de programa* e *memória de armazenamento de informações* (dados, bytes, bits etc.).

- a) *Memória de armazenamento de programa:* todo microcomputador deve ter uma área de memória na qual será armazenado o programa que definirá as tarefas que o microcomputador/CPU deve executar. Esse programa define a função de um microcomputador e também diz ao microcomputador, instrução por instrução, o que ele deve fazer ao longo do tempo. Esse programa deve ser implementado por um programador técnico, conhecedor do hardware do sistema microcomputadorizado e das instruções do microprocessador. O microcomputador não funcionará se não existir um programa armazenado em sua memória de programa. Assim, uma vez energizado o microcomputador, esse programa será buscado e executado pelo microprocessador. Esse programa define a característica funcional do microcomputador (p. ex.: operar como um forno de microondas, como uma máquina de lavar roupas ou como uma injeção eletrônica de automóvel etc.). Esse programa não pode ser perdido na ausência de energia elétrica, pois isso acarretaria a perda da funcionalidade desse

microcomputador, para o qual foi projetado. Assim, o tipo de memória em que esse programa deve ser armazenado é a memória *não-volátil*. Dessa maneira, sempre que o microcomputador for 'desenergizado', esse programa não poderá ser perdido e quando ele for 'energizado' novamente, deverá voltar a funcionar com a mesma característica funcional. Memórias não-voláteis normalmente utilizadas em microcomputadores são as memórias ROM (*Read Only Memory*), PROM/OTP (*Programmable Read Only Memory/One Time Programming*), EPROM (*Erasable and Programmable, Read Only Memory*) e EEPROM (*Electrical, Erasable and Programmable, Read Only Memory*). Geralmente, essas memórias são utilizadas somente para leitura e consequentemente não é possível realizar operações de escrita.

- b) *Memória de armazenamento de informações:* todo microcomputador deve ter uma área de armazenamento de informações. Essa memória deve permitir a escrita e a leitura de informações. Para entender melhor, considere um microcomputador controlando uma determinada atividade. Caso existam informações definidas pelo mundo externo por meio de chaves, sensores, teclados etc., elas devem ser lidas e armazenadas na memória, para depois serem lidas novamente e analisadas (tratadas) pelo microprocessador por meio de um programa. Como exemplo, considere o acionamento das teclas do teclado de um microcomputador, ou seja, uma vez acionada uma tecla, a CPU deverá ler a informação que foi gerada pelo hardware associado a essa tecla e executar o armazenamento dessa informação na memória. Depois, deverá fazer a leitura dessa informação, a fim de analisá-la (interpretá-la), para verificar qual tecla foi acionada. Além disso, a informação deverá ser mostrada no monitor de vídeo. Repare que foram necessárias, nesse exemplo, várias operações de leitura e escrita na memória, para a realização dessa tarefa. Assim, é preciso utilizar uma memória que tenha a característica de realizar leitura e escrita de informações. Portanto, deve-se utilizar memórias RAM (*Random Access Memory*). Lembre-se de que essas memórias apresentam a característica de serem voláteis. Portanto, sempre que o microcomputador for desenergizado, as informações contidas nesse tipo de dispositivo serão perdidas.

Unidade de entrada e saída (E/S): também chamada pela abreviação de I/O (*Input/Output*), é responsável pelo interfaceamento das informações entre o microcomputador e o mundo externo. Geralmente são circuitos integrados (CIs) capazes de ler e armazenar as informações vindas do mundo externo, por exemplo, informações originárias do teclado, do canal de comunicação serial de um microcomputador etc.; e que também podem definir informações do microcomputador para o mundo externo, por exemplo, o acionamento de bips, alto-falantes, leds indicadores, displays, interfaces de comunicação seriais e paralelas etc.

1.2.3 - Arquitetura de sistemas microprocessados/microcontrolados: a Figura 1.2 mostra como os três blocos básicos se comunicam entre si para dar funcionalidade ao microcomputador. As vias de comunicação entre os blocos básicos são chamadas de barramentos. Cada barramento é constituído por um número de vias (trilhas de cobre), nas quais fluem informações em paralelo. São três os barramentos existentes: o barramento de endereços, o barramento de temporização e controle e o barramento de dados.

Barramento de endereços: a CPU utiliza esse barramento para definir os endereços das posições de memória de programa em que ela vai buscar as instruções a serem executadas e também para definir os endereços de memória de dados ou dos dispositivos de entrada e saída para a troca de informações. Esse barramento é unidirecional.

Barramento de temporização e controle: a CPU utiliza esse barramento para definir os sinais de temporização e controle para gerenciar o tempo e a direção do fluxo de informações nas operações de leitura e escrita nos dispositivos (unidades de memória ou unidades de entrada e saída). Esse barramento é unidirecional, por exemplo, o sinal de leitura (RD/ - *read*), o sinal de escrita (WR/ - *write*) etc.

Barramento de dados: a CPU utiliza esse barramento para receber as informações vindas da memória ou dos dispositivos de entrada e saída ou então para definir as informações para a memória ou para os dispositivos de entrada e saída. Esse barramento é bidirecional. É importante frisar que, em alguns microprocessadores ou microcontroladores, a memória não troca informações diretamente com os dispositivos de entrada e saída.

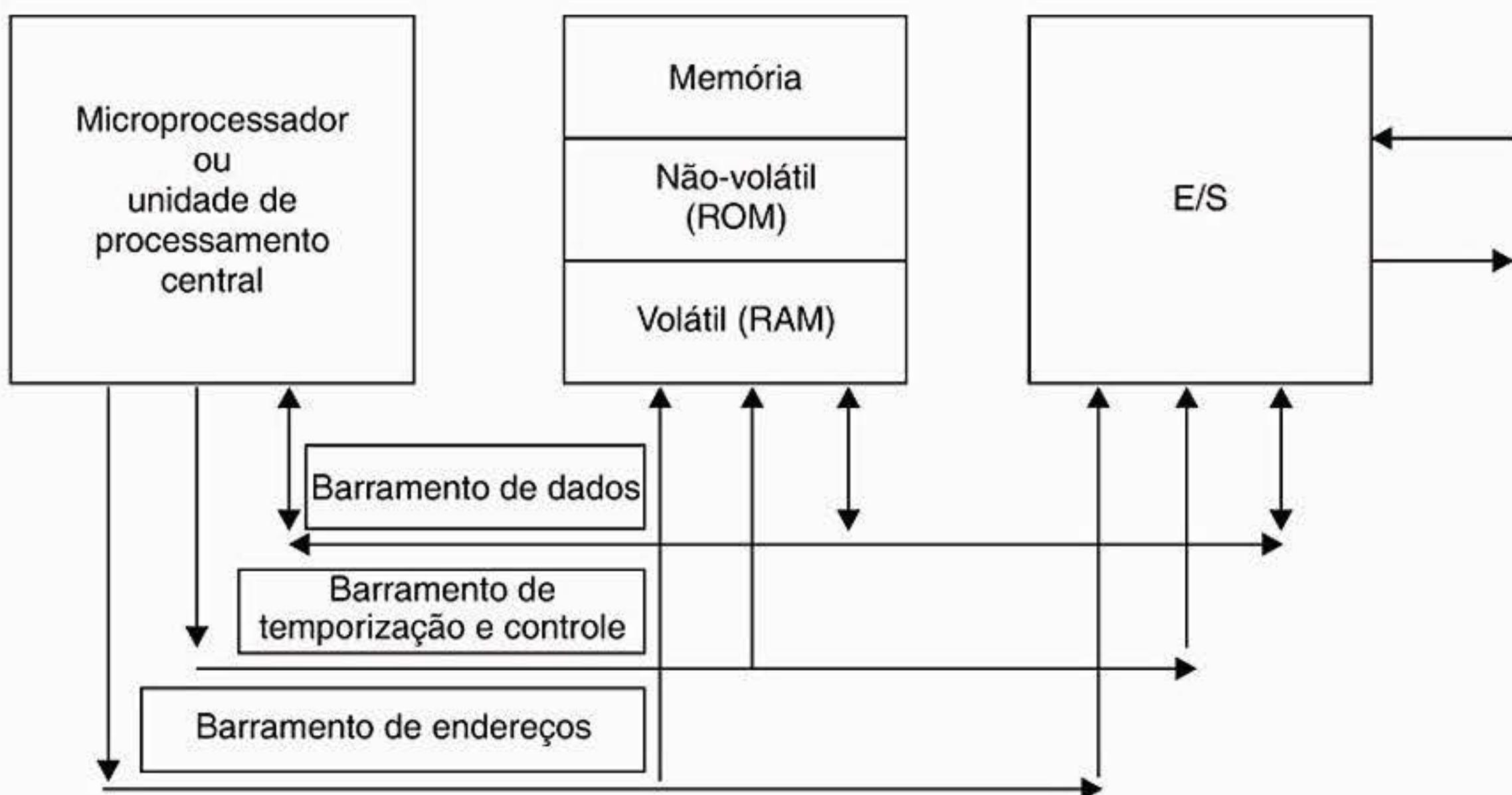


Figura 1.2 Arquitetura de sistemas microprocessados/microcontrolados.

No sistema apresentado na Figura 1.2, cada memória ou dispositivo de entrada e saída tem, pelo menos, um endereço. O microprocessador utiliza o barramento de endereços para acessar a posição de memória ou o dispositivo de entrada e saída. Assim, a seqüência de eventos para que o microprocessador execute uma operação de leitura na memória ou em um dispositivo de entrada e saída é:

- ◆ a CPU, por meio do bloco de temporização e controle, define o endereço de memória ou do dispositivo de entrada e saída no barramento de endereços por um certo tempo. Com isso, o hardware agregado ao sistema selecionará o dispositivo e uma posição de memória específica;
- ◆ a CPU, por meio do bloco de temporização e controle, define o sinal de controle de leitura (*Read-RD/*, por exemplo) no barramento de temporização e controle por um certo tempo;
- ◆ a memória ou o dispositivo de entrada e saída disponibiliza a informação (um byte, por exemplo), por um certo tempo, no barramento de dados, para que a CPU possa capturá-la (operação de leitura) e armazená-la em um dos seus registradores internos, para depois ser processada e analisada.

Da mesma maneira, a seqüência de eventos para que o microprocessador execute uma operação de escrita na memória ou em um dispositivo de entrada e saída é:

- ◆ a CPU define o endereço de memória ou do dispositivo de entrada e saída no barramento de endereços. Com isso, o hardware agregado ao sistema selecionará o dispositivo e uma posição de memória específica;
- ◆ a CPU define a informação (um byte, por exemplo) no barramento de dados;
- ◆ a CPU, por meio do bloco de temporização e controle, define o sinal de controle (*Write-WR/*, por exemplo) no barramento de temporização e controle, para que essa informação seja armazenada na memória ou no dispositivo de entrada e saída.

1.2.4 - Funcionamento de um sistema microprocessado/microcontrolado: o microcomputador é uma máquina eletrônica capaz de buscar e executar instruções de programas alocados em memória. Cada instrução tem uma função bem-definida. Após a energização de um microcomputador, é gerado um sinal de inicialização do sistema, geralmente chamado de sinal de *reset* (por exemplo, o sinal de 1 lógico por um certo tempo e depois fica estável, igual a 0 lógico), por um circuito externo obrigatório ao sistema, chamado de circuito de *reset*. Esse circuito externo está ligado ao pino *reset* do microprocessador (CPU) ou do microcontrolador (qualquer microprocessador ou microcontrolador obrigatoriamente tem esse pino). Uma das principais funções desse sinal é a inicialização de um registrador interno do microprocessador (CPU) ou do microcontrolador, chamado de contador de programa (PC - *Program Counter* ou similar). Esse registrador existe em todos os microprocessadores ou microcontroladores e, por definição, ele sempre contém o endereço da próxima instrução a ser buscada na memória de programa e executada pelo microprocessador ou microcontrolador, cujo valor inicial, após o sinal de *reset*, é predefinido pelo fabricante (p. ex.: igual a 0000h para o microprocessador 8085, como também para os microcontroladores da família MCS-51 da Intel). Para que o sistema funcione adequadamente nesse endereço é necessário que o projetista de hardware tenha mapeado uma memória não-volátil (por exemplo, EPROM de 8Kx8, cujo endereço inicial é 0000h) e que exista um programa previamente gravado a partir desse endereço de memória de programa (0000h), que definirá a funcionalidade do microcomputador. Assim, sempre que o microcomputador for energizado existirá um sinal de *reset* que inicializará o contador de programa. O microprocessador ou microcontrolador fará uma operação de leitura da instrução no endereço de memória definido pelo contador de programa (PC) e decodificará e executará essa instrução. Cada

vez que uma instrução é executada, o contador de programa (PC) é ajustado para apontar para a próxima instrução do programa. Esse processo é cíclico, e o programa alocado na memória é executado por inteiro, instrução por instrução, fazendo com que o microcomputador alcance seu objetivo funcional desejado.

Os eventos que ocorrem no sistema microprocessado/microcontrolado para que seja realizada a busca e a execução de uma única instrução são chamados de *ciclos de instrução*. O ciclo de instrução é composto por outros dois ciclos: o *ciclo de busca* e o *ciclo de execução*. Para explicar os ciclos de busca e de execução detalhadamente, considere um microprocessador de 8 bits.

A - Ciclo de busca:

A1 - De instruções de um byte (instruções que ocupam mais de uma posição de memória na memória de programa):

- a) a CPU faz a operação de leitura de um byte (código de máquina da instrução) no endereço da memória de programa especificado pelo registrador contador de programa (PC) e depois armazena esse byte no registrador de função especial, chamado registrador de instrução (IR). Ambos os registradores (PC e IR) são registradores internos de controle da CPU;
- b) a CPU decodifica e interpreta tal byte. Decodificar e interpretar um byte de instrução significa reconhecer a instrução, ou seja, se é uma operação de movimentação de informações, de adição, de subtração etc. Depois de reconhecida a instrução, a CPU gera um conjunto de sinais de controle para a execução da instrução (p. ex.: mover um byte da CPU para um endereço de memória, adicionar dois bytes etc.);
- c) a CPU adiciona uma unidade ao conteúdo do registrador *contador de programa* (PC). Isso significa definir o endereço de memória de programa da próxima instrução a ser buscada e executada no próximo ciclo de instrução.

A1 - De instruções de mais de um byte (instruções que ocupam mais de uma posição de memória na memória de programa):

- a) idem ao item a anterior;
- b) idem ao item b anterior, exceto que são feitas outras operações de leitura na memória de programa, em endereços subsequentes, dos argumentos de certas instruções dos microprocessadores ou microcontroladores

(p. ex.: um byte de valor constante a ser adicionado em algum registrador interno da CPU, uma certa posição de memória de dados em que será armazenado um dado etc.). Esses argumentos são definidos na própria memória de programa;

- c) a CPU adiciona a quantidade de bytes que compõem a instrução ao conteúdo do registrador contador de programa (PC); por exemplo, adiciona 2 ou 3 bytes ao conteúdo do registrador PC, se a instrução for de 2 ou 3 bytes, respectivamente.

B - Ciclo de execução: é o ciclo que executa a instrução. A execução consiste em operações de leitura e escrita de informações em diferentes registradores internos da CPU, entre a CPU e memórias ou dispositivos de entrada e saída, e vice-versa, em operações lógicas e aritméticas etc.

Nesse sistema microprocessado ou microcontrolado não podem existir dois dispositivos (memória ou unidade de entrada e saída) com o mesmo endereço. Isso acarretaria a queima de um dos dispositivos por curto-circuito, por meio do barramento de dados, durante uma operação de leitura de informações nesse endereço, caso esses dispositivos não apresentem as mesmas informações.

Exercícios resolvidos

1 – Dê exemplos de equipamentos inteligentes.

Resposta: os de uso pessoal: relógios digitais, calculadoras, agendas eletrônicas, telefones celulares, handhelds etc.; os de uso residencial: forno de microondas, máquina de lavar, televisão, sistemas de controle de portão automático etc.; os de uso industrial: microcomputadores aplicados ao controle de manufatura, CLPs, robôs etc.; eletrônica embarcada: alarmes de carro, computadores de bordo, injeção eletrônica etc.

2 – Quais as vantagens de se utilizar um microcomputador?

Resposta: a sua grande rapidez de processamento e sua alta confiabilidade nas tarefas executadas.

Exercícios e questões propostos

- 1 - O que é um microcomputador?
- 2 - O que é um programa de computador?
- 3 - O que é uma instrução?
- 4 - Defina software, firmware e hardware.
- 5 - Qual é a diferença entre software e firmware?
- 6 - Qual é a estrutura básica de um microcomputador? Descreva a função de cada bloco básico.
- 7 - O que é um microprocessador? Quais são os outros nomes com que geralmente ele é chamado?
- 8 - Quais são as partes que compõem o microprocessador? Quais as funções de cada parte?
- 9 - Qual é a diferença entre um microcomputador e um microprocessador?
- 10 - Qual é a diferença entre um microprocessador e um microcontrolador?
- 11 - Quais são os barramentos em um sistema microprocessado? Qual é a função de cada um deles?
- 12 - Como um programa é executado por um microcomputador?
- 13 - O que é ciclo de instrução?
- 14 - O que é ciclo de busca?
- 15 - O que é ciclo de execução?
- 16 - Descreva uma operação de escrita em memória.
- 17 - Descreva uma operação de leitura em memória.
- 18 - O que é um microcontrolador?
- 19 - Qual é a diferença entre um microcomputador e um microcontrolador?

PÁGINA EM BRANCO

2.1 Objetivos

- ❖ Apresentação da família de microcontroladores MCS-51 da Intel
- ❖ Descrição da pinagem do microcontrolador 8051/31
- ❖ Organização da memória nos dispositivos da família de microcontroladores MCS-51 da Intel
- ❖ Mapeamento de memória para uso de memória externa
- ❖ Mapeamento de E/S como memória
- ❖ Uso de E/S externa
- ❖ O sinal de inicialização (*reset*) da família de microcontroladores MCS-51 da Intel
- ❖ O sinal de relógio (*clock*) da família de microcontroladores MCS-51 da Intel
- ❖ A operação de executar um programa passo a passo
- ❖ Operação de redução da potência consumida (modo *Idle* e de baixa potência)

2.2 Introdução teórica

2.2.1 – Apresentação da família de microcontroladores MCS-51 da Intel: o microcontrolador 8051 é membro original da família de microcontroladores MCS-51 da Intel. Suas características principais são:

- ◆ CPU de 8 bits otimizada para aplicações de controle;
- ◆ poderosa capacidade de processamento booleano, incluindo lógica individual de bits;
- ◆ 64 Kbytes de endereçamento de memória de programa;

- ◆ 64 Kbytes de endereçamento de memória de dados;
- ◆ 4 Kbytes de memória de programa interna;
- ◆ 128 bytes de memória RAM de dados interna;
- ◆ 32 linhas de E/S bidirecionais, endereçáveis individualmente;
- ◆ 2 *timers*/contadores de 16 bits;
- ◆ 5 entradas de interrupções (três internas e duas externas) com dois níveis de prioridade;
- ◆ 1 oscilador interno de relógio.

A arquitetura básica do membro original da família de microcontroladores MCS-51 da Intel, ou seja, a do microcontrolador 8051, é mostrada na Figura 2.1.

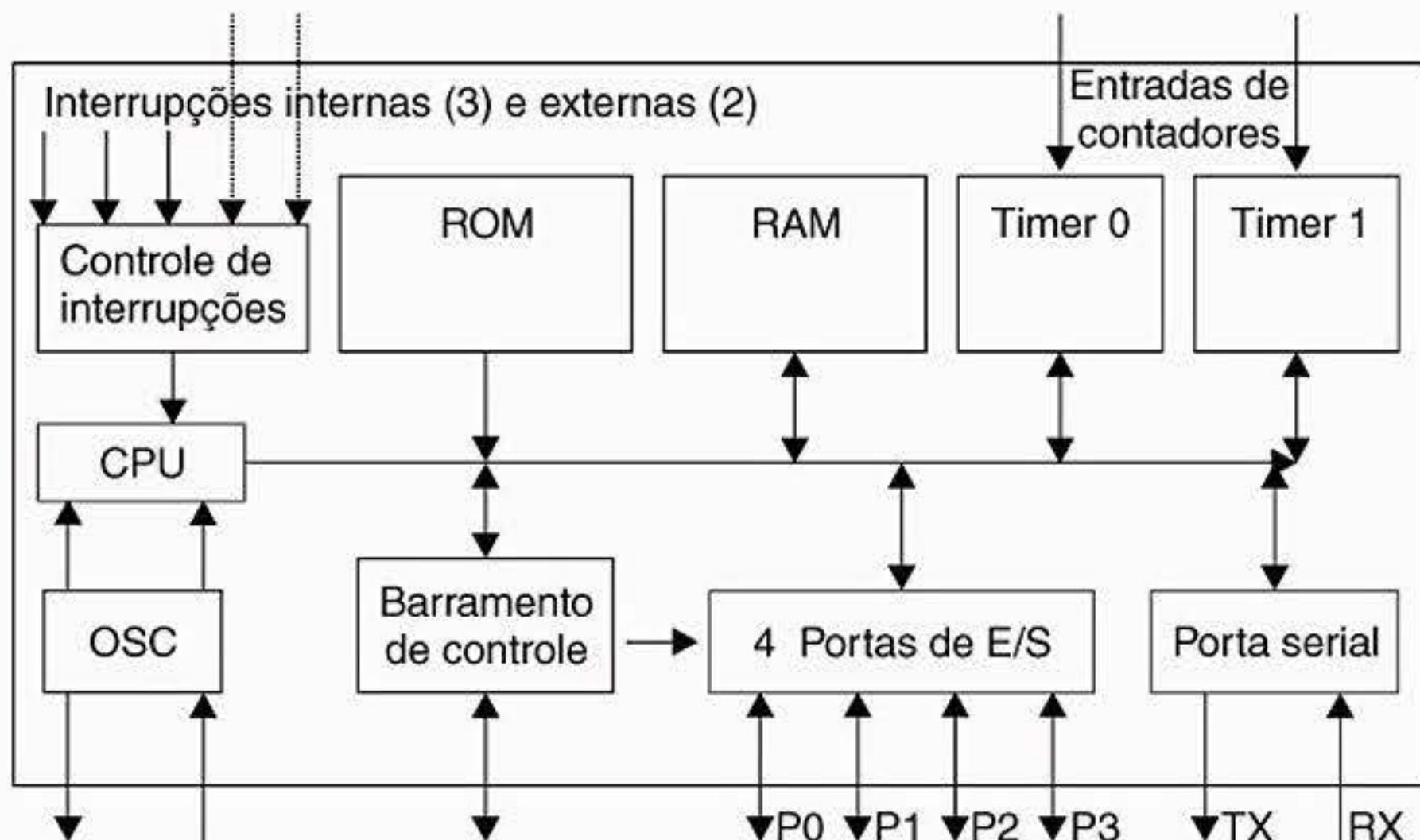


Figura 2.1 Arquitetura básica do microcontrolador 8051.

A Tabela 2.1, na página 20, apresenta a família de microcontroladores MCS-51 da Intel.

Dispositivos CHMOS: a letra C no meio do nome do 8051 (80C51) significa que os dispositivos foram implementados com tecnologia CHMOS e que são totalmente compatíveis com o 8051 além de absorverem menos corrente do que o HMOS. Eles têm dois modos de potência reduzida:

- Modo Idle* (ocioso) programado por software: a CPU é desligada, enquanto a RAM e outros dispositivos internos continuam a operar. Nesse modo, a corrente é reduzida em 15 por cento
- Modo de baixa potência* programado por software: todos os dispositivos internos são desligados. A RAM interna continua a operar. Nesse modo, o consumo de corrente elétrica é inferior a 10 uA.

Tabela 2.1 A família de microcontroladores MCS-51 de 8 bits da Intel.

Dispositivo	Versão sem ROM	Versão com EPROM	Capacidade da ROM	Capacidade da RAM (em bytes)	Portas de E/S de 8 bits	Timers/Contadores de 16 bits	Matrizes de contadores programáveis	UART	Canais de DMA	Canais A/D	Fontes de interrupção	Modos de baixo consumo e Idle
8051	8031	-	4K	128	4	2		✓			6/5	
8051AH	8031AH	8751AH	4K	128	4	2		✓			6/5	
-	-	8751BH	-	-	-	-		-			-	-
8052AH	8032AH	8752BH	8K	256	4	3		✓			8/6	
80C51BH	80C31BH	87C51	4K	128	4	2		✓			6/5	✓
80C52	80C32	-	8K	256	4	3		✓			8/6	✓
83C51FA	80C51FA	87C51FA	8K	256	4	3	✓	✓			14/7	✓
83C51FB	80C51FA	87C51FB	16K	256	4	3	✓	✓			14/7	✓
83C152JA	80C152JA	-	8K	256	5	2		✓	2		19/11	✓
-	80C152JB	-	-	256	7	2		✓	2		19/11	✓
83C152JC	80C152JC	-	8K	256	5	2		✓	2		19/11	✓
-	80C152JD	-	-	256	7	2		✓	2		19/11	✓
83C452	80C452	87C452P	8K	256	5	2		✓			9/8	✓

As letras BH são funcionalmente compatíveis com o HMOS. As diferenças são de projeto, e é possível fazer o intercâmbio entre os dispositivos HMOS e CMOS.

2.2.2 - Pinagem: o 8051/31 é um chip de 40 pinos (Figura 2.2).

A pinagem desse dispositivo é descrita na Tabela 2.2.

Entrada do sinal externo de <i>reset</i> .	9	Reset	EA/	31	Sinal de saída para acesso externo à memória de programa.
Entrada 2 do circuito oscilador externo a cristal.	18	X2	ALE	30	Sinal de saída habilitador de captura externa do byte de endereço menos significativo que está multiplexado com o byte de dados.
Entrada 1 do circuito oscilador externo a cristal.	19	X1	PSEN/	29	Sinal de saída de controle de leitura da memória de programa externa (ROM)
Porta 0 ou barramento de endereços menos significativos e dados (multiplexados): Bit 0 da porta 0 ou A0 e D0.	39	P0.0/AD0	P2.0/A8	21	Porta 2 ou barramento de endereços mais significativos: Bit 0 da porta 2 ou A8
Bit 1 da porta 0 ou A1 e D1	38	P0.1/AD1	8	P2.1/A9	Bit 1 da porta 2 ou A9
Bit 2 da porta 0 ou A2 e D2	37	P0.2/AD2		P2.2/A10	Bit 2 da porta 2 ou A10
Bit 3 da porta 0 ou A3 e D3	36	P0.3/AD3		P2.3/A11	Bit 3 da porta 2 ou A11
Bit 4 da porta 0 ou A4 e D4	35	P0.4/AD4		P2.4/A12	Bit 4 da porta 2 ou A12
Bit 5 da porta 0 ou A5 e D5	34	P0.5/AD5	0	P2.5/A13	Bit 5 da porta 2 ou A13
Bit 6 da porta 0 ou A6 e D6	33	P0.6/AD6		P2.6/A14	Bit 6 da porta 2 ou A14
Bit 7 da porta 0 ou A7 e D7	32	P0.7/AD7		P2.7/A15	Bit 7 da porta 2 ou A15
Bit 0 da porta 1	1	P1.0	5	P3.0/RXD	Bit 0 da porta 3 ou pino de recepção de dados seriais.
Bit 1 da porta 1	2	P1.1		P3.1/TXD	Bit 1 da porta 3 ou pino de transmissão de dados seriais.
Bit 2 da porta 1	3	P1.2		P3.2/INT0/	Bit 2 da porta 3 ou sinal de entrada externa da interrupção 0.
Bit 3 da porta 1	4	P1.3	1	P3.3/INT1/	Bit 3 da porta 3 ou sinal de entrada externa da interrupção 1.
Bit 4 da porta 1	5	P1.4		P3.4/T0	Bit 4 da porta 3 ou sinal de entrada de <i>clock</i> do <i>timer</i> 0.
Bit 5 da porta 1	6	P1.5		P3.5/T1	Bit 5 da porta 3 ou sinal de entrada de <i>clock</i> do <i>timer</i> 1.
Bit 6 da porta 1	7	P1.6		P3.6/WR/	Bit 6 da porta 3 ou sinal de saída de controle de escrita de memória de dados externa (RAM).
Bit 7 da porta 1	8	P1.7		P3.7/RD/	Bit 7 da porta 3 ou sinal de saída de controle de leitura de memória de dados externa (RAM).

Figura 2.2 Pinagem do 8051.

Tabela 2.2 Descrição da Pinagem do 8051/31.

Nome	Símbolo	Numeração	Descrição/Função dos Pinos
Fonte de alimentação	VCC	40	Entrada do positivo da fonte de alimentação
Terra	VSS	20	Entrada do terra do circuito (GND)
Porta 0 ou barramento de endereços menos significativos e dados multiplexados no tempo, quando se utiliza memória ROM e/ou RAM externa	P0.0 a P0.7 ou AD0-AD7	39 a 32	A porta 0 é uma porta de entrada e saída bidirecional de 8 bits de dreno aberto. Operando como uma porta de saída, cada pino pode absorver oito entradas LS TTL. Escrevendo o 1 lógico nos pinos da porta 0, eles flutuam, e esses estados podem ser utilizados como entradas de alta impedância. A porta zero é também o barramento de endereços menos significativos multiplexados no tempo com o barramento de dados durante o acesso a uma memória de programa ou de dados externa. Nessa aplicação, utilizam-se <i>pull-ups</i> internos ao escrever o 1 lógico e pode-se fornecer ou absorver até oito entradas LS TTL. A porta 0 também recebe os bytes de códigos durante a programação da EPROM e envia os bytes de códigos durante a verificação do programa gravado na ROM e EPROM. <i>Pull-ups</i> externos são necessários durante a verificação do programa.
Porta 1 ou T2, T2EX (entrada de <i>clock</i> externo do <i>timer</i> 2, somente para o 80932/52), P1.2 a P1.7	P1.0 a P1.7 e P1.0 = T2 e P1.1 = T2EX	1 a 8	A porta 1 é uma porta de entrada e saída bidirecional de 8 bits com <i>pull-ups</i> internos. Os buffers de saída podem fornecer ou absorver quatro entradas LS TTL. Os pinos da porta 1, que têm o 1 lógico em suas saídas, são levados a 1 lógico pelos <i>pull-ups</i> internos e, nesse caso, podem ser utilizados como entrada. Com a porta 1 funcionando como entrada, seus pinos, que são externamente levados para zero lógico, fornecerão corrente devido aos seus <i>pull-ups</i> internos. A porta 1 também recebe o byte de endereço menos significativo durante a programação da EPROM e durante a verificação do programa gravado da ROM e EPROM. No 8032AH/52AH, os pinos P1.0 e P1.1 da porta 1 também servem às funções de T2 e T2EX.
Porta 2 ou barramento de endereços mais significativos quando se utiliza a memória ROM e/ou RAM externa	P2.0 a P2.7 ou A8 a A15	21 a 28	A porta 2 é uma porta de entrada e saída bidirecional de 8 bits com <i>pull-ups</i> internos. Os buffers de saída podem fornecer ou absorver quatro entradas LS TTL. Os pinos da porta 2, que têm o 1 lógico em suas saídas, são levados a 1 lógico pelos <i>pull-ups</i> internos e, nesse caso, podem ser utilizados como entrada. Com a porta 2 funcionando como entrada, seus pinos, que são externamente levados para zero lógico, fornecerão corrente devido aos seus <i>pull-ups</i> internos. A porta 2 também emite o byte do endereço mais significativo durante a programação da EPROM, durante a busca da memória de programa externa e durante o acesso à memória de dados externa, que utiliza 16 bits de endereçamento (MOVX @DPTR). Nessa aplicação, a porta utiliza <i>pull-ups</i> internos quando emite 1 lógico. Durante o acesso à memória de dados externa, que utiliza endereçamento de 8 bits (MOVX @Ri), a porta 2 emite o conteúdo do registrador de função especial P2. A porta 2 também recebe o byte do endereço mais significativo durante a programação da EPROM e durante a verificação da ROM e EPROM.

Tabela 2.2 Descrição da Pinagem do 8051/31. (*Continuação*)

Nome	Símbolo	Numeração	Descrição/Função dos Pinos
Porta 3 ou pinos de receção e transmissão serial, interrupção externa 1 e 0, entrada de <i>clock</i> externa do <i>timer</i> 0 e 1, sinal de escrita e leitura de memória RAM externa	P3.0 a P3.7 ou RXD, TXD, INT0\, INT1\, T0, T1, WR\, RD\	10 a 17	A porta 3 é uma porta de entrada e saída bidirecional de 8 bits com <i>pull-ups</i> internos. Os buffer de saída podem fornecer ou absorver quatro entradas LS TTL. Os pinos da porta 3 que têm 1 lógico em suas saídas, são levadas a 1 lógico pelos <i>pull-ups</i> internos e, nesse caso, podem ser utilizados como entrada. Com a porta 3 funcionando como entrada, seus pinos, que são externamente levados para 0 lógico, fornecerão corrente devido aos seus <i>pull-ups</i> internos. A porta 3 também serve às funções de várias características da família MCS-51, como está indicado ao lado.
<i>Reset</i>	RST	9	Entrada de <i>Reset</i> . Um nível lógico alto nesse pino por dois ciclos de máquina: enquanto o oscilador está sendo executado, reseta o dispositivo (inicializa alguns registradores internos com valores predefinidos pelo fabricante).
<i>Address Latch Enable/PROG\</i> (pulso habilitador de captura de endereço)	ALE/PROG\	30	Pulso de saída que indica a um dispositivo externo que ele deve captar o sinal de endereço no barramento de endereço e os dados que estão multiplexados no tempo. Esse pino também serve como entrada do pulso de programação durante a programação da EPROM. Em operação normal, o sinal de ALE é emitido a uma razão constante de 1/6 da frequência do oscilador e pode ser utilizado para fins de <i>clock</i> ou temporizador externo. Observe que um pulso de ALE é emitido a cada acesso à memória de dados externa.
<i>Program Store Enable</i> (pulso habilitador de armazenamento de programa)	ALE/PROG\	29	É o pulso de leitura para a memória de programa externa. Quando o dispositivo está executando códigos da memória de programa externa, PSEN\ é ativado duas vezes a cada ciclo de máquina, exceto quando existe acesso à memória de dados externa.
<i>External Access Enable Programming Supply Voltage</i>	EA\ /VPP	31	EA\ deve ser ligado a VSS para habilitar o dispositivo a buscar códigos da memória de programa externa no endereço inicial de 0000h até FFFFh. EA\ deve ser ligado a VSS para a execução do programa contido na memória ROM/EPROM interna. Note, entretanto, que, se o <i>Security Bit</i> (bit de segurança) na EPROM é programado, o dispositivo não buscará códigos de qualquer local de memória de programa externo. Esse pino também recebe a fonte de alimentação de programação de 21V durante a programação da EPROM
Entrada do amplificador oscilador inversor	XTAL1	19	Entrada para o amplificador oscilador inversor

Tabela 2.2 Descrição da Pinagem do 8051/31. (*Continuação*)

Nome	Símbolo	Numeração	Descrição/Função dos Pinos
Saída do amplificador oscilador inversor	XTAL2	18	Entrada para o amplificador oscilador inversor

2.2.3 – Organização da memória nos microcontroladores da família MCS-51 da Intel: a função principal da memória de programa é armazenar o firmware dos equipamentos inteligentes e da memória de dados simplesmente para armazenar as informações (bytes) que serão utilizadas para o gerenciamento de controle que se deseja executar.

A memória da família de microcontroladores MCS-51 separa logicamente o endereçamento relativo à memória de programa e à memória de dados, como é descrito a seguir.

2.2.4 – Separação lógica da memória de programa e da memória de dados: todos os dispositivos da família de microcontroladores MCS-51 têm endereçamentos separados da memória de programa e da memória de dados, como é mostrado na Figura 2.3.

A separação lógica permite que a memória de dados seja acessada por endereçamento de 8 bits (menor tempo de acesso) e por endereçamento indireto de 16 bits, gerados por meio do registrador DPTR (*Data Pointer*).

Também é possível acessar 64 Kbytes de memória de programa. Na versão com ROM e EPROM, os primeiros 4, 8 ou 16 Kbytes são fornecidos internamente. Na versão sem ROM ou EPROM, toda a memória de programa é externa. PSEN/ (*Program Store Enable*) é o sinal de leitura para a memória de programa externa.

A memória de dados tem um endereçamento separado da memória de programa. Pode-se endereçar até 64 Kbytes de memória RAM externa. A CPU gera os sinais RD/ e WR/ durante o acesso à RAM externa.

As memórias de programa e de dados externas podem ser combinadas pela aplicação dos sinais PSEN/ e RD/ à entrada de uma porta AND e utilizar a saída como sinal de leitura para a memória de programa e dados.

Memória de programa: a Figura 2.4 mostra o mapa da parte inferior da memória de programa. Após um sinal de *reset*, o conteúdo do contador de programa (PC) é inicializado com o valor 0000h e faz com que a CPU inicie a busca e a execução do programa a partir do endereço 0000h.

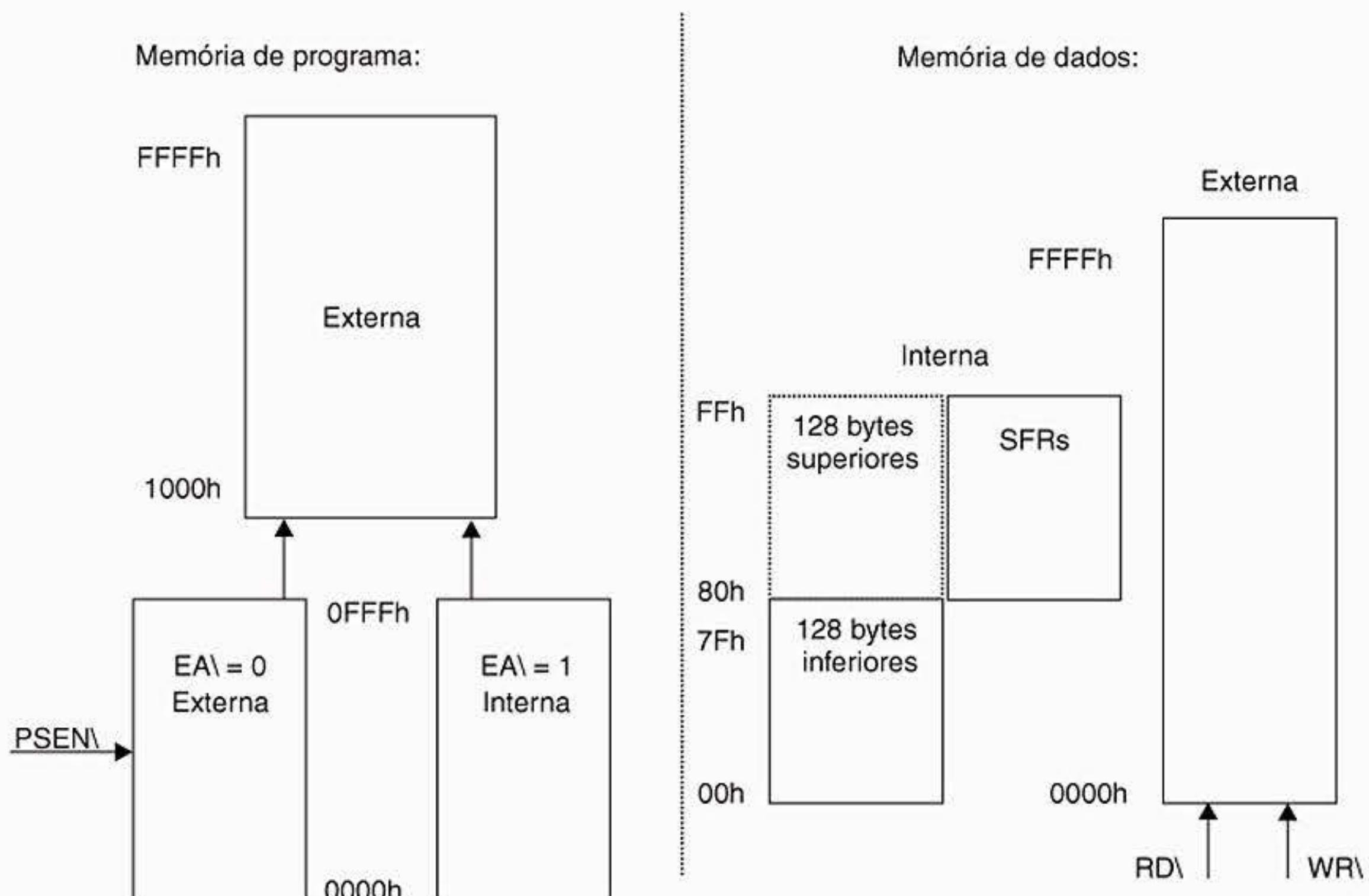


Figura 2.3 Organização da memória da família de microcontroladores MCS-51 da Intel.

Existem cinco ou seis fontes de interrupções na família de microcontroladores MCS-51, dependendo do membro da família, e essas fontes são: fonte de interrupção externa 0, fonte de interrupção do *timer/contador 0*, fonte de interrupção externa 1, fonte de interrupção do *timer/contador 1* e fonte de interrupção do canal de comunicação serial (recepção e transmissão), para os membros da família de microcontroladores que têm a terminação /51 ou /31, e mais uma fonte de interrupção do *timer/contador 2*, para os membros da família de microcontroladores que têm a terminação /52 ou /32.

As fontes de interrupções têm a finalidade de interromper o fluxo de processamento principal que está sendo executado pela CPU e desviar tal processamento para uma sub-rotina chamada *rotina de atendimento à fonte de interrupção*. Essas sub-rotinas de atendimento às fontes de interrupção devem ser armazenadas em posições de memória específicas, definidas pelo fabricante. Depois de executada a sub-rotina de atendimento à fonte de interrupção, a CPU volta a processar o programa principal a partir da posição de memória de programa em que ela foi interrompida.

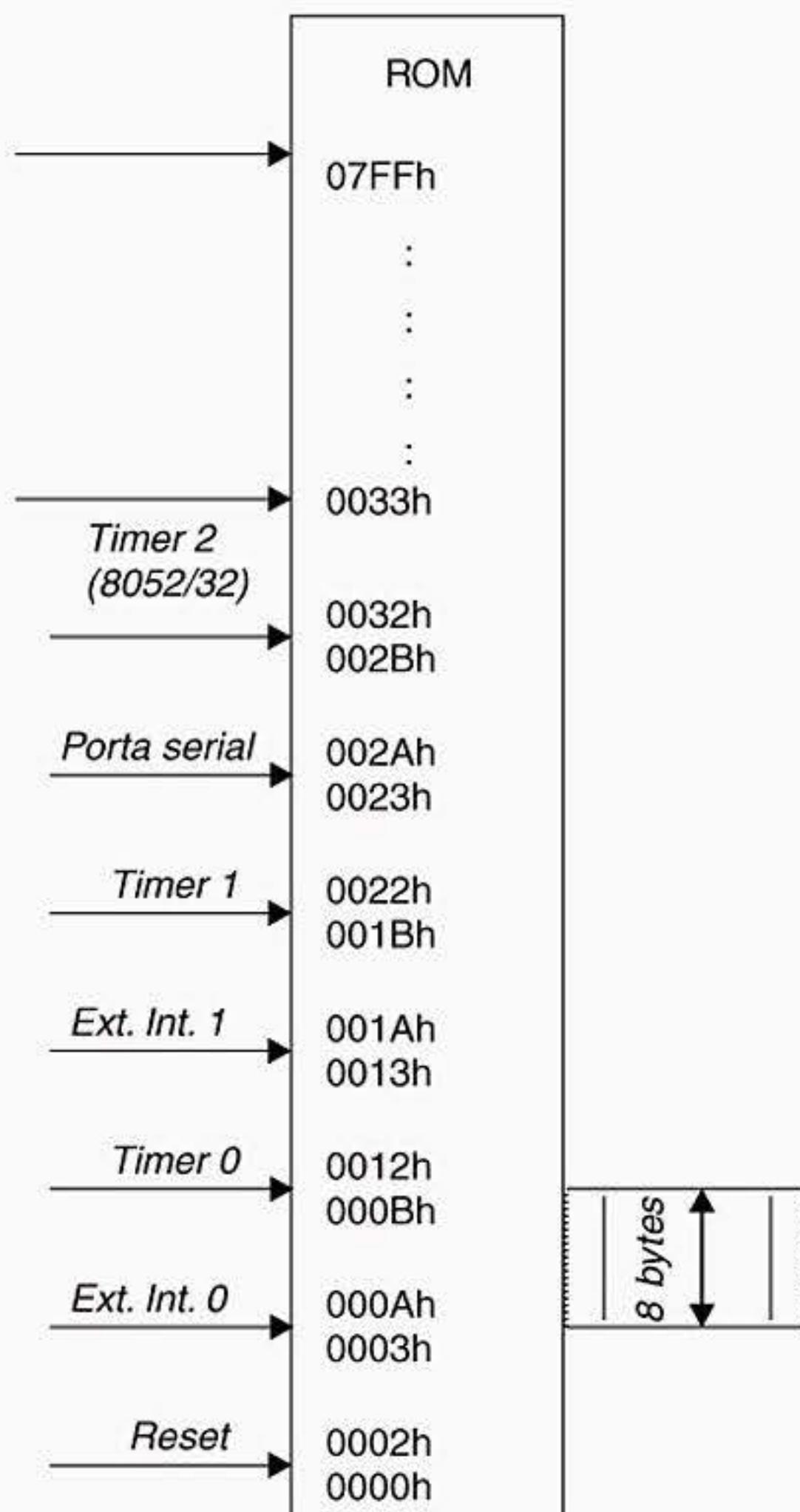


Figura 2.4 Memória de programa do MCS-51.

As fontes de interrupção são geradas por sinais elétricos que advêm de sinais elétricos criados por interfaces ligadas às fontes de interrupção externas 0 e 1 e do canal de comunicação serial ou de sinais elétricos internos ao microcontrolador, criados pelos *timers/contadores* 0, 1 e 2.

Para cada fonte de interrupção, existe um espaço reservado de memória de programa, no qual deve ser escrita uma sub-rotina de atendimento à fonte de interrupção: 0003h a 000Ah para a interrupção externa 0, 000Bh a 0012h para a interrupção do *timer/contador* 0, 0013h a 001Ah para a interrupção externa 1, 001Bh a 0022h para a interrupção do *timer/contador* 1, 0023h a 002Ah para a interrupção do canal de comunicação serial de transmissão e recepção e de 002Bh a 0032h para a interrupção do *timer/contador* 2 (sómente para o 8032/52).

O espaço disponível para escrever as rotinas de atendimento para cada fonte de interrupção é de 8 bytes (p. ex.: para a interrupção 0, o espaço vai do endereço 0003h a 000Ah; para a interrupção 1, o espaço vai do endereço 000Bh a 0012h etc.).

No caso de uma sub-rotina ocupar mais de 8 bytes, ela pode continuar em outra posição de memória, utilizando uma instrução de salto incondicional dentro do espaço alocado para cada fonte de interrupção ou estar alocada inteiramente em outra posição de memória, na qual ela será ativada também pelo uso de uma instrução de salto incondicional, situada no endereço inicial do local em que se escreve a sub-rotina de atendimento à fonte de interrupção. A segunda maneira é a mais usual.

Os 4 Kbytes, 8 Kbytes ou 16 Kbytes da memória de programa, dependendo do membro da família, podem estar na ROM interna ou na ROM externa. Essa seleção é feita ao se ligar o pino EA\ ou EA_{barra} (*External Access*) no VCC ou no GND, respectivamente.

Nos dispositivos com 4 Kbytes de ROM, se EA_{barra} for ligado ao VCC, a CPU buscará o programa do endereço de 0000h a 0FFFh na ROM interna e buscará o endereço de 1000h a FFFFh na ROM externa.

Nos dispositivos com 8 Kbytes de ROM, se EA_{barra} for ligado ao VCC, a CPU buscará o programa do endereço de 0000h a 1FFFh na ROM interna e buscará o endereço de 2000h a FFFFh na ROM externa.

Nos dispositivos com 16 Kbytes de ROM, se EA_{barra} for ligado ao VCC, a CPU buscará o programa do endereço de 0000h a 3FFFh na ROM interna e buscará o endereço de 4000h a FFFFh na ROM externa.

Se EA_{barra} for ligado ao VSS (GND), a CPU fará a busca na memória ROM externa. Para os dispositivos da família MCS-51, sem ROM interna, esse pino deve ser sempre ligado ao VSS, para que o programa seja buscado integralmente na memória ROM externa e, assim, funcione adequadamente.

O sinal de controle de leitura da ROM externa é o sinal PSEN\=PSEN_{barra}, que não é ativo para a busca das instruções na memória de programa interna.

Na Figura 2.5 é mostrada a configuração do hardware para a utilização de memória de programa externa.

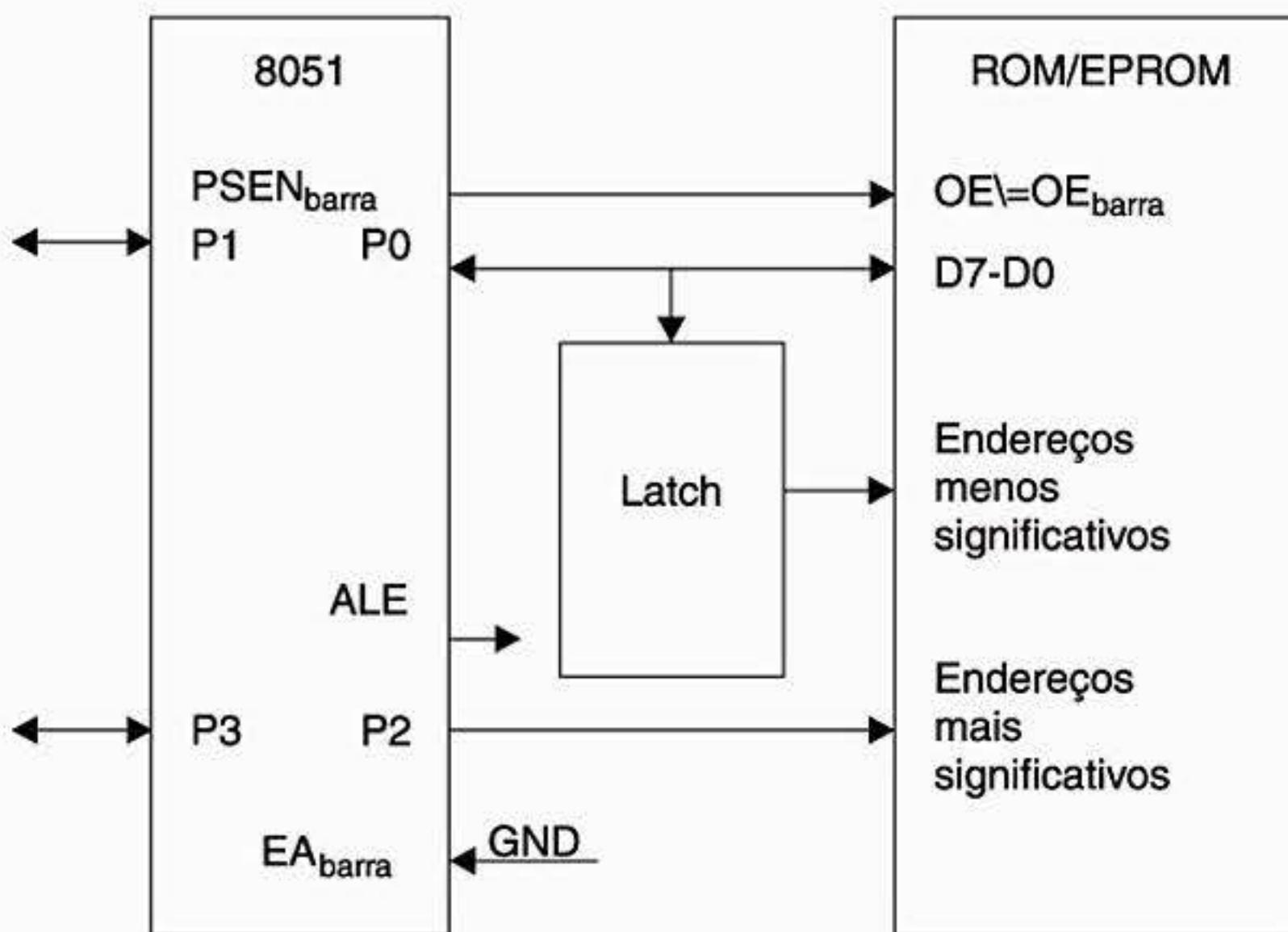


Figura 2.5 Hardware para memória de programa externa.

Observe que dezesseis linhas de entrada e saída são dedicadas à função especial de barramento de endereços e dados para a busca do programa na memória de programa externa. A porta 0 funciona como um barramento formado pelos endereços menos significativos multiplexados com os dados. Ele emite primeiro a informação definida pelo PCL (parte menos significativa do contador de programa) como um endereço e depois fica flutuante, esperando a chegada de um byte de código da memória de programa. Durante o tempo em que o endereço menos significativo do contador de programa (PCL) é válido em P0, um sinal de *sincronismo* de ALE (Address Latch Enable) permite a captura desses sinais em um dispositivo *latch* externo.

A porta 2 emite o byte mais significativo do contador de programa (PCH) durante todo o tempo da operação de leitura.

Depois disso, o sinal de PSEN\ é gerado para habilitar a operação de leitura da ROM/EPROM, e o byte de código é lido a partir da memória de programa para o microprocessador do microcontrolador.

O endereçamento da memória de programa é sempre de 16 bits de largura.

Memória de dados: a Figura 2.6 mostra uma configuração de hardware para acessar até 2 Kbytes de RAM externa. A CPU, nesse caso, está executando um programa da ROM interna. A porta 0 serve como barramento do byte de endereços menos significativos e do byte de dados multiplexados, e as três linhas da porta 2 estão sendo utilizadas para paginar a RAM

de 256 em 256 bytes. A CPU gera os sinais de controle de leitura $RD\bar{ } = RD_{\text{barra}}$ e de escrita $WR\bar{ } = WR_{\text{barra}}$ para acessar a RAM externa.

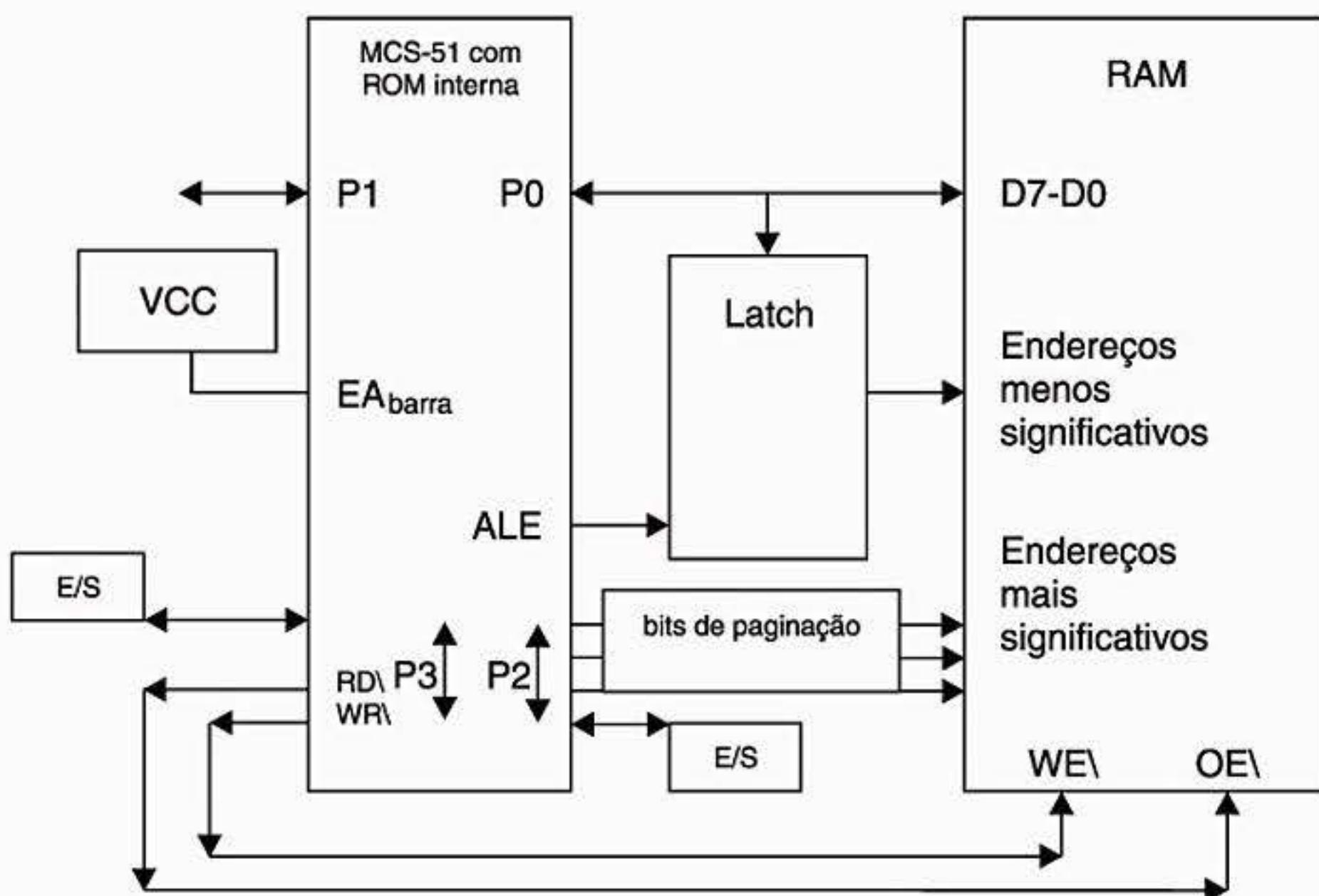


Figura 2.6 Hardware para memória de dados externa.

Observação: se a memória de programa é interna ($EA_{\text{barra}} = 1$), os bits da porta 2 são definidos como portas de entrada e saída e NÃO emitem o valor de PCH (bits de endereços mais significativos).

Podem existir até 64 Kbytes de memória de dados externa. A largura dos endereços da memória de dados externa pode ser de 1 ou 2 bytes. A largura de endereços de 1 byte geralmente é utilizada com a combinação de uma ou mais linhas de entrada e saída para 'paginar' o endereçamento da RAM de 256 em 256 bytes. A largura de endereços de 2 bytes também pode ser utilizada e, nesse caso, os 8 bits mais significativos do barramento de endereços são emitidos pela porta 2 ($EA_{\text{barra}} = 0$).

A memória de dados interna está mapeada na Figura 2.7. O espaço de memória é dividido em três blocos:

- ◆ os 128 bytes inferiores;
- ◆ os 128 bytes superiores;
- ◆ os SFRs (*Special Function Registers*), registradores de funções especiais.

A memória de dados interna é sempre endereçada com 8 bits de largura. Isso implica endereçar somente 256 bytes de memória. Entretanto, os

modos de endereçamento para a RAM interna podem acomodar até 384 bytes (256 bytes de memória RAM e 128 bytes disponibilizados para os registradores de funções especiais), utilizando um truque simples. Por meio do modo de endereçamento direto superior a 7Fh, é acessado um espaço de memória; e um modo de endereçamento indireto superior a 7Fh acessa um espaço de endereçamento de memória diferente.

A Figura 2.7 também mostra os 128 bytes superiores e o espaço dos SFRs ocupando o mesmo bloco de endereços (de 80h a FFh), porém são entidades separadas fisicamente.

Os 128 bytes inferiores da RAM estão presentes em todos os microcontroladores da família MCS-51 da Intel e estão divididos em três partes principais, como está representado na Figura 2.8.

A primeira parte é constituída por quatro bancos de registradores de 8 bits, que ocupam os endereços de 00h a 1Fh.

A segunda parte é constituída por 16 bytes, que ocupam os endereços de 20h a 2Fh, em que essas posições de memória podem ser acessadas por bytes ou bits.

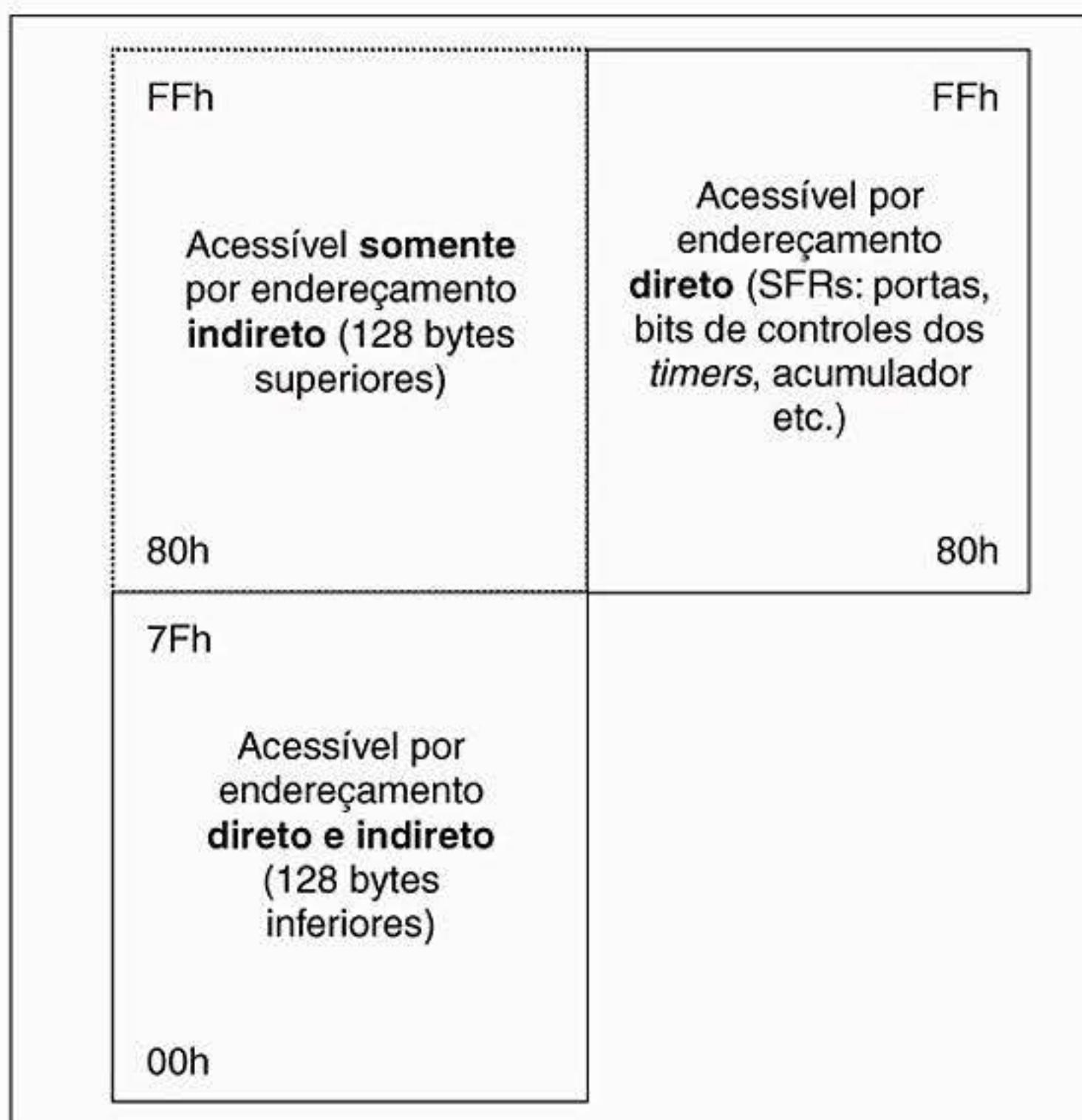


Figura 2.7 Memória de dados interna.

	7Fh	Endereçável por byte
	30h	
	2Fh	Endereçável por bit e/ou byte
	20h	
Valor inicial do SP (<i>Stack Pointer</i> – ponteiro de pilha) após um sinal de reset.	1Fh	R7
	18h	Banco 3 R0
	17h	R7
	10h	Banco 2 R0
	0Fh	R7
	08h	Banco 1 R0
	07h	Banco 0 R7
	00Hh	R0

Figura 2.8 128 bytes inferiores da RAM interna.

A terceira parte é a que ocupa os endereços de 30h a 7Fh, em que são acessadas simplesmente por byte.

Os 32 bytes inferiores, cujos endereços vão de 00h até 1Fh, estão agrupados em quatro bancos de oito registradores. As instruções de programa referenciam esses registradores como R0, R1, R2, R3, R4, R5, R6 e R7.

Esses bancos de registradores só podem ser acessados um de cada vez. A seleção do banco é feita por meio de 2 bits selecionadores de bancos de registradores chamados *RS1* e *RS0*, localizados nos bits 3 e 4 do registrador de função especial PSW (*Program Status Word*). Dependendo dos valores de *RS1* e *RS2*, será selecionado um único banco de registradores para serem acessados, como foi representado anteriormente (Figura 2.9).

O uso desses registradores permite uma maior eficiência dos programas, já que as instruções que utilizam tais registradores são mais velozes, pois são acessadas por endereçamento direto e estão próximas à CPU, além de ocuparem menos posições de memória de programa.

bits (PSW) =	7	6	5	4	3	2	1	0
	C	AC	F0	RS1	RS0	OV	-	P

RS1	RS0	Banco selecionado	Registradores selecionados	Endereços de memória selecionados
0	0	0	R0 a R7	00h a 07h
0	1	1	R0 a R7	08h a 0Fh
1	0	2	R0 a R7	10h a 17h
1	1	3	R0 a R7	18h a 1Fh

RS1 RS0	Endereço	Registrador	Banco
1 1	1Fh	R7	3
1 1	1Eh	R6	3
1 1	1Dh	R5	3
1 1	1Ch	R4	3
1 1	1Bh	R3	3
1 1	1Ah	R2	3
1 1	19h	R1	3
1 1	18h	R0	3
1 0	17h	R7	2
1 0	16h	R6	2
1 0	15h	R5	2
1 0	14h	R4	2
1 0	13h	R3	2
1 0	12h	R2	2
1 0	11h	R1	2
1 0	10h	R0	2
0 1	0Fh	R7	1
0 1	0Eh	R6	1
0 1	0Dh	R5	1
0 1	0Ch	R4	1
0 1	0Bh	R3	1
0 1	0Ah	R2	1
0 1	09h	R1	1
0 1	08h	R0	1
0 0	07h	R7	0
0 0	06h	R6	0
0 0	05h	R5	0
0 0	04h	R4	0
0 0	03h	R3	0
0 0	02h	R2	0
0 0	01h	R1	0
0 0	00h	R0	0

Figura 2.9 Banco de registrador.

Os 16 bytes seguintes, acima dos bancos de registradores, cujos endereços de memória vão de 20h a 2Fh, formam um bloco de memória que é endereçado por byte e por bit. Os endereços desses bits vão de 00h a 7Fh e são representados como é indicado na Figura 2.10.

Endereço de memória	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
2Fh	7Fh	7Eh	7Dh	7Ch	7Bh	7Ah	79h	78h
2Eh	77h	76h	75h	74h	73h	72h	71h	70h
2Dh	6Fh	6Eh	6Dh	6Ch	6Bh	6Ah	69h	68h
2Ch	67h	66h	65h	64h	63h	62h	61h	60h
2Bh	5Fh	5Eh	5Dh	5Ch	5Bh	5Ah	59h	58h
2Ah	57h	56h	55h	54h	53h	52h	51h	50h
29h	4Fh	4Eh	4Dh	4Ch	4Bh	4Ah	49h	48h
28h	47h	46h	45h	44h	43h	42h	41h	40h
27h	3Fh	3Eh	3Dh	3Ch	3Bh	3Ah	39h	38h
26h	37h	36h	35h	34h	33h	32h	31h	30h
25h	2Fh	2Eh	2Dh	2Ch	2Bh	2Ah	29h	28h
24h	27h	26h	25h	24h	23h	22h	21h	20h
23h	1Fh	1Eh	1Dh	1Ch	1Bh	1Ah	19h	18h
22h	17h	16h	15h	14h	13h	12h	11h	10h
21h	0Fh	0Eh	0Dh	0Ch	0Bh	0Ah	09h	08h
20h	07h	06h	05h	04h	03h	02h	01h	00h

Figura 2.10 Faixa de memória endereçável por byte e bit.

Observe que existem 128 endereços de bits nessa faixa de memória, que vão do endereço 00h até 7Fh, ou em decimais, de 0_{10} a 127_{10} , que ocupam as posições de memória RAM interna de 20h a 2Fh.

Pode-se endereçar um bit de duas maneiras diferentes. Uma delas é por meio do endereço do bit, que vai de 00h = 0_{10} (bit 0 da posição de memória 20h) até o endereço 7Fh = 128_{10} (bit 7 da posição de memória 2Fh), como foi representado anteriormente. A outra maneira de representá-lo é por meio do endereço de memória a que ele pertence e de sua localização dentro dessa posição de memória; por exemplo, o bit cujo endereço é 00h também pode ser representado como 20h.0, ou seja, esse bit está localizado no conteúdo da posição de memória 20h e sua posição corresponde ao bit menos significativo ou ao bit 0. Da mesma maneira, o bit cujo endereço é 6Ch também pode ser representado pelo endereço 2Dh.4.

Caso se conheça uma das duas maneiras de endereçamento, é possível obter a outra. A primeira conversão, da representação ZWh para a representação 2Xh.Y ($ZWh \rightarrow 2Xh.Y$), pode ser feita de acordo com o seguinte procedimento:

Como são 16 as posições de memória que vão do endereço 20h até 2Fh, e como cada posição de memória tem 8 bits, pode-se representar o endereço de um bit, cujo endereço vai de 00h até 7Fh, em função do endereço de memória

e de sua posição dentro desse endereço, transformando o endereço do bit (00h - 7Fh) em decimal, dividindo-o por 8, obtendo-se o quociente (q) e o resto (r) em hexadecimal e por fim, aplicando-se a fórmula: $[20h + qh] h \cdot r = 2qh.r$.

Exemplo: Obter a outra representação do endereço do bit 5Dh?

Solução:

- ◆ Transformar o endereço do bit de hexadecimal para decimal: $5 \cdot 16^1 + 13 \cdot 16^0 = 93_{10}$.
- ◆ Dividir o valor do endereço em decimal (93_{10}) por 8: $93 / 8$, obtendo o quociente $q = 11_{10}$ e o resto $r = 5_{10}$.
- ◆ Transformar o quociente (q) e o resto (r) em hexadecimal: $q = 11_{10} = B_{16}$ e $r = 5_{10} = 5_{16}$.
- ◆ Aplicar a fórmula: $5Dh = [20h + Bh]h.5 = 2Bh.5$, ou seja, o endereço do bit 5Dh é igual a 2Bh.5.

Para transformar o endereço de um bit da representação $2Xh.Y$ para ZWh ($2Xh.Y \rightarrow ZWh$), basta seguir o seguinte procedimento:

- ◆ Obter o quociente e o resto. O quociente (q) é obtido subtraindo $2Xh$ de $20h$ e transformando-o em decimal, e o resto é obtido através do número após o ponto (.) de tal representação. Multiplicando o quociente em decimal por 8 e somando com o resto em decimal, obtém-se o endereço em decimal. O valor do endereço em decimal deve ser transformado para hexadecimal.

Exemplo: Como obter a outra maneira de representar o endereço do bit 2Ah.3?

- ◆ Obter o quociente (q) e o resto (r): $q = 2Ah - 20h = Ah$ e $r = 3h$.
- ◆ Transformar q e r de hexadecimal para decimal:
 $q = Ah = 10_{10}$ e $r = 3h = 3_{10}$.
- ◆ Multiplicar o quociente por 8 e somar o resultado com o resto em decimal: $10 \cdot 8 + 3 = 83_{10}$.
- ◆ Transformar o endereço de decimal para hexadecimal:
 $83_{10} = 5 \cdot 16^1 + 3 \cdot 16^0 = 53h$.

Essas posições de memória podem ser utilizadas como posições de memória de 8 bits, com propósito geral, ou podem ser empregadas principalmente como bits individuais (0 - *resetado*, 1 - *setado*). Elas podem ser testadas por instruções de testes de bits do tipo JB bit e JNB bit. Esses bits são muito

utilizados para representar condições digitais, ou seja, como *flags* sinalizadores de condição (bits que sinalizam condições de controle), por exemplo, sistema de iluminação ligado ou desligado, chave acionada ou desacionada, resultado de uma operação maior que zero ou menor que zero etc.).

A terceira faixa de memória, que vai do endereço 30h até 7Fh para o microcontrolador 8051 e de 30h até FFh para os membros da família de microcontroladores MCS-51 da Intel, que têm 256 bytes de memória interna, só pode ser acessada de 8 em 8 bits, por endereçamento direto até 7Fh e por endereçamento indireto de 80h a FFh. Seu uso é de propósito geral.

A faixa de 80h até FFh, para o microcontrolador 8051, contempla os registradores de função especiais, chamados de SFRs (*Special Function Registers*). A Figura 2.11 mostra alguns registradores de funções especiais (SFRs). Esses registradores só podem ser acessados por endereçamento direto, e alguns deles também são endereçáveis por bit.

FFh	
E0h: ACC	
B0h: Porta 3	
A0h: Porta 2	
90h: Porta 1	
80h: Porta 0	

Endereços que terminam com 0h ou 8h são também endereçáveis por bit.

Figura 2.11 Endereçamento de alguns SFRs.

As tabelas 2.3 e 2.4 mostram os registradores de função especiais (SFRs) com seus respectivos endereçamentos e nomes.

Tabela 2.3 Registradores de funções especiais (SRFs).

Registradores	Endereço	Nome dos registradores
A ou ACC	E0h	Acumulador
B*	F0h	Registrador B
DPL	82h	Byte menos significativo do ponteiro de dados
DPH	83h	Byte mais significativo do ponteiro de dados
IE*	A8h	Habilitador de interrupções
IP	B8h	Priorizador de interrupções
SCON*	98h	Controlador da comunicação serial
SBUF	99h	Buffer de dados serial
PSW*	D0h	Palavra de status de programa
PCON	87h	Controle de potência
TCON*	88h	Controle do <i>timer</i> /contador
TMOD	89h	Modo de operação de <i>timer</i> /contador
TH0	8Ch	Byte mais significativo do <i>timer</i> /contador 0
TL0	8Ah	Byte menos significativo do <i>timer</i> /contador 0
TH1	8Dh	Byte mais significativo do <i>timer</i> /contador 1
TL1	8Bh	Byte menos significativo do <i>timer</i> /contador 1
P0*	80h	Porta 0
P1*	90h	Porta 1
P2*	A0h	Porta 2
P3*	B0h	Porta 3

Tabela 2.4 Registradores de funções especiais somente disponíveis para a família 8052.

Registradores	Endereço	Nome dos registradores
T2CON*	C8h	Controle de <i>timer</i> /contador 2
TH2	CDh	Byte mais significativo do <i>timer</i> /contador 2
TL2	CCh	Byte menos significativo do <i>timer</i> /contador 2
RCAP2H	CBh	Byte mais significativo do <i>timer</i> /contador de captura 2
RCAP2L	Cah	Byte menos significativo do <i>timer</i> /contador de captura 2

* Registradores endereçados por bit.

2.2.5 - Mapeamento de memória para uso de memória externa: cada célula de memória e cada registrador de um dispositivo externo, mapeado como memória, deve ter pelo menos um endereço.

Se houver mais de um dispositivo respondendo a um determinado endereço, isso causará um conflito de sinais no barramento de dados e, consequentemente, a queima desses dispositivos. O projetista de hardware, responsável pelo projeto do sistema de decodificação dos endereços, deve evitar que dispositivos diferentes tenham endereços iguais.

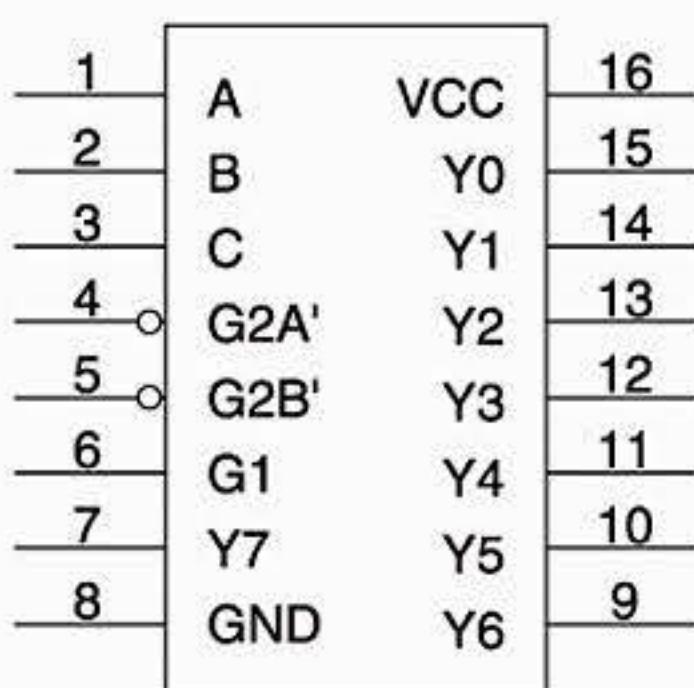
Geralmente, um sistema com microprocessador ou microcontrolador tem mais que um dispositivo de memória de entrada e saída. Assim, para cada dispositivo, deve haver endereçamentos diferentes entre eles, para que não existam conflitos de sinais. O sistema de endereçamento deve selecioná-los quando os endereçamentos aparecerem no barramento de endereços do microprocessador ou microcontrolador. O método tradicional para fazer a decodificação dos endereços é por meio do uso de um decodificador para separar o espaço de endereçamento em partes iguais. Um decodificador muito utilizado, na prática, para fazer a decodificação do espaçamento de memória ou de dispositivos de entrada e saída (mapeamento de memória ou de dispositivos de entrada e saída) é o decodificador de três entradas e oito saídas, o 74138. A Tabela 2.5 mostra sua 'tabela-verdade'.

Tabela 2.5 Tabela-verdade do 74138.

Entradas							Saídas							
G2A'	G2B'	G1	A2	A1	A0		Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7
1	X	X	X	X	X		1	1	1	1	1	1	1	1
X	1	X	X	X	X		1	1	1	1	1	1	1	1
X	X	0	X	X	X		1	1	1	1	1	1	1	1
0	0	1	0	0	0		0	1	1	1	1	1	1	1
0	0	1	0	0	1		1	0	1	1	1	1	1	1
0	0	1	0	1	0		1	1	0	1	1	1	1	1
0	0	1	0	1	1		1	1	1	0	1	1	1	1
0	0	1	1	0	0		1	1	1	1	0	1	1	1
0	0	1	1	0	1		1	1	1	1	1	0	1	1
0	0	1	1	1	0		1	1	1	1	1	1	0	1
0	0	1	1	1	1		1	1	1	1	1	1	1	0

0: 0 lógico; 1: 1 lógico; X: pode ser 0 ou 1 lógico; ': representa a entrada ativa em 0 lógico.

A representação esquemática e a pinagem do 74138 são mostradas na Figura 2.12.



74138

Figura 2.12 Decodificador 74138.

Esse decodificador não habilita nenhuma saída, caso uma das entradas habilitadoras, $G2A'$ ou $G2B'$, seja igual a 1 lógico ou se $G1$ for igual a 0 lógico. Quando $G2A' = G2B' = 0$ lógico e $G1 = 1$ lógico, somente uma saída em 0 lógico é ativada, correspondente à combinação das entradas $A0$, $A1$ e $A2$ (p. ex.: $A2 = A1 = A0 = 0$ faz com que $Y0 = 0$ e as demais saídas fiquem iguais a 1 lógico; $A2 = A1 = A0 = 1$ fará com que $Y7 = 0$ e as demais saídas fiquem iguais a 1 lógico etc.).

Assim, considere um exemplo de mapeamento de memória ou dispositivo de entrada e saída, com os sinais do 8051 ligados ao 74138, como está representado na Figura 2.13.

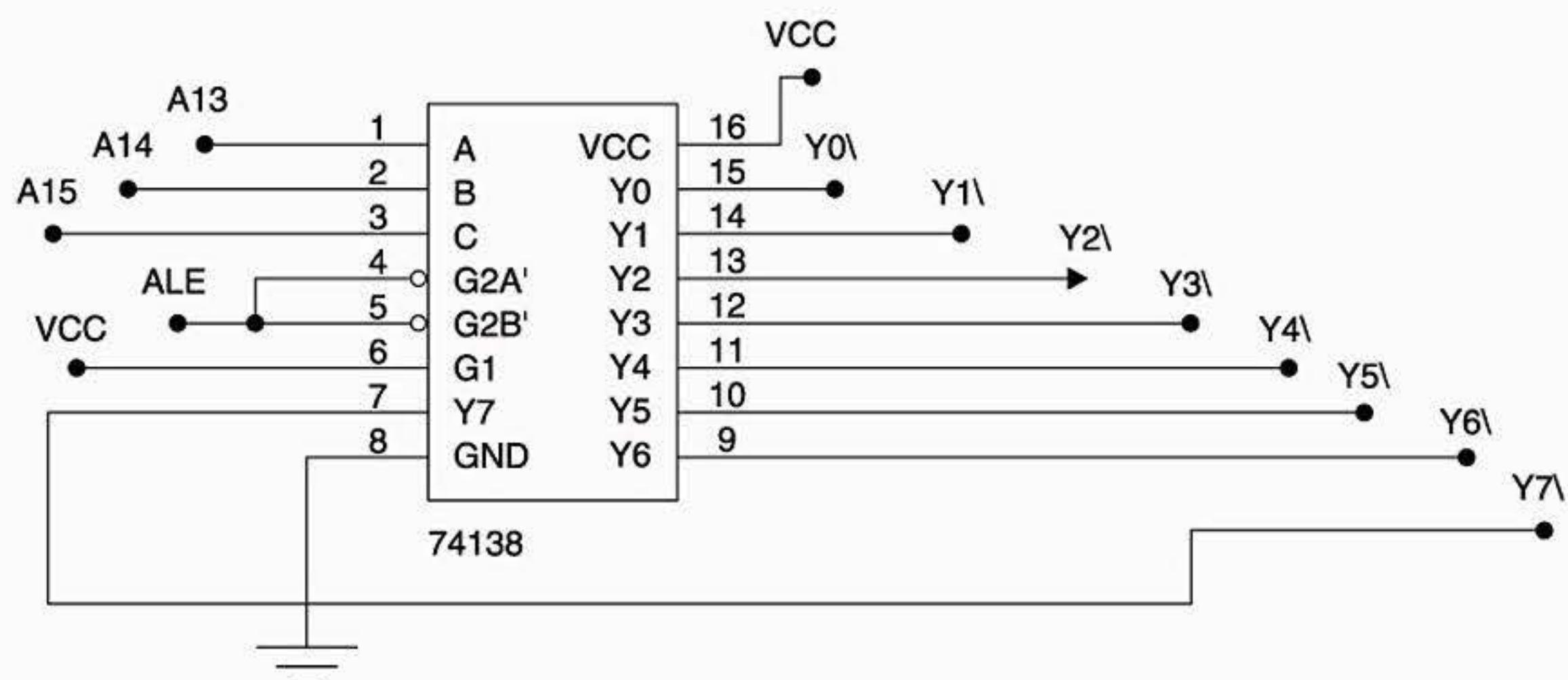


Figura 2.13 Exemplo de mapeamento de memória com o 74138.

Com esse hardware, o mapeamento de memória ou de dispositivo de entrada e saída, gerado por cada saída $YX\backslash$ ($X = 0, 7$), é mostrado na Tabela 2.6.

Por meio dessa tabela, pode-se afirmar que:

- A saída $Y0\backslash$ é habilitada em 0 lógico sempre que no barramento de endereços do microcontrolador forem definidos os endereços da faixa de 0000h a 1FFFh. A saída $Y1\backslash$ é habilitada em 0 lógico sempre que no barramento de endereços do microcontrolador forem definidos os endereços da faixa de 2000h a 3FFFh e assim por diante.
- Para esse hardware, cada saída do decodificador, quando habilitada, define uma faixa de 8 Kbytes de endereços. Ou seja, pode-se dizer que esse sistema de mapeamento divide os 64 Kbytes possíveis em oito partes, e que cada parte tem uma largura de 8 Kbytes de endereços.

Tabela 2.6 Mapa de endereçamento de memória ou dispositivo de E/S mapeado com memória.

A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	Endereço inicial e final	Saída do decodificador habilitada	Quantidade de bytes por saída
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000H	Y0\=0	
0	0	0	X	X	X	X	X	X	X	X	X	X	X	X	0	Y0\=0	8K	
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1FFFH	Y0\=0	
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	2000H	Y1\=0	
0	0	1	X	X	X	X	X	X	X	X	X	X	X	X	0	Y1\=0	8K	
0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	3FFFH	Y1\=0	
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4000H	Y2\=0	
0	1	0	X	X	X	X	X	X	X	X	X	X	X	X	0	Y2\=0	8K	
0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	5FFFH	Y2\=0	
0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	6000H	Y3\=0	
0	1	1	X	X	X	X	X	X	X	X	X	X	X	X	0	Y3\=0	8K	
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	7FFFH	Y3\=0	
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	8000H	Y4\=0	
1	0	0	X	X	X	X	X	X	X	X	X	X	X	X	0	Y4\=0	8K	
1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	9FFFH	Y4\=0	
1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	A000H	Y5\=0	
1	0	1	X	X	X	X	X	X	X	X	X	X	X	X	0	Y5\=0	8K	
1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	BFFFH	Y5\=0	
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C000H	Y6\=0	
1	1	0	X	X	X	X	X	X	X	X	X	X	X	X	0	Y6\=0	8K	
1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	DFFFH	Y6\=0	
1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	E000H	Y7\=0	
1	1	1	X	X	X	X	X	X	X	X	X	X	X	X	0	Y7\=0	8K	
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	FFFFH	Y7\=0	
Dígito + sign. do endereço				Dígito + sign. do endereço				Dígito - sign. do endereço				Dígito - sign. do endereço						

- c) Se, por exemplo, for definido o endereço 912Dh no barramento de endereços do microcontrolador, será habilitada a saída Y4\ do decodificador. Se, por exemplo, for definido o endereço E999h no barramento de endereços do microcontrolador, será habilitada a saída Y7\ do decodificador e assim por diante.

	C	B	A	Decodificador 74138														
End.	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	Y	
912Dh	1	0	0	1	0	0	0	1	0	0	1	0	1	1	0	1	Y4\	
E999h	1	1	1	0	1	0	0	1	1	0	0	1	1	0	0	1	Y7\	

- d) Observe que os endereços A15, A14 e A13 são os bits que definem a faixa de endereçamento de cada saída do decodificador.
- e) Os bits A0 a A12 NÃO influenciam a ativação das saídas dos decodificadores.

É importante dizer que qualquer dispositivo ligado a um sistema microprocessado ou microcontrolado deve ter um pino físico chamado *chip enable* ou similar. Esse pino faz com que o dispositivo desempenhe sua função somente quando o mesmo estiver habilitado.

Assim, conectando, por exemplo, a saída Y0\ ao *chip enable* de uma memória de 8Kx8, os endereços dessa memória irão de 0000h a 1FFFh. Ou se, por exemplo, a saída Y6\ for conectada a uma memória de 2Kx8, os endereços dessa memória irão de C000h a C7FFh ($= C000h + 2K = C000h + 07FFh$), de C800h a CFFFh ($= C800h + 2K = C800h + 07FFh$), de D000h a D7FFh ($= D000h + 2K = D000h + 07FFh$) ou, ainda, de D800h a DFFFh ($= D800h + 2K = D800h + 07FFh$).

2.2.6 – Mapeamento de interfaces de E/S como memória: pode-se mapear uma interface de E/S como memória utilizando-se a saída do decodificador conectada a um circuito integrado que tem a função de ser um dispositivo de entrada e saída. O endereçamento da E/S é obtido da mesma maneira como foi obtido para o caso dos endereçamentos de memória.

2.2.7 – Uso de interfaces de E/S externa: é possível adicionar dispositivos de entrada e saída (portas paralelas, relógios, controladores de display de cristal líquido, timers/contadores etc.), além daqueles que já existem no microcontrolador. Eles podem ser adicionados como dispositivos mapeados como memória.

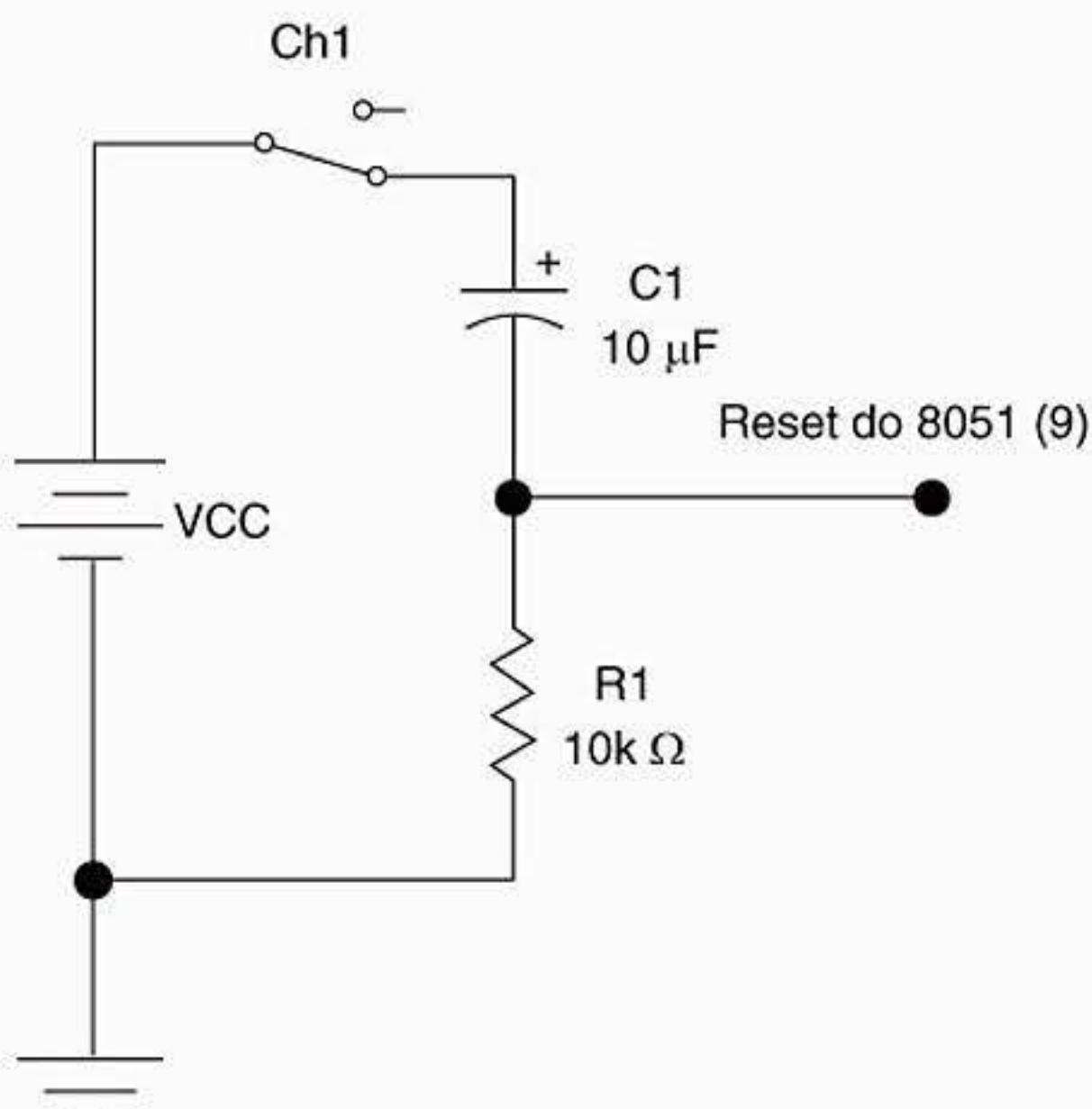


Figura 2.14 Circuito de *reset* para a família de microcontroladores MCS-51 da Intel.

2.2.8 – O sinal de reiniciar da família de microcontroladores MCS-51 da Intel: o *reset* da família de microcontroladores MCS-51 é ativado quando o pino 9 permanecer em nível alto por, no mínimo, dois ciclos de máquina. Um possível hardware para gerar o sinal elétrico de *reset* é mostrado na Figura 2.14.

Ao energizar o sistema microcomputadorizado inicialmente o capacitor está descarregado e a tensão sobre o resistor é igual a VCC. Após um tempo $t = 10\mu F \cdot 10k\Omega = 0,1$ segundo, a tensão do capacitor fica aproximadamente igual a VCC e a tensão no resistor fica igual a zero. Assim, logo após a energização do sistema microcomputadorizado, um sinal de *reset* é gerado pelo circuito em nível lógico 1 durante um período t e depois fica em zero lógico.

Após um sinal de *reset*, internamente à CPU, são inicializados alguns registradores de funções especiais, como descrito a seguir:

- ◆ O contador de programa (PC) é inicializado com o valor 0000h. Como a função do registrador contador de programa (PC) é a de armazenar sempre o endereço da próxima instrução a ser buscada e executada pelo microprocessador do microcontrolador, quando ocorrer o sinal de *reset*, a próxima instrução a ser executada será aquela armazenada no conteúdo do endereço da memória de programa 0000h (ROM/EPROM). Assim, o projetista de hardware deve mapear sempre a memória ROM/

EPROM a partir do endereço 0000h, e o projetista de software também deve escrever sempre um programa a partir do endereço 0000h;

- ◆ O registrador de função especial *ponteiro de pilha* (SP – Stack Pointer) é inicializado com o valor de 07h. Todos os dados armazenados na memória RAM por meio de instruções que utilizam a pilha serão armazenados a partir do endereço de memória 07h. As instruções que utilizam a pilha são: PUSH, POP e LCALL, ACALL, RET e RETI.
- ◆ As portas P0, P1, P2 e P3 serão inicializadas com FFh (inicialmente, ficam programadas como entradas).
- ◆ O registrador de controle do canal de comunicação serial SCON é 'zerado'.
- ◆ Os 5 bits menos significativos dos registradores de controle de interrupção IE e IP são 'zerados'.

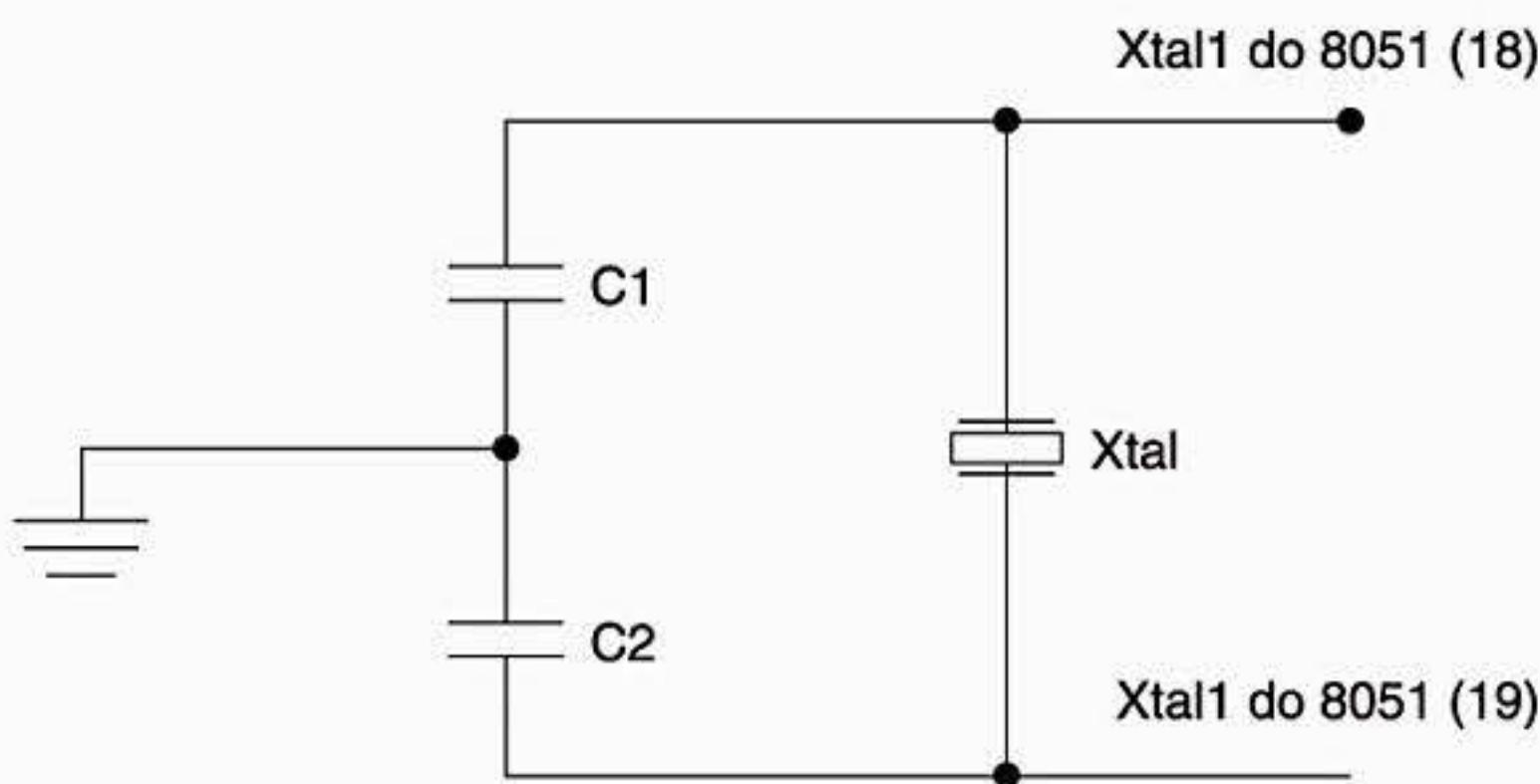


Figura 2.15 Circuito do sinal de relógio (clock).

2.2.9 – O sinal de relógio (clock) do MCS-51: todos os membros da família de microcontroladores MCS-51 têm um oscilador interno que pode ser utilizado como um sinal de relógio (fonte de *clock*) para o microprocessador (CPU). Assim, para utilizá-lo como oscilador interno, um cristal ou um ressonador cerâmico deve ser conectado entre os pinos XTAL1 e XTAL2 do microcontrolador, como é mostrado na Figura 2.15. Quando o sinal de relógio for gerado com osciladores externos, o hardware será diferente, e cada tipo de tecnologia terá um tipo de configuração, como se segue:

a) *HMOS ou CHMOS*

Sinal de relógio externo ligado aos pinos XTAL1 e XTAL2; o sinal de relógio invertido (uso de porta NOT).

b) *Somente HMOS*

Sinal de relógio externo diretamente ligado aos pinos XTAL2 e XTAL1 aterrados.

c) *Somente CHMOS*

Sinal de relógio externo diretamente ligado aos pinos XTAL1 e XTAL2; não deve ser ligado a nada.

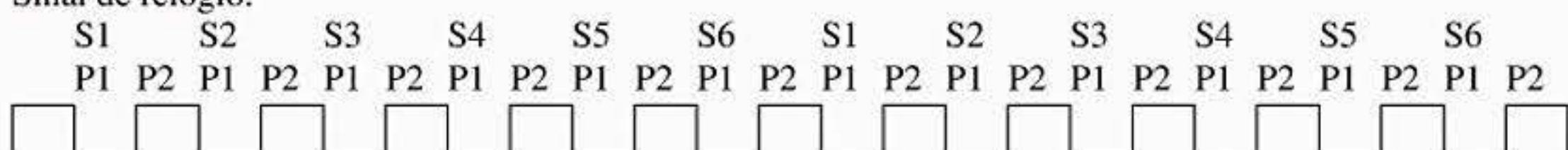
O gerador de sinal de relógio interno define a seqüência de estados que formam o ciclo de máquina da família de microcontroladores MCS-51.

2.2.10 - Ciclos de máquina: o ciclo de máquina consiste na seqüência de seis estados, numerados de S1 a S6. O tempo de cada estado corresponde a dois períodos de relógio (*clock*). Assim, um ciclo de máquina tem doze períodos de relógio. Por exemplo, caso seja utilizado um cristal de 12 MHz, o ciclo de máquina tem uma freqüência de $12\text{MHz}/12 = 1\text{MHz}$ e T_{clock} (período de *clock*) = $1/1\text{MHz} = 1\text{ ms}$.

Cada ciclo de máquina está dividido em duas meias fases (Fase 1 e Fase 2), cada uma delas correspondendo a um período de relógio.

A seguir, há um exemplo de seqüência de busca e execução em estados e fases para vários tipos de instruções.

Sinal de relógio:



Sinal de ALE:



1- Um ciclo de instrução de um byte. Exemplo: INC A.

Lê Opcode Lê o próximo Lê o próximo

 Opcode (descarta) Opcode novamente

S1	S2	S3	S4	S5	S6	S1	S2	S3	S4	S5	S6
----	----	----	----	----	----	----	----	----	----	----	----

2- Um ciclo de instrução de dois bytes. Exemplo: ADD A,#data.

Lê Opcode Lê o segundo Lê o próximo

 Byte Opcode

S1	S2	S3	S4	S5	S6	S1	S2	S3	S4	S5	S6
----	----	----	----	----	----	----	----	----	----	----	----

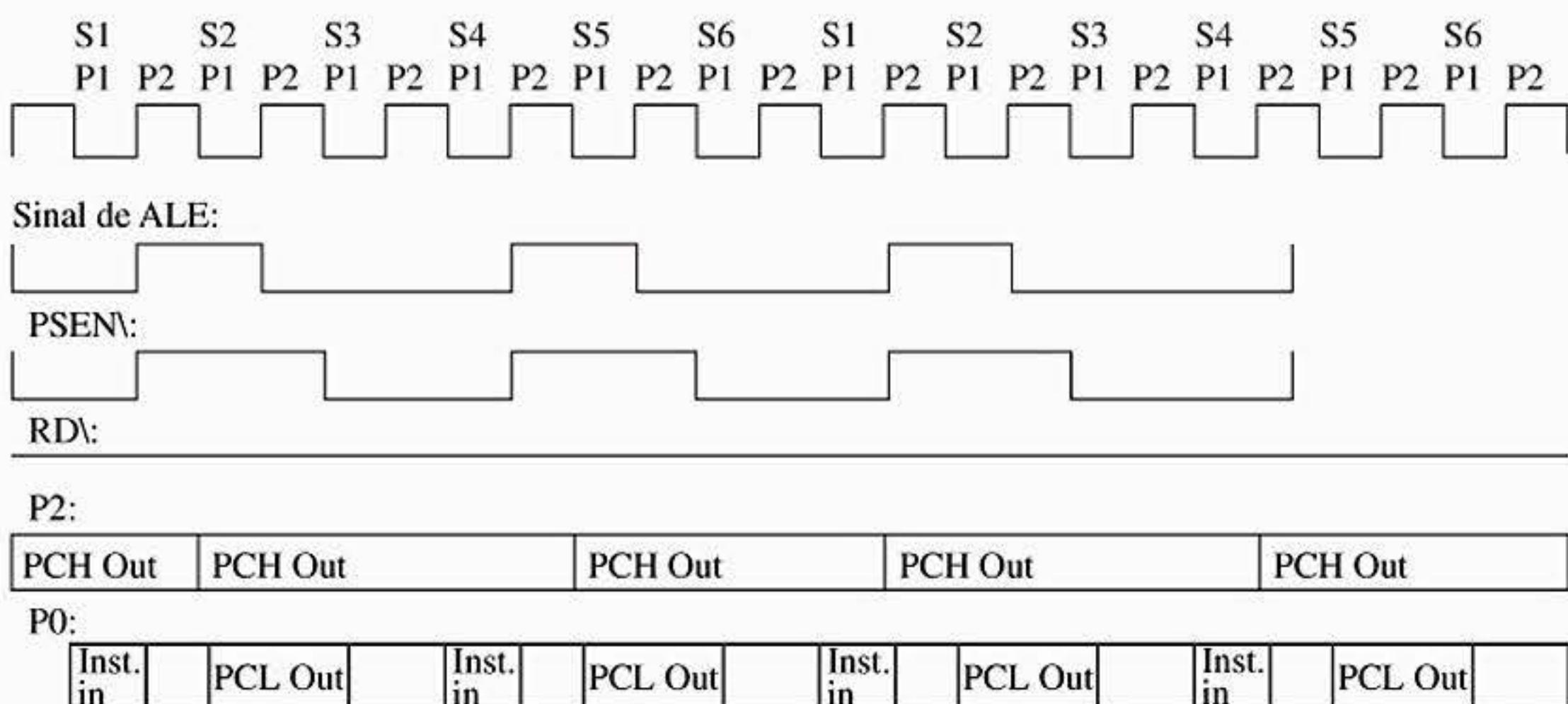
3- Dois ciclos de instrução de um byte. Exemplo: INC DPTR

Lê Opcode Lê o próximo Lê o próximo Lê o próximo

 Opcode (descarta) Opcode (descarta) Opcode (descarta)

S1	S2	S3	S4	S5	S6	S1	S2	S3	S4	S5	S6
----	----	----	----	----	----	----	----	----	----	----	----

Veja, a seguir, os sinais e os tempos envolvidos na busca do programa em memória de programa externa durante o uso da instrução MOVX.



Se ocorrer um acesso à memória de dados externa, não existirão dois pulsos de $PSEN\ = PSEN_{barra}$, pois os barramentos de endereço e de dados estão sendo utilizados para o acesso à memória de dados, com o uso da instrução MOVX. Dessa maneira, o ciclo para o acesso à memória de dados é duas vezes maior que o ciclo da memória de programa.

Quando o microprocessador (CPU) está executando um programa da memória de programa interna, $PSEN_{barra}$ não é ativado e o endereçamento da memória de programa externa não é emitido. Entretanto, o sinal ALE continua sendo ativado duas vezes por ciclo de máquina e, assim, ele pode ser utilizado como um sinal de saída de relógio. Por exemplo, se o cristal utilizado for de 12 MHz, cada sinal de relógio será de $12\text{ MHz}/12 = 1\text{MHz}$. Como o sinal de ALE é formado por três sinais de relógio, então ALE pode ser utilizado como um sinal de relógio de 333,33 KHz.

2.2.11 - Operação de execução de um programa passo a passo: a estrutura do sistema de interrupções dos microcontroladores da família MCS-51 permite a execução de programas passo a passo, com pouquíssimo trabalho referente a software.

Uma fonte de interrupção só é atendida pelo microprocessador em duas condições: a) quando não estiver sendo executada uma outra interrupção de mesmo nível; b) quando não estiver sendo executada a instrução RETI. Assim, sempre que for atendida uma interrupção, a mesma não poderá ser novamente processada, enquanto não for executada uma instrução do programa principal.

Um modo de utilizar essa característica de operação passo a passo é programar uma fonte de interrupção externa, como INT0\ para ser ativada

por nível. A rotina de atendimento à fonte de interrupção INT0\=INT0_{barra} deve terminar com a seguinte codificação, por exemplo:

JNB P3.2,\$	Aguarda que o bit P3.2 vá para 1 lógico
JB P3.2,\$	Aguarda que o bit P3.2 vá para 0 lógico
RETI	Retorna da rotina de atendimento da fonte de interrupção INT0\

Agora, se o pino INT0_{barra}, que deverá estar conectado ao pino P3.2, está normalmente em um nível lógico baixo, a CPU executará a sub-rotina de atendimento da fonte de interrupção INT0_{barra} e 'ficará lá' até que INT0_{barra} apresente um sinal pulsado (de 0 lógico para 1 lógico e para 0 lógico novamente). Então, ela executará a instrução RETI, voltará para o programa principal, executará uma única instrução e, imediatamente, ocorrerá uma nova interrupção. Novamente, a CPU executará a rotina de atendimento da fonte de interrupção INT0_{barra} e aguardará um sinal pulsado de P3.2. Uma instrução de programa será executada por vez, sempre que P3.2 pulsar.

2.2.12 - Operação de baixa potência em microcontroladores HMOS: durante a operação normal, a RAM interna consome sua potência de VCC. Mas se a tensão em RST/VPD exceder a VCC, ela se tornará a fonte de potência para a RAM interna. Isso permite que uma fonte de potência secundária (backup) possa ser utilizada para reter os dados da RAM em um evento de falha na fonte de potência referente a VCC.

Para aproveitar essa característica, o sistema do usuário, sob detecção de falha na fonte de potência referente a VCC, interromperia a CPU via INT0_{barra} ou INT1_{barra}, para transferir dados relevantes para a RAM e habilitaria a fonte de potência secundária (backup) para o pino RST\VPT, antes que a fonte de potência principal referente a VCC chegasse ao seu limite de operação. Quando a fonte de potência retornasse, VPD necessitaria ficar um tempo suficientemente longo para gerar um sinal de *reset*, para que novamente o sistema viesse a funcionar com sua fonte de potência principal referente a VCC.

2.2.13 - Modos de redução de potência em microcontroladores CHMOS: nessa versão, são apresentados dois modos de redução de potência: *Idle* e de *baixa potência*.

- a) *Modo Idle*: no modo Idle (IDL = 1 lógico), o oscilador funciona somente para as fontes de interrupções externas do canal de comunicação serial e dos timers/contadores e o oscilador para a CPU está desativado e não existe sinal de *clock* para a mesma.

Nessa condição, a CPU é preservada com relação ao conteúdo dos registradores de funções especiais *Stack Pointer* (SP), *Program Counter* (PC), *Program Status Word* (PSW), *Acumulador A* ou ACC e todos os outros registradores mantêm seus dados durante o modo *Idle*.

Existem dois modos de finalizar o modo *Idle*:

- ◆ Quando qualquer fonte de interrupção for ativada, desde que esteja habilitada, zerando (PCON.0) por hardware. Dessa maneira, a interrupção será atendida e, após executar a instrução RETI, o sistema entra novamente no modo *Idle*. Os flags GF1 e GF0 podem ser utilizados para indicar se as fontes de interrupções ocorreram durante o modo normal de operação ou durante o modo *Idle*.
 - ◆ Quando é gerado um sinal de *reset* pelo hardware.
- b) *Modo baixa potência*: no modo de baixa potência (*Power Down*), o oscilador interno é desativado, todas as funções são interrompidas e somente a RAM interna fica energizada e operante (os registradores de funções especiais não ficam energizados). A única maneira de encerrar o modo de baixa potência é por meio de um sinal de *reset* emitido pelo hardware.

O VCC pode ser reduzido para minimizar o consumo de potência. É preciso tomar o cuidado para assegurar que o VCC não seja reduzido antes que o modo de baixa potência esteja ativado e que o VCC seja restaurado para o nível normal de operação, antes que o modo de baixa potência seja encerrado.

O sinal de *reset* é o que finaliza o modo de baixa potência e também libera o oscilador. O sinal de *reset* não deveria ser ativado antes de o VCC ser recuperado.

O modo *Idle* e o de baixa potência são ativados por meio dos bits que pertencem ao registrador de função especial PCON (87h), endereçável por bit.

Registrador PCON:

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
SMOD	-	-	-	GF1	GF0	PD	IDL

- ◆ **SMOD**: bit de duplo *baud rate* (freqüência de recepção e transmissão do canal de comunicação serial). Quando for igual a 1 lógico, o *baud rate* será duplicado se o canal de comunicação serial estiver no modo 1, 2 ou 3.

- ◆ GF1: *flag* para uso de finalidade geral.
- ◆ GF0: *flag* para uso de finalidade geral.
- ◆ PD: bit de baixa potência (*Power Down*). Tornando esse bit igual a 1 lógico, o modo de baixa potência é ativado.
- ◆ IDL: bit do modo *Idle*. Tornando esse bit igual a 1 lógico, o modo *Idle* é ativado.

Exercícios resolvidos

1 - Considere que você é o projetista de um equipamento didático que utilizará um dos microcontroladores da família MCS-51 da Intel. Por meio de um estudo preliminar, você concluiu que necessitará de 2 Kbytes de memória de programa interna, 200 posições de memória de dados, 5 fontes de interrupções, 2 *timers/contadores* e 27 portas de entrada e saída. Observando a Tabela 2.1, qual membro da família de microcontroladores você utilizaria?

Resposta: uma das opções poderia ser o microcontrolador 80C52AH. Ele atende a todos os requisitos de sua conclusão descrita anteriormente.

2 - Considere o endereço do bit 4Ch. Onde está localizado esse bit (endereço de memória e posição do bit)? Qual é a outra maneira de representar o endereço desse bit?

Solução: 4Ch = ?

- ◆ Transformar o endereço do bit de hexadecimal para decimal: $4.16^1 + 12.16^0 = 76_{10}$.
- ◆ Dividir o valor do endereço em decimal (76_{10}) por 8: $76 / 8$ e obter o quociente (q) = 9_{10} e o resto (r) = 4_{10} .
- ◆ Transformar o quociente (q) e o resto (r) em hexadecimal: $q = 9_{10} = 9h$ e $r = 4_{10} = 4h$.
- ◆ Aplicar a fórmula: $4Ch = [20h + 9h]h.4 = 29h.4$, ou seja, o endereço do bit 4Ch é igual a 29h.4.

3 - Considere que o conteúdo do registrador de função especial (PSW) foi inicializado com o valor E3h. Qual banco de registradores foi selecionado?

Solução:

bits	7	6	5	4	3	2	1	0
(PSW) =	C	AC	F0	RS1	RS0	OV	-	P
E3h =	1	1	1	0	0	0	1	1

Como RS1 = 0 e RS0 = 0, o banco de registradores selecionado foi o banco 0; os registradores selecionados foram os registradores de R0 a R7, cujos endereços vão de 00h a 07h.

Exercícios e questões propostos

- 1 - Qual é o membro original da família de microcontroladores MCS-51 da Intel e quais são suas características principais?
- 2 - Utilizando a Tabela 2.1, selecione qual membro da família MCS-51 melhor corresponde a um projeto que necessita de 4 Kbytes de memória ROM, 96 bytes de memória RAM e baixo consumo de potência?
- 3 - Descreva os pinos de 32 a 39 do microcontrolador 8051.
- 4 - Descreva a função do sinal PSEN_{barra} da família de microcontroladores MCS-51.
- 5 - Quanto de memória RAM e ROM o microcontrolador 8051 pode endereçar? Desenhe a organização da memória da família de microcontroladores MCS-51.
- 6 - Represente os endereços de memória ROM que apresentam funções específicas após o sinal de *reset* e após os sinais de interrupção. Descreva também como ocorre o processo de interrupção e de *reset*.
- 7 - Qual é o espaço, em bytes de memória ROM, para se escrever rotinas de atendimento às fontes de interrupção? Caso uma rotina de atendimento às fontes de interrupção seja maior que esse espaço disponível, como proceder para alocar a mesma?
- 8 - Qual é a representação elétrica do hardware para a memória de programa externa? Explique a função de cada sinal e dos componentes utilizados.
- 9 - Qual é a representação elétrica do hardware para memória de dados externa? Explique a função de cada sinal e dos componentes utilizados.

- 10 - Qual é o diagrama de blocos da estrutura da memória de dados interna? Explique cada um dos blocos integrantes.
- 11 - Dados os endereçamentos dos bits da memória RAM interna, calcule a outra maneira de representar seu endereço.
- a) 2Ch
 - b) 2Fh.6
 - c) 71h
 - d) 24h.6
- 12 - Quais são os registradores de função especial encontrados nos microcontroladores 8051 e 8052? Descreva-os.
- 13 - Qual é a função de um decodificador em um sistema que utiliza microcontrolador, memórias externas e dispositivos de entrada e saída?
- 14 - Desenhe o diagrama de blocos (8051, latch, decodificador, sinais de controle etc.) de um projeto que consiste em um microcontrolador 8051 de duas memórias ROM externas de 4Kx8 e uma memória RAM de 32Kx8.
- 15 - Projete um mapeamento de memória (mapa de endereçamento) para dividir o espaço de endereçamento externo de dados de 16Kx8 em 16Kx8. Desenhe as interconexões do decodificador com o microcontrolador 8051.
- 16 - Explique o esquema elétrico de um circuito que deve ser conectado ao pino de *reset* do microcontrolador 8051 e descreva o que ocorre quando um sinal de *reset* é gerado.
- 17 - Descreva e explique o diagrama de tempo de uma instrução de um ciclo de máquina e de um byte.

O CONJUNTO DE INSTRUÇÕES DA FAMÍLIA DE MICROCONTROLADORES MCS-51 DA INTEL

3.1 Objetivos

- ❖ Definir o registrador de função especial *Program Status Word* (PSW)
- ❖ Definir os diferentes modos de endereçamento da família de microcontroladores MCS-51
- ❖ Definir o conjunto de instruções da família de microcontroladores MCS-51

3.2 Introdução teórica

Inicialmente, será apresentado o registrador de função especial PSW (*Program Status Word* ou registrador de condição do programa). Esse registrador é de fundamental importância para a implementação de programas que tomam decisões e, consequentemente, dão inteligência ao sistema microcontrolado.

3.2.1 - O registrador de função especial Program Status Word, PSW: é composto por 4 bits de condição (*flags* – bandeiras de sinalização) que refletem a condição atual do processamento de um programa: **C**: *carry bit*, **AC**: *auxiliar carry bit*, **OV**: *overflow bit* e **P**: *parity bit*. Ele também é composto por um bit para utilização geral (F0), que pode ser *setado* (= 1) ou *resetado* (= 0) por software. Além desses bits, ele também é formado pelos dois bits selecionadores de banco de registradores, RS1 e RS2. É importante destacar que os bits sinalizadores da condição do programa (*flags*) são influenciados sempre que a unidade lógica e aritmética (ULA) é utilizada por instruções aritméticas e lógicas. Assim, sempre que for executada uma instrução aritmética ou lógica, o resultado da operação ficará armazenado em um determinado registrador ou posição de memória e, simultaneamente, esses *flags* serão alterados,

refletindo a condição do resultado da operação (se ocorreu um comando vai 0/1 do bit mais significativo, se ocorreu um vai 0/1 do bit 3, se houve uma condição de erro e se a paridade do resultado obtido é par ou ímpar etc.). A Tabela 3.1 mostra as instruções que afetam os *flags* do registrador de função especial PSW:

Tabela 3.1 Instruções que afetam os *flags* do registrador de função especial PSW (*Program Status Word*).

Instrução	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		
MOV C,/bit	X		
CJNE	X		

A seguir, é mostrado o registrador de função especial PSW, que pode ser endereçado por byte ou por bit.

(PSW) =	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
	C	AC	F0	RS1	RS0	OV	-	P

Onde:

Carry bit flag (C) ou (PSW.7): após operações aritméticas de adição e de subtração significa o ‘vai um’ ou o ‘vai zero’ do bit 7 do valor do resultado. Após operações lógicas, esse bit é resetado [(C) = 0].

Significado do valor de *carry bit flag* após uma operação de adição:

- ◆ (C) = 0: significa que o número pode ser representado com 8 bits (valor $\leq 255_{10}$ (FFh)).

- ◆ (C) = 1: significa que o resultado não pode ser representado com 8 bits (valor $> 255_{10}$). Isso quer dizer que o *carry bit flag* deverá ser utilizado como um nono bit para determinar o resultado da operação de adição ($C * 2^8$ adicionado ao valor dos outros 8 bits do resultado).

Significado do *carry bit flag* após uma operação de subtração [(x) - (y)]:

- ◆ (C) = 0: significa que o 1º operando é \geq 2º operando.
- ◆ (C) = 1: significa que o 1º operando é $<$ 2º operando.

O *carry bit flag* também é utilizado na conversão de um número binário ou hexadecimal em BCD (representação decimal utilizando quatro bits), ou seja:

- ◆ (C) = 0: não é necessário adicionar o valor 6 nos quatro bits mais significativos do resultado.
- ◆ (C) = 1: é necessário adicionar o valor 6 nos quatro bits mais significativos do resultado.

Dica

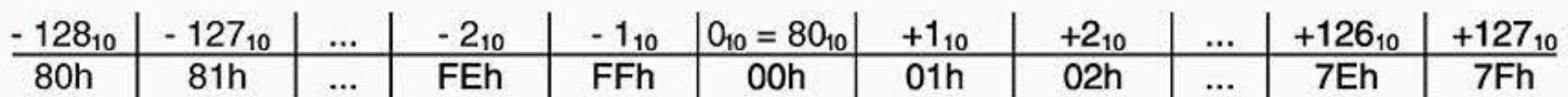
- a) Para a família de microcontroladores MCS-51, a operação de subtração não é feita diretamente, como o ser humano está acostumado a fazer em decimal.
- b) A operação de subtração é simulada por meio da operação de adição do conteúdo do acumulador (A) e do complemento de dois (C_2) do valor que se deseja subtrair. Essa simulação só é válida se, e somente se, o *carry bit flag* (C) e o *auxiliar carry flag* (AC) forem complementados. Ou seja, pode-se representar a operação de subtração da seguinte maneira:

$$(A) \leftarrow (A) - (C) - (<\text{src} - \text{byte}>) = (A) + [(C) + (<\text{src} - \text{byte}>)]_{C_2} \Leftrightarrow (C) \leftarrow \text{not}(C) \text{ e } (AC) \leftarrow \text{not}(AC)$$

- c) A família de microcontroladores MCS-51 trabalha com *números sinalizados*, ou seja, com uma escala de valores que vai de - 128 a +127. Dos 8 bits de informações, o mais significativo (bit 7) é o 'bit de sinal', e o restante dos bits (bit 6 - bit 0) define o valor numérico por meio de 7 bits do byte.
- d) Todo número negativo está representado em complemento de 2. Para determinar o valor numérico de um número negativo (bit 7 = 1), sempre é preciso determinar o valor de seu complemento de 2.

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	Valor
positivo	0	1	1	1	1	1	1	1	$7Fh = 127_{10}$
positivo	0	:	:	:	:	:	:	:	:
positivo	0	0	0	0	0	0	0	0	$00h = 0_{10}$
negativo	1	1	1	1	1	1	1	1	$FFh = -1_{10}$
negativo	1	:	:	:	:	:	:	:	:
negativo	1	0	0	0	0	0	0	0	$80h = -128_{10}$

Escala de números sinalizados



Exemplo: Considere que $(A) = 20h$, $(R0) = 08h$ e $(C) = 0b$, execute a SUBB A,R0.

Solução:

A representação simbólica dessa instrução é:

$$(A) \leftarrow (A) - (C) - (R0) = (A) + [(R0) + (C)]_{C2} \Leftrightarrow (C) = \text{not}(C) \text{ e } (AC) = \text{not}(AC)$$

Assim:

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
$(R0) = 08h = 8_{10}$	0	0	0	0	1	0	0	0	+
$(C) = 0b$									0
$(R0) + (C) =$	0	0	0	0	1	0	0	0	$= 08h = 8_{10}$
$[(R0) + (C)]_{C1}$	1	1	1	1	0	1	1	1	$= F7h = -8_{10}$
$[(R0) + (C)]_{C2}$	¹ 1	¹ 1	⁰ 1	⁰ 1	1	0	0	0	$+ = F8h = -8_{10}$
$(A) = 20h = 32_{10}$	0	0	1	0	0	0	0	0	$+ = 18h = 24_{10}$
$(A) + [(R0) + (C)]_{C2}$	¹ 0	0	0	1	1	0	0	0	

Complementando o $(C) = \text{not}(C) = \text{not}(1) = 0 \Rightarrow (C) = 0$.

Complementando o $(AC) = \text{not}(AC) = \text{not}(0) = 1 \Rightarrow (AC) = 1$.

Repare que o $(C) = 0$ em uma operação de subtração significa que o 1º operando ($20h$) é \geq 2º operando ($08h$).

Auxiliar carry bit flag (AC) ou (PSW.6): após operações aritméticas de adição ou subtração significa o 'vai um' ou 'vai zero' do bit 3 do valor do resultado. Após operações lógicas, esse bit é *setado* (AC) = 1.

Significado do valor do *auxiliar carry flag* após uma operação de adição e subtração:

- ◆ (AC) = 0: não é somado o valor 6 nos quatro bits menos significativos do resultado, em operações de ajuste decimal;
- ◆ (AC) = 1: é somado o valor 6 nos quatro bits menos significativos do resultado, em operações de ajuste decimal.

General purpose flag (F0) ou (PSW.5): utilizado para propósito geral. O usuário pode *setar* ou *resetar* esse *flag* por software para sinalizar ou indicar alguma condição desejada representando ativado ou desativado, respectivamente.

Register bank select flags (RS1) ou (PSW.4) e (RS0) ou (PSW.3): seletores dos bancos de registradores (RS1, RS0). Quando RS1 e RS0 forem iguais a 00, será selecionado o banco 0 (R0 a R7); quando forem iguais a 01, será selecionado o banco 1; quando forem iguais a 10, será selecionado o banco 2 e quando forem iguais a 11 será selecionado o banco 3. Somente um banco estará disponível para o acesso pela CPU durante a execução de um programa. Caso você queira acessar um banco de registradores diferente do atual é necessário modificar por software RS1 e RS0 para outro valor.

Overflow flag (OV) ou (PSW.2): é calculado da seguinte maneira:

$$(OV) = ('vai 0/1' do bit 7 do resultado) \text{ OR-EX } ('vai 0/1' do bit 6 do resultado)$$

$$(OV) = (C) \text{ OR-EX } ('vai 0/1' do bit 6 do resultado)$$

Significado do valor de *overflow flag* após uma operação de adição e subtração:

- ◆ (OV) = 0: o resultado pode ser representado como um número sinalizado - bit mais significativo como bit de sinal (0 = positivo e 1 = negativo) e 7 bits que representam o valor. Essa maneira de representação binária define uma escala de -128 a +127, quando se considera 8 bits.

Observação: todo número negativo é representado em complemento de dois (C_2).

- ◆ (OV) = 1: mostra uma condição de erro, ou seja, o resultado não pode ser representado como um número sinalizado. Essa condição de erro é gerada a partir de duas condições após uma operação de adição:
 - ◆ quando são somados dois números positivos, resultando em um número negativo;
 - ◆ quando são somados dois números negativos, resultando em um número positivo.

Um raciocínio análogo é considerado após a execução de uma operação de subtração, ou seja:

- a) quando é subtraído um número negativo de um número positivo e o resultado é positivo;
- b) quando é subtraído um número positivo de um número negativo e o resultado é negativo.

Parity flag (P) ou (PSW.0): chamado de *flag de paridade*. Sempre reflete a condição do registrador acumulador (ACC ou A) quanto à quantidade de números 1 existente no mesmo.

- ◆ (P) = 0: se a quantidade de números 1 do conteúdo do acumulador for par (0, 2, 4 ou 6), a paridade (P) será considerada paridade par;
- ◆ (P) = 1: se a quantidade de números 1 do conteúdo do acumulador for ímpar (1, 3, 5 ou 7), a paridade (P) será considerada paridade ímpar.

O *flag de paridade* geralmente é utilizado na comunicação serial assíncrona (recepção e transmissão) de dados, com o objetivo de reduzir o risco de erros. No Capítulo 8 serão fornecidos todos os detalhes com relação a esse assunto.

Exemplo: Determinação dos *flags* de sinalização após a execução de algumas instruções aritméticas, considerando-se as seguintes condições iniciais:

$$(A) = 3Dh; (PSW) = 00h = 00000000b; (30h) = 1Fh.$$

1 - ADD A,30h?

Solução:

A representação simbólica dessa instrução é:

$$(A) \leftarrow (A) + (30h)$$

Assim:

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
(A) = 3Dh = 61 ₁₀	0	0	1	1	1	1	0	1	+
(30h) = 1Fh = 31 ₁₀	0	0	0	1	1	1	1	1	
(A) ← (A) + (30h) =	0	1	0	1	1	1	0	0	= 5Ch = 92 ₁₀

Além de o conteúdo do acumulador mudar do valor 3Dh para 5Ch, o conteúdo do registrador *Program Status Word* (PSW) mudará para:

(PSW) =	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
	C	AC	F0	RS1	RS0	OV	-	P	
	0	1	0	0	0	0	0	0	= 44h

Em que:

- ◆ (C) = 'vai 0' do bit 7 do resultado da operação de adição = 0. Significa que o resultado pode ser representado com 8 bits (resultado = 92₁₀ < 255₁₀).
- ◆ (AC) = 'vai 1' do bit 3 do resultado da operação de adição = 1. Caso seja feita uma operação de ajuste decimal, será somado o valor 6 aos 4 bits menos significativos, para executar a conversão.
- ◆ (F0): continua com o valor inicial = 0. O flag (F0) não é afetado por essa instrução.
- ◆ RS1, RS0 = continua com os valores iniciais = 0,0, que não são afetados por essa instrução. Continua selecionando o primeiro banco de registradores.
- ◆ (OV) = ('vai 0' do bit 7) or-ex ('vai 0' do bit 6) = 0 or-ex 0 = 0. Significa que não houve condição de erro na representação numérica do resultado dentro da escala de - 128 a +127, pois foram somados dois números positivos resultando em um número positivo.
- ◆ (P): quatro números 1 no resultado da operação de adição = paridade par = 0.

3.2.2 – Modos de endereçamento: existe uma variedade de modos de endereçamento rápidos para acessar as diferentes posições de memória de um sistema microcontrolado. Os membros da família de microcontroladores MCS-51 têm cinco modos de endereçamento diferentes: por registrador, direto, indireto por registrador, imediato e por registrador-base mais indireto indexado por registrador. Veja na Tabela 3.2 a relação dos métodos de endereçamento com o espaço de memória associado.

Tabela 3.2 Modo de endereçamento relacionado com o espaçamento de memória.

Modo de endereçamento	Operandos (registradores e memória)
Endereçamento imediato	Memória de programa
Por registrador	R0-R7 e ACC (A), B, C (<i>carry bit</i>) e DPTR
Direto	Os 128 bytes menos significativos da RAM interna e registradores de funções especiais
Indireto por registrador	RAM interna (@R0, @R1 e SP) e memória de dados externa (@R0, @R1 e @DPTR)
Registrador base mais indireto indexado por registrador	memória de programa (@DPTR+A e @PC+A)

Observação: quando o registrador acumulador (A) não for um dos argumentos de uma instrução e a mesma apresentar dois diferentes tipos de endereçamento, obtém-se o chamado *endereçamento combinado* ou *misto* e o tipo de endereçamento dessa instrução será definido pelo endereçamento de ambos. Como exemplo disso, considere a seguinte instrução: MOV @R0,30h. Ela apresenta um *endereçamento misto*, definido pelos dois endereçamentos, ou seja, indireto ou indexado por registrador e pelo endereçamento direto.

- a) *Endereçamento por registradores*: nesse tipo de endereçamento, a instrução especifica diretamente um dos registradores do banco de registradores selecionado. Essas instruções transportam por meio de seu código de máquina (código-objeto) pelo menos 3 bits correspondentes ao registrador selecionado (de R0 a R7). O registrador B, o registrador *data pointer*, DPTR; o *carry bit*, C e o acumulador, ACC, do processador booleano também podem ser endereçados como registradores. A seguir, são fornecidos alguns exemplos desse tipo de endereçamento.

Exemplo 1: MOV A,R1

- ◆ A representação simbólica dessa instrução é: $(A) \leftarrow (R1)$.
- ◆ O significado dessa instrução é: *no conteúdo do registrador acumulador será copiado (operação de escrita) o conteúdo do registrador R1.*

Exemplo 2: MOV R2,A

- ◆ A representação simbólica dessa instrução é: $(R2) \leftarrow (A)$.
- ◆ O significado dessa instrução é: *no conteúdo do registrador R2 será copiado (operação de escrita) o conteúdo do registrador acumulador.*

- b) *Endereçamento direto*: nesse tipo de endereçamento, o operando é especificado por um campo de endereços de memória RAM de 8 bits dentro da instrução. Somente os registradores de funções especiais e os 128 bytes menos significativos da RAM interna podem ser endereçados de forma diferente. A seguir, são fornecidos alguns exemplos desse tipo de endereçamento.

Exemplo 1: MOV A,30h

- ◆ A representação simbólica dessa instrução é: $(A) \leftarrow (30h)$.
- ◆ O significado dessa instrução é: *no conteúdo do registrador acumulador será copiado (operação de escrita) o conteúdo da posição de memória, cujo endereço é 30h.*

Exemplo 2: MOV 7Ah,A

- ◆ A representação simbólica dessa instrução é: $(7Ah) \leftarrow (A)$.
 - ◆ O significado dessa instrução é: *no conteúdo da posição de memória, cujo endereço é 7Ah, será copiado (operação de escrita) o conteúdo do registrador acumulador.*
- c) *Endereçamento indireto indexado por registrador*: nesse tipo de endereçamento, a instrução utiliza o conteúdo do registrador R0 ou do registrador R1, do banco selecionado, como um ponteiro de localização dentro de um bloco de memória de 256 bytes: 128 bytes menos significativos da memória RAM interna; 128 bytes mais significativos da memória RAM interna (8032/8052) ou 256 bytes menos significativos da memória RAM externa.

Os registradores de funções especiais não são acessados por esse tipo de endereçamento. O acesso aos 64 Kbytes de endereçamento de memória RAM externa é feito por meio do uso do registrador de 16 bits, o *data pointer* (DPTR). As instruções de PUSH <*direct*> e POP <*direct*> também utilizam esse tipo de endereçamento.

O registrador denominado ponteiro de pilha, *Stack Pointer* (SP), pode estar em qualquer lugar da RAM interna.

A seguir, são fornecidos alguns exemplos desse tipo de endereçamento.

Exemplo 1: MOV A,@R0

- ◆ A representação simbólica dessa instrução é: $(A) \leftarrow ((R0))$.
- ◆ O significado dessa instrução é: *no conteúdo do registrador acumulador será copiado (operação de escrita) o conteúdo da posição de memória, cujo endereço é dado pelo conteúdo do registrador R0.*

Exemplo 2: MOV @R1, A

- ◆ A representação simbólica dessa instrução é: $((R1)) \leftarrow (A)$.
 - ◆ O significado dessa instrução é: *no conteúdo da posição de memória, cujo endereço é dado pelo conteúdo do registrador R1, será copiado (operação de escrita) o conteúdo do registrador acumulador.*
- d) *Endereçamento imediato:* nesse tipo de endereçamento, as constantes podem fazer parte da instrução na memória de programa. A seguir, são fornecidos alguns exemplos desse tipo de endereçamento.

Exemplo 1: MOV A,#0Ah

- ◆ A representação simbólica dessa instrução é: $(A) \leftarrow \#3Ah$.
 - ◆ O significado dessa instrução é: *o conteúdo do registrador acumulador será inicializado (operação de escrita) com o valor constante 3Ah.*
- e) *Endereçamento indireto por registrador-base mais o registrador indexado:* nesse tipo de endereçamento, o byte pode ser acessado a partir da memória de programa, por meio de um endereço indireto de uma posição de memória de programa endereçada pela adição de um registrador-base (DPTR ou PC) e o registrador indexado (A). Esse modo facilita o acesso a tabelas.

A seguir, são fornecidos alguns exemplos desse tipo de endereçamento.

Exemplo 1: MOVC @A+DPTR

- ◆ A representação simbólica dessa instrução é: $(A) \leftarrow ((A) + (DPTR))$.
- ◆ O significado dessa instrução é: *no conteúdo do registrador acumulador será copiado (operação de escrita) o conteúdo da posição de memória de programa (ROM/EPROM/EEPROM), cujo endereço é dado pela soma do conteúdo do registrador acumulador ACC com o conteúdo do registrador data pointer, DPTR.*

A seguir, são fornecidos alguns exemplos do tipo de endereçamento combinado (misto).

- ◆ MOV R0,20h: modo de endereçamento combinado por registrador e direto;
- ◆ MOC 30h,R4: modo de endereçamento combinado, direto e por registrador;
- ◆ MOV R1,#40h: modo de endereçamento combinado, por registrador e imediato;

- ◆ MOV 60h,#55h: modo de endereçamento combinado, direto e imediato;
- ◆ MOV @R1,#33h: modo de endereçamento combinado, indireto ou indexado por registrador e imediato;
- ◆ MOV 50h,@R1: modo de endereçamento combinado, direto e indireto ou indexado por registrador;
- ◆ MOV @R0,50h: modo de endereçamento combinado, indireto indexado por registrador e direto;
- ◆ MOV 40h,50h: modo de endereçamento combinado, direto e direto.

3.2.3 - O conjunto de instruções da família de microcontroladores MCS-51: todos os membros da família de microcontroladores MCS-51 executam o mesmo conjunto de instruções de 8 bits. As instruções que esse microcontrolador, por meio de seu microprocessador, pode executar são: operações de transferência de dados, aritméticas, lógicas, booleanas, de salto incondicional, de salto condicional, de chamadas a sub-rotinas, de retorno a sub-rotinas, entre outras.

A - Operações de transferência de dados

- Da RAM interna:* a Tabela 3.3 mostra o conjunto de instruções que estão disponíveis para mover dados dentro da memória RAM interna. Caso seja utilizado um cristal de 12 MHz, todas essas instruções serão executadas em 1 μ s ou 2 μ s.

Tabela 3.3 Operações de transferência de dados da RAM interna.

Mnemônico	Operação	Modos de endereçamento				Tempo de exec. (μs)
		Dir.	Ind.	Reg.	Imed.	
MOV A,<src>	(A) \leftarrow <src>	X	X	X	X	1
MOV <dest>, A	<dest> \leftarrow (A)	X	X	X		1
MOV <dest>,<src>	<dest> \leftarrow <src>	X	X	X	X	2
MOV DPTR,#data ₁₆	(DPTR) \leftarrow #data ₁₆				X	2
PUSH <src>	(SP) \leftarrow (SP) + 1, ((SP)) \leftarrow <src>	X				2
POP <dest>	<dest> \leftarrow ((SP)), (SP) \leftarrow (SP) - 1	X				2
XCH A,<byte>	(A) \leftrightarrow <byte>	X	X	X		1
XCHD A,@Ri	(A) _{3..0} \leftrightarrow ((Ri) _{3..0})		X			1

- b) *Da RAM externa:* a Tabela 3.4 mostra o conjunto de instruções que estão disponíveis para mover dados para dentro da memória RAM externa de dados. Somente o endereçamento indireto pode ser utilizado. A única escolha a ser feita é entre utilizar um endereçamento de um byte, @R_i (R0 ou R1 do banco selecionado) ou um endereçamento de 16 bits (2 bytes), @DPTR. A desvantagem de utilizar um endereçamento de 2 bytes pode ser percebida quando é necessário acessar apenas alguns Kbytes da RAM externa e, dessa maneira, todos os bits da Porta 2 são utilizados como barramento de endereços. Quando se utiliza o endereçamento de 8 bits, não é necessário 'sacrificar' todos os bits da Porta 2. Com um cristal de 12 MHz, todas as instruções precisam de 2ms para serem executadas.

Tabela 3.4 Operações de transferência de dados da RAM externa.

Mnemônico	Operação	Largura do endereçamento	Tempo de exec. (μs)
MOVX A,@R _i	(A) ← ((R _i))	8 bits	2
MOVX@R _i ,A	((R _i)) ← (A)	8 bits	2
MOVX A,@DPTR	(A) ← ((DPTR))	16 bits	2
MOVX@DPTR,A	((DPTR)) ← (A)	16 bits	2

Os sinais de controle de leitura e escrita da memória RAM externa são ativados somente durante a execução da instrução MOVX. Normalmente, esses sinais estão inativos e os pinos podem ser utilizados como linhas extras de E/S.

- c) *Tabelas de procura:* a Tabela 3.5 mostra duas instruções disponíveis para a procura de tabelas na memória de programa. Se o acesso à tabela é feito por meio da memória de programa externa, então o sinal de controle de leitura é PSEN\.

Tabela 3.5 Operações de transferência de dados da ROM.

Mnemônico	Operação	Tempo de execução (μs)
MOVC A,@A + DPTR	(A) ← ((A) + (DPTR))	2
MOVC A,@A + PC	(A) ← ((A) + (PC))	2

A primeira instrução da Tabela 3.5 pode acomodar até 256 bytes, numerados de 0 a FFh. O número do byte desejado deve ser carregado no conteúdo do registrador acumulador (A), e o conteúdo do *Data Pointer* (DPTR) deve ser inicializado com o endereço do início da tabela.

A outra instrução utiliza o conteúdo do *Program Counter* (PC) como o endereço da tabela-base, e a tabela é acessada por meio de uma sub-rotina. Em primeiro lugar, deve-se inicializar o conteúdo do acumulador (A) com o número da entrada desejada e a sub-rotina deve ser ativada. A tabela deve começar logo após a instrução RET da sub-rotina e precisa ter, no máximo, um tamanho de 255 bytes.

MOV A, NUMERODESEJADO
ACALL TABELA
TABELA: MOVC A,@A+PC
RET

B - Operações aritméticas

A Tabela 3.6 relaciona as instruções aritméticas disponíveis para a família de microcontroladores MCS-51.

Tabela 3.6 Operações aritméticas da família de microcontroladores MCS-51.

Mnemônico	Operação	Modos de endereçamento				Tempo de execução (μs)
		Dir.	Ind.	Reg.	Imed.	
ADD A,<byte>	(A) \leftarrow (A) + <byte>	X	X	X	X	1
ADDC A,<byte>	(A) \leftarrow (A) + <byte> + (C)	X	X	X	X	1
SUBB A,<byte>	(A) \leftarrow (A) - <byte> - (C)	X	X	X	X	1
INC A	(A) \leftarrow (A) + 1			Só (A)		1
INC<byte>	(byte) \leftarrow <byte> + 1	X	X	X		1
INC DPTR	(DPTR) \leftarrow (DPTR) + 1			Só (DPTR)		2
DEC A	(A) \leftarrow (A) - 1			Só (A)		1
DEC<byte>	(byte) \leftarrow <byte> - 1	X	X	X		1
MUL AB	(B) \leftarrow + sign (A*B) (A) \leftarrow - sign (A*B)			Só (A) e (B)		4
DIV AB	(A) \leftarrow int (A/B) (A) \leftarrow mod (A/B)			Só (A) e (B)		4
DA A	Ajuste decimal			Só (A)		1

C - Operações lógicas

A Tabela 3.7 relaciona as instruções lógicas disponíveis para a família de microcontroladores MCS-51.

Tabela 3.7 Operações lógicas da família de microcontroladores MCS-51.

Mnemônico	Operação	Modos de endereçamento				Tempo de execução (μs)
		Dir.	Ind.	Reg.	Imed.	
ANL A,<byte>	(A) \leftarrow (A) and <byte>	X	X	X	X	1
ANL <byte>,A	<byte> \leftarrow <byte> and (A)	X				1
ANL <byte>,#data	<byte> \leftarrow <byte> and #data	X				2
ORL A,<byte>	(A) \leftarrow (A) or <byte>	X	X	X	X	1
ORL <byte>,A	<byte> \leftarrow <byte> or (A)	X				1
ORL <byte>,#data	<byte> \leftarrow <byte> or #data	X				2
XRL A,<byte>	(A) \leftarrow (A) or-ex <byte>	X	X	X	X	1
XRL <byte>,A	<byte> \leftarrow <byte> or-ex (A)	X				1
XRL <byte>,#data	<byte> \leftarrow <byte> or-ex #data	X				2
CLR A	(A) \leftarrow #00h			Só (A)		1
CPL A	(A) \leftarrow not (A)			Só (A)		1
RL A	Rotaciona (A) um bit para a esquerda			Só (A)		1
RLC A	Rotaciona (A) um bit para a esquerda por meio do (C)			Só (A)		1
RR A	Rotaciona (A) um bit para a direita			Só (A)		1
RRC A	Rotaciona (A) um bit para a direita por meio do (C)			Só (A)		1
SWAP A	(A) _{3,0} \leftrightarrow (A) _{7,4}			Só (A)		1

D - Operações booleanas

A Tabela 3.8 relaciona as instruções booleanas (operações de somente um bit) disponíveis para a família de microcontroladores MCS-51.

Tabela 3.8 Operações booleanas da família de microcontroladores MCS-51.

Mnemônico	Operação	Tempo de execução (μs)
ANL C,bit	(C) \leftarrow (C) and (bit)	2
ANL C,/bit	(C) \leftarrow (C) and (not (bit))	2
ORL C,bit	(C) \leftarrow (C) or (bit)	2
ORL C,/bit	(C) \leftarrow (C) or (not (bit))	2
MOV C,bit	(C) \leftarrow (bit)	1
MOV bit,C	(bit) \leftarrow (C)	2
CLR C	(C) \leftarrow #0	1
CLR bit	(bit) \leftarrow #0	1
SETB C	(C) \leftarrow #1	1
SETB bit	(bit) \leftarrow #1	1
CPL C	(C) \leftarrow not (C)	1
CPL bit	(bit) \leftarrow not (bit)	1
JC rel	(PC) \leftarrow (PC) + rel, se (C) = 1	2
JNC rel	(PC) \leftarrow (PC) + rel, se (C) = 0	2
JB rel	(PC) \leftarrow (PC) + rel, se (bit) = 1	2
JNB rel	(PC) \leftarrow (PC) + rel, se (bit) = 0	2
JBC rel	(PC) \leftarrow (PC) + rel, se (bit) = 1 e (bit) \leftarrow #0	2

E - Operações de salto incondicional

O endereço de destino dessas instruções é especificado na linguagem de programação Assembly, por um rótulo (*label*) ou por um endereço atual da memória de programa. Entretanto, o endereço de destino trabalha como um valor relativo. Ele é um byte sinalizado (representado em complemento de dois), que é adicionado ao conteúdo do registrador *Program Counter* (PC). A escala de saltos vai de - 128 a +127 da memória de programa, relativamente ao primeiro byte que segue à instrução.

A Tabela 3.9 mostra a lista de instruções de salto incondicional.

Tabela 3.9 Operações de salto incondicional da família de microcontroladores MCS-51.

Mnemônico	Operação	Tempo de exec. (μs)
SJMP <i>addr</i>	Salta para <i>addr</i> (- 128 a + 127)	2
AJMP <i>addr</i>	Salta para <i>addr</i> (dentro do espaço de - 1024 a +1024)	2
LJMP <i>addr</i>	Salta para <i>addr</i> (64K)	3
JMP @A+DPTR	Salta para A+DPTR	2
ACALL <i>addr</i>	Chama a sub-rotina para o endereço <i>addr</i> (dentro do espaço de - 1024 a +1024)	2
LCALL <i>addr</i>	Chama a sub-rotina para o endereço <i>addr</i> (64K)	3
RET	Retorna da sub-rotina	2
RETI	Retorna da rotina de atendimento a interrupção	2
NOP	Nenhuma operação (<i>No operation</i>)	1

F - Operações de salto condicionais

Todas essas instruções usam o endereço de destino relativo e assim são limitadas a um salto de - 128 a +127 a partir da mesma.

A Tabela 3.10 mostra a lista de instruções de salto condicional.

Tabela 3.10 Operações salto condicional da família de microcontroladores MCS-51.

Mnemônico	Operação	Modos de endereçamento				Tempo de exec. (μs)
		Dir.	Ind.	Reg.	Imed.	
JZ rel	(PC) ← (PC) + rel, se (A) = 0			Só (A)		2
JNZ rel	(PC) ← (PC) + rel, se (A) ≠ 0			Só (A)		2
DJNZ <byte>, rel	<byte> ← <byte> - 1 e (PC) ← (PC) + rel, se (byte) ≠ 0	X		X		2
CJNE A,<byte>, rel	(PC) ← (PC) + rel, se (A) ≠ <byte>	X			X	2
CJNE A,#data, rel	(PC) ← (PC) + rel, se (A) ≠ #data		X	X		2

Na Tabela 3.11, são mostradas todas as instruções da família de microcontroladores MCS-51.

Tabela 3.11 Conjunto de instruções da família de microcontroladores CS-51.

Operações de transferência de dados				
Instrução	Byte(s)	Ciclos	Codificação	Representação simbólica
MOV A,Rn	1	1	1110 1rrr	$(PC) \leftarrow (PC) + 1$ $(A) \leftarrow (Rn)$
MOV A,direct	2	1	1110 0101 direct address	$(PC) \leftarrow (PC) + 2$ $(A) \leftarrow (\text{direct})$
MOV A,@Ri	1	1	1110 011I	$(PC) \leftarrow (PC) + 1$ $(A) \leftarrow ((Ri))$
MOV A,#data	2	1	0111 0100 immediate data	$(PC) \leftarrow (PC) + 2$ $(A) \leftarrow \#data$
MOV Rn,A	1	1	1111 1rrr	$(PC) \leftarrow (PC) + 1$ $(Rn) \leftarrow (A)$
MOV Rn,direct	2	2	1010 1rrr direct address	$(PC) \leftarrow (PC) + 2$ $(Rn) \leftarrow (\text{direct})$
MOV Rn,#data	2	1	0111 1rrr immediate data	$(PC) \leftarrow (PC) + 2$ $(Rn) \leftarrow \#data$
MOV direct,A	2	1	1111 0101 direct address	$(PC) \leftarrow (PC) + 2$ $(\text{direct}) \leftarrow (A)$
MOV direct,Rn	2	2	1000 1rrr direct address	$(PC) \leftarrow (PC) + 2$ $(\text{direct}) \leftarrow (Rn)$
MOV direct,direct	3	2	1000 0101 dir.adr.src dir.adr.dest	$(PC) \leftarrow (PC) + 3$ $(\text{direct}) \leftarrow (\text{direct})$
MOV direct,@Ri	2	2	1000 011i direct address	$(PC) \leftarrow (PC) + 2$ $(\text{direct}) \leftarrow ((Ri))$
MOV direct,#data	3	2	0111 0101 dir. adr. immed. data	$(PC) \leftarrow (PC) + 3$ $(\text{direct}) \leftarrow \#data$
MOV @Ri,A	1	1	1111 011i	$(PC) \leftarrow (PC) + 1$ $((Ri)) \leftarrow (A)$
MOV @Ri,direct	2	2	1010 011i direct address	$(PC) \leftarrow (PC) + 2$ $((Ri)) \leftarrow (\text{direct})$
MOV @Ri,#data	2	1	0111 011i immediate data	$(PC) \leftarrow (PC) + 2$ $((Ri)) \leftarrow \#data$
MOV DPTR,#data ₁₆	3	2	1001 0000 imme. data _{15,8} imme. data _{7,0}	$(PC) \leftarrow (PC) + 3$ $(DPH) \leftarrow \#data_{15,8}$ $(DPL) \leftarrow \#data_{7,0}$
MOVC A,@A+DPTR	1	2	1001 0011	$(PC) \leftarrow (PC) + 1$ $(A) \leftarrow ((A) + (DPTR))$
MOVC A,@A+PC	1	2	1000 0011	$(PC) \leftarrow (PC) + 1$ $(A) \leftarrow ((A) + (PC))$
MOVXA,@Ri	1	2	1110 001i	$(PC) \leftarrow (PC) + 1$ $(A) \leftarrow ((Ri))$

Tabela 3.11 Conjunto de instruções da família de microcontroladores CS-51. (Continuação)

Operações de transferência de dados				
Instrução	Byte(s)	Ciclos	Codificação	Representação simbólica
MOVX A,@DPTR	1	2	1110 0000	$(PC) \leftarrow (PC) + 1$ $(A) \leftarrow ((DPTR))$
MOVX @Ri,A	1	2	1111 000i	$(PC) \leftarrow (PC) + 1$ $((Ri)) \leftarrow (A)$
MOVX @DPTR,A	1	2	1111 0000	$(PC) \leftarrow (PC) + 1$ $((DPTR)) \leftarrow (A)$
PUSH direct	2	2	1100 0000 direct address	$(PC) \leftarrow (PC) + 2$ $(SP) \leftarrow (SP) + 1$ $((SP)) \leftarrow (direct)$
POP direct	2	2	1101 0000 direct address	$(PC) \leftarrow (PC) + 2$ $(direct) \leftarrow ((SP))$ $(SP) \leftarrow (SP) - 1$
XCH A,Rn	1	1	1100 1rrr	$(PC) \leftarrow (PC) + 1$ $(A) \leftarrow (Rn)$
XCH A,direct	2	1	1100 0101 direct address	$(PC) \leftarrow (PC) + 2$ $(A) \leftarrow (direct)$
XCH A,@Ri	1	1	1100 011i	$(PC) \leftarrow (PC) + 1$ $(A) \leftarrow ((Rn))$
XCHD A,@Ri	1	1	1101 011i	$(PC) \leftarrow (PC) + 1$ $(A_{3..0}) \leftarrow ((Rn_{3..0}))$

Operações aritméticas				
Instrução	Byte(s)	Ciclos	Codificação	Representação simbólica
ADD A,Rn	1	1	0010 1rrr	$(PC) \leftarrow (PC) + 1$ $(A) \leftarrow (A) + (Rn)$
ADD A,direct	2	1	0010 0101 direct address	$(PC) \leftarrow (PC) + 2$ $(A) \leftarrow (A) + (direct)$
ADD A,@Ri	1	1	0010 011I	$(PC) \leftarrow (PC) + 1$ $(A) \leftarrow (A) + ((Ri))$
ADD A,#data	2	1	0010 0100 immediate data	$(PC) \leftarrow (PC) + 2$ $(A) \leftarrow (A) + \#data$
ADDC A,Rn	1	1	0011 1rrr	$(PC) \leftarrow (PC) + 1$ $(A) \leftarrow (A) + (C) + (Rn)$
ADDC A,direct	2	1	0011 0101 direct address	$(PC) \leftarrow (PC) + 2$ $(A) \leftarrow (A) + (C) + (direct)$
ADDC A,@Ri	1	1	0011 011I	$(PC) \leftarrow (PC) + 1$ $(A) \leftarrow (A) + (C) + ((Ri))$
ADDC A,#data	2	1	0011 0100 immediate data	$(PC) \leftarrow (PC) + 2$ $(A) \leftarrow (A) + (C) + \#data$
SUBB A,Rn	1	1	1001 1rrr	$(PC) \leftarrow (PC) + 1$ $(A) \leftarrow (A) - (C) - (Rn)$

Tabela 3.11 Conjunto de instruções da família de microcontroladores CS-51. (Continuação)

Operações aritméticas				
Instrução	Byte(s)	Ciclo	Codificação	Representação simbólica
SUBB A,direct	2	1	1001 0101 direct address	$(PC) \leftarrow (PC) + 2$ $(A) \leftarrow (A) - (C) - (\text{direct})$
SUBB A,@Ri	1	1	1001 010i	$(PC) \leftarrow (PC) + 1$ $(A) \leftarrow (A) - (C) - ((Ri))$
SUBB A,#data	2	1	1001 0100 immediate data	$(PC) \leftarrow (PC) + 2$ $(A) \leftarrow (A) - (C) - \#data$
INCA	1	1	0000 0100	$(PC) \leftarrow (PC) + 1$ $(A) \leftarrow (A) + 1$
INC Rn	1	1	0000 1rrr	$(PC) \leftarrow (PC) + 1$ $(Rn) \leftarrow (Rn) + 1$
INC direct	2	1	0000 0101 direct address	$(PC) \leftarrow (PC) + 2$ $(\text{direct}) \leftarrow (\text{ditect}) + 1$
INC @Ri	1	1	0000 011i	$(PC) \leftarrow (PC) + 1$ $((Ri)) \leftarrow ((Ri)) + 1$
INC DPTR	1	2	1010 0011	$(PC) \leftarrow (PC) + 1$ $(DPTR) \leftarrow (DPTR) + 1$
DECA	1	1	0001 0010	$(PC) \leftarrow (PC) + 1$ $(A) \leftarrow (A) - 1$
DEC Rn	1	1	0001 1rrr	$(PC) \leftarrow (PC) + 1$ $(Rn) \leftarrow (Rn) - 1$
DEC direct	2	1	0001 0101 direct address	$(PC) \leftarrow (PC) + 2$ $(\text{direct}) \leftarrow (\text{ditect}) - 1$
DEC @Ri	1	1	0001 011i	$(PC) \leftarrow (PC) + 1$ $((Ri)) \leftarrow ((Ri)) - 1$
MUL AB	1	4	1010 0100	$(PC) \leftarrow (PC) + 1$ $(A) \leftarrow 8 \text{ bits - significativos}$ $(B) \leftarrow 8 \text{ bits + significativos}$
DIV AB	1	4	1000 0100	$(PC) \leftarrow (PC) + 1$ $(A) \leftarrow \text{quociente}$ $(B) \leftarrow \text{resto}$
DAA	1	1	1101 0100	$(PC) \leftarrow (PC) + 1$ 1°) IF $\{(A_{3-0}) > 9\}$ OR $(AC)=1$ } then $(A_{3-0}) \leftarrow (A_{3-0}) + 6;$ 2°) IF $\{(A_{7-4}) > 9\}$ OR $(C)=1$ } then $(A_{7-4}) \leftarrow (A_{7-4}) + 6;$

Tabela 3.11 Conjunto de instruções da família de microcontroladores CS-51. (Continuação)

Operações lógicas				
Instrução	Byte(s)	Ciclos	Codificação	Representação simbólica
ANLA,Rn	1	1	01011rr	$(PC) \leftarrow (PC) + 1$ $(A) \leftarrow (A) \text{and} (Rn)$
ANLA,direct	2	1	01010101 directaddress	$(PC) \leftarrow (PC) + 2$ $(A) \leftarrow (A) \text{and} (\text{direct})$
ANLA,@Ri	1	1	0101011I	$(PC) \leftarrow (PC) + 1$ $(A) \leftarrow (A) \text{and} ((Ri))$
ANLA,#data	2	1	01010100 immediate data	$(PC) \leftarrow (PC) + 2$ $(A) \leftarrow (A) \text{and} \#data$
ANL direct,A	2	1	01010010 direct address	$(PC) \leftarrow (PC) + 2$ $(\text{direct}) \leftarrow (\text{direct}) \text{and} (A)$
ANL direct,#data	3	2	01010011 direct address immed.data	$(PC) \leftarrow (PC) + 3$ $(\text{direct}) \leftarrow (\text{direct}) \text{and} \#data$
ORLA,Rn	1	1	01001rr	$(PC) \leftarrow (PC) + 1$ $(A) \leftarrow (A) \text{or} (Rn)$
ORLA,direct	2	1	01000101 directaddress	$(PC) \leftarrow (PC) + 2$ $(A) \leftarrow (A) \text{or} (\text{direct})$
ORLA,@Ri	1	1	0100011I	$(PC) \leftarrow (PC) + 1$ $(A) \leftarrow (A) \text{or} ((Ri))$
ORLA,#data	2	1	01000100 immediate data	$(PC) \leftarrow (PC) + 2$ $(A) \leftarrow (A) \text{or} \#data$
ORL direct,A	2	1	01000010 direct address	$(PC) \leftarrow (PC) + 2$ $(\text{direct}) \leftarrow (\text{direct}) \text{or} (A)$
ORL direct,#data	3	2	01000011 direct address immed.data	$(PC) \leftarrow (PC) + 3$ $(\text{direct}) \leftarrow (\text{direct}) \text{or} \#data$
XRLA,Rn	1	1	01101rr	$(PC) \leftarrow (PC) + 1$ $(A) \leftarrow (A) \text{or-ex} (Rn)$
XRLA,direct	2	1	01100101 directaddress	$(PC) \leftarrow (PC) + 2$ $(A) \leftarrow (A) \text{or-ex} (\text{direct})$
XRLA,@Ri	1	1	0110011I	$(PC) \leftarrow (PC) + 1$ $(A) \leftarrow (A) \text{or-ex} ((Ri))$
XRLA,#data	2	1	01100100 immediate data	$(PC) \leftarrow (PC) + 2$ $(A) \leftarrow (A) \text{or-ex} \#data$
XRL direct,A	2	1	01100010 direct address	$(PC) \leftarrow (PC) + 2$ $(\text{direct}) \leftarrow (\text{direct}) \text{or-ex} (A)$
XRL direct,#data	3	2	01100011 direct address immed.data	$(PC) \leftarrow (PC) + 3$ $(\text{direct}) \leftarrow (\text{direct}) \text{or-ex} \#data$
CLRA	1	1	11100100	$(PC) \leftarrow (PC) + 1$ $(A) \leftarrow 00h$
CPLA	1	1	11110100	$(PC) \leftarrow (PC) + 1$ $(A) \leftarrow \text{not}(A)$
RLA	1	1	00100011	$(PC) \leftarrow (PC) + 1$ $(An+1) \leftarrow (An) p/n=0a6$ $(A_v) \leftarrow (A_v)$
RLCA	1	1	00110011	$(PC) \leftarrow (PC) + 1$ $(An+1) \leftarrow (An) p/n=0a6$ $(A_v) \leftarrow (C) c(C) \leftarrow (A_v)$

Tabela 3.11 Conjunto de instruções da família de microcontroladores CS-51. (Continuação)

Operações lógicas				
Instrução	Byte(s)	Ciclos	Codificação	Representação simbólica
RR A	1	1	0000 0011	$(PC) \leftarrow (PC) + 1$ $(An) \leftarrow (An+1)$ p/ n = 0 a 6 $(A_0) \leftarrow (A_7)$
RRC A	1	1	0001 0011	$(PC) \leftarrow (PC) + 1$ $(An) \leftarrow (An+1)$ p/ n = 0 a 6 $(A_7) \leftarrow (C)$ e $(C) \leftarrow (A_0)$
SWAP A	1	1	1100 0100	$(PC) \leftarrow (PC) + 1$ $(A_{0-3}) \leftarrow (A_{7-4})$

Manipulação de variáveis booleanas				
Instrução	Byte(s)	Ciclos	Codificação	Representação simbólica
CLR C	1	1	1100 0011	$(PC) \leftarrow (PC) + 1$ $(C) \leftarrow 0b$
CLR bit	2	1	1100 0010 bit address	$(PC) \leftarrow (PC) + 2$ $(bit) \leftarrow 0b$
SETB C	1	1	1101 0011	$(PC) \leftarrow (PC) + 1$ $(C) \leftarrow \#1$
SETB bit	2	1	1101 0011 bit address	$(PC) \leftarrow (PC) + 2$ $(bit) \leftarrow \#1$
CPL C	1	1	1011 0011	$(PC) \leftarrow (PC) + 1$ $(C) \leftarrow \text{not} (C)$
CPL bit	2	1	1011 0010	$(PC) \leftarrow (PC) + 2$ $(bit) \leftarrow \text{not} (bit)$
ANL C,bit	2	2	1000 0010 bit address	$(PC) \leftarrow (PC) + 2$ $(C) \leftarrow (C) \text{ and } (\text{bit})$
ANL C,/bit	2	2	1000 0000 bit address	$(PC) \leftarrow (PC) + 2$ $(C) \leftarrow (C) \text{ and } (\text{not} (\text{bit}))$
ORL C,bit	2	2	0111 0010 bit address	$(PC) \leftarrow (PC) + 2$ $(C) \leftarrow (C) \text{ or } (\text{bit})$
ORL C,/bit	2	2	1010 0000 bit address	$(PC) \leftarrow (PC) + 2$ $(C) \leftarrow (C) \text{ or } (\text{not} (\text{bit}))$
MOV C, bit	2	1	1010 0010 bit address	$(PC) \leftarrow (PC) + 2$ $(C) \leftarrow (\text{bit})$
MOV bit, C	2	2	1001 0010 bit address	$(PC) \leftarrow (PC) + 2$ $(\text{bit}) \leftarrow (C)$
JC rel	2	2	0100 0000 rel address	$(PC) \leftarrow (PC) + 2$ If $(C) = 1$ then $(PC) \leftarrow (PC) + \text{rel}$
JNC rel	2	2	0101 0000 rel address	$(PC) \leftarrow (PC) + 2$ If $(C) = 0$ then $(PC) \leftarrow (PC) + \text{rel}$

Tabela 3.11 Conjunto de instruções da família de microcontroladores CS-51. (*Continuação*)

Manipulação de variáveis booleanas				
Instrução	Byte(s)	Ciclos	Codificação	Representação simbólica
JB bit, rel	3	2	0010 0000 bit address rel address	$(PC) \leftarrow (PC) + 3$ If (bit) = 1 then $(PC) \leftarrow (PC) + rel$
JNB bit, rel	3	2	0011 0000 bit address rel address	$(PC) \leftarrow (PC) + 3$ If (bit) = 0 then $(PC) \leftarrow (PC) + rel$
JBC bit, rel	3	2	0001 0000 bit address rel address	$(PC) \leftarrow (PC) + 3$ If (bit) = 1 then (bit) $\leftarrow 0$ $(PC) \leftarrow (PC) + rel\}$
Instruções de salto				
Instrução	Byte(s)	Ciclos	Codificação	Representação simbólica
ACALL addr ₁₁	2	2	a ₁₀ a ₉ a ₈ 1 0001 a ₇ a ₆ a ₅ a ₄ a ₃ a ₂ a ₁ a ₀	$(PC) \leftarrow (PC) + 2$ $(SP) \leftarrow (SP) + 1$ $((SP)) \leftarrow (PC_{7-0})$ $(SP) \leftarrow (SP) + 1$ $((SP)) \leftarrow (PC_{15-8})$ $(PC) \leftarrow addr_{11}$
LCALL addr ₁₆	3	3	0001 0010 addr ₁₅₋₈ addr ₇₋₀	$(PC) \leftarrow (PC) + 3$ $(SP) \leftarrow (SP) + 1$ $((SP)) \leftarrow (PC_{7-0})$ $(SP) \leftarrow (SP) + 1$ $((SP)) \leftarrow (PC_{15-8})$ $(PC) \leftarrow addr_{16}$
RET	1	2	0010 0010	$(PC) \leftarrow (PC) + 1$ $(PC_{15-8}) \leftarrow ((SP))$ $(SP) \leftarrow (SP) - 1$ $(PC_{7-0}) \leftarrow ((SP))$ $(SP) \leftarrow (SP) - 1$
RETI	1	2	0011 0010	$(PC) \leftarrow (PC) + 1$ $(PC_{15-8}) \leftarrow ((SP))$ $(SP) \leftarrow (SP) - 1$ $(PC_{7-0}) \leftarrow ((SP))$ $(SP) \leftarrow (SP) - 1$
AJMP addr ₁₁	2	2	a ₁₀ a ₉ a ₈ 0 0001 a ₇ ... a ₀	$(PC) \leftarrow (PC) + 2$ $(PC_{10-0}) \leftarrow addr_{11}$
LJMP addr ₁₆	3	2	0000 0010 addr ₁₅₋₀ addr ₇₋₀	$(PC) \leftarrow addr_{15-0}$
SJMP rel	2	2	1000 000 rel address	$(PC) \leftarrow (PC) + 2$ $(PC) \leftarrow (PC) + rel$
JMP @A+DPTR	1	2	0111 0011	$(PC) \leftarrow (A) + (DPTR)$
JZ rel	2	2	0110 0000 rel address	$(PC) \leftarrow (PC) + 2$ If (A) = 0 then $(PC) \leftarrow (PC) + rel$

Tabela 3.11 Conjunto de instruções da família de microcontroladores CS-51. (Continuação)

Instruções de salto				
Instrução	Byte(s)	Ciclos	Codificação	Representação simbólica
JNZ rel	2	2	0111 0000 rel address	$(PC) \leftarrow (PC) + 2$ If (A) $\neq 0$ then $(PC) \leftarrow (PC) + rel$
CJNE A,direct, rel	3	2	1011 0101 direct address rel address	$(PC) \leftarrow (PC) + 3$ IF (A) \neq (direct) then $(PC) \leftarrow (PC) + rel$ IF (A) $<$ (direct) then $(C) \leftarrow 1$ else $(C) \leftarrow 0$
CJNE A,#data, rel	3	2	1011 0100 immed. address rel address	$(PC) \leftarrow (PC) + 3$ IF (A) \neq #data then $(PC) \leftarrow (PC) + rel$ IF (A) $<$ #data then $(C) \leftarrow 1$ else $(C) \leftarrow 0$
CJNE Rn,#data, rel	3	2	1011 1rrr immed. address rel address	$(PC) \leftarrow (PC) + 3$ IF (Rn) \neq #data then $(PC) \leftarrow (PC) + rel$ IF (Rn) $<$ #data then $(C) \leftarrow 1$ else $(C) \leftarrow 0$
CJNE @Ri,#data,rel	3	2	1011 011i immed. address rel address	$(PC) \leftarrow (PC) + 3$ IF ((Ri)) \neq #data then $(PC) \leftarrow (PC) + rel$ IF ((Ri)) $<$ #data then $(C) \leftarrow 1$ else $(C) \leftarrow 0$
DJNZ Rn,rel	2	2	1101 1rrr rel address	$(PC) \leftarrow (PC) + 2$ $(Rn) \leftarrow (Rn) - 1$ IF (Rn) 0 then $(PC) \leftarrow (PC) + rel$
DJNZ direct,rel	3	2	1101 010 direct address rel address	$(PC) \leftarrow (PC) + 3$ $(direct) \leftarrow (direct) - 1$ IF (direct) 0 then $(PC) \leftarrow (PC) + rel$
NOP	1	1	0000 0000	$(PC) \leftarrow (PC) + 1$

Exercícios resolvidos

1 - A partir das condições iniciais dadas a seguir, execute (processe) teoricamente as instruções, mostrando os registradores e as posições de memória que são afetados pelo processamento de tais instruções e justifique o resultado obtido, considerando as escalas de 0 a 255 e de -128 a +127:

(A) = A6h; (PSW) = 9Ah; (PC) = 1000h; (20h) = 2Eh; (21h) = 7Dh; (22h) = 6Bh; (23h) = 5Ah; (24h) = 49h; (25h) = 38h; (26h) = 27h; (R0) = 37h; (R1) = 34h; (R2) = FBh; (R3) = 59h; (R4) = 4Eh; (R5) = 72h; (R6) = 3Ch; (R7) = 81h.

a) ADDC A,21h;

Solução: observando-se a Tabela 3.11, verifica-se que essa instrução apresenta um endereçamento direto. Assim, pode-se obter as seguintes informações:

ADDC A,direct	2	1	0011 0101 direct address	$(PC) \leftarrow (PC) + 2$ $(A) \leftarrow (A) + (C) + (\text{direct})$
---------------	---	---	--------------------------	--

Sua representação simbólica é dada por:

$$(PC) \leftarrow (PC) + 2$$

O conteúdo do registrador *Program Counter* (PC) é adicionado de duas unidades (aponta para o endereço de memória de programa da próxima instrução a ser executada pela CPU).

$$(A) \leftarrow (A) + (C) + (\text{direct})$$

Armazenará, no conteúdo do registrador acumulador, o resultado da operação de adição entre o conteúdo do registrador acumulador A, o conteúdo do *carry bit flag* e o conteúdo da posição de memória cujo endereço é 20h.

Assim, as condições iniciais dos registradores são:

$(A) = A6h$; $(C) = 1$, pois o conteúdo do registrador de função especial PSW é igual a 9Ah, como indicado a seguir:

	bit 7	bit 6	bit 5	Bit 4	bit 3	bit 2	bit 1	bit 0
$(PSW) = 9Ah =$	C	AC	F0	RS1	RS0	OV	-	P
	1	0	0	1	1	0	0	1

E o conteúdo da posição de memória cujo endereço 21h é igual a 7Dh. Portanto:

$(A) = A6h = 166_{10}$	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	+
$(21h) = 7Dh = 125_{10}$	'1	'0	'1	'0	'0	'1	'1	0	
$(C) = 1$, pois	0	1	1	1	1	1	0	1	
$(A) \leftarrow (A) + (C) + (21h) =$									
	'0	0	1	0	0	1	0	0	$= 24h = 36_{10}$

Além de o conteúdo do acumulador mudar do valor A6h para 36h pela Tabela 3.5, o conteúdo do registrador *Program Status Word* (PSW), por meio dos bits (C), (OV) e (AC) também muda para:

- ◆ (C) = foi um do bit 7 = 1. O resultado não pode ser armazenado em 8 bits.
- ◆ (AC) = foi um do bit 3 = 1.
- ◆ (F0) = não é alterado por essa operação e permanece com o estado anterior = 0.
- ◆ (RS1) = não é alterado por essa operação e permanece com o estado anterior = 1.
- ◆ (RS0) = não é alterado por essa operação e permanece com o estado anterior = 1. A composição dos dois bits define o banco 3 de registradores (R0 - endereço de memória 17h → R7 - endereço de memória 1Fh).
- ◆ (OV) = (foi um do bit 7) OR-EX (foi um do bit 6) = 1 OR-EX 1 = 0. Está na faixa de - 128 a +127.
- ◆ (P) = reflete a quantidade de números 1 contidos no conteúdo do acumulador A, que é igual a dois (2) = paridade par = 0.

Assim, o conteúdo do registrador PSW fica sendo:

(PSW) =	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
	C	AC	F0	RS1	RS0	OV	-	P
	1	1	0	1	1	0	0	0

= D8h

- ◆ Justificativa do resultado encontrado, considerando-se a escala de 0 a 255:

$$\begin{array}{rcl}
 (A) = & A6h = & 166_{10} \\
 (C) = & 1 = & 1_{10} \\
 (21h) = & 7Dh = & 125_{10} \\
 \hline
 & 124h = & 292_{10} = \\
 & & 1.16^2 + 2. 16^1 + 4. 16^0 > 255_{10}
 \end{array}
 \quad (\textbf{C})$$

- ◆ Justificativa do resultado encontrado, considerando-se a escala de - 128 a +127:

Como o conteúdo do registrador A é igual a A6h = 1010 0110_b, esse número é negativo, pois apresenta seu bit mais significativo igual a 1 e está representado em complemento de dois. Sendo assim, considerando-se o valor do conteúdo do registrador A na escala de - 128 a +127:

$$\begin{array}{r}
 (A) = A6h = 1010\ 0110 \\
 (A)_{C1} = 59h = 0101\ 1001 \\
 1h = 1 \\
 \hline
 (A)_{C2} = 5Ah = 0101\ 1010
 \end{array}
 = -(5 \cdot 16^1 + 10 \cdot 16^0) = -90_{10} \text{ (menos noventa) na escala de -128 a +127}$$

Assim, o resultado da adição de tais registradores é:

$$\begin{array}{r}
 (A) = A6h = -90_{10} \\
 (C) = 1 = + 1_{10} \\
 (21h) = 7Dh = +125_{10} \\
 \hline
 '24h = 36_{10}
 \end{array}
 = 2 \cdot 16^1 + 4 \cdot 16^0 < 127 \text{ (dentro da escala de -128 a +127) } \Rightarrow (OV) = 0$$

2 - Defina o tipo de endereçamento de cada uma das instruções dadas a seguir:

a) ADDC A,20h

Resposta: endereçamento direto.

b) MOV @R0,55h

Resposta: endereçamento combinado (misto), formado pelos endereçamentos direto e indireto ou indexados por registrador.

3 - Crie um programa que faça a operação *OR-Exclusivo* entre o conteúdo do acumulador (A) e a constante 33h. O resultado deve ser armazenado no conteúdo do registrador R5 do segundo banco de registradores.

Solução:

XRL	A,#33h	(A) \leftarrow (A) or-ex #33h
MOV	PSW,#08h	(PSW) \leftarrow #08h = # 0000 1000 \Rightarrow (RS1) = 0 e (RS0) = 1 \Rightarrow define segundo banco de registradores (B1)
MOV	R5,A	Armazena, no conteúdo do registrador R5 do segundo banco de registradores, o conteúdo do acumulador A.

Exercícios e questões propostos

1 - Defina o registrador de função especial *Program Status Word* (PSW), bem como todos os seus bits e descreva a finalidade de cada um deles.

2 - A partir das condições iniciais oferecidas a seguir, execute (processe) teoricamente as instruções, mostrando os registradores e as posições de memória afetados pelo processamento de tais instruções:

(A) = E7h; (PSW) = BCh; (20h) = 2Fh; (21h) = 7Eh; (22h) = 8Ch; (23h) = 5Bh; (24h) = 6Ah; (25h) = 39h; (26h) = 4Ch; (R0) = 26h; (R1) = 22h; (R2) = E1h; (R3) = 7Dh; (R4) = 8Ch; (R5) = 7Bh; (R6) = 6Ah; (R7) = 59h.

- a) ADDC A,21h
- b) SUBB A,20h
- c) ADD A,22h
- d) ANL A,#23h
- e) ORL A,26h
- f) INC A
- g) DEC A
- h) SWAP A
- i) MUL AB
- j) DIV AB
- k) RRA
- l) RLCA
- m) MOV @R1,A
- n) ADDC A,@R1
- o) SUBB A,@R0
- p) XCHD A,@R0

3 - Defina os diferentes modos de endereçamento das instruções do item 2.

4 - Defina os tipos de endereçamento de cada uma das instruções mostradas a seguir:

- a) ADDC A,20h
- b) SUBB A,@R0
- c) ADD A,#23h
- d) ANL A,24h
- e) ORL A,25h
- f) INC A
- g) DEC A
- h) SWAP A
- i) MUL AB
- j) DIV AB
- k) RRA
- l) RLC A
- m) MOV @R1,A
- n) ADDC A,@R1
- o) SUBB A,@R0
- p) XCHD A,@R0
- q) MOV 20H,R6

5 - Considerando as condições iniciais do Exercício 1, calcule o conteúdo do registrador *Program Counter* (PC) para as seguintes instruções:

Endereço	Mnemônico	Argumento(s)	(PC) = ?
0142h	MOV	23h,@R1	
0208h	LJMP	30h	
0319h	DJNZ	R0, 0FAh	

6 - Crie um programa que adicione o conteúdo do acumulador (A) à constante ACh e que armazene o resultado no conteúdo da posição de memória, cujo endereço é 4Bh.

7 - Crie um programa que execute a operação lógica AND dos conteúdos: do acumulador A, da posição de memória cujo endereço é dado pelo conteúdo do registrador R1 e do registrador R5. Armazene o resultado no conteúdo do registrador R1 do terceiro banco de registradores.

- 8 - Crie um programa que faça a operação lógica OR-EX dos conteúdos: da posição de memória cujo endereço é 78h e da posição de memória cujo endereço é 57h. O resultado deve ser armazenado no conteúdo do registrador R3 do segundo banco de registradores.
- 9 - Crie um programa que execute a rotação de um bit do conteúdo da posição de memória cujo endereço é 26h. O resultado deve ser somado com o *carry bit flag* e com o conteúdo da posição de memória cujo endereço é 44h. Com o resultado, faça uma operação lógica OR-EX com a constante 7Dh. O resultado deve ser armazenado na posição de memória cujo endereço é 48h.

4.1 Objetivos

- ❖ Estratégias de elaboração de programas em Assembly
- ❖ Fluxogramas de programas simples e programas com loop
- ❖ Exemplos de programas simples e com loop

4.2 Introdução teórica

Este capítulo é de fundamental importância para quem deseja adquirir conhecimentos básicos em programação (software). Aqui, é ensinado passo a passo o procedimento da elaboração de programas simples e com loop, utilizando a linguagem Assembly aplicada aos microcontroladores da família MCS-51 da Intel.

Uma vez que o leitor aprenda e pratique as técnicas de programação ensinadas neste capítulo, ele poderá aplicar tal metodologia de implementação de programas a qualquer outra linguagem de programação, seja ela de baixo, médio ou alto nível.

Existem muitas linguagens de programação que podem ser utilizadas na programação de sistemas inteligentes, utilizando microcontroladores, além do Assembly (nível baixo), tais como as linguagens estruturadas de programação C (nível médio), BASIC e PL/M (nível alto). A escolha de uma delas na elaboração de um projeto de software é muito importante e envolve o conhecimento técnico especializado.

A característica de uma linguagem de nível baixo, como o Assembly, é a de apresentar o mesmo conjunto de instruções que aquele definido pelo fabricante da família de microprocessadores ou microcontroladores, com a qual

se deseja desenvolver os projetos de programa. Isso permite um total controle sobre os registradores internos do dispositivo e as posições de memória e, consequentemente, são gerados programas muito mais compactos em relação às outras linguagens. Assim, sempre que existirem limitações de memória (pouca memória) e limitações de velocidade de processamento (máxima velocidade de processamento), recomenda-se a utilização da linguagem de programação de nível baixo, ou seja, o Assembly. Portanto, pelas próprias características dos microcontroladores, é muito comum a utilização do Assembly na elaboração de projetos de equipamentos inteligentes. Em contrapartida, a única desvantagem da linguagem Assembly em relação às outras linguagens é sua característica de ser rica em detalhes, solicitando dos programadores um conhecimento bastante grande sobre o dispositivo a ser programado.

A característica de uma linguagem de nível médio, como o C, é a de apresentar os aspectos de uma linguagem de nível baixo (manipulação bit a bit ou byte a byte dos registradores internos e das posições de memória), como também as características de uma linguagem de nível alto, que definimos a seguir. Atualmente, essa é uma das linguagens de programação mais utilizadas em projetos de equipamentos inteligentes.

A característica de uma linguagem de nível alto é a de ser muito mais próxima da linguagem do ser humano. Trata-se de uma linguagem descritiva, cujas instruções são palavras escritas em inglês e que estão muito longe das instruções em Assembly. Cada instrução de uma linguagem de nível alto é representada por muitas instruções de uma linguagem de nível baixo. Programar utilizando uma linguagem de nível alto é muito mais fácil que programar com uma linguagem de nível baixo ou médio. Mas quando se utiliza uma linguagem de programação de nível alto, são gerados programas maiores e com menor velocidade de processamento. Outra desvantagem da linguagem de nível alto, que muitas vezes é interpretada como vantagem, é a de não exigir do programador o conhecimento completo e detalhado do dispositivo que se deseja programar.

Um programa em código de máquina (código-objeto) é composto por bytes que representam as instruções do dispositivo. É a linguagem que o microcontrolador sabe e pode executar. Assim, sempre que for desenvolvido um programa em uma determinada linguagem de programação, é necessário fazer a *compilação* desse programa para transformá-lo em linguagem de máquina. O programa em linguagem de máquina deve ser simulado por meio de um simulador (AVSIM51, Pinnacle etc.) ou gravado em uma memória

para verificar seu funcionamento por meio da utilização do próprio hardware do produto.

A seguir, inicia-se o estudo das técnicas de programação em Assembly para a família de microcontroladores MCS-51 da Intel.

4.2.1 - Estratégias de elaboração de programas: é muito importante para um programador, antes de implementar um programa, seguir uma determinada metodologia. A seguir, é descrita uma metodologia de elaboração de programas que é independente da linguagem de programação.

A – Pré-requisitos do técnico que desenvolverá o projeto

- a) conhecer a estrutura interna (hardware) do dispositivo com que se deseja desenvolver o projeto;
- b) conhecer o conjunto de instruções do dispositivo.

B – Viabilidade do projeto

- a) obter e entender claramente as especificações do cliente (desejos do cliente, características do produto e do sistema, modo de funcionamento etc.);
- b) elaborar diferentes estratégias de soluções para o projeto. Isso pode ser feito utilizando-se diagramas de fluxo de dados (DFD), diagramas de blocos, desenhos etc.;
- c) Elaborar a viabilidade do projeto com relação ao hardware e ao software para verificar a possibilidade de sua implementação junto ao cliente. Algumas perguntas são utilizadas para garantir a viabilidade do projeto: Quais são as necessidades de hardware? Qual será o custo final do produto (o cliente pagará por esse valor)? Qual será a quantidade de equipamentos a ser fabricada (a fábrica comporta essa quantidade)? Como serão feitas as entregas do produto? Existem fornecedores de componentes no Brasil? Quais componentes devem ser importados? Existem concorrentes? etc.;
- d) Escolher a melhor solução junto com o cliente, ou seja, aquela que utilizará o menor número de componentes e a que apresenta maior velocidade de processamento. Isso afeta diretamente o custo final do hardware e traz uma maior flexibilidade funcional do produto.

C - Implementação do projeto

- a) elaborar o fluxograma do programa em Assembly: os fluxogramas devem ser implementados desde o nível de 'macrodetalhamento' até o nível de 'microdetalhamento'. Um fluxograma representando o nível de 'macrodetalhamento' deve conter as idéias e os objetivos dentro dos blocos representativos. Um fluxograma representando o nível de 'microdetalhamento' deve utilizar as representações simbólicas dos *mнемônicos* das instruções dentro dos blocos representativos. Exemplo: dentro de um bloco do fluxograma, representar (A) ← (B): significa a instrução MOV A,B;
- b) a partir do fluxograma, gerar o programa-fonte na linguagem de programação escolhida (arquivo em código ASCII que contém as instruções que compõem o programa);
- c) compilar o programa-fonte para gerar o arquivo binário que corresponde ao programa em códigos de máquina;
- d) fazer a *linkagem* dos arquivos binários para agrupar, de maneira organizada, os diferentes blocos de programa que compõem o programa final.
- e) efetuar a simulação do programa para verificar seu funcionamento e corrigir os possíveis erros de lógica. Nessa fase, não é necessário o hardware do produto;
- f) realizar a gravação da EPROM, inserção no hardware do produto e teste físico de sua funcionalidade;
- g) efetuar a emulação do programa (é necessário um sistema emulador) para corrigir erros de lógica ou eventuais falhas de algum componente, utilizando o próprio hardware do produto;
- h) fazer testes rápidos de bancada para verificar o funcionamento do produto final;
- i) realizar testes de longa duração (confiabilidade), em condições de 'estresse', para a homologação do produto.

4.2.2 - Fluxogramas: essa ferramenta de programação é fundamental para a implementação de qualquer rotina de um programa de microcomputador. É por meio dessa ferramenta que se pode observar como foi implementada a estratégia de solução de uma necessidade de programação elaborada por um programador. O fluxograma representa como o fluxo de informações será processado pelo microprocessador.

Neste livro, serão utilizados apenas quatro símbolos básicos para a representação de um fluxograma: a linha de conexão, a elipse, o retângulo e o losango. A seguir, são definidos cada um dos blocos integrantes de um fluxograma.

- a) *Linha de conexão*: representa o fluxo de informações a ser processado pelo microprocessador. É também utilizado para conectar os blocos básicos de um fluxograma. As setas para cima (\uparrow), para baixo (\downarrow), para o lado direito (\rightarrow) e para o lado esquerdo (\leftarrow) são empregadas para esse fim.
- b) *Elipse*: representa o início e o fim do programa. Quando utilizada no início do programa, deve conter o nome da rotina em seu interior. É importantíssimo que esse nome sempre esteja associado ao seu objetivo ou à sua função. Isso facilita o entendimento e a consulta à documentação do programa. Por exemplo, para uma rotina que lê as teclas de um teclado, um possível nome seria 'Ler teclado', como mostra a Figura 4.1.

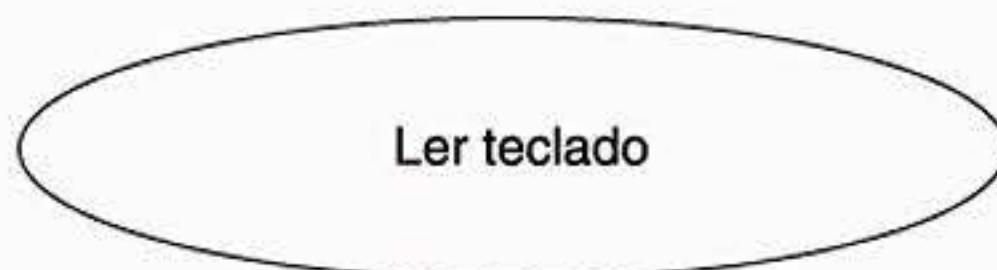


Figura 4.1 Representação, por meio de uma elipse, do início de um programa chamado 'Ler teclado'.

Quando a elipse for utilizada no fim da rotina, deve-se escrever 'FIM e o nome da rotina', como mostra a Figura 4.2.

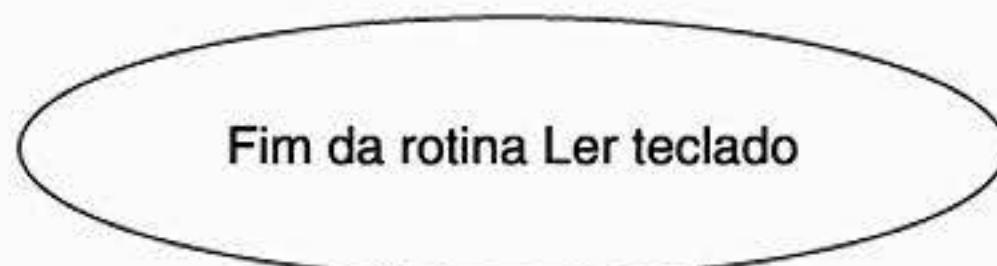


Figura 4.2 Representação, por meio de uma elipse, do fim de um programa chamado 'Ler teclado'.

- c) *Retângulo*: representa o processamento das informações, ou seja, as operações de movimentação de dados e endereços, as operações aritméticas, lógicas e de rotação. Um exemplo é mostrado na Figura 4.3.

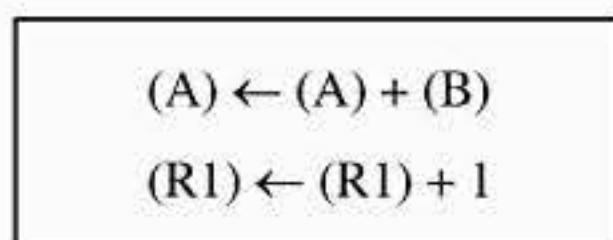


Figura 4.3 Representação do processamento de informações por meio de um retângulo.

- d) *Losango*: representa uma tomada de decisão. Isso é feito por meio de um teste comparativo da informação que está disponível no conteúdo de um bit de um registrador ou de uma posição de memória com um determinado valor. Se a condição for satisfeita, será definido um determinado caminho para que a informação seja processada; caso contrário, será definido outro caminho para que a informação seja processada. A Figura 4.4 exemplifica duas maneiras de representação de tomada de decisão utilizadas em um losango.

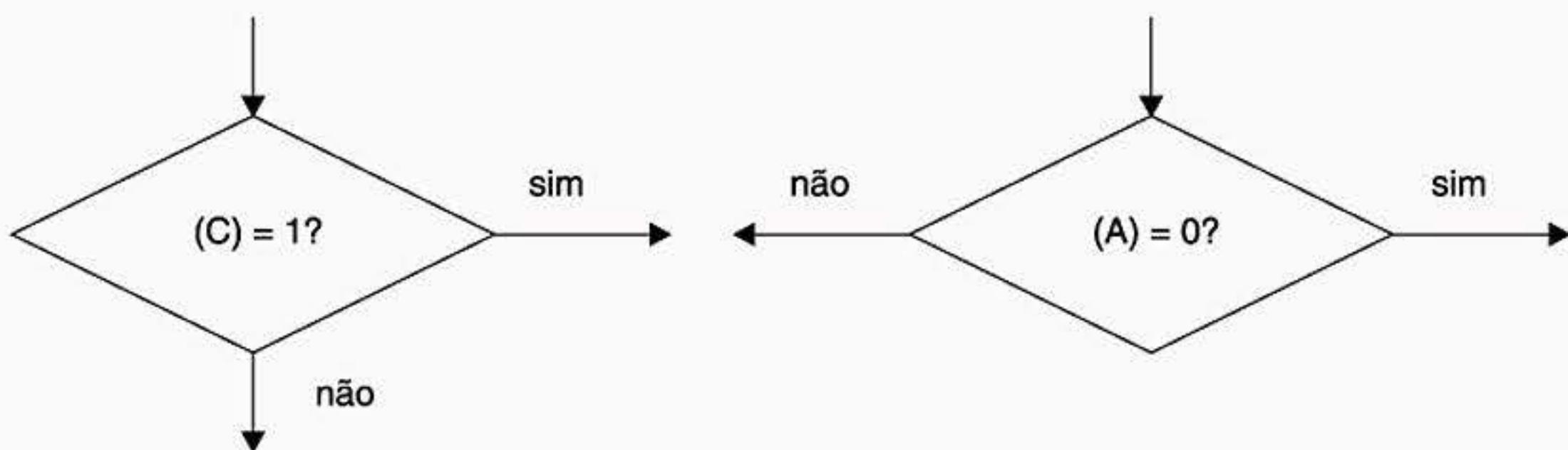


Figura 4.4 Representações de blocos de tomada de decisão por meio de um losango.

4.2.3 – Programas simples: o fluxograma de uma rotina de um programa simples (Figura 4.5) sempre deve conter, no mínimo, cinco blocos básicos:

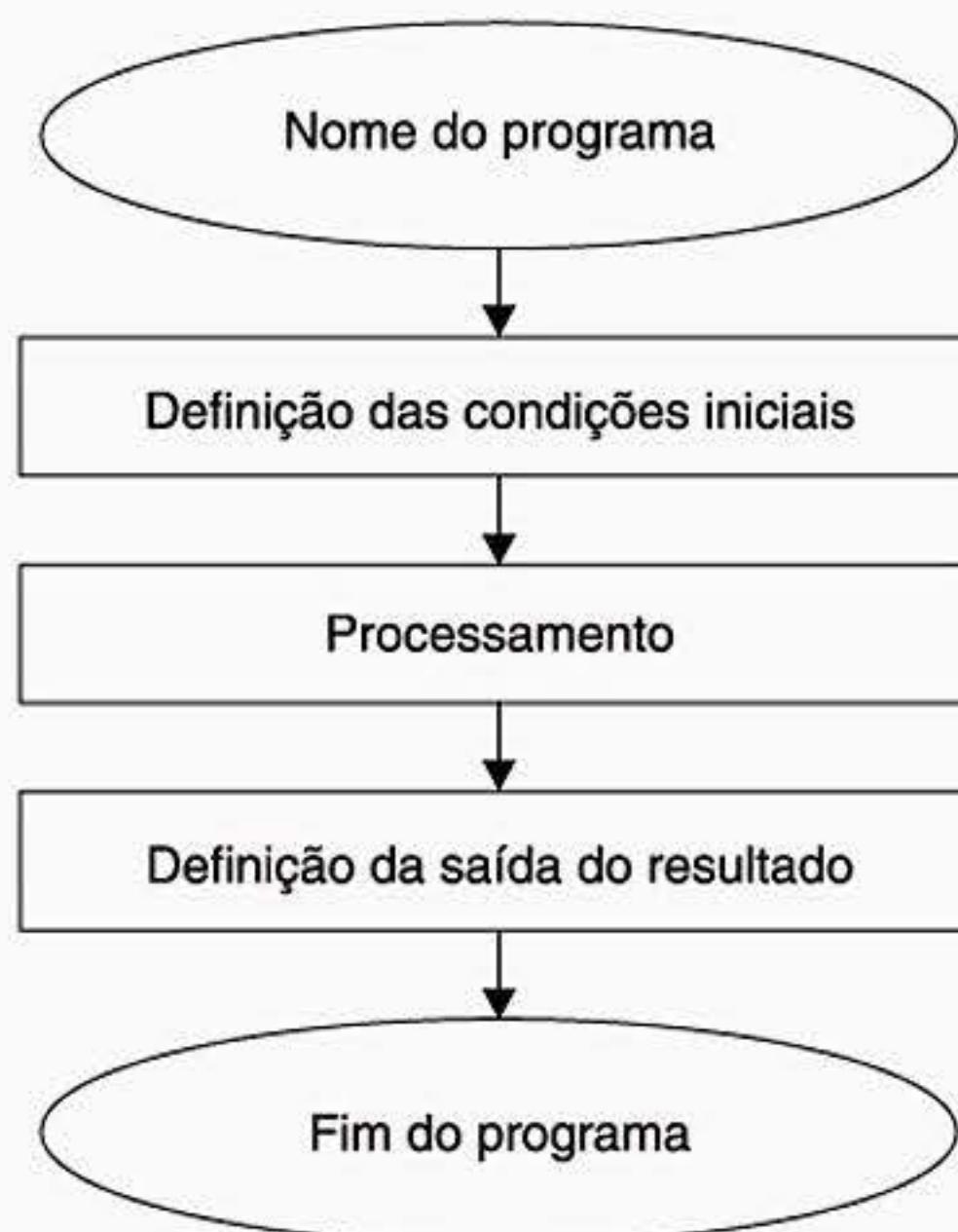


Figura 4.5 Representação típica de um fluxograma de uma rotina de um programa simples.

- 1 - Um primeiro bloco deve conter o nome do programa associado ao seu objetivo ou a sua função (elipse).
- 2 - Um segundo bloco deve conter as condições iniciais dos bits, dos registradores e das posições de memória que serão utilizados pelo programa. Esses bits, registradores e posições de memória terão uma função bastante específica dentro do programa e poderão ser considerados como variáveis de entrada do programa. As condições iniciais devem ser representadas dentro de um retângulo.
- 3 - Um terceiro bloco faz o processamento da informação, que pode ser realizado por meio de operações de movimentação de informações, de operações aritméticas e lógicas, de rotação etc., que devem ser representadas dentro de um retângulo.
- 4 - Um quarto bloco deve definir onde será armazenada a informação resultante do processamento. Esse armazenamento poderá ser feito em um registrador, em uma posição de memória ou em uma unidade de entrada e saída. Devem ser representadas dentro de um retângulo.
- 5 - Um último bloco deve simbolizar o fim do programa (elipse).

4.2.4 - Programas com loop: um possível fluxograma de um programa com loop (Figura 4.6) deve sempre conter no mínimo oito blocos básicos:

- 1 - Um primeiro bloco deve conter o nome do programa, que sempre deve estar associado ao seu objetivo ou a sua função (elipse).
- 2 - Um segundo bloco deve conter as condições iniciais dos bits, dos registradores ou das posições de memória e, agora, também das informações do buffer de memória de dados a serem analisados. Um buffer de memória de dados é o local no qual são armazenadas as informações que serão processadas. A característica de um buffer de memória de dados é que as informações estão armazenadas em posições de memória subsequentes, ou seja, em endereços de memória subsequentes. Geralmente, pode-se definir matematicamente um buffer de memória de três diferentes maneiras:
 - ◆ por meio de seu endereço inicial e final;
 - ◆ por meio de seu endereço inicial e da quantidade de elementos do *buffer* de memória;
 - ◆ por meio da quantidade de elementos do *buffer* de memória e de seu endereço final.

As condições iniciais devem ser representadas dentro de um retângulo.

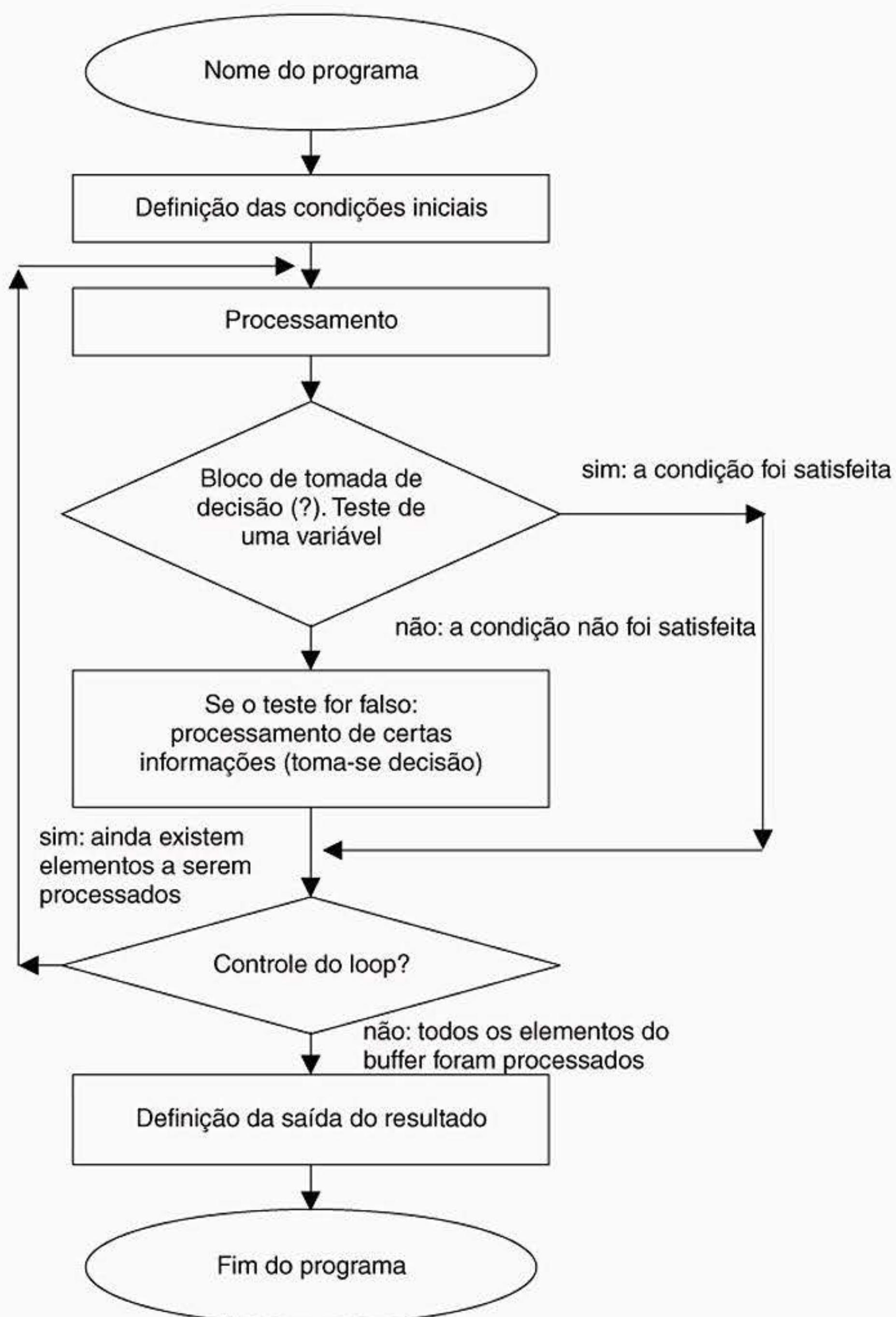


Figura 4.6 Representação típica de um fluxograma de uma rotina de um programa com loop.

3 - Um terceiro bloco terá a função de processar a informação. Esse processamento é feito basicamente por meio de uma operação de movimentação entre registradores e posições de memória, em combinação com

operações aritméticas e lógicas. Ao fazer uma operação aritmética ou lógica utilizando a ULA, no caso dos microcontroladores da família MCS-51, sabe-se que o registrador de função especial *Program Status Word* (PSW) também é influenciado por meio de seus bits de sinalização *carry bit* (C), *auxiliar carry bit* (AC), *overflow* (OV) e *parity* (paridade) (P). Assim, esse bloco define a condição matemática ou lógica do programa por meio do conteúdo do registrador de função especial PSW. Esse bloco deve ser representado dentro de um retângulo.

- 4 - Um quarto bloco, o de tomada de decisão. Esse bloco é responsável pelo teste da condição matemática ou lógica do programa definida no bloco anterior. Esse teste pode consistir na verificação dos bits de sinalização *carry bit* (C), *auxiliar carry bit* (AC), *overflow* (OV) e *parity* (paridade) (P) do registrador de função especial PSW, da condição matemática ou lógica do conteúdo do registrador acumulador (A), da condição lógica de bits de registradores de funções especiais endereçáveis por bit ou de posições de memória etc. Caso a condição seja satisfeita (verdadeira), será definido um caminho de processamento da informação a ser executado pelo microprocessador; caso contrário, será definido um outro caminho de processamento de informação. O teste deve ser representado dentro de um losango.
- 5 - Um quinto bloco conterá as instruções que devem ser executadas quando a condição do item 4 não for satisfeita.
- 6 - Um sexto bloco fará o controle do loop, verificando se todas as informações do buffer foram processadas (analisadas). Geralmente, isso é feito de duas maneiras, dependendo de como se deseja controlar o buffer de memória:
 - a) quando o buffer de memória é controlado pela sua posição inicial e por sua quantidade de elementos, o controle é feito apontando para a próxima posição do buffer de memória e diminuindo a quantidade de elementos a serem analisados, cada vez que for processada uma informação do buffer. Quando a quantidade de elementos do buffer de memória for diferente de zero, o caminho de processamento será desviado para o início do processamento do programa (para o item 3) e, dessa maneira, será realizado um novo processamento com o próximo conteúdo da posição de memória do buffer (outro elemento do buffer de memória). Caso a quantidade de elementos do buffer a serem analisados seja igual a zero, isso significa dizer que todos os elementos do buffer já foram processados, que o processamento será encerrado;

- b) Quando o buffer de memória é controlado pela sua posição inicial e pela sua posição final, o controle é feito apontando-se para a próxima posição do buffer de memória, cada vez que uma informação for processada, e deve-se comparar tal endereço com o endereço final do buffer, adicionado de uma unidade. Enquanto o endereço da última informação analisada pelo programa é diferente do endereço final adicionado de uma unidade, o caminho de processamento é desviado para o início do processamento (para o item 3) e, dessa maneira, será feito um novo processamento com o próximo conteúdo da posição de memória do buffer. Caso o endereço da última informação analisada pelo programa seja igual ao endereço final do buffer adicionado de uma unidade, o processamento será encerrado.

7 - Um sétimo bloco deve definir onde será armazenada a informação resultante do processamento. Essa informação poderá ser armazenada em um registrador, em uma posição de memória ou em uma unidade de entrada e saída.

8 - Um último bloco deve simbolizar o fim do programa.

4.2.5 - Exemplo de programa simples: fazer um fluxograma e um programa-fonte em Assembly para o microcontrolador 8051, que sejam capazes de realizar a operação de adição entre 2 bytes que estão localizados nos conteúdos das posições de memória, cujos endereços são 45h e 5Ah. O resultado da operação de adição deve ser armazenado no conteúdo da posição de memória, cujo endereço é 6Dh.

Solução: em primeiro lugar, é necessário entender perfeitamente o que está sendo solicitado. Um desenho em forma de rascunho (Figura 4.7) é uma ferramenta importante, nesse caso.

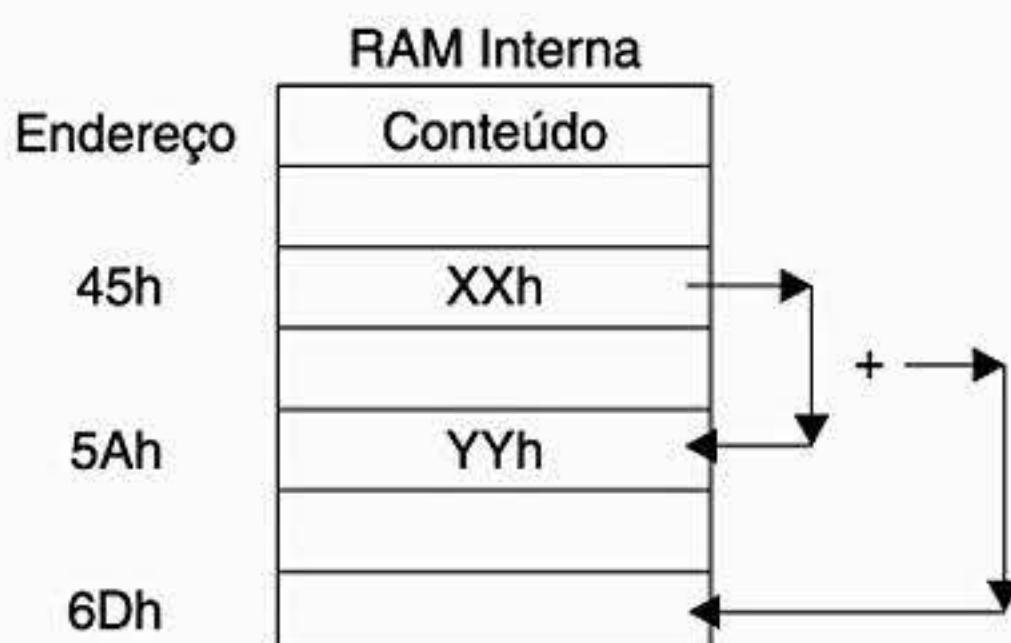


Figura 4.7 Desenho em forma de rascunho para ajudar no entendimento do que está sendo solicitado.

Depois disso, é necessário montar a estratégia de solução do problema, o que exige verificar quais são as instruções capazes de fazer a operação de adição. Verifique na Tabela 3.11, do Capítulo 3, o conjunto de instruções da família de microcontroladores MCS-51, para obter quatro instruções que realizam a operação de adição:

- 1 - ADD A,Rn $\Rightarrow (A) \leftarrow (A) + (Rn)$: no conteúdo do acumulador será armazenado o resultado da operação de adição entre o conteúdo do acumulador e o conteúdo do registrador R0.

Estratégia de solução: repare que a instrução dada anteriormente utiliza dois registradores para fazer uma operação de adição: o conteúdo do registrador acumulador A e o conteúdo de um dos registradores de um dos bancos selecionados (R0 a R7). Dessa maneira, para implementar um programa utilizando essa instrução, é necessário realizar duas operações de movimentação de informações, da memória para o acumulador e da memória para um registrador, antes de executar essa instrução. Assim, transferindo o conteúdo da posição de memória, cujo endereço é 45h, para o conteúdo do acumulador A (MOV A,45h), e do conteúdo da posição de memória, cujo endereço é 5Ah, para o conteúdo do registrador R0, por exemplo (MOV R0, 5Ah), pode-se executar a instrução ADD A,R0. Como o resultado da operação de adição fica armazenado no conteúdo do acumulador (A), ele deve ser armazenado no conteúdo da posição de memória, cujo endereço é 6Dh (MOV 6Dh, A). O fluxograma e o programa-fonte são apresentados na Figura 4.8.

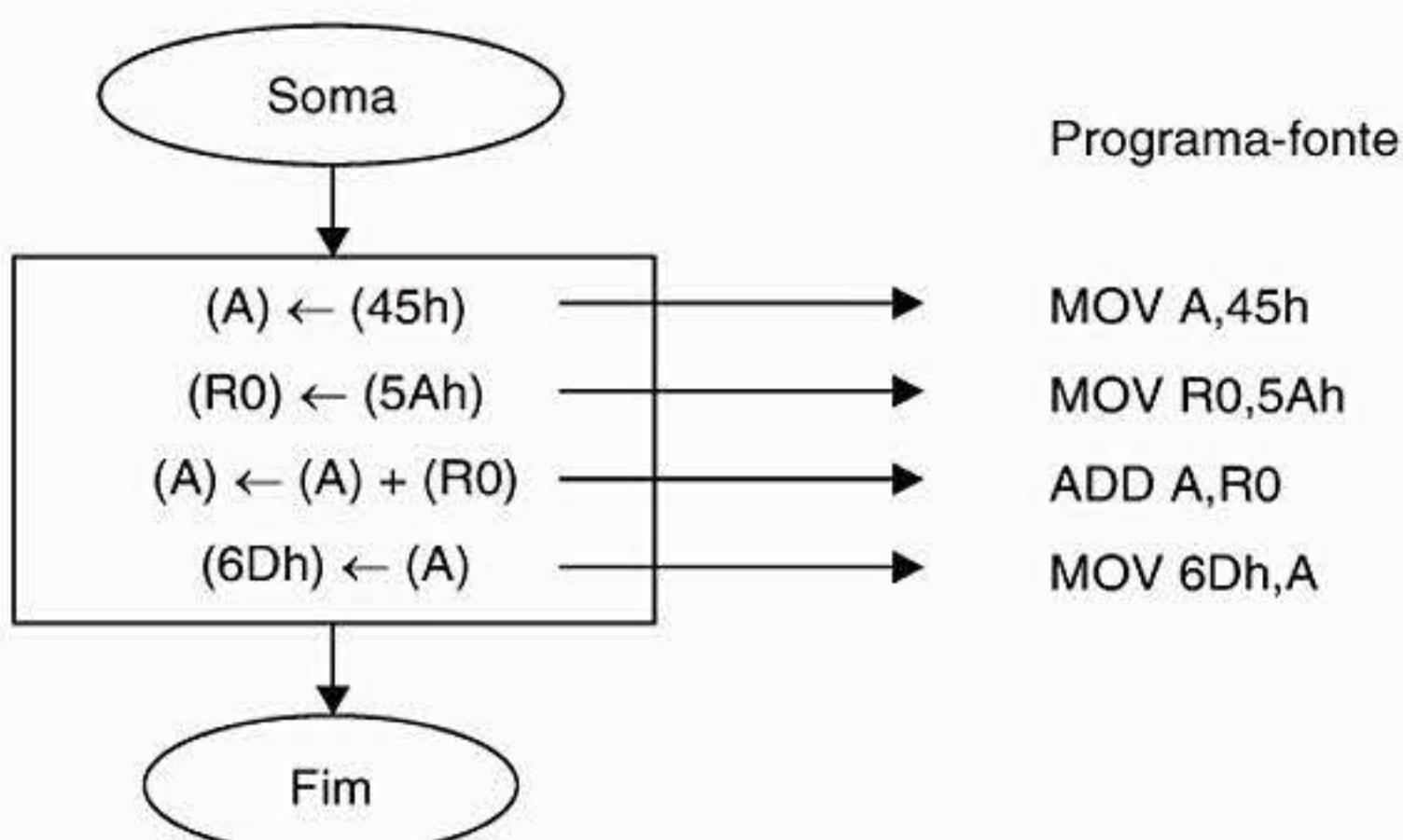


Figura 4.8 Programa de adição usando a instrução ADD A,Rn.

- 2 - ADD A,*direct* $\Rightarrow (A) \leftarrow (A) + (\text{direct})$: no conteúdo do acumulador será armazenado o resultado da operação de adição entre o conteúdo do acumulador e o conteúdo da posição de memória, cujo endereço é *direct*.

Estratégia de solução: repare novamente que o conteúdo do acumulador A é um dos registradores envolvidos nessa instrução e o outro argumento é o conteúdo de uma posição de memória definido por endereçamento direto. Assim, para resolver esse problema utilizando essa instrução, é necessário fazer apenas uma operação de movimentação de informação, isto é, do conteúdo da posição de memória, cujo endereço é 45h, para o conteúdo do acumulador A (MOV A,45h). Após isso, basta utilizar a instrução dada anteriormente, definindo 5Ah como *direct* (argumento da instrução), ou seja, ADD A,5Ah. Como o resultado fica armazenado no conteúdo do acumulador A, ele deve ser armazenado no conteúdo da posição de memória, cujo endereço é 6Dh. (MOV 6Dh, A). O fluxograma e o programa-fonte são apresentados na Figura 4.9.

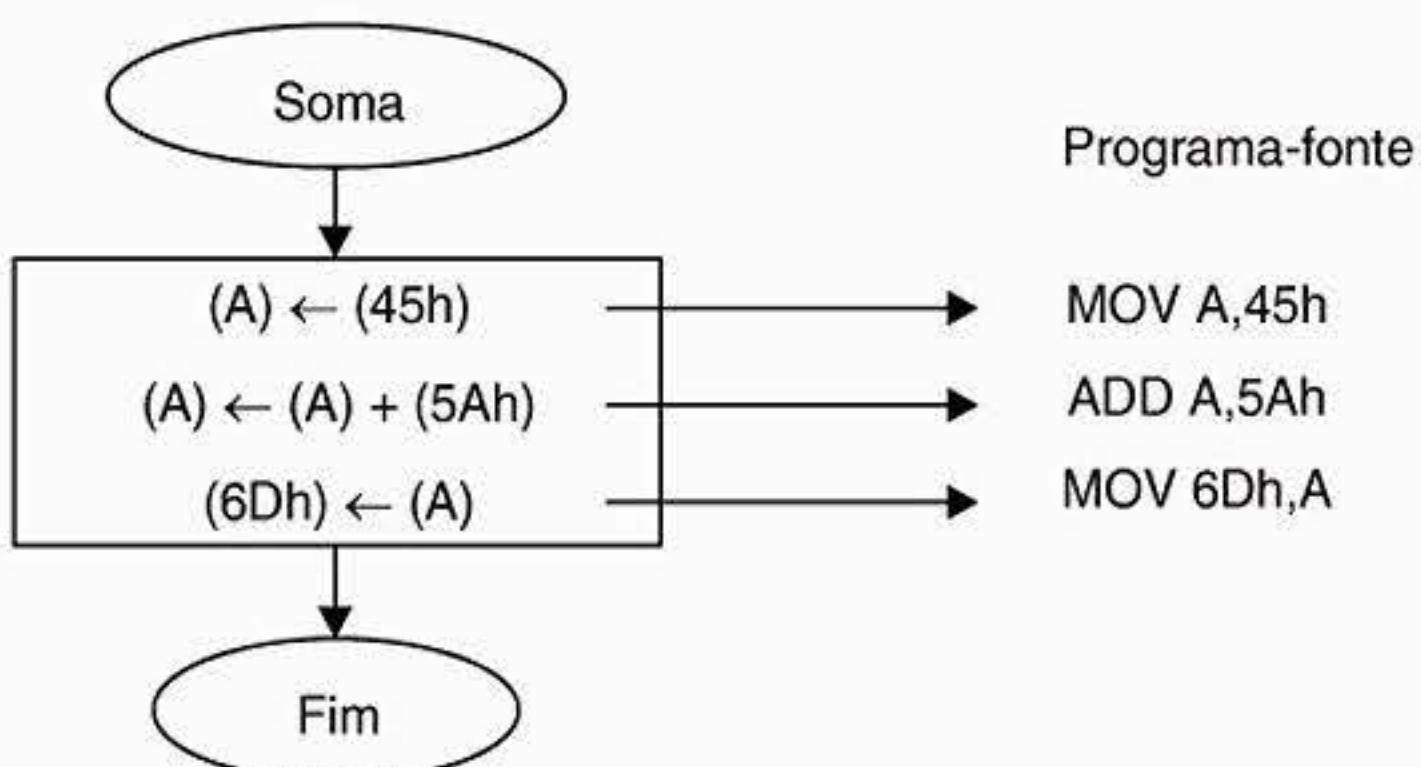


Figura 4.9 Programa de adição usando a instrução ADD A,*direct*.

- 3 - $\text{ADD } A, @Ri \Rightarrow (A) \leftarrow (A) + ((Ri))$: no conteúdo do acumulador, será armazenado o resultado da operação de adição entre o conteúdo do acumulador e o conteúdo da posição de memória, cujo endereço é dado pelo conteúdo do registrador R0.

Estratégia de solução: novamente, o conteúdo do registrador acumulador é um dos argumentos da instrução, e o outro é dado por um endereçamento indireto ou indexado por registrador. Para criar o programa utilizando essa instrução, são necessárias duas operações de movimentação de informação, isto é, uma do conteúdo da posição de memória, cujo endereço é 45h, para o conteúdo do acumulador A (MOV A,45h), e a outra consiste em inicializar o conteúdo do registrador R0, por exemplo, com o valor do endereço da memória 5Ah (MOV R0, #5Ah). Após isso, basta fazer a operação de adição utilizando a instrução ADD A,@R0. Como o resultado é armazenado no conteúdo do acumulador A, ele deve ser armazenado no conteúdo da posição de memória cujo endereço é 6Dh

(MOV 6Dh,A). O fluxograma e o programa-fonte são apresentados na Figura 4.10.

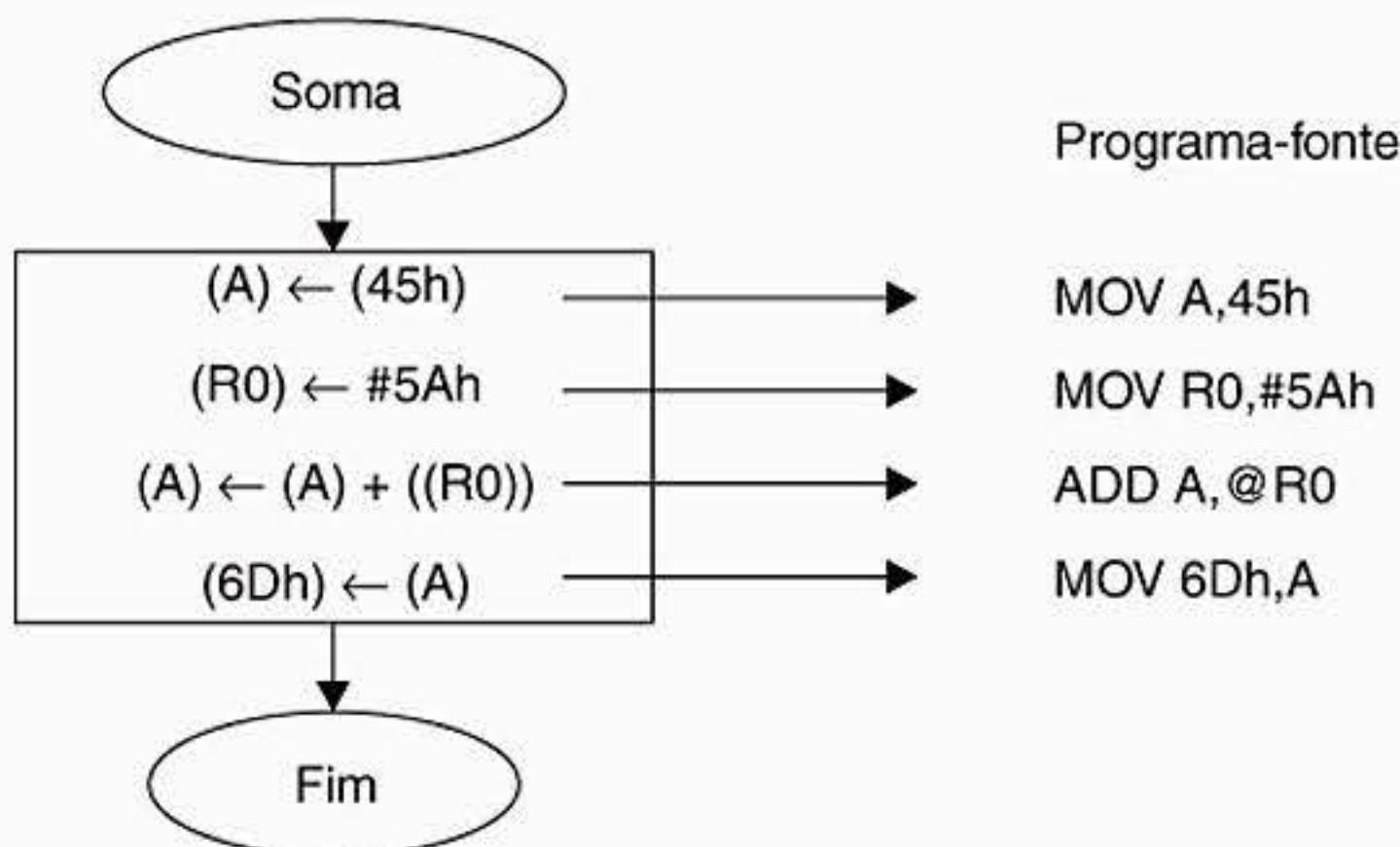


Figura 4.10 Programa de adição usando a instrução ADD A,@Ri.

- 4 - ADD A, #data. $\Rightarrow (A) \leftarrow (A) + \#data$: no conteúdo do acumulador será armazenado o resultado da operação de adição entre o conteúdo do acumulador e a constante *data*.

Estratégia de solução: não há solução, pois não é possível saber o conteúdo de uma das posições de memória, ou seja, essa posição de memória simula uma variável e, dessa maneira, utilizando essa instrução, não é possível resolver esse problema.

Conclusões:

- Embora existam quatro tipos de instruções de adição (ADD) diferentes, há somente três soluções;
- A melhor solução, entre as apresentadas anteriormente, é aquela que utiliza a instrução de adição com endereçamento direto (ADD A,*direct*), pois o programa final em Assembly só utiliza três instruções e as demais soluções utilizam quatro instruções. Isso resulta em um menor gasto de memória de programa e em uma maior velocidade de processamento que as demais, preservando, assim, a melhor relação custo/benefício;
- Sempre é preciso escolher a melhor solução (solução ótima). Isso significa escolher aquela que utiliza menos posições de memória de programa e que, geralmente, é a mais rápida. Essa é a função principal do engenheiro projetista de software. O resultado é um produto mais barato e eficaz quanto à sua funcionalidade.

4.2.6 - Exemplo de programa com loop: fazer um fluxograma e um programa-fonte em Assembly, para um membro da família de microcontroladores MCS-51, que calculem a quantidade de números menores que 38h de um buffer de memória que vai do endereço de memória 60h até o 7Ah. A quantidade de elementos menores que 38h deve ser armazenada no conteúdo da posição de memória cujo endereço é 7Bh.

Solução: existem muitas maneiras de resolver esse problema. Uma delas será apresentada a seguir.

Observe que se o problema for resolvido por meio de um programa seqüencial serão utilizadas muitas instruções de programa e, consequentemente, serão empregadas muitas posições de memória de programa, pois existem, no buffer de memória, $7Ah - 60h + 1 = 1Bh = 27_{10}$ elementos a serem analisados. Assim, é inviável a solução do problema por esse método para que seja preservada a relação custo/benefício. A solução que se emprega, nesse caso, é a que utiliza um programa com loop. Para solucionar o problema será utilizado o fluxograma com loop, sugerido na seção anterior.

Em primeiro lugar, é necessário entender perfeitamente o que está sendo solicitado. Um desenho em forma de rascunho (Figura 4.11) representa de maneira bastante simplificada o que o programa está pedindo.

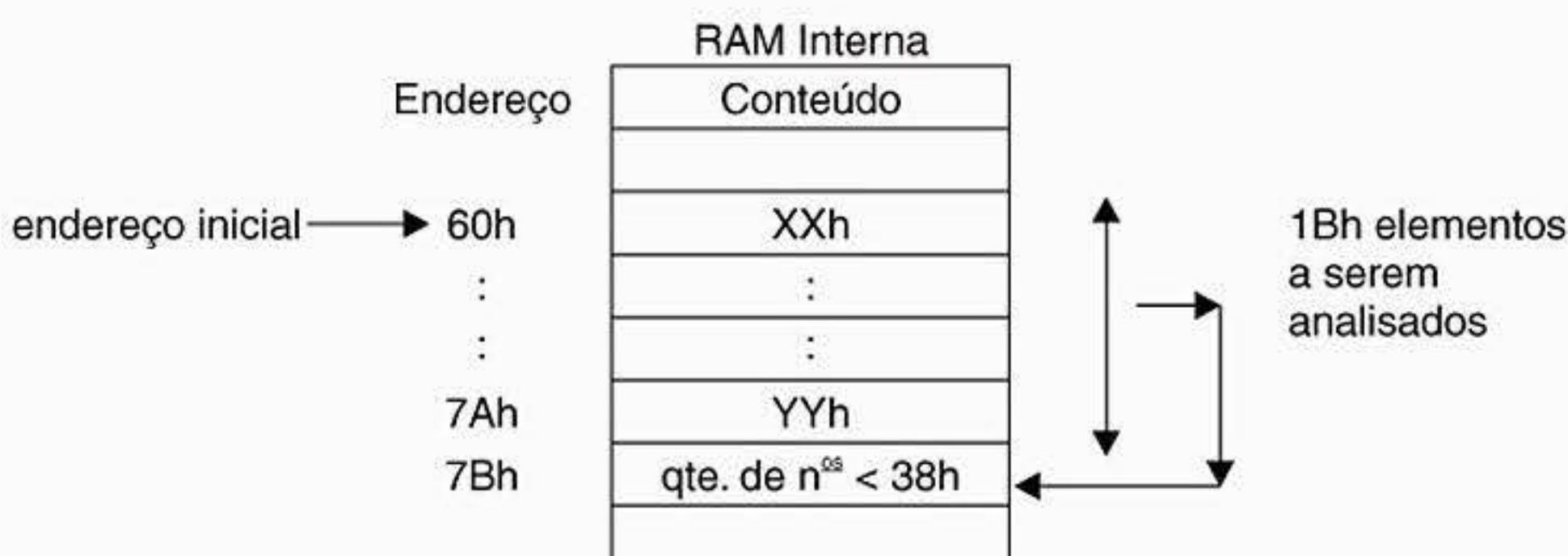


Figura 4.11 Desenho em forma de rascunho para ajudar no entendimento do que está sendo solicitado.

Seguindo o fluxograma com loop, necessitamos preencher cada um de seus blocos.

- ◆ *Nome do programa:* quantidade de números < 38h.
- ◆ *Condições iniciais:* definições das variáveis do programa, que são:
 - ◆ controles do buffer de memória:

- ◆ os endereços do buffer de memória serão controlados por meio do registrador R0. Ele é inicializado com o valor 60h, que é o endereço inicial do buffer de memória;
- ◆ a quantidade de elementos do buffer de memória será controlada por meio do registrador R1. Ele é inicializado com o valor 1Bh.
- ◆ definição de um contador, cujo valor inicial é zero. Toda vez que for analisado um elemento do buffer de memória e isso resultar em um número menor que 38h, esse contador deverá ser adicionado de uma unidade. Como contador, adota-se o conteúdo da posição de memória cujo endereço é 7Bh por conveniência, visto que foi solicitado que a quantidade de elementos menores que 38h fosse armazenada nessa posição.
- ◆ *Processamento, teste e tomada de decisão:* essa etapa consiste em definir a estratégia de solução do problema. Uma possível estratégia é analisar cada conteúdo de posição do buffer de memória e fazer uma comparação com o valor 38h. A operação de comparação é feita por meio de uma operação de subtração e pelo teste de *flag carry bit*. Como foi visto no Capítulo 3, após uma operação de subtração, o *flag carry bit* sinaliza se um número é maior ou igual (\geq) ou menor ($<$) que outro. Aproveitando essa informação, elabora-se a estratégia de solução para o problema. Assim, se após a operação de subtração do conteúdo da posição de memória de um elemento do buffer com o valor 38h resultar um valor 0 no conteúdo do *flag carry bit*, isso significa que o conteúdo da posição de memória é maior ou igual (\geq) a 38h e, dessa maneira, não se deve adicionar 1 ao contador de números menores que 38h (7Bh). Caso resulte o valor 1 no conteúdo do *flag carry bit*, isso significa que o conteúdo da posição de memória é menor ($<$) que 38h e, dessa maneira, é necessário adicionar 1 ao contador de números menores que 38h (7Bh).

A única operação de subtração que existe para essa família de microcontroladores e que é capaz de fazer a subtração de uma determinada constante é a instrução SUBB A,#*data* \Rightarrow (A) \leftarrow (A) - (C) - #38h, na qual o registrador acumulador A é um de seus argumentos. Assim, antes de executar uma operação de subtração, é necessário trazer um elemento do conteúdo do buffer de memória para o conteúdo do acumulador A. Em programas com loop, recomenda-se utilizar o endereçamento indireto ou indexado por registrador, ou seja, por meio da utilização da

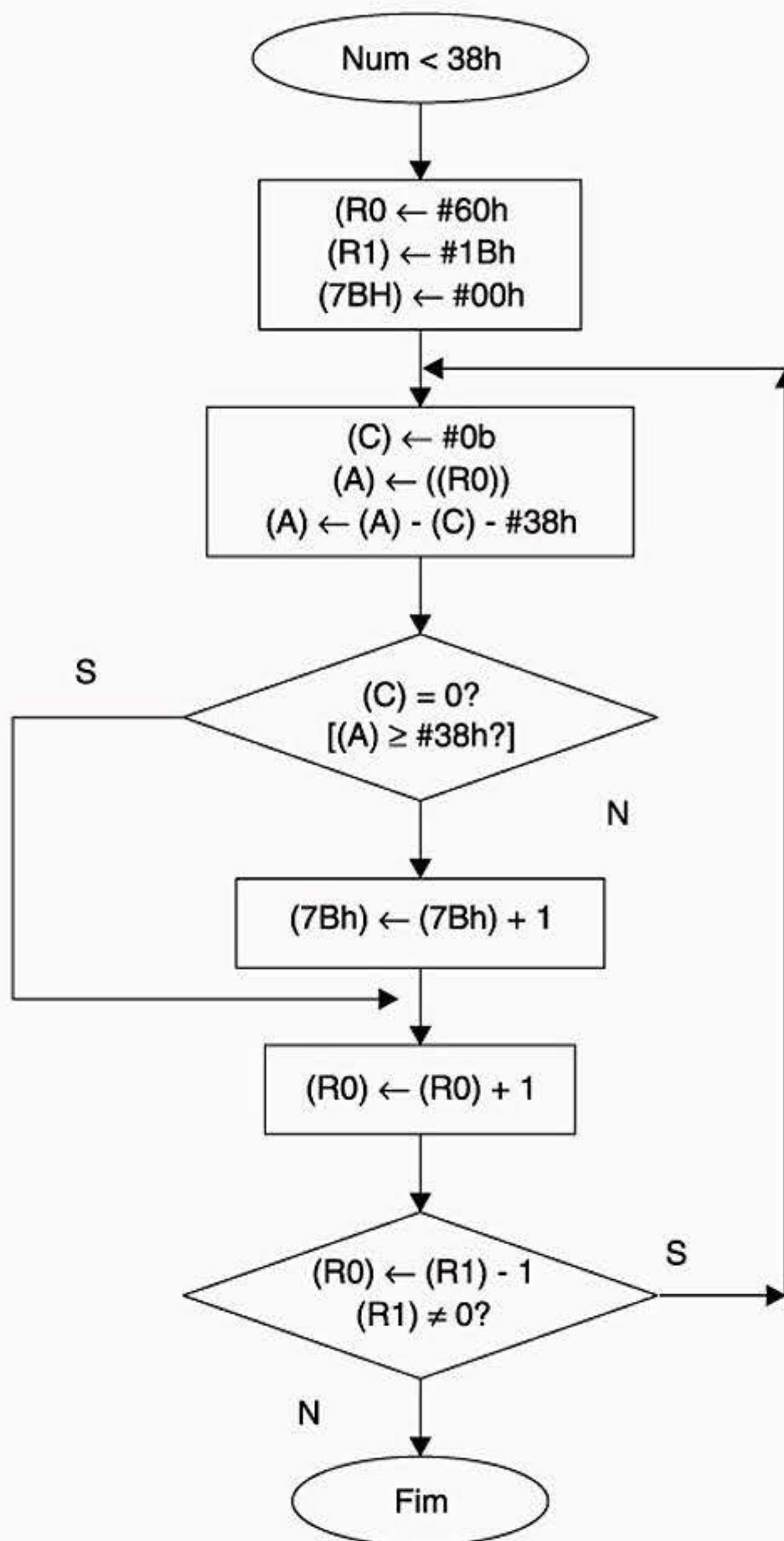
instrução $MOV A,@Ri \Rightarrow (A) \leftarrow ((Ri))$, antes do processamento da operação de subtração. Esse tipo de endereçamento facilita a obtenção do próximo elemento do buffer. Isso é feito utilizando uma instrução para adicionar uma unidade ao conteúdo do registrador Ri ($R0$ ou $R1$).

Repare, também, em um detalhe bastante importante: a instrução que faz a operação de subtração [$SUBB A,#data \Rightarrow (A) \leftarrow (A) - (C) - \#38h$] também utiliza o conteúdo do *flag carry bit* como argumento. Assim, é necessário 'zerar' esse *flag* (C) antes que a instrução de subtração seja executada, para que o resultado não seja influenciado pelo *flag*.

Controle do loop: é preciso ajustar as variáveis de controle do buffer de memória para que seja possível analisar os próximos elementos do buffer de memória. Como foi visto, a estratégia de solução (trazer o conteúdo do buffer de memória para o conteúdo do acumulador, 'zerar' o *flag carry bit*, fazer a operação de subtração, testar o conteúdo do *flag carry bit* e adicionar ou não 1 ao contador, dependendo se for menor que $38h$ ou não) deve ser feita para todos os conteúdos das posições de memória do buffer. Dessa maneira, para fazer o ajuste das variáveis que controlam o buffer de memória, tem-se:

- ◆ toda vez que for processado um elemento do buffer, deve-se adicionar uma unidade ao conteúdo do registrador que controla os endereçamentos do buffer ($R0$), para que o próximo elemento do buffer seja analisado no loop seguinte;
- ◆ deve-se verificar também se todos os elementos do buffer foram analisados. Isso é feito decrementando-se uma unidade ao conteúdo do registrador que contém a quantidade de elementos do buffer de memória ($R1$) e verificar se ele é igual a zero. Se não for igual a zero, deve-se desviar o fluxo de programa para a parte do processamento para que se repita a análise, agora, com esse novo elemento do buffer. Caso seja igual a zero, o processamento deve ser finalizado, pois isso sinaliza que todos os elementos do buffer foram analisados.
- ◆ *Definição da saída do programa:* já especificada armazenada no conteúdo da posição de memória, cujo endereço é $7Bh$. Esse bloco não é necessário.
- ◆ *Fim:* fim do programa.

A seguir é apresentado o fluxograma que calcula a quantidade de números menores que 38h e o programa-fonte em Assembly.



Início do programa que calcula a quantidade de elementos < #38h

	MOV R0,#60h	;Endereço inicial do buffer de memória
	MOV R1,#1Bh	;Quantidade de elementos do buffer de memória
	MOV 7Bh,#00	;'Zera' o contador que armazena a quantidade de números < #38h
ADR2:	CLR C	;Limpa o conteúdo do carry bit
	MOV A,@R0	;Armazena no (A) um elemento do buffer
	SUBB A,#38h	;Subtrai o (A) da constante #38h e define o flag (C)
	JNC ADR1	;Se (C) = 0 ($A \geq #38h$) => (PC) = ADR1 [não soma um no (7Ah)]
	INC 7Bh	;Se (C) = 1 ($A < #38h$), soma um no (7Bh)
ADR1:	INC R0	;Aponta para a próxima posição de memória a ser analisada
	DJNZ R1,ADR2	;Se a quantidade de elementos a serem analisados for $\neq 0$, então ;'salta' para ADR2
	END	;Termina o programa se (R1) = 0.

Exercícios resolvidos

1 – Faça um fluxograma e um programa-fonte, em Assembly, para um membro da família de microcontroladores MCS-51, que executem a operação de adição entre os conteúdos dos registradores R0 e R3 e o conteúdo da posição de memória cujo endereço é 55h. O resultado deve ser armazenado no conteúdo do registrador R6 do segundo banco de registradores.

Resposta: a solução é apresentada na Figura 4.13.

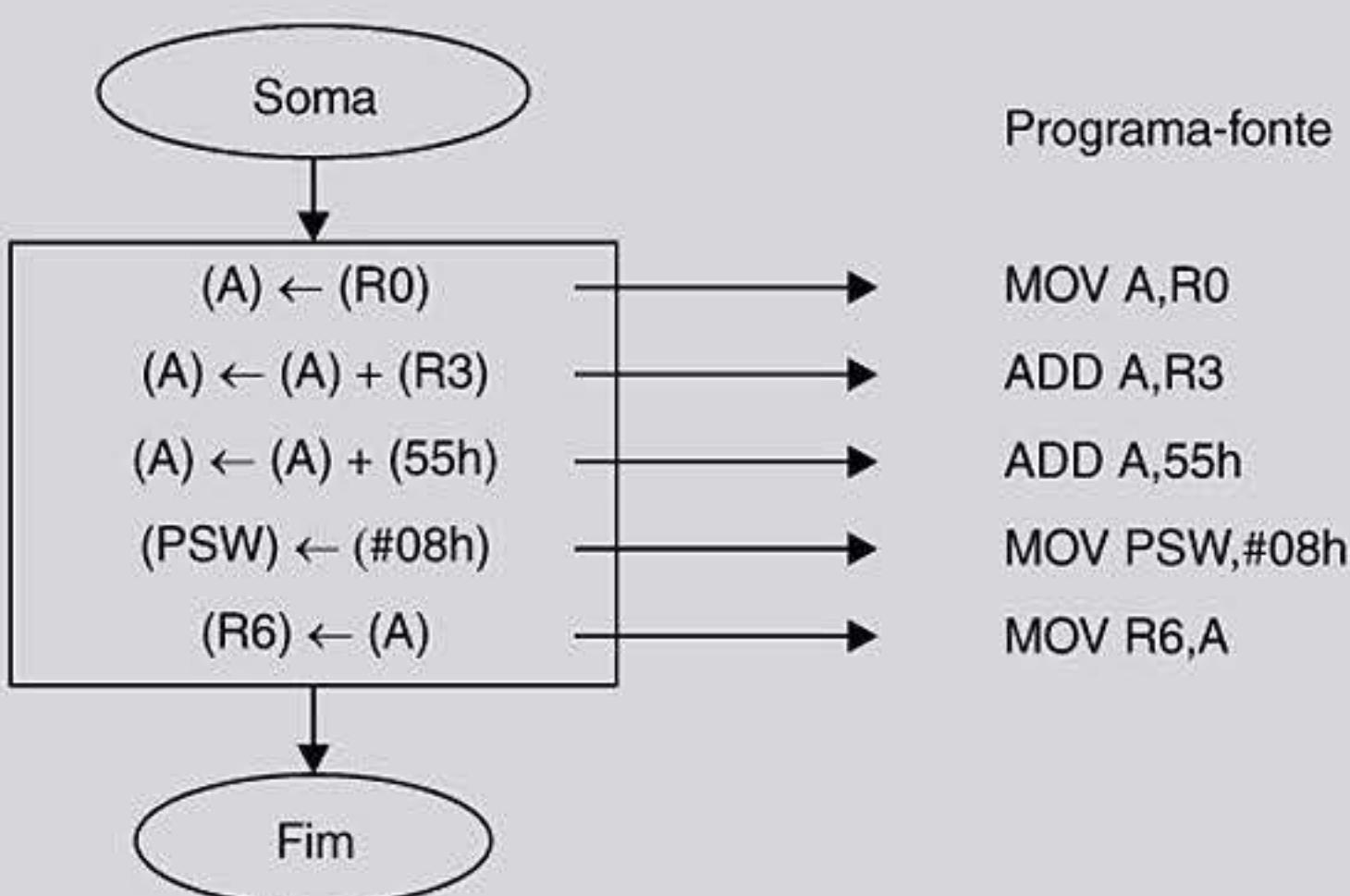


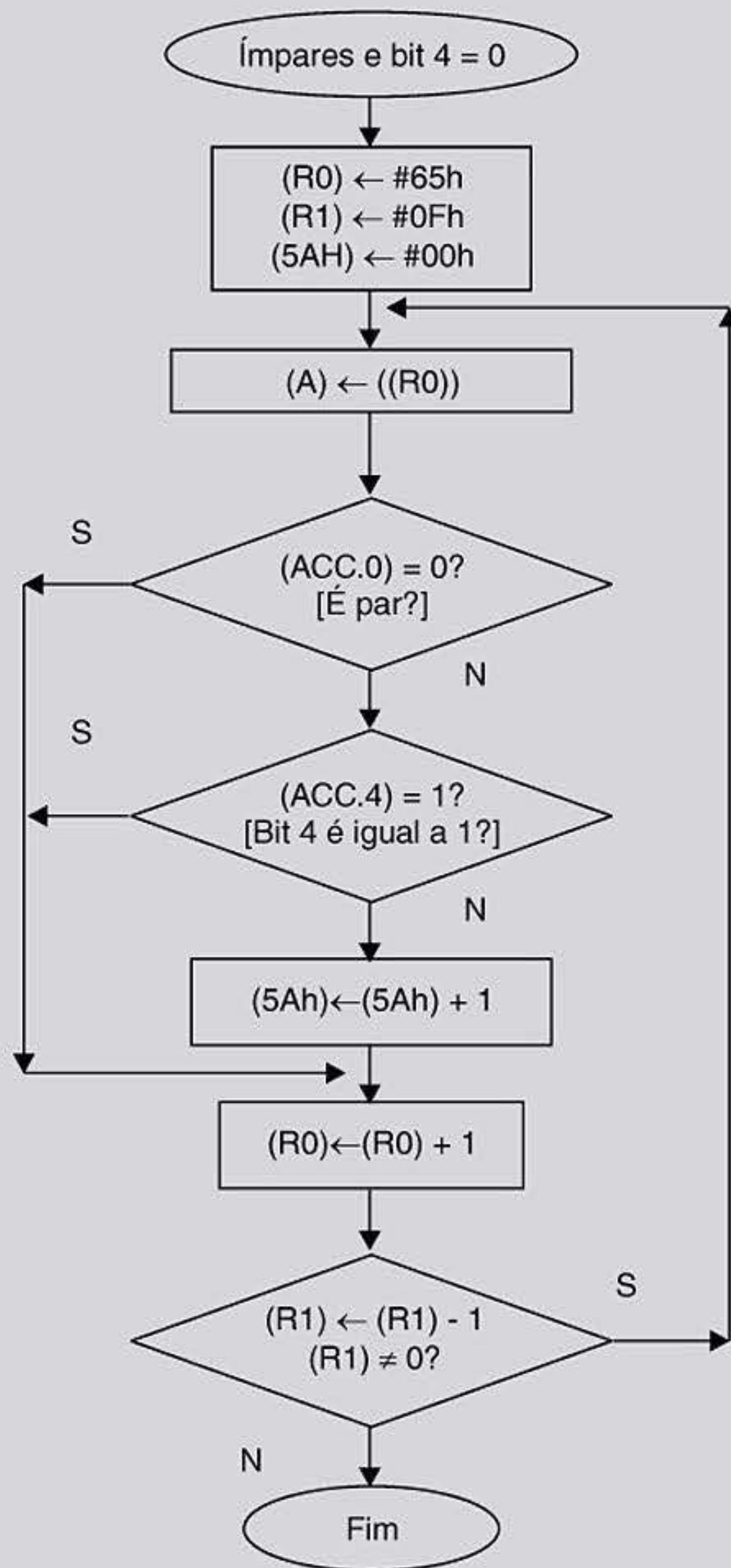
Figura 4.13 Solução do Exercício 1.

2 – Faça um fluxograma e um programa-fonte, em Assembly, para o microcontrolador 8051, que calculem a quantidade de números ímpares e apresentem o bit 4 igual a 0 de um buffer de memória

que vai do endereço 65h até o endereço 73h. O resultado deve ser armazenado no conteúdo da posição de memória cujo endereço é 5Ah.

Solução: Um número ímpar é aquele em que o bit menos significativo é igual a 1.

A solução é apresentada a seguir.



Início do programa que calcula a quantidade de elementos ímpares e apresenta o bit 4 igual a 0.

MOV	R0,#65h	;Endereço inicial do buffer
MOV	R1,#0Fh	;Quantidade de elementos do buffer
MOV	5Ah,#00	;‘Zera’ o conteúdo da posição de memória que armazena a ;quantidade de números ímpares e apresenta o bit 4 = 0
ADR2:	MOV A@R0	;Armazena no (A) um elemento do buffer
	JNB ACC.0,ADR1	;Se (ACC.0)=0 (par) \Rightarrow (PC)=ADR1 [não soma um ao (5Ah)]
	JB ACC.4,ADR1	;Se (ACC.4)=1 \Rightarrow (PC)=ADR1 [não soma um ao (5Ah)]
	INC 5Ah	;Adiciona uma unidade ao contador (5Ah)
ADR1:	INC R0	;Aponta para a próxima posição de memória do buffer
	DJNZ R1,ADR2	;Se a quantidade de elementos a serem analisados for $\neq 0$, ;então ‘salta’ para ADR2
	END	;Termina o programa, se (R1)=0

Exercícios e questões propostos

- 1 – Faça um fluxograma e um programa-fonte, em Assembly, para o 8031, que executem a operação de subtração entre 2 bytes localizados nos conteúdos das posições de memória cujos endereços são 5Fh e 7Ch. O resultado deve ser armazenado no conteúdo do registrador R5 do último banco de registradores.
- 2 – Modificar o programa do item 1 para executar uma operação lógica OR-EX entre tais conteúdos.
- 3 – Modificar o programa do item 1 para executar uma operação lógica AND entre tais conteúdos.
- 4 – Modificar o programa do item 1, considerando os conteúdos das posições de memória cujos endereços são 44h e 8Dh.
- 5 – Faça um fluxograma e um programa-fonte, em Assembly, para o microcontrolador 8051, que executem a operação de adição com *carry bit* entre 2 bytes. O primeiro byte está localizado no conteúdo do registrador R3 do terceiro banco de registradores, e o outro byte está localizado no conteúdo da posição de memória cujo endereço é 38h. O resultado deve ser armazenado no conteúdo do registrador R2 do segundo banco de registradores.
- 6 – Faça um fluxograma e um programa-fonte, em Assembly, para o microcontrolador 8051, que façam uma operação lógica OR-Exclusivo entre o conteúdo da posição de memória cujo endereço é 25h e o conteúdo do registrador R5 do penúltimo banco de registradores. O resultado obtido deve ser multiplicado por 4, adicionado ao valor 37h

e dividido por 8 e depois deve ser armazenado no conteúdo do registrador R1 do antepenúltimo banco de registradores.

- 7 - Considerando o programa que calculava a quantidade de números menores que 38h, modifique-o, da forma que o buffer de memória foi modificado. Agora, ele vai do endereço 3Fh até a posição de memória cujo endereço é 6Ch.
- 8 - Considerando o programa que calculava a quantidade de números menores que 38h, modifique-o de maneira que ele deva armazenar o resultado no conteúdo do registrador B.
- 9 - Considerando o programa que calculava a quantidade de números menores que 38h, modifique-o, de maneira que ele calcule a quantidade de números maiores que 6Ch.
- 10 - Considerando o programa que calculava a quantidade de números menores que 38h, modifique-o de maneira que ele calcule a quantidade de números ímpares.
- 11 - Faça um fluxograma e um programa-fonte, em Assembly, para o microcontrolador 8051, que calculem a quantidade de números pares de um buffer que vai do endereço 56h até 6Dh. O resultado deve ser armazenado no conteúdo da posição de memória cujo endereço é 5Ah.
- 12 - Faça um fluxograma e um programa-fonte, em Assembly, para o microcontrolador 8051, que calculem a quantidade de números positivos de um buffer que vai do endereço 36h ao 74h. O resultado deve ser armazenado no conteúdo da posição de memória cujo endereço é 1Bh.
- 13 - Faça um fluxograma e um programa-fonte, em Assembly, para o microcontrolador 8051, que calculem a quantidade de números negativos de um buffer que vai do endereço 59h até 71h. O resultado deve ser armazenado no conteúdo da posição de memória cujo endereço é 4Fh.
- 14 - Faça um fluxograma e um programa-fonte, em Assembly, para o microcontrolador 8031, que calcule a quantidade de números que apresentam paridade ímpar de um buffer de memória que vai do endereço 2Eh até 64h. O resultado deve ser armazenado no conteúdo da posição de memória cujo endereço é 12h.

- 15 - Faça um fluxograma e um programa-fonte, em Assembly, para o microcontrolador 8051, que calculem a quantidade de números que apresentam paridade par e sejam diferentes de 56h, de um buffer que vai do endereço 49h até 71h. O resultado deve ser armazenado no conteúdo da posição de memória cujo endereço é 79h.
- 16 - Faça um fluxograma e um programa-fonte, em Assembly, para o microcontrolador 8051, que calculem a quantidade de números diferentes de zero que apresentem o bit 5 igual a um e que sejam negativos, de um buffer de memória que vai do endereço 2Bh até 39h. O resultado deve ser armazenado no conteúdo da posição de memória cujo endereço é 6Fh.
- 17 - Resolva o Exercício 16 utilizando a instrução CJNE para fazer o controle do loop. Compare a solução obtida agora com a solução anterior do Exercício 16 e diga qual delas é a melhor quanto ao gasto de memória de programa e à velocidade de processamento.
- 18 - Faça um fluxograma e um programa-fonte, em Assembly, para o microcontrolador 8031, que transformem o buffer de memória do Exercício 17 em números negativos, somente para aqueles elementos do buffer que apresentem o bit 5 igual a 0 ou sejam negativos.
- 19 - Faça um fluxograma e um programa-fonte, em Assembly, para o microcontrolador 8051, que transformem o buffer de memória do Exercício 17 em números ímpares, somente para aqueles elementos que apresentem paridade ímpar, apresentem o bit 1 igual a 1 e sejam menores que 44h.
- 20 - Faça um fluxograma e um programa-fonte, em Assembly, para o microcontrolador 8031, que transformem o buffer de memória do Exercício 17 num buffer de memória em que os elementos ímpares e positivos devam ser substituídos pelos seus valores multiplicados por 32.
- 21 - Faça um fluxograma e um programa-fonte, em Assembly, para o microcontrolador 8051, que copiem o conteúdo do buffer de memória do Exercício 17 para um buffer de memória cujo endereço inicial seja 4Ah.

- 22 - Faça um fluxograma e um programa-fonte, em Assembly, para o microcontrolador 8031 que façam a operação lógica *OR-EX* dos elementos do buffer de memória do Exercício 17. O resultado deve ser armazenado no conteúdo do registrador R0 do primeiro banco de registradores.
- 23 - Faça a operação de subtração entre dois buffers de memória. O primeiro buffer de memória começa no endereço 13h e termina no endereço 2Ah. O endereço inicial do segundo buffer de memória é 34h. O resultado deve ser colocado em um buffer de memória cujo endereço inicial seja 56h.

PÁGINA EM BRANCO

5.1 Objetivos

- ❖ Definição e princípio de funcionamento de sub-rotina
- ❖ Definição de pilha (LIFO) e fila (FIFO)
- ❖ As instruções que trabalham com a pilha (PUSH, POP, ACALL, LCALL, RET e RETI)
- ❖ Estruturação da linguagem de programação Assembly
- ❖ Exemplos de programas estruturados com sub-rotinas, aplicados à família de microcontroladores MCS-51 da Intel

5.2 Introdução teórica

Este capítulo completa o estudo de programação iniciado no Capítulo 4.

O software implementado de maneira estruturada, por meio da utilização de sub-rotinas, apresenta as seguintes características:

- ◆ é extremamente compacto e flexível, obtendo-se alto desempenho e boa relação custo/benefício;
- ◆ é modular e, consequentemente, de fácil implementação, simulação, teste e manutenção;
- ◆ as sub-rotinas podem ser colocadas em bibliotecas e aproveitadas para a implementação de novos produtos que utilizam tecnologia similar, reduzindo, assim, o tempo de desenvolvimento.

Dessa forma, este capítulo é muito importante para o leitor, pois ensinará as técnicas de programação estruturada, utilizando-se a linguagem de programação Assembly, aplicadas à família de microcontroladores MCS-51 da

Intel para a implementação de projetos profissionais de software de alto desempenho e boa relação custo/benefício.

A seguir, são apresentadas algumas definições extremamente importantes para o desenvolvimento deste capítulo.

5.2.1 - Definição de sub-rotina: é um conjunto de instruções com características específicas. As sub-rotinas devem ser alocadas em uma área de memória de programa **separada** da área de memória do **programa principal**. Assim, sempre que se discutir sub-rotina, pode-se pensar em uma estrutura de programa que tem, pelo menos, duas áreas de memória. A primeira área de memória de programa contém as sub-rotinas e a segunda contém o chamado **programa principal**, que deve conter **instruções específicas** em conjunto, principalmente, com as **instruções de chamadas às sub-rotinas** que, no caso da família de microcontroladores MCS-51 da Intel são as instruções ACALL *endereço inicial da sub-rotina* e LCALL *endereço inicial da sub-rotina*.

Uma sub-rotina se origina sempre que ela for utilizada mais que uma vez ao longo de um programa. Para exemplificar, considere que é necessário resolver uma expressão matemática várias vezes em um programa. Essa função matemática é constituída por diversos códigos de máquina (*mнемônicos*). Repare que cada vez que era necessário utilizá-la, era preciso reescrevê-la, o que aumentava consideravelmente o tamanho do programa. Quanto mais essa sub-rotina era utilizada, maior era o programa e, consequentemente, uma grande quantidade de memória de programa era necessária. Dessa maneira, a relação custo/benefício do projeto de software acabava ficando bastante alta em termos de hardware, como mostramos a seguir.

Programa principal:

```

    instrução 1
    :
    instrução N
    rotina que calcula uma expressão matemática (tamanho de 60 bytes, por exemplo)
    instrução N+1
    :
    instrução M
    rotina que calcula uma expressão matemática (tamanho de 60 bytes, por exemplo)
    instrução M
    :
    instrução O
    rotina que calcula uma expressão matemática (tamanho de 60 bytes, por exemplo)
    etc.
    END
  
```

A seguir o mesmo programa utilizando agora a sub-rotina:

Sub-rotina:

End1: sub-rotina que calcula uma expressão matemática (tamanho de 60 bytes, por exemplo)

■

RET

Programa Principal:

Prog P: instrução 1

■

instrução N

Chamada à sub-rotina que calcula uma expressão matemática (ACALL end1/LCALL end1 - 2 ou 3 bytes)

instrução N+1

■

instrução M

Chamada à sub-rotina que calcula uma expressão matemática (ACALL end1/LCALL end1 - 2 ou 3 bytes)

instrução M

■

instrução O

Chamada à sub-rotina que calcula uma expressão matemática (ACALL_end1/LCALL_end1 - 2 ou 3 bytes)

ato

END

Observe que um programa implementado com sub-rotinas utiliza bem menos posições de memória de programa (exemplo anterior, aproximadamente $60 + 10 = 70$ bytes) do que um programa que não é implementado com sub-rotinas (exemplo da página 102, aproximadamente $3 \times 60 + 10 = 190$ bytes).

A estrutura de projeto de um programa (software) utilizando várias subrotinas é apresentada a seguir.

Área de
memória de
programa 1

: Área de memória de programa das 'sub-rotinas'

: Sub-rotina 1:

instru

; Sub-rotina 2:
endereço inicial da sub-rotina 2 : instrução 1

Instrução 2

100

<pre>; Sub-rotina M: endereço inicial da sub-rotina M : instrução 1 instrução 2 : instrução N RET</pre>	<pre>Área de memória de programa 2 : Área de memória de programa do 'programa principal' endereço inicial do programa principal : instrução 1 instrução 2 : ;chamada à 'sub-rotina 1' ACALL/LCALL endereço inicial da sub-rotina 1 ;chamada à 'sub-rotina 2' ACALL/LCALL endereço inicial da sub-rotina 2 : ;chamada à 'sub-rotina M' ACALL/LCALL endereço inicial da sub-rotina M ; instrução K END</pre>
---	--

Toda sub-rotina é caracterizada por seu endereço inicial e pelo fato de sua última instrução ser sempre a instrução RET (retorno de sub-rotina).

A instrução RET (RETurn, em inglês = RETorno de sub-rotina) deve sempre ser utilizada em conjunto com as instruções de chamada a sub-rotinas ACALL $addr_{11}$ e LCALL $addr_{16}$.

Analogamente, como uma interrupção funciona de maneira similar a uma sub-rotina, a instrução RETI (RETurn Interruption, em inglês = RETorno de uma sub-rotina de Interrupção), ela deve ser sempre utilizada para finalizar uma sub-rotina de atendimento à fonte de interrupção das cinco fontes de interrupções existentes nos microcontroladores da família MCS-51 - duas entradas de interrupções externas (IE0 e IE1), duas entradas de interrupções dos timers/contadores (TF0 e TF1) e uma entrada de interrupção do canal de comunicação serial, referente à interrupção de recepção e transmissão de informações (RI+TI).

Uma fonte de interrupção de hardware é sempre ativada por um sinal elétrico.

Uma sub-rotina elaborada para atender a uma fonte de interrupção é chamada de *sub-rotina de atendimento à fonte de interrupção*.

Todo programa principal também é caracterizado pelo seu endereço inicial e por apresentar, na sua última linha de programa, a diretriz END do Assembly para indicar o fim do programa.

Vantagens da utilização das sub-rotinas:

- ◆ reduz a quantidade de memória de programa (ROM ou EPROM), diminuindo também o custo do projeto de hardware;
- ◆ apresenta a característica de ser modular. Cada módulo corresponde a uma sub-rotina que deve ser independente. Isso facilita a implementação, a simulação, o teste físico, a manutenção e a modificação do projeto de software;
- ◆ o projeto do programa final fica fácil de ser entendido;
- ◆ os módulos podem fazer parte de uma biblioteca de sub-rotinas e podem ser utilizados na implementação de novos projetos que empregam a mesma tecnologia de microcontroladores.

5.2.2 - Princípio de funcionamento de uma sub-rotina: considere a estrutura de um projeto de software, dada no final da página 103 e começo da 104.

Para executar esse programa é necessário inicializar o conteúdo do registrador *Program Counter* (PC) com o valor do *endereço inicial do programa principal*. Isso significa dizer que a próxima instrução a ser executada será a *instrução 1* da área de memória de programa 2, até executar a instrução de chamada à sub-rotina 1, ACALL/LCALL *endereço inicial da sub-rotina 1*. Ao executar essa instrução, o programa processará a *sub-rotina 1* a partir do *endereço inicial da sub-rotina 1*, localizada na área de memória de programa 1. Assim, serão executadas as instruções contidas na sub-rotina 1, isto é, as instruções 1, 2 até a N e finalmente será executada a instrução RET da sub-rotina 1. Essa instrução é responsável pelo *retorno* do processamento para o programa principal. O retorno ao programa principal é feito por meio da execução da instrução contida no endereço de memória imediatamente após a instrução ACALL ou LCALL, que foi executada. No exemplo, esse é o endereço da instrução de chamada à sub-rotina 2, que aponta para a instrução ACALL/LCALL *endereço inicial da sub-rotina 2*. Assim, a próxima instrução a ser executada, será a instrução ACALL/LCALL *endereço inicial da sub-rotina 2*. Da mesma maneira, o programa executará a *sub-rotina 2*, a partir do *endereço inicial da sub-rotina 2*, localizada na área de memória de programa 1. Em seguida, serão executadas as instruções 1, 2 até a N e finalmente será executada a instrução RET da sub-rotina 2, que é responsável pelo *retorno* ao programa principal por meio da execução da instrução contida no endereço de memória imediatamente após a instrução ACALL ou LCALL, que foi executada. Tal endereço é o da instrução de chamada à sub-rotina 3, que aponta para a instrução ACALL/LCALL *endereço inicial da sub-rotina 3* e assim por diante.

Mas como uma sub-rotina é executada fisicamente pelo microprocessador?

Em primeiro lugar, é necessário definir o que é *Pilha* (*LIFO – Last-In, First-Out*). O conceito de pilha vem da definição de empilhamento físico de objetos, ou seja, ao empilhar objetos, o primeiro objeto da pilha será o último a sair, para que a mesma não seja destruída. Por exemplo, uma pilha de latas de leite em pó em uma prateleira de supermercado. Deve-se, em primeiro lugar, retirar a última lata que foi colocada na pilha, para que a pilha não seja destruída.

A pilha deve ser endereçada em uma área de memória RAM disponível no hardware do sistema microprocessado ou microcontrolado. Essa área pode ser da memória RAM interna ou da RAM externa.

Nessa área de memória RAM, serão armazenadas informações (bytes ou endereços), utilizando instruções que empregam endereçamento indireto indexado ao conteúdo do registrador denominado *Stack Pointer* (Ponteiro de Pilha – SP).

Após o sinal de *reset*, o conteúdo do registrador *Stack Pointer* (SP) é inicializado com o valor 07h (definido pelo fabricante).

A pilha sempre aumenta de tamanho no sentido ascendente dos endereçamentos de memória RAM e diminui de tamanho no sentido descendente dos endereços de memória.

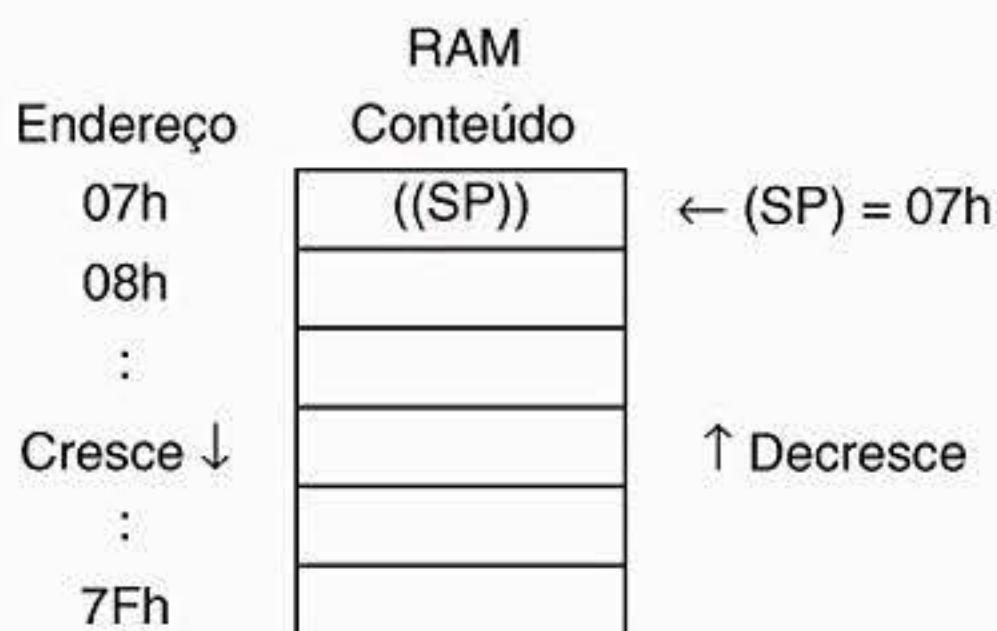


Figura 5.4 Representação esquemática da pilha (LIFO) após um sinal de *reset*.

As instruções pertencentes à família de microcontroladores MCS-51 que utilizam a pilha são:

PUSH direct, POP direct, ACALL address, LCALL address, RET e RETI.

Define-se Fila (*FIFO – First-In, First-Out*) como um buffer de memória com o princípio de escrita e leitura de bytes com base na idéia de que o primeiro byte que entra será o primeiro a sair. O conceito de fila pode ser associado a uma fila física de pessoas, ou seja, ao se formar uma fila, a primeira pessoa que

entra nessa fila será a primeira a sair, para que a mesma não seja destruída. Por exemplo, uma fila de aposentados do INSS em dia de pagamento.

Esse tipo de armazenamento geralmente é utilizado em comunicação serial, em que os bytes estão armazenados em um determinado buffer de memória com o conceito de fila, ou seja, os primeiros bytes armazenados serão os primeiros a serem lidos e transmitidos.

A seguir, serão apresentadas as instruções PUSH *direct* e POP *direct*.

PUSH direct: escreve/armazena o conteúdo da posição de memória cujo endereço é *direct* na pilha. Isso é feito pelo microprocessador, da seguinte maneira: o conteúdo do *Stack Pointer* é adicionado de uma unidade $[(SP) \leftarrow (SP)+1]$ e depois o conteúdo da posição de memória, cujo endereço é *direct*, é armazenado no conteúdo da posição de memória RAM da pilha, cujo endereço é dado pelo conteúdo do registrador *Stack Pointer* $[(SP) \leftarrow (\text{direct})]$.

POP direct: lê/restaura um byte da pilha. Isso é feito pelo microprocessador da seguinte maneira: o conteúdo da posição de memória, cujo endereço é *direct*, será armazenado com o conteúdo da posição de memória, cujo endereço é dado pelo conteúdo do *Stack Pointer* $[(\text{direct}) \leftarrow ((SP))]$ e depois, o conteúdo do *Stack Pointer* será decrementado de uma unidade $[(SP) \leftarrow (SP) - 1]$.

A Tabela 5.1 descreve as instruções PUSH *direct* e POP *direct*.

Tabela 5.1 Descrição das instruções PUSH *direct* e POP *direct*.

Instrução	Bytes	Ciclos	Codificação	Operação
PUSH direct	2	2	1100 0000 direct address	$(PC) \leftarrow (PC) + 2$ $(SP) \leftarrow (SP) + 1$ $((SP)) \leftarrow (\text{direct})$
POP direct	2	2	1101 0000 direct address	$(PC) \leftarrow (PC) + 2$ $(\text{direct}) \leftarrow ((SP))$ $(SP) \leftarrow (SP) - 1$

Como exemplo da função de armazenamento de bytes na pilha, considere que o conteúdo do *Stack Pointer* (SP) foi inicializado com o valor 50h e que se deseja armazenar o conteúdo do registrador R0 na pilha, que vale 23h. Inicialmente, como a pilha foi definida a partir do endereço 50h da memória RAM interna do microcontrolador, pois $(SP) = 50h$, ela começa a partir do endereço 50h da memória RAM como é mostrado a seguir.

Stack Pointer (SP)	Endereço da memória RAM interna	Conteúdo da memória RAM interna
50h →	50h	XXh (dado qualquer)
	51h	YYh (dado qualquer)
	52h	ZZh (dado qualquer)

A instrução para armazenar o conteúdo do registrador R1 na pilha é PUSH 01h (endereçamento direto, considerando que R1 pertence ao primeiro banco de registradores). Assim, após essa instrução ser processada, obtém-se:

$$(SP) \leftarrow (SP) + 1 \leftarrow 50h + 1 = 51h;$$

$$((SP)) \leftarrow (\text{direct}) \Leftrightarrow ((SP)) = (51h) \leftarrow (01h) = (R1) = 23h.$$

A seguir, são mostradas as condições finais do conteúdo do *Stack Pointer* (SP) e do conteúdo da pilha.

Stack Pointer (SP)	Endereço da memória RAM interna	Conteúdo da memória RAM interna
	50h	XXh (dado qualquer)
51h →	51h	23h
	52h	ZZh (dado qualquer)

Para ler (recuperar) o conteúdo do registrador R1 da pilha, a instrução a ser utilizada é POP 01h. Assim, executando-se essa instrução, tem-se:

$$(\text{direct}) \leftarrow ((SP)) \Leftrightarrow (01h) = (R0) \leftarrow ((SP)) = (51h) = 23h;$$

$$(SP) \leftarrow (SP) - 1 \leftarrow 51h - 1 = 50h.$$

Stack Pointer (SP)	Endereço da memória RAM interna	Conteúdo da memória RAM interna
50h →	50h	XXh (dado qualquer)
	51h	23h
	52h	ZZh (dado qualquer)

Considerações gerais sobre esse exemplo:

- ◆ Quando se armazena bytes na pilha e depois os mesmos são lidos, o conteúdo do *Stack Pointer* volta à sua condição inicial.
- ◆ Os bytes que foram armazenados na pilha continuam em suas posições de memória até que a pilha seja novamente utilizada.
- ◆ A pilha aumenta de tamanho no sentido ascendente do endereçamento da memória RAM.

- ◆ A pilha diminui de tamanho no sentido descendente do endereçamento da memória RAM.
- ◆ O conteúdo da posição de memória inicial da pilha nunca é utilizado.

A seguir, serão apresentadas as instruções de chamadas a sub-rotinas ACALL $addr_{11}$ e LCALL $addr_{16}$ e de retorno às sub-rotinas RET e RETI:

- ◆ ACALL $addr_{11}$:
 - ◆ *Função: Absolute Call* (chamada à sub-rotina absoluta = memória de programa).
 - ◆ *Descrição:* ACALL chama incondicionalmente uma sub-rotina localizada no endereço indicado por $addr_{11}$. Inicialmente, a instrução ACALL $addr_{11}$ incrementa o conteúdo do registrador *Program Counter* (PC) em duas unidades, pois a instrução ACALL é formada por 2 bytes. Isso é feito para calcular o endereço da instrução seguinte, que deverá ser executada logo após a sub-rotina ser processada por inteiro. Dessa maneira, o conteúdo do registrador *Program Counter* (PC) contém o endereço de retorno da sub-rotina. Depois disso, a instrução ACALL $addr_{11}$ armazena o conteúdo do registrador PC (16 bits) dentro da pilha por endereçamento indireto ou indexado, utilizando o conteúdo do registrador *Stack Pointer* (SP). Isso é feito armazenando-se, em primeiro lugar, o byte menos significativo do conteúdo do registrador *Program Counter* (PC), chamado PCL, em que L significa *Low* (menos significativo), na pilha, na posição de memória cujo endereço é dado pelo conteúdo do *Stack Pointer*. Depois, o conteúdo do registrador *Stack Pointer* é adicionado de uma unidade. E finalmente, o byte mais significativo do conteúdo do registrador *Program Counter* (PC), chamado PCH, em que H significa *high* (mais significativo), é armazenado na pilha, na posição de memória cujo endereço é dado pelo conteúdo do *Stack Pointer*.

O endereço de destino é obtido ao se concatenar (juntar/compor) os 5 bits mais significativos do conteúdo do *Program Counter* (PC), que estão junto com os bits de 7 - 5 do *opcode* (código da instrução) e o segundo byte da instrução. A sub-rotina deverá estar localizada a uma distância máxima de 2 Kbytes dentro da memória de programa. Nenhum *flag* do conteúdo do registrador PSW é afetado por essa instrução.

A Tabela 5.2 descreve as instruções ACALL $addr_{11}$ e LCALL $addr_{16}$.

Tabela 5.2 Descrição das instruções ACALL $addr_{11}$ e LCALL $addr_{16}$.

Instrução	Bytes	Ciclos	Codificação	Operação
ACALL $addr_{11}$	2	2	a10 a9 a8 1 0001 a7.....a0	$(PC) \leftarrow (PC) + 2$ $(SP) \leftarrow (SP) + 1$ $((SP)) \leftarrow (PC_{7..0})$ $(SP) \leftarrow (SP) + 1$ $((SP)) \leftarrow (PC_{15..8})$ $(PC_{10..0}) \leftarrow addr_{11}$
LCALL $addr_{16}$	3	2	0001 0010 a15....a8 a7a0	$(PC) \leftarrow (PC) + 3$ $(SP) \leftarrow (SP) + 1$ $((SP)) \leftarrow (PC_{7..0})$ $(SP) \leftarrow (SP) + 1$ $((SP)) \leftarrow (PC_{15..8})$ $(PC_{15..0}) \leftarrow addr_{16}$

Exemplo: Inicialmente, considere que o conteúdo do registrador *Stack Pointer* (SP) vale 07h e que a instrução ACALL *sub rtn* está armazenada no conteúdo da posição de memória ROM, cujo endereço é 0123h, e o nome *sub rtn* corresponde ao endereço de memória ROM 0345h, como é indicado a seguir.

Endereço de memória	Mnemônico
0123h	ACALL <i>sub rtn</i> (0345h)

Stack Pointer (SP)	Endereço da memória RAM interna	Conteúdo da memória RAM interna
07h →	07h	XXh (dado qualquer)
	08h	YYh (dado qualquer)
	09h	ZZh (dado qualquer)

Depois de executar a instrução ACALL *sub rtn*, o conteúdo do registrador *Stack Pointer* (SP) será igual a 09h, e os conteúdos das posições de memória 08h e 09h conterão os valores 25h e 01h, respectivamente; o conteúdo do registrador *Program Counter* (PC) será igual a 0345H, como é mostrado a seguir.

Endereço de memória	Mnemônico			
0123h	ACALL <i>sub rtn</i> ⇒			
	ACALL 0345h ⇒	$(PC) \leftarrow (PC) + 2$	= 0123h+2	= 0125h (endereço de retorno)
		$(SP) \leftarrow (SP) + 1$	= 07h+1	= 08h
		$((SP)) \leftarrow (PC_{7..0})$	<=> (08h)	= 25h
		$(SP) \leftarrow (SP) + 1$	= 08h+1	= 09h
		$((SP)) \leftarrow (PC_{15..8})$	<=> (09h)	= 01h
		$(PC_{10..0}) \leftarrow addr_{11}$	<=> (PC _{10..0})	= 0345h

Stack Pointer (SP)	Endereço da memória RAM interna	Conteúdo da memória RAM interna
	07h	XXh (dado qualquer)
	08h	25h
09h →	09h	01h

◆ *LCALL addr₁₆*:

- ◆ *Função:* Long Call (chamada à sub-rotina longo).
- ◆ *Descrição:* LCALL chama incondicionalmente uma sub-rotina localizada no endereço indicado por *addr₁₆*. Inicialmente, a instrução LCALL *addr₁₆* incrementa o conteúdo do registrador *Program Counter* (PC) em três unidades, pois a instrução LCALL é constituída de 3 bytes. Isso é feito para calcular o endereço da instrução seguinte, que deve ser executada logo após a sub-rotina ser processada. dessa maneira, o conteúdo do registrador *Program Counter* (PC) contém o endereço de retorno da sub-rotina. Depois disso, a instrução ACALL *addr₁₆* armazena o conteúdo do registrador PC (16 bits) dentro da pilha por endereçamento indireto ou indexado, utilizando o conteúdo do registrador *Stack Pointer* (SP). Isso é feito armazenando-se, em primeiro lugar, o byte menos significativo do conteúdo do registrador *Program Counter* (PCL) na pilha, na posição de memória, cujo endereço é dado pelo conteúdo do *Stack Pointer*. Depois, ao conteúdo do registrador *Stack Pointer* é adicionada uma unidade. E finalmente, o byte mais significativo do conteúdo do registrador *Program Counter* (PCH) é armazenado na pilha, na posição de memória cujo endereço é dado pelo conteúdo do *Stack Pointer* (SP). Depois disso, o conteúdo do registrador de função especial *Program Counter*, (PC), é inicializado com o endereço inicial da sub-rotina. Isso faz com que a próxima instrução a ser executada seja aquela que pertence à sub-rotina, mudando o fluxo de processamento do programa principal para a sub-rotina.

A sub-rotina deve estar localizada a uma distância máxima de 64 Kbytes dentro da memória de programa. Nenhum *flag* do conteúdo do registrador *Program Status Word* (PSW) é afetado por essa instrução.

Exemplo: Inicialmente, considere que o conteúdo do registrador *Stack Pointer* (SP) seja igual a 07h. A instrução LCALL *subrtn* está armazenada no conteúdo da posição de memória, cujo endereço é 0123h, e o nome *subrtn* é o endereço de memória 1234h.

<i>Endereço de memória</i>	<i>Mnemônico</i>
0123h	LCALL <i>sub rtn</i> (1234h)

Stack Pointer (SP)	Endereço da memória RAM interna	Conteúdo da memória RAM interna
07h →	07h	XXh (dado qualquer)
	08h	YYh (dado qualquer)
	09h	ZZh (dado qualquer)

Depois de executar a instrução LCALL *sub rtn* o conteúdo do registrador *Stack Pointer* (SP) será igual a 09h, e os conteúdos das posições de memória 08h e 09h conterão os valores 26H e 01H, respectivamente; o conteúdo do registrador *Program Counter* (PC) será igual a 1234h, como é mostrado a seguir.

<i>Endereço de memória</i>	<i>Mnemônico</i>			
0123h	LCALL <i>sub rtn</i> ⇒			
0123h	LCALL 1234h ⇒	(PC) ← (PC)+3 = 0123h+3 = 0126h (endereço de retorno)		
		(SP) ← (SP)+1 = 07h+1 = 08h		
		((SP)) ← (PC _{7,0}) <=> (08h) = 26h		
		(SP) ← (SP)+1 = 08h+1 = 09h		
		((SP)) ← (PC _{15,8}) <=> (09h) = 01h		
		(PC _{15,0}) ← addr ₁₆ <=> (PC _{15,0}) = 1234h		

Stack Pointer (SP)	Endereço da memória RAM interna	Conteúdo da memória RAM interna
	07h	XXh (dado qualquer)
	08h	26h
09h →	09h	01h

◆ RET:

- ◆ **Função:** retorno de sub-rotina.
- ◆ **Descrição:** essa instrução é utilizada principalmente em conjunto com as instruções de chamada às sub-rotinas ACALL $addr_{11}$ e LCALL $addr_{16}$.

A instrução RET é responsável por ler (recuperar) o endereço de retorno da sub-rotina para o programa principal, que foi armazenando pelas instruções de chamadas a sub-rotinas ACALL $addr_{11}$ e LCALL $addr_{16}$, da pilha. Assim, a instrução RET lê 2 bytes sucessivos da pilha, utilizando o endereçamento indireto ou indexado por meio do conteúdo do registrador *Stack Pointer* (SP) e armazenando-os no conteúdo do registrador *Program Counter* (PC) na sua parte mais significativa (PCH) e na sua parte menos significativa (PCL).

Essa instrução é executada pelo microprocessador por meio da operação da leitura do conteúdo da posição de memória, cujo endereço é dado pelo conteúdo do *Stack Pointer* (SP), e o armazena na parte mais significativa do conteúdo do registrador *Program Counter* (PCH). Depois, o conteúdo do registrador *Stack Pointer* (SP) é diminuído em uma unidade. Novamente, o microprocessador faz uma operação de leitura do conteúdo da posição de memória, cujo endereço é dado pelo conteúdo do registrador *Stack Pointer* (SP), e o armazena na parte menos significativa do conteúdo do registrador *Program Counter* (PCL). Depois, mais uma vez, o conteúdo do registrador *Stack Pointer* (SP) é diminuído em uma unidade. Assim, a próxima instrução a ser executada será aquela contida no conteúdo da posição de memória, cujo endereço é aquele imediatamente posterior à instrução ACALL ou LCALL que chamou tal sub-rotina.

Nenhum *flag* do conteúdo do registrador PSW é afetado por essa instrução.

A Tabela 5.3 descreve as instruções RET e RETI.

Tabela 5.3 Descrição das instruções RET e RETI.

Instrução	Bytes	Ciclos	Codificação	Operação
RET	1	2	0010 0010	$(PC_{15:8}) \leftarrow ((SP))$ $(SP) \leftarrow (SP) - 1$ $(PC_{7:0}) \leftarrow ((SP))$ $(SP) \leftarrow (SP) - 1$
RETI	1	2	0011 0010	$(PC_{15:8}) \leftarrow ((SP))$ $(SP) \leftarrow (SP) - 1$ $(PC_{7:0}) \leftarrow ((SP))$ $(SP) \leftarrow (SP) - 1$

Exemplo: Inicialmente, considere que o conteúdo do registrador *Stack Pointer* (SP) seja igual a 0Bh e que os conteúdos das posições de memória, cujos endereços da RAM interna são 0Ah e 0Bh, tenha os valores 23h e 01h, respectivamente, como é mostrado a seguir.

Stack Pointer (SP)	Endereço da memória RAM interna	Conteúdo da memória RAM interna
	09h	XXh (dado qualquer)
	0Ah	23h
0Bh →	0Bh	01h

Depois de executar a instrução RET, o conteúdo do registrador *Stack Pointer* (SP) será igual a 09h, e o conteúdo do registrador *Program Counter* (PC) será igual a 0123h, como é mostrado a seguir.

$$\begin{array}{lll} \text{RET} \Rightarrow & (\text{PC}_{15..8}) \leftarrow ((\text{SP})) & \Leftrightarrow (\text{PCH}) \leftarrow (0\text{Bh}) = 01\text{h} \\ & (\text{SP}) \leftarrow (\text{SP}) - 1 & = 0\text{Bh} - 1 = 0\text{Ah} \\ & (\text{PC}_{7..0}) \leftarrow ((\text{SP})) & \Leftrightarrow (\text{PCL}) \leftarrow (0\text{Ah}) = 23\text{h} \\ & (\text{SP}) \leftarrow (\text{SP}) - 1 & = 0\text{Ah} - 1 = 09\text{h} \end{array}$$

Stack Pointer (SP)	Endereço da memória RAM interna	Conteúdo da memória RAM interna
09h →	09h	XXh (dado qualquer)
	0Ah	23h
	0Bh	01h

Assim, a próxima instrução a ser executada será aquela localizada no conteúdo da posição de memória, cujo endereço é dado pelo conteúdo do registrador *Program Counter* (PC), que vale 0123h.

◆ *RETI:*

- ◆ *Função:* retorno de sub-rotina de interrupção.
- ◆ *Descrição:* essa instrução é utilizada principalmente em conjunto com a instrução de chamada à sub-rotina de interrupção LCALL *vector address* (0003h-IE0, 000Bh-TF0, 0013h-IE1, 001Bh-TF1, 0023h-RI+TI e 002Bh-TF2+EXF2-8032/52), que ocorre quando uma interrupção é gerada por um sinal elétrico externo (IE0 ou IE1) ou pelos *timers/contadores* (TF0 ou TF1) ou pela interrupção do canal de comunicação serial (RI ou TI), por meio da recepção ou transmissão de informação ou, ainda, pelo *timer/contador* 2 para o 8051/32. Quando ocorre um desses sinais de hardware, o microprocessador finaliza a instrução que está sendo executada, geralmente localizada no programa principal, e gera uma instrução de LCALL *vector address* correspondente à interrupção ocorrida (externa, pelos *timers/contadores* ou por comunicação serial). Assim, a instrução de LCALL *vector address* armazena na pilha o endereço de retorno para o programa principal. Com isso, a instrução RETI fica responsável por recuperar o endereço de retorno para o programa principal que foi armazenado na pilha pela instrução de chamada a sub-rotinas LCALL *vector address*. A instrução RETI lê 2 bytes sucessivos da pilha, utilizando o endereçamento indireto ou indexado por meio do conteúdo do registrador *Stack Pointer* (SP) e os armazena no conteúdo do

registrador *Program Counter* (PC) na sua parte mais significativa (PCH) e na sua parte menos significativa (PCL), respectivamente. Essa instrução é executada pelo microprocessador por meio da operação de leitura do conteúdo da posição de memória, cujo endereço é dado pelo conteúdo do *Stack Pointer* (SP), e o armazena na parte mais significativa do conteúdo do registrador *Program Counter* (PCH). Depois, o conteúdo do registrador *Stack Pointer* (SP) é diminuído em uma unidade. Novamente, o microprocessador faz uma operação de leitura do conteúdo da posição de memória, cujo endereço é dado pelo conteúdo do registrador *Stack Pointer* (SP) e o armazena na parte menos significativa do conteúdo do registrador *Program Counter* (PCL). Depois, mais uma vez, o conteúdo do registrador *Stack Pointer* (SP) é diminuído em uma unidade. Assim, a próxima instrução a ser executada será aquela contida no conteúdo da posição de memória, cujo endereço é o imediatamente posterior à última instrução que foi executada pelo microprocessador, antes que a interrupção ocorresse. Nenhum *flag* do conteúdo do registrador *Program Status Word* (PSW) é afetado por essa instrução.

A Tabela 5.3 descreve as instruções RETI.

Exemplo: Inicialmente, considere que o conteúdo do registrador *Stack Pointer* (SP) seja igual a 30h e que o conteúdo das posições de memória, cujos endereços da RAM interna são 2Fh e 30h, possua os valores 34h e 12h, respectivamente, como é mostrado a seguir.

Stack Pointer (SP)	Endereço da memória RAM interna	Conteúdo da memória RAM interna
	2Eh	XXh (dado qualquer)
	2Fh	34h
30h →	30h	12h

Depois de executar a instrução RETI, o conteúdo do registrador *Stack Pointer* será igual a 2Eh e o conteúdo do registrador *Program Counter* (PC) será igual a 1234h, como é mostrado a seguir.

$$\begin{aligned}
 \text{RET} \Rightarrow & \quad (\text{PC}_{15,8}) \leftarrow ((\text{SP})) \quad \Leftrightarrow (\text{PCH}) \leftarrow (30h) = 12h \\
 & (\text{SP}) \leftarrow (\text{SP}) - 1 \quad = 30h - 1 \quad = 2Fh \\
 & (\text{PC}_{7,0}) \leftarrow ((\text{SP})) \quad \Leftrightarrow (\text{PCL}) \leftarrow (2Fh) = 34h \\
 & (\text{SP}) \leftarrow (\text{SP}) - 1 \quad = 2Fh - 1 \quad = 2Eh.
 \end{aligned}$$

Stack Pointer (SP)	Endereço da memória RAM interna	Conteúdo da memória RAM interna
2Eh →	2Eh	XXh (dado qualquer)
	2Fh	34h
	30h	12h

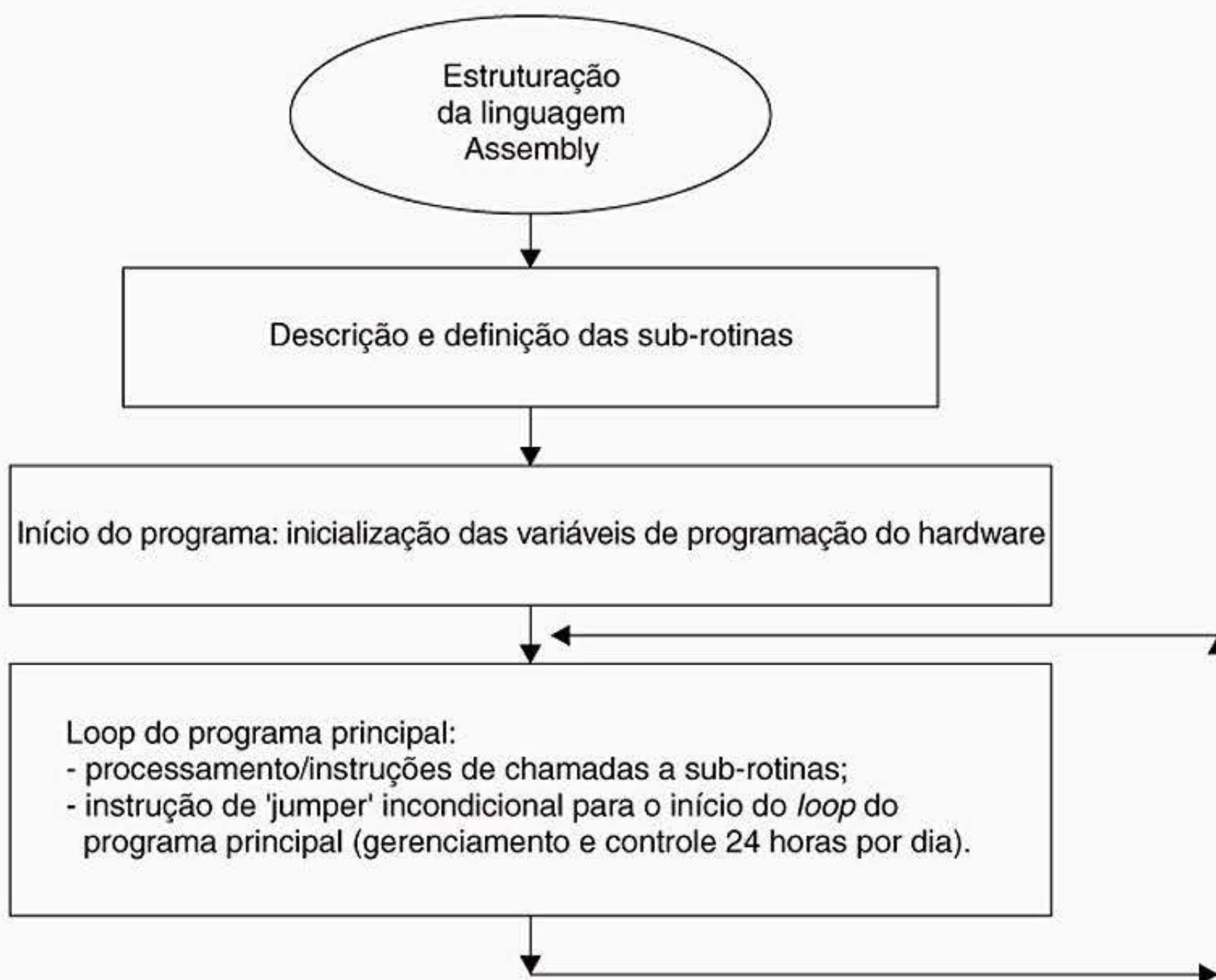
Assim, a próxima instrução a ser executada será aquela localizada no conteúdo da posição de memória, cujo endereço é dado pelo conteúdo do registrador *Program Counter* (PC), que vale 1234h.

5.3 Estruturação da linguagem de programação Assembly

Com base no conceito de sub-rotinas e aproveitando suas características de modularidade, a estruturação da linguagem Assembly consiste na elaboração de um programa formado por três partes principais quanto ao seu armazenamento na memória de programa ROM ou EPROM. Essas partes são:

- 1 - *Primeira parte do programa (área de programa 1)*: essa área é reservada para a alocação de todas as sub-rotinas que farão parte do projeto do programa final. Exemplo: rotinas de teclado, de display, de comunicação, de controle de um determinado processo etc.
- 2 - *Segunda parte do programa (área de programa 2)*: início do programa principal. Essa área é utilizada para alocar as instruções de inicialização de todas as variáveis que serão utilizadas pelo programa e é também onde são feitas todas as programações das interfaces de entrada e saída utilizadas pelo hardware. Essa programação é feita por meio da inicialização do conteúdo dos registradores de controle e de modo de operação das interfaces de entrada e saída (*timers/contadores, canal de comunicação serial* etc.).
- 3 - *Terceira parte do programa (área de programa 3)*: pertence também ao programa principal e contém o looping do mesmo. Nessa área, está contido o processamento propriamente dito, ou seja, é aqui que são definidas as instruções de chamadas às sub-rotinas, de forma organizada, para executar um determinado controle ou tarefa específica. A última instrução dessa parte do programa consiste em um **jumper incondicional** para a posição de memória do início do loop do programa principal. Com isso, consegue-se um controle contínuo de máquina ou de processo, ou seja, um controle durante 24 horas por dia.

O fluxograma a seguir representa a estruturação da linguagem Assembly.



Estruturação da linguagem Assembly:

;Sub-rotinas - primeira parte:

```

addr1: Sub-rotina 1
      RET
      :
addrN: Sub-rotina N
      RET
  
```

; Inicialização das variáveis do programa e do hardware – segunda parte, que pertence ao
; programa principal:

endereço do início do programa principal: instruções 1

;

instrução N

; Loop do programa principal - terceira parte, que pertence ao programa principal:

loop: instrução 1

;

instrução N

call adr1 ; chama a sub-rotina 1

;

call adrN ; chama a sub-rotina N

;

sjmp loop ; 'Fim do looping do programa

END ; principal'

O programa mostrado anteriormente está estruturado em um formato chamado *down-top* (em primeiro lugar, estão definidas as sub-rotinas e depois, o programa principal e seu loop). Da mesma maneira, também é possível estruturar um programa em Assembly em uma estrutura *top-down* (em primeiro lugar, é definido o programa principal e seu loop, e depois, as sub-rotinas).

Pelos grandes benefícios e vantagens que a estruturação de uma linguagem de programação produz, atualmente, todas as linguagens de programação disponíveis no mercado exigem tal formatação. Como exemplo, podemos citar as linguagens de programação C, Visual C, Visual Basic, Pascal, PL/M, JAVA, Visual JAVA, LISP etc.

Vantagens da utilização da linguagem de programação estruturada:

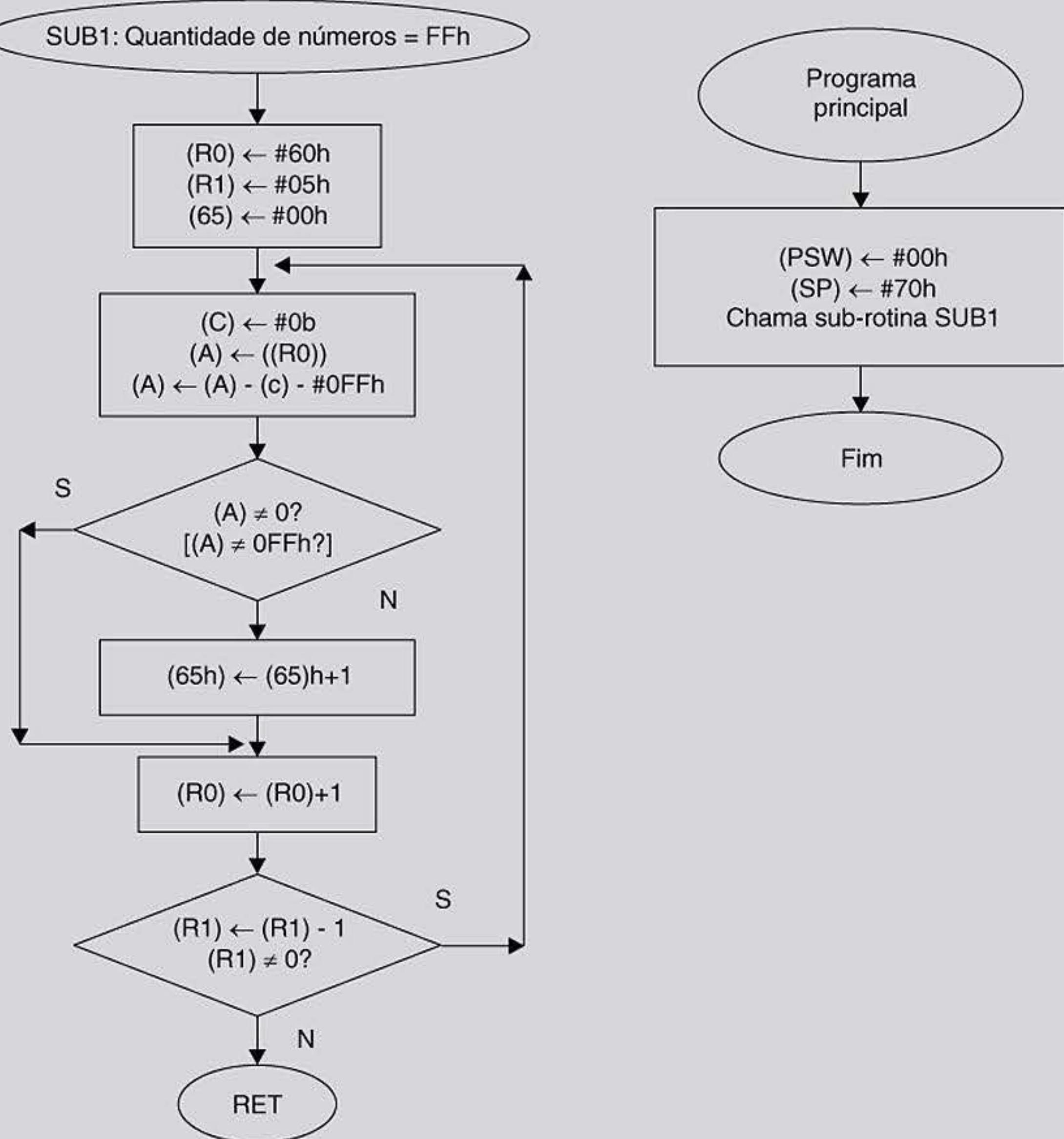
- ◆ metodologia sistemática de implementação de projeto de programa;
- ◆ para um projeto completo, constituído de muitas funções ou tarefas, a solução é facilmente implementada por meio da divisão do projeto completo em pequenos módulos (simples sub-rotinas) que são responsáveis por tarefas específicas e bem-definidas;
- ◆ o tempo de implementação é extremamente reduzido, pois é necessário apenas definir as diferentes tarefas (sub-rotinas) que o projeto deve executar, e essas tarefas devem ser chamadas no loop do programa principal, de maneira organizada e seqüencial;
- ◆ grande confiabilidade na implementação das sub-rotinas;
- ◆ tempo de desenvolvimento reduzido;
- ◆ tempo de simulação reduzido;
- ◆ tempo de emulação reduzido;
- ◆ tempo de manutenção reduzido;
- ◆ projeto modular;
- ◆ flexibilidade na implementação de novas características ao produto final, por meio da inserção e retirada dos módulos que compõem o projeto do programa;
- ◆ inserção dos módulos já implementados em bibliotecas de sub-rotinas. Na implementação de novos projetos, as mesmas bibliotecas podem ser utilizadas imediatamente;
- ◆ fácil manutenção, pois, quando ocorre um problema, este pode ser analisado diretamente em sua respectiva sub-rotina.

Exercícios resolvidos

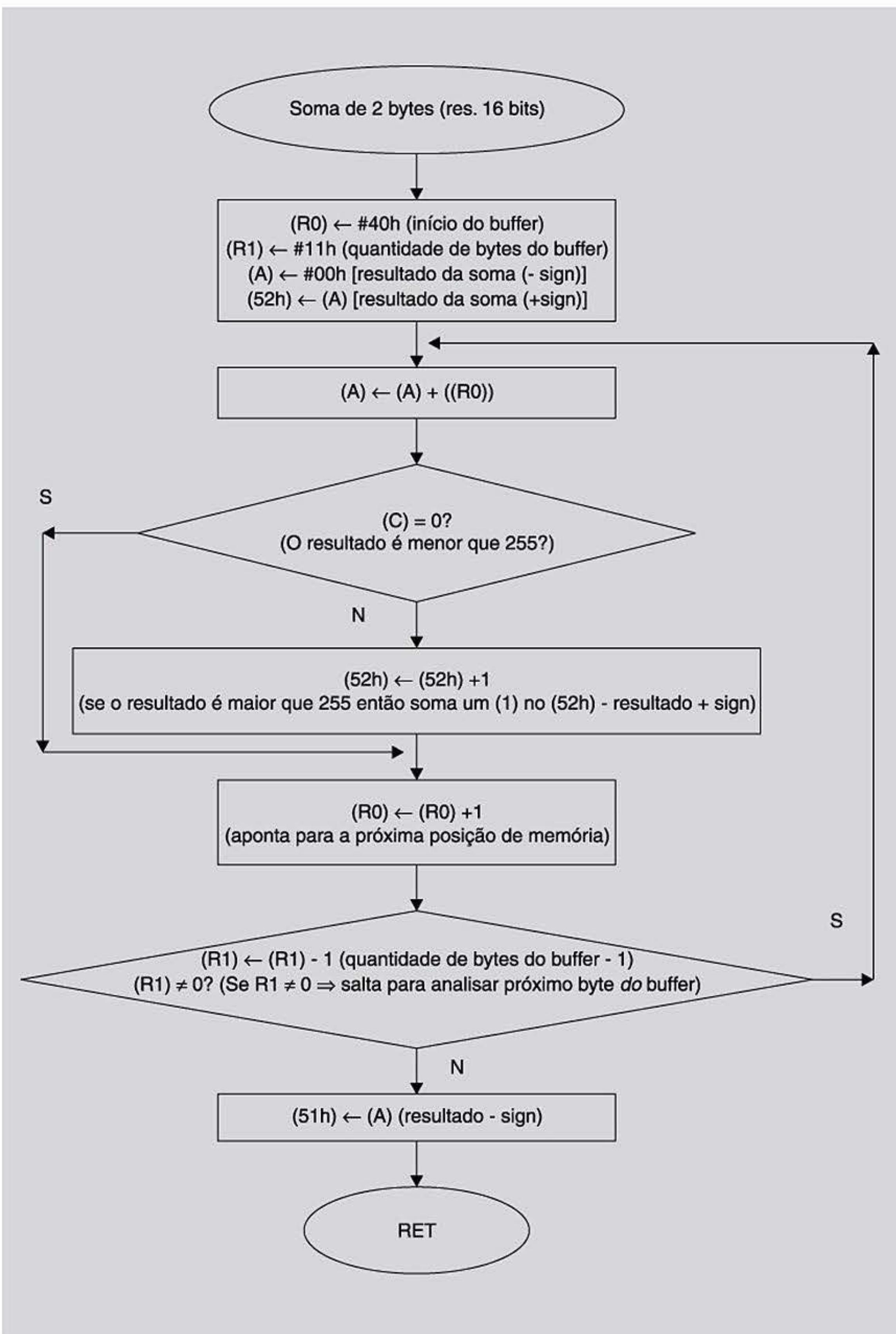
1 - Faça um programa estruturado (programa-fonte e fluxograma) em Assembly, utilizando o microcontrolador 8051 da Intel que calcula a quantidade de números iguais a FFh de uma faixa de memória que vai do conteúdo da posição de memória, cujo endereço é 60h, ao conteúdo da posição de memória, cujo endereço é 64h. O resultado deve ser armazenado no conteúdo da posição de memória cujo endereço é 65h.

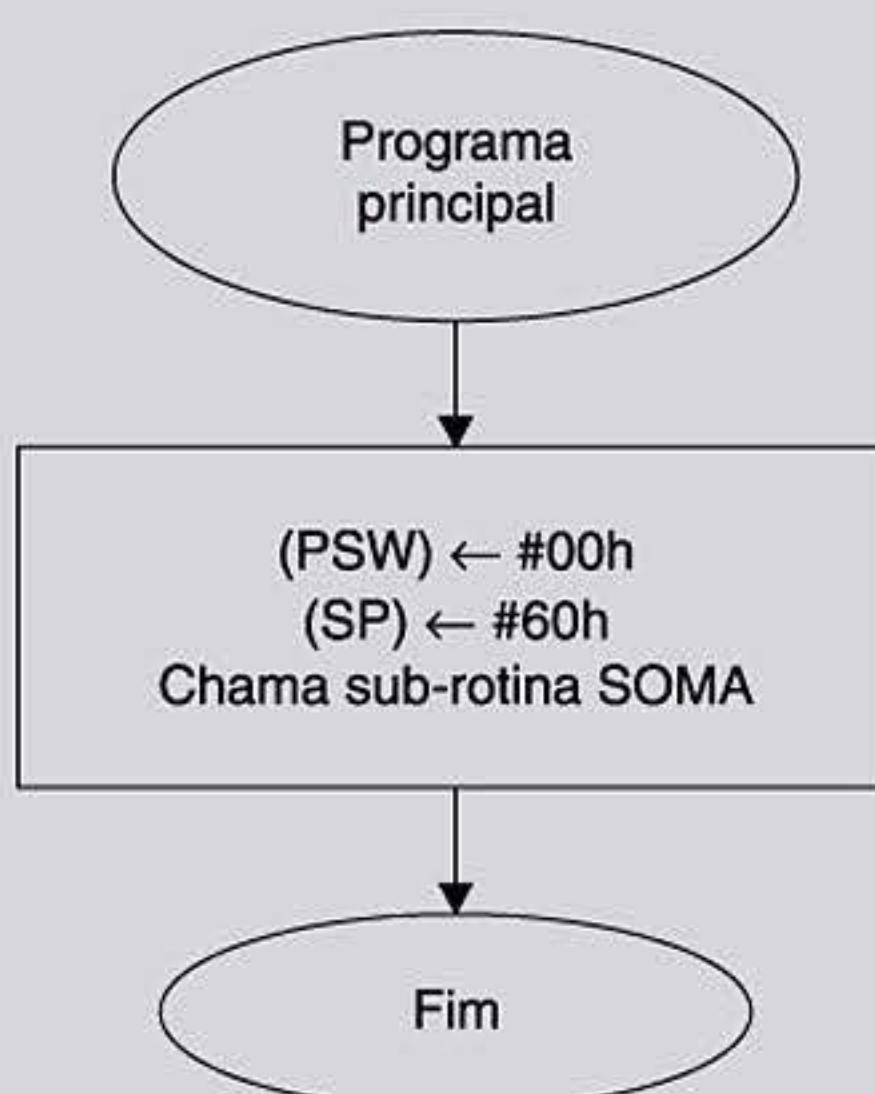
Solução:

DEFSEG	EXEMPLO, ABSOLUTE
SEG	EXEMPLO
ORG <i>0100h</i>	
<i>; Sub-rotina que calcula a quantidade de num = 0FFh</i>	
SUB1: MOV R0,#60h	; Endereço inicial do buffer
MOV R1,#05h	; Quantidades de elementos do buffer
MOV 65h,#00	; 'Zera' o contador de num = 0FFh
ADR2: CLR C	; 'Zera' o (C): não influenciar a oper. de sub.
MOV A,@R0	; Carrega (A) com o cont. do buffer de memória
SUBB A,#0FFh	; Subtrai o (A) da const. 0FFh
JNZ ADRI	; Se (A) ⁱ 0 (A ⁱ #0FFh) \Rightarrow (PC) = ADRI [não soma 1 no (65h)]
INC 65h	; Se (A) = 0 (A = #0FFh), soma um no (65h)
ADR1: INC R0	; Aponta p/ a próx. posição de memória a ser analisada
DJNZ R1,ADR2	; Existem mais dados a serem analisados? Se existirem, salta para <i>ADR2</i>
RET	; Se não existirem, finaliza a sub-rotina, se (R1) = 0 e retorna.
<i>; Início do programa principal</i>	
PROGP:MOV PSW,#00h	; Faz (PSW) \leftarrow #00h.
MOV SP,#70h	; Define (SP) = 70h (início da pilha)
ACALL SUB1	; Chama a sub-rotina <i>SUB1</i> .
END	; Fim do programa principal



2 - Faça um programa estruturado (fluxograma e programa-fonte) em Assembly, utilizando o microcontrolador 8051, que soma os bytes dos conteúdos das posições de memória do buffer de memória, que vai do endereço 40h ao endereço 50h. Considere que o resultado dessa soma pode ser maior que 255. Dessa maneira, a parte menos significativa do resultado deve ser armazenada no conteúdo da posição de memória cujo endereço é 51h, e a parte mais significativa do resultado deve ser armazenada no conteúdo da posição de memória cujo endereço é 52h.





; Programa-fonte:

DEFSEG EXEMPLO, ABSOLUTE

SEG EXEMPLO

ORG 0100h ; O programa será armazenado a partir do conteúdo da posição de memória ; ROM/EPROM de programa.

SOMA: MOV R0,#40h ; (R0) é o ponteiro do buffer. O valor inicial é igual a 40h (início da posição de memória do buffer).

MOV R1,#11h ; (R1) é um contador descendente, que contém a quantidade de bytes do buffer a serem somados (= 50h - 40h + 1).

MOV A,#00h ; 'Zera' o (A) que conterá a parte menos significativa do resultado de 16 bits.

MOV 52h,A ; 'Zera' o conteúdo da posição de memória, cujo endereço é 52h, que será responsável pelo armazenamento da parte mais significativa do resultado de 16 bits.

ADR2: ADD A,@R0 ; Soma o resultado - sign. (A) ao conteúdo da posição de memória cujo endereço é dado pelo conteúdo do registrador R0.

JNC ADR1 ; Se o resultado da soma for menor do que 255, salta (não soma 1 ao conteúdo da posição de memória cujo endereço é 52h (resultado + sign)).

INC 52h ; Se o resultado da soma for maior que 255, soma 1 ao conteúdo da posição de memória, cujo endereço é 52h (resultado + sign).

ADR1: INC R0 ; Aponta para o próximo byte da memória a ser somado

DJNZ R1,ADR2 ; Diminui em uma unidade o conteúdo do registrador R1 e salta, se ele for diferente de zero, isto é, houver mais dados a serem somados a partir da memória.

MOV 51h,A ; Armazena a parte - sign no conteúdo da posição de memória cujo endereço é 51h.

RET ; Retorno da sub-rotina por meio da leitura da pilha do endereço de retorno.

PROGP: MOV PSW,#00h ; Faz (PSW) ← #00h.

MOV SP,#60h ; Define (SP) = 60h (início da pilha)

LCALL SOMA ; Chama a sub-rotina SOMA

END ; Fim do programa principal

Exercícios e questões propostos

- 1 - Faça um programa estruturado (fluxograma e programa-fonte) em Assembly, utilizando o microcontrolador 8031, que transforme o buffer de memória em números pares sempre que um byte desse buffer for menor ou igual a 7Ah. Os endereços inicial e final do buffer são 3Ah e 68h, respectivamente.
- 2 - Faça um programa estruturado (fluxograma e programa-fonte) em Assembly, utilizando o microcontrolador 8051, que calcule a quantidade de números que apresentam o bit 5 igual a 1, do buffer de memória que vai da posição de memória 15h à posição 31h. O resultado deve ser colocado no conteúdo do registrador R2 do penúltimo banco de registradores. Esse mesmo programa deve determinar o menor número do buffer de memória que vai da posição de memória 34h à posição 52h. O resultado deve ser colocado no conteúdo da posição de memória cujo endereço é 69h.
- 3 - Faça um programa estruturado (fluxograma e programa-fonte) em Assembly, utilizando o microcontrolador 8051, que:
 - a) determine o maior número do buffer de memória que vai da posição de memória 34h à posição 61h. O resultado deve ser acrescentado ao conteúdo do registrador R7 do segundo banco de registradores;
 - b) determine a quantidade de números maiores que 77h do buffer de memória que vai da posição de memória 4Ah a 71h. O resultado deve ser acrescentado ao conteúdo do registrador B;
 - c) Transforme o buffer de memória em números que apresentam o bit 2 igual a 1, sempre que um byte desse buffer for igual a ABh. Os endereços inicial e final do buffer são 4Ah e 77h, respectivamente.
- 4 - Faça um programa estruturado (fluxograma e programa-fonte) em Assembly, utilizando o microcontrolador 8031, que calcule a quantidade de números maiores que 22h do buffer de memória que vai da posição de memória 33h à posição 41h. O resultado deve ser colocado no conteúdo do registrador R1 do segundo banco de registradores. O programa também deve calcular a quantidade de números que apresentam o bit 6 igual a 0 e o bit 2 igual a 1, do buffer de memória

que vai da posição de memória 45h à posição 51h. O resultado deve ser colocado no conteúdo da posição de memória cujo endereço é 3Ch. O programa também deve calcular a quantidade de números maiores que 65h e que o bit 4 seja igual a 1 do buffer de memória que vai da posição de memória 5Fh a 7Eh. O resultado deve ser colocado no conteúdo da posição de memória cujo endereço é 7Dh.

- 5 - Faça um programa estruturado (fluxograma e programa-fonte) em Assembly, utilizando o microcontrolador 8051, que realize a operação lógica *OR-Exclusivo* entre dois buffers de memória. O primeiro buffer vai do endereço 21h a 35h. O segundo buffer começa a partir da posição de memória 36h. O resultado da operação lógica entre esses dois buffers de memória deve ser colocado em um outro buffer, cujo endereço inicial é 65h. Transforme o buffer resultante em números que apresentam o bit 6 igual a 0.

capítulo

6

As PORTAS (PORTES) DE ENTRADA E SAÍDA E SUAS APLICAÇÕES NO CONTROLE DIGITAL DE MÁQUINA E DE PROCESSO

6.1 Objetivos

- ❖ Conhecer a arquitetura interna das portas da família de microcontroladores MCS-51
- ❖ Programar as portas atuando como entrada e como saída
- ❖ Ler informações das portas de entrada
- ❖ Escrever informações nas portas de saída
- ❖ Como detectar acionamentos por meio de um bit programado como entrada de dados
- ❖ Como detectar acionamentos por meio de vários bits programados como entradas de dados
- ❖ Como gerar tempo por meio das rotinas de temporização (atraso) por software
- ❖ Eliminando o ruído (*bounce*) gerado por chaves mecânicas
- ❖ Processar as informações das portas de entrada (identificação de acionamento de bits, de padrão de informação de entrada, do número de acionamento/desacionamento de chaves)
- ❖ Exemplos de projeto utilizando as portas de entrada e saída

6.2 Introdução teórica

6.2.1 - A estrutura e a operação das portas: todas as quatro portas, na família de microcontroladores MCS-51, são bidirecionais. Cada uma delas consiste de um *latch* (registradores de funções especiais – P0 a P3), um driver e um buffer de entrada.

Os drivers de saída das portas 0 e 2 e o buffer de entrada da porta 0 são utilizados no acesso à memória externa. A função da porta 0 é definir o byte menos significativo do endereço de memória externa multiplexado no tempo com o byte que será escrito ou lido por meio da memória externa, e a função da porta 2 é definir o byte mais significativo do endereço de memória externa ao longo do ciclo de instrução.

Alguns dos drivers de saída dos buffers de entrada da porta 1 (P1) e todos os bits da porta 3 (P3) são multifuncionais, ou seja, além de poderem trabalhar normalmente como portas de entrada e saída, eles também podem assumir funções alternativas, como as descritas na Tabela 6.1.

Tabela 6.1 Descrição das funções alternativas que alguns bits podem assumir da porta P1 do 8052/32 e da porta P3.

Pinos das portas	Funções alternativas
P1.0 *	T2 (entrada externa do <i>timer/contador</i> 2)
P1.1 *	Captura/ <i>trigger</i> de recarregamento do <i>timer/contador</i> 2
P3.0	RXD (porta de entrada serial)
P3.1	TXD (porta de saída serial)
P3.2	INT0/ (interrupção externa 0)
P3.3	INT1/ (interrupção externa 1)
P3.4	T0 (entrada externa do <i>timer/contador</i> 0)
P3.5	T1 (entrada externa do <i>timer/contador</i> 1)
P3.6	WR/ (sinal de escrita de memória de dados externa)
P3.7	RD/ (sinal de leitura de memória de dados externa)

* P1.0 e P1.1 somente no 8052/32.

As funções alternativas só podem ser ativadas se o *latch* correspondente da respectiva porta for definido com 1 lógico. Caso o *latch* da porta seja definido com 0 lógico, sua função alternativa estará desativada e sua função principal será a de reproduzir a condição do conteúdo do registrador de função especial P3 para o 8051/31 e P1.0 e P1.1 para o 8052/32.

No modo *timer* ou contador, ocorrerá o recarregamento automático do *timer/contador* 2 dos registradores RLDH e RLDL, somente se a opção de recarregamento estiver selecionada (CP/RL2barra = 0).

As figuras de 6.1 a 6.4 ilustram o diagrama funcional do *latch* de um bit típico e um buffer de entrada e saída em cada uma das quatro portas existentes na família de microcontroladores MCS-51. O *latch* de um bit (um bit do registrador de função especial de uma porta) é representado como um *flip-flop* do tipo D, em que sua entrada de *clock* está ligada a um sinal de escrita do *latch* originário da CPU. A saída Q do *flip-flop* é colocada no barramento interno em resposta a um sinal de leitura do *latch* originário da CPU. O nível do pino da porta é colocado no barramento interno em resposta a um sinal de

leitura do pino originário da CPU. Algumas instruções lêem uma porta, outras ativam o sinal de leitura do *latch* e outras ativam o sinal de leitura do pino.

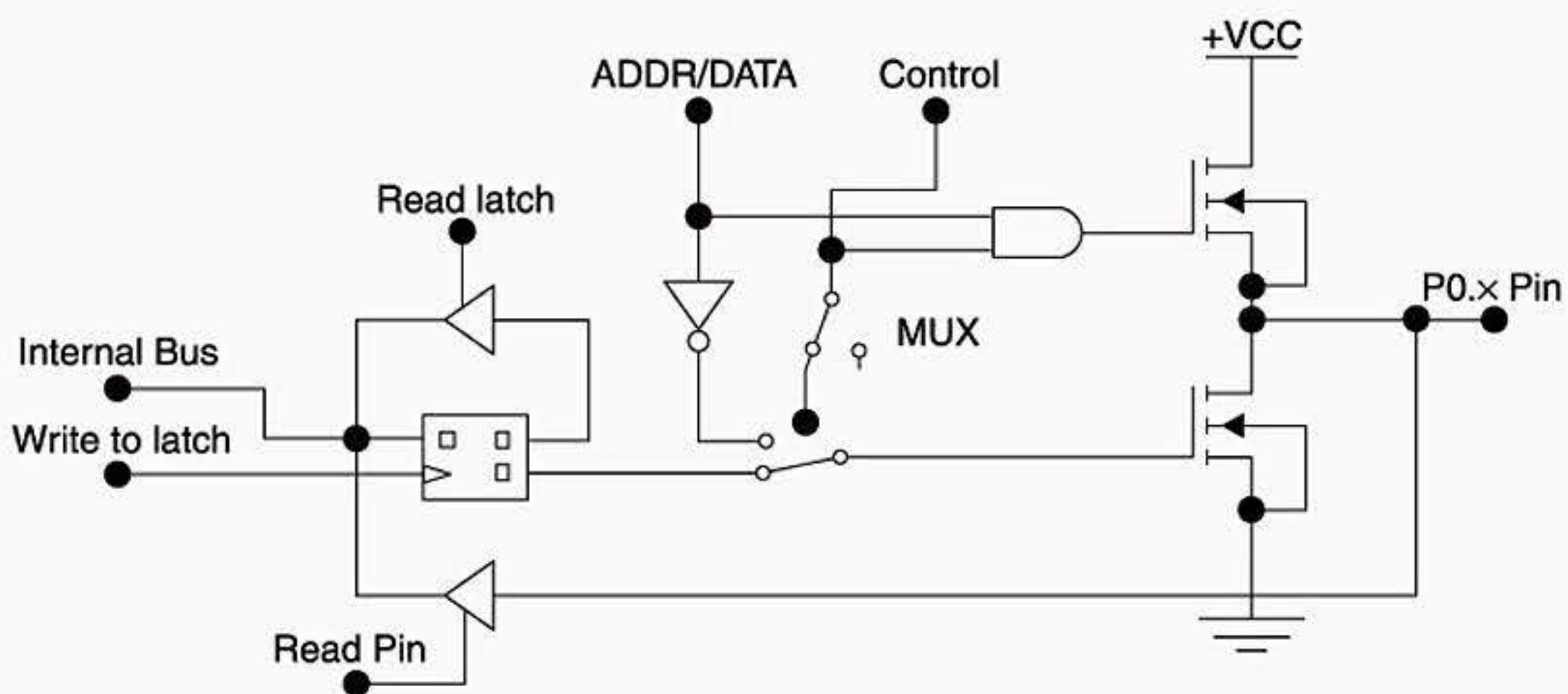


Figura 6.1 Arquitetura interna dos bits da porta P0 da família de microcontroladores MCS-51 da Intel.

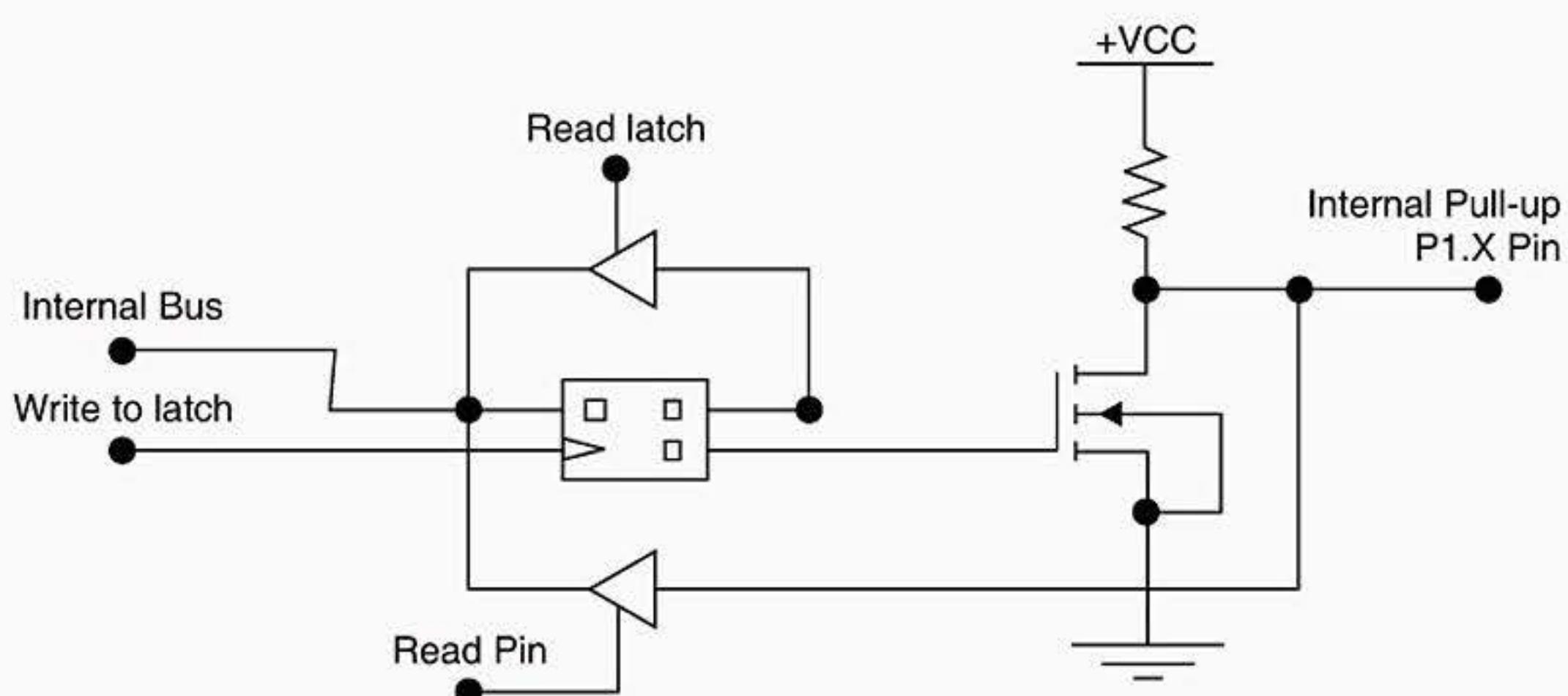


Figura 6.2 Arquitetura interna dos bits da porta P1 da família de microcontroladores MCS-51.

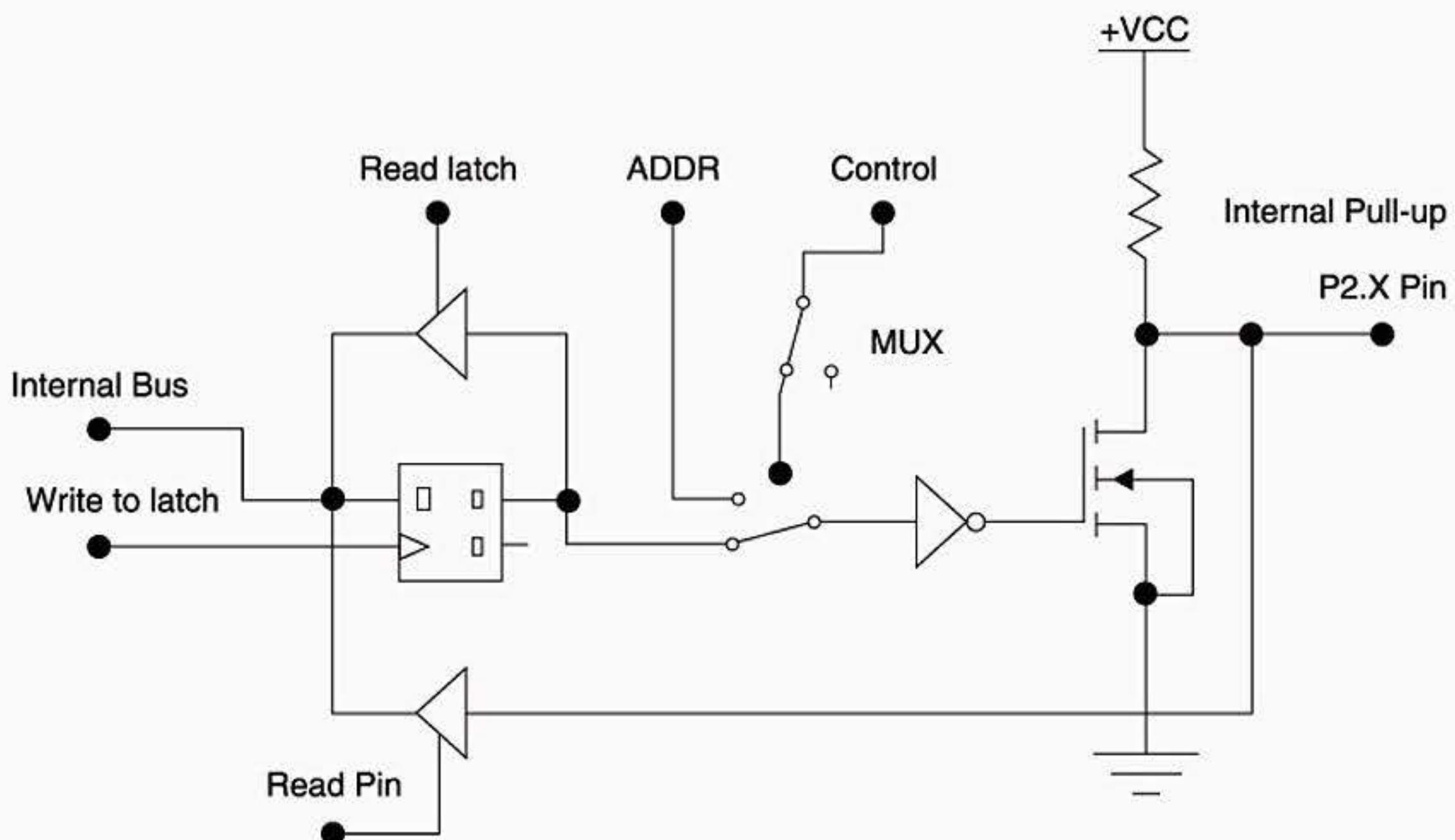


Figura 6.3 Arquitetura interna dos bits da porta P2 da família de microcontroladores MCS-51.

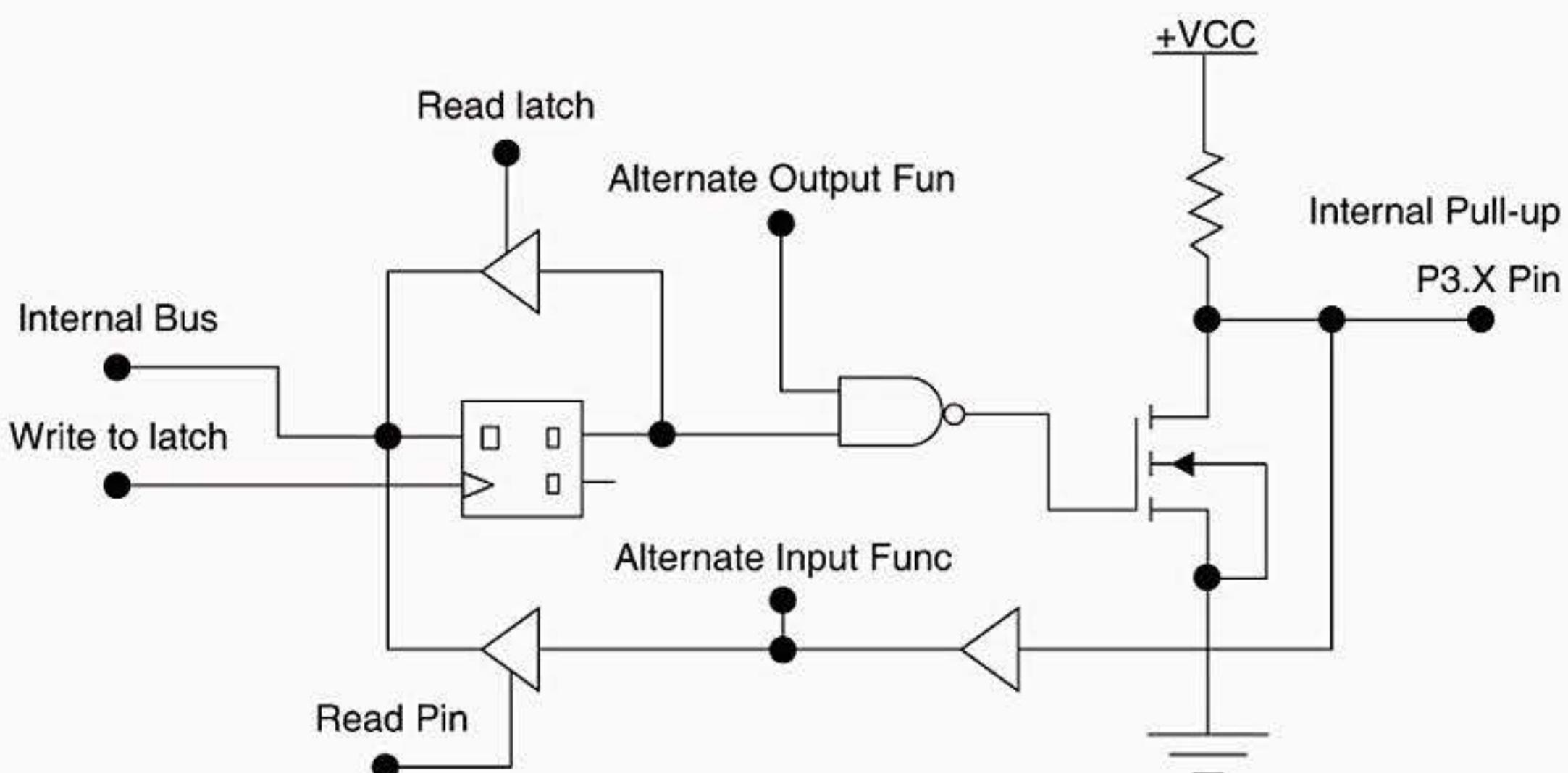


Figura 6.4 Arquitetura interna dos bits da porta P3 da família de microcontroladores MCS-51 (circuito pseudobidirecional).

As portas 1, 2 e 3 têm *pull-ups* internos. A porta 0 tem saída de dreno aberto. Cada linha de entrada e saída pode ser utilizada independentemente como entrada ou como saída.

As portas 0 e 2 não podem ser utilizadas como entrada e saída de finalidade geral, quando estão sendo empregadas como barramento de endereço e de dados. Para serem utilizadas como entrada, os *latches* das portas

deverão apresentar em sua saída o nível 1 lógico, que desaciona o FET do driver de saída. Então, para as portas 1, 2 e 3, o pino é colocado em 1 lógico por um *pull-up* interno. E também pode ser colocado em 0 lógico por uma fonte externa.

A porta 0 difere por não ter *pull-ups* internos. O FET ligado ao *pull-up* do driver de saída de P0 é utilizado somente quando a porta está emitindo 1 lógico durante o acesso à memória externa. Caso contrário, os FETs que estão ligados aos *pull-ups* estarão desligados. Conseqüentemente, as linhas de P0 que estão sendo utilizadas como linhas de saída são dreno aberto. Escrever 1 lógico no *latch* deixa os FETs de saída desligados, assim o pino flutua. Nessa condição, ele pode ser utilizado como uma entrada de alta impedância.

Quando um bit de uma porta é considerado como a fonte de informação nas instruções que manipulam as portas, a informação é definida pelo nível lógico do próprio pino, em vez da saída do *latch*. Esse nível lógico é sempre afetado pelo microcontrolador e pela entrada externa. O valor lido é essencialmente a função *OR* do transistor e do dispositivo externo. Se o dispositivo externo apresenta alta impedância (entrada de *gate* de um transistor MOSFET ou saída em alta impedância), ao ler um pino de entrada, será refletido o nível lógico prévio da saída. Para utilizar um pino como entrada, o correspondente *latch* de saída deverá estar em 1 lógico. Assim, a saída barrada do *latch* ficará em 0 lógico, e o transistor se comportará como uma chave aberta. Com isso, o dispositivo externo pode fornecer ao pino de entrada do buffer de entrada um sinal lógico alto ou baixo. Dessa maneira, a mesma porta pode ser utilizada como entrada e saída por meio da escrita de 1 lógico em todos os pinos utilizados como entrada nas operações de saída, ignorando todos os pinos utilizados como saída nas operações de entrada.

Nas instruções de um operando (INC, DEC, DJNZ e CPL), a saída do *latch*, em vez do pino de entrada, é utilizada como a fonte de dados. Similarmente, as instruções de dois operandos utilizando a porta como origem e destino (ANL, ORL e XRL) empregam as saídas dos *latches*. Isso assegura que os bits dos *latches* correspondentes aos pinos utilizados como entrada não serão 'resetados' no processo de execução dessas instruções.

As instruções booleanas JBC testam a saída dos *latches*, em vez dos pinos de entrada.

Se o *latch* da porta P3 contém 1 lógico, então o nível de saída é controlado pelo sinal chamado *função de saída alternativa*. O nível do pino P3.X está sempre disponível para a função de entrada alternativa.

Devido ao fato de as portas 1, 2 e 3 terem *pull-ups* fixos, elas são chamadas portas *quase bidirecionais*. Quando configuradas como entrada, elas vão

para 1 lógico e fornecerão corrente (IIL), se externamente colocadas em 0 lógico. A porta 0, de outra maneira, é considerada verdadeiramente bidirecional, pois, quando configurada como entrada, ela flutua.

Após um sinal de *reset*, todas as saídas dos *latches* das portas da família de microcontroladores MCS-51 são definidas como 1 lógico. Dessa maneira, todas as portas ficam programadas como entrada. Se um 0 lógico for subsequentemente escrito em um *latch* de uma porta, ele fica programado como saída. Essa mesma porta pode ser novamente reconfigurada como entrada, se uma nova operação de escrita de alto nível lógico for executada.

a) Característica de ler-modificar-escrever

As instruções que lêem, modificam e escrevem no *latch* são chamadas de instruções de 'ler-modificar-escrever':

- ANL (p. ex.: ANL P1,A);
- ORL (p. ex.: ORL P2,A);
- XRL (p. ex.: XRL P3,A);
- JBC (p. ex.: JBC P1.1, LABEL);
- CPL (p. ex.: CPL P3.0);
- INC (p. ex.: INC P2);
- DEC (p. ex.: DEC P2);
- DJNZ (p. ex.: DJNZ P3, LABEL);
- MOV PX,Y,C (p. ex.: MOV P0.1,C);
- CLR PX,Y (p. ex.: CLR P1.2);
- SETB PX,Y (p. ex.: SETB P2.0).

As instruções mostradas anteriormente são direcionadas aos *latches*, em vez de às portas, para evitar a má interpretação do nível de tensão do pino. Por exemplo, um bit de uma porta pode ser usado para fornecer tensão à base de um transistor. Quando um 1 lógico é escrito no bit, o transistor é ligado. Se o microprocessador ler o mesmo bit da porta por meio de seu pino físico, em vez da saída do *latch*, ele lerá a tensão na base do transistor e, dessa maneira, interpretará como 0 lógico. Lendo o *latch*, em vez do pino, é retornado o valor correto de 1 lógico.

b) Programação dos conteúdos das portas dos microcontroladores MCS-51

Um bit de uma porta é *programado como entrada* por meio de uma operação de escrita de 1 lógico nos *latches* da porta. Como foi visto na Figura 6.1, a saída do *latch* Q fica em 1 lógico e a outra saída Q complementada fica em 0 lógico. Quando isso ocorre, o transistor opera como uma chave aberta, e a informação do pino pode ser capturada pelo microprocessador por meio do barramento de dados. Nesse caso, o conteúdo das portas poderá ler informações de determinadas interfaces de entrada. As interfaces de entrada, normalmente utilizadas nos sistemas microprocessados ou microcontrolados,

utilizam sensores com o objetivo de controlar determinadas variáveis de um processo que se deseja gerenciar. Exemplos de sensores utilizados em interfaces de entrada são os de temperatura, de pressão, de umidade, ópticos, magnéticos, indutivos etc. Outros exemplos de interfaces de entrada geralmente utilizadas como entrada de informação do mundo externo para o sistema microprocessado ou microcontrolado são as interfaces com teclado, interfaces com chaves *push-button*, interfaces com *dip switches*, interfaces com mouse, interfaces com scanners etc.

b.1) Exemplos de programação das portas:

A seguir, são mostrados alguns exemplos de programação das portas para trabalharem como entrada de informação, definidas por interfaces externas ao sistema microcontrolado (mundo externo):

1 - Programação de todos os bits das portas P0 para trabalharem como bits de entrada de informação do mundo externo para o microcontrolador.

- ◆ MOV P0,#0FFh

- ◆ *Representação simbólica:* $(P0) \leftarrow \#FFh$.
- ◆ *Significado da instrução:* o conteúdo da porta P0 será inicializado com o valor constante FFh.
- ◆ *Explicação:* essa instrução escreverá 1 lógico nos 8 *latches* da porta P0 e, dessa maneira, todos os bits da porta P0 serão programados para funcionar como entrada de informação do mundo externo para o sistema microcontrolado.

2 - Programação do bit mais significativo da porta P0 para trabalhar como entrada de informações do mundo externo para o microcontrolador; e o restante, como bits programados como saída de informações do sistema microcontrolado para o mundo externo.

- ◆ MOV P1,#80h

- ◆ *Representação simbólica:* $(P1) \leftarrow \#80h = \#1000\ 0000_b$
- ◆ *Significado da instrução:* o conteúdo da porta P1 será inicializado com o valor constante 80h.
- ◆ *Explicação:* essa instrução escreverá 1 lógico no bit mais significativo do *latch* da porta P1 e 0 lógico nos demais bits do *latch* da porta P1; dessa maneira, o bit mais significativo da porta P1 estará programado como entrada e os demais bits serão programados como saída.

3 - Programação dos bits pares da porta P0 para trabalharem como bits de entrada de informação do mundo externo para o microcontrolador e dos bits ímpares da porta P0 para trabalharem como bits de saída de informação do microcontrolador para o mundo externo.

◆ **MOV P2,#55h**

- ◆ *Representação simbólica:* $(P2) \leftarrow \#55h = \#0101\ 0101_b$
- ◆ *Significado da instrução:* o conteúdo da porta P2 será inicializado com o valor constante 55h;
- ◆ *Explicação:* essa instrução escreverá 1 lógico nos bits 0, 2, 4 e 6 do *latch* da porta P2 e 0 lógico nos bits 1, 3, 5 e 7 do *latch* da porta P2; dessa maneira, os bits pares da porta P2 serão programados como entrada e os bits ímpares serão programados como saída.

c) *Operações de leitura nas portas da família de microcontroladores MCS-51*

Antes de fazer a leitura do conteúdo das portas, é necessário programar as mesmas para operarem como bits de entrada de informação do mundo externo para o sistema microcontrolado, como foi visto no item anterior. Depois disso, pode-se fazer as operações de leitura de informações.

Observação: após um sinal de *reset*, todas as saídas dos *latches* das portas são definidas por hardware com nível lógico alto e consequentemente todas as portas ficam programadas como entradas.

As instruções *MOV A, PX* ($X = 0, 1, 2$ e 3), *MOV Rn, PX*, *MOV @Ri, PX* são exemplos de instruções que lêem o conteúdo das portas.

c.1) *Exemplo de operação de leitura de 7 portas:*

MOV A,#0FFh: $(A) \leftarrow \#0FFh \Rightarrow$ Byte de programação para fazer todos os bits da porta P0 trabalharem como bits de entrada de informação do mundo externo para o sistema microcontrolado.

MOV P0,A : $(P0) \leftarrow (A) \Rightarrow$ Programa todos os bits da porta P0 como entrada de informação do mundo externo.

MOV A,P0 : $(A) \leftarrow (P0) \Rightarrow$ Lê o conteúdo da porta P0 com a informação definida pelo mundo externo. Essa informação pode ter sido definida por uma interface constituída de um *dip switch* que foi ligado à porta P0, por exemplo.

d) *Operação de escrita nas portas da família de microcontroladores MCS-51*

Antes de fazer a escrita nos conteúdos das portas, é necessário programar as mesmas como bits de saída de informação do sistema microcontrolado para o mundo externo, como foi visto no item B).

As instruções *MOV PX, A*, *MOV PX, Rn*, *MOV PX,@Ri*, *ANL*, *ORL*, *XRA* são exemplos de instruções que escrevem informações nas portas.

d.1) *Exemplo de operação de escrita em portas:*

MOV A,#00h : (A) ← #00h ⇒ Byte de programação para fazer todos os bits da porta P0 trabalharem como bits de saída de informação do sistema microcontrolado para o mundo externo.

MOV P0,A : (P0) ← (A) ⇒ Programa todos os bits da porta P0 como saída de informação para o mundo externo.

MOV P0,R5 : (P0) ← (R5) ⇒ Escreve a informação existente no conteúdo do registrador R5 no conteúdo da porta P0, que será disponibilizada para o mundo externo. Caso a porta P0 esteja ligada com lógica positiva a um conjunto de 8 leds, os que estiverem ligados aos bits em zero lógico ficarão desativados e os leds ligados aos bits em 1 lógico ficarão ativados.

e) *Monitorando acionamentos gerados por interfaces de entrada implementadas por chaves e sensores por meio das portas*

Uma das maneiras mais simples de detectar acionamentos por meio das portas de entrada é utilizar as instruções "JB PX.Y, \$" (Se $PX.Y = 1$ então $(PC) \leftarrow \$$) e "JNB PX.Y,\$" (Se $PX.Y = 0$ então $(PC) \leftarrow \$$). O símbolo de cifrão (\$), utilizado como argumento nas instruções JB e JNB, simboliza o próprio endereço da instrução.

Assim, a instrução *JB PX.Y,\$* simplifica a escrita do programa-fonte e pode ser representada como:

addr1: JB PX.Y, addr1 ⇔ Se $PX.Y = 1$ então $(PC) \leftarrow addr1$

addr1: JNB PX.Y, addr1 ⇔ Se $PX.Y = 0$ então $(PC) \leftarrow addr1$

e.1) *Exemplos de monitoração de acionamentos de interfaces de entrada utilizando as portas:*

1 - Monitoração de apenas um bit específico de uma porta:

1.1 -

$JB P1.0, \$ \Rightarrow (PC) \leftarrow (PC) + 3 \Rightarrow$ isso significa calcular o endereço da instrução seguinte, pois são somadas três unidades ao conteúdo do registrador *Program Counter* (PC), uma vez que a instrução JB P1.0, \$ é constituída de três bytes;

\Rightarrow Se $(P1.0) = 1$ então $(PC) \leftarrow \$ \Rightarrow$ significa saltar para o endereço da própria instrução, ou seja, enquanto o $(P1.0) = 1$ lógico, o programa prossegue no próprio endereço da instrução JB P1.0, \$, isto é, fica em loop até que o conteúdo desse bit mude para 0 lógico. Isso corresponde a um acionamento em lógica negativa $1 \rightarrow 0$. Quando o conteúdo do bit 0 da porta P1 mudar para 0 lógico, o programa prosseguirá no endereço imediatamente seguinte a essa instrução, que corresponde ao endereço da próxima instrução.

1.2 -

$JNB P2.4, \$ \Rightarrow (PC) \leftarrow (PC) + 3 \Rightarrow$ isso significa calcular o endereço da próxima instrução, pois são somadas três unidades ao conteúdo do registrador *Program Counter* (PC), uma vez que a instrução JB P2.4, \$ é constituída de 3 bytes;

\Rightarrow Se $(P2.4) = 0$ então $(PC) \leftarrow \$ \Rightarrow$ significa saltar para o endereço da própria instrução, ou seja, enquanto o $(P2.4) = 0$ lógico, o programa prossegue no próprio endereço da instrução JB P2.4, \$, isto é, fica em loop até que o conteúdo desse bit mude para 1 lógico. Isso corresponde a um acionamento em lógica positiva $0 \rightarrow 1$. Quando o conteúdo do bit 4 da porta P2 mudar para 1 lógico, o programa prosseguirá no endereço imediatamente seguinte a essa instrução que corresponde ao endereço da próxima instrução.

2 - Monitoração de vários acionamentos de interfaces de entrada utilizando as portas:

A instrução $CJNE A, direct, rel [(PC) \leftarrow (PC) + 3; \text{se } (A) \neq (\text{direct}) \text{ então } (PC) \leftarrow (PC) + rel; \text{ se } (A) < (\text{direct}) \text{ então } (C) \leftarrow 1 \text{ caso contrário } (C) \leftarrow 0]$ pode ser utilizada para detectar o acionamento de mais de um bit de uma porta. Para isso, devemos definir um padrão (byte) no conteúdo do registrador acumulador (A) e comparar com o conteúdo da porta (*direct*). Caso o conteúdo do registrador acumulador (A) seja diferente (\neq) do conteúdo da porta, o programa deve 'saltar' para o endereço da própria instrução (\$), ou seja, enquanto o conteúdo do registrador acumulador (A) e o conteúdo da porta forem diferentes, o programa ficará em loop, executando essa mesma instrução. Quando o conteúdo da porta for ajustado por interferência externa (por exemplo, o operador), com um valor igual ($=$) ao conteúdo do registrador acumulador (A), o programa prosseguirá no endereço imediatamente subsequente a essa instrução.

Como exemplo, considere que à porta P1 esteja ligada uma interface constituída de um *dip switch* de oito chaves, que o acionamento é feito por lógica negativa ($1 \rightarrow 0$) e que, inicialmente, todos os bits da porta P1 estão em 1 lógico, pois as chaves se encontram desativadas. Assim, o programa mostrado a seguir aguarda que o conteúdo dos bits 0 e 7 da porta P1 seja acionado.

MOV A,#01111110b	$\Rightarrow (A) \leftarrow \#7Eh = 0111\ 1110_b \Rightarrow$ \Rightarrow define padrão (byte) = #7Eh (aguarda que os bits 7 e 0 sejam acionados e os demais bits não).
CJNE A,P1,S	\Rightarrow compara (A) com o (P1). Se $(A) \neq (P1) \Rightarrow (PC) \leftarrow \$$, ou seja, 'salta' para o mesmo endereço da instrução. Essa instrução aguarda que o conteúdo da porta P1 seja igual ao conteúdo do acumulador (A), ou seja, fica em loop. Quando $(A) = (P1)$, o programa sai desse loop e executa a instrução imediatamente seguinte a essa.

3 - Monitoração de um acionamento qualquer de um bit de uma porta:

A seguir, há um exemplo de rotina que monitora um acionamento qualquer de um dos bits de uma porta, considerando o acionamento com lógica negativa:

```

WAIT: MOV  A,#0FFh ; (A) ← #0FFh [ Inicialmente, chaves desativadas  $\Rightarrow (P1) = 0FFh$  ]
      CLR  C       ; (C) ← #0b (para não influenciar a operação de subtração)
      SUBB A,P1    ; (A) ← (A) - (C) - (P1) [ subtrai (A) de (P1) com (C) = 0 ]
      JZ   WAIT     ; Se (A) = 0  $\Rightarrow (PC) \leftarrow$  WAIT [se nenhuma chave foi acionada, significa que
                  ; (A) = (P1), ou seja, não ocorreu nenhum acionamento, e o programa é desviado
                  ; para o endereço WAIT até que ocorra um acionamento. Dessa maneira, quando
                  ; (A)  $\neq$  (P1), o programa sai do loop].
  
```

Outra solução possível:

```

WAIT: MOV  A,P1    ; (A) ← (P1). Considere inicialmente que todas as portas do (P1) =
      ; FFh (nenhuma chave está acionada em zero lógico - lógica negativa)
      INC  A       ; (A) ← (A) + 1 (se nenhuma chave foi acionada (A) ← FFh + 1 = 00h e, no
                  ; entanto, se alguma chave foi acionada, (P1) será diferente de FFh.
                  ; Então (A) ← (A) + 1
                  ; jamais será igual a zero
      JZ   WAIT     ; Se (A) = 0  $\Rightarrow (PC) \leftarrow$  WAIT (aguarda o acionamento de qualquer 'bit' da
                  ; porta se (A) = 0, pois nenhuma chave foi acionada).
  
```

6.2.2 - Como gerar rotinas de tempo utilizando software: em projetos de controle simples utilizando microcontroladores ou em projetos com microcontroladores que não têm timers/contadores, muitas vezes, é necessário gerar rotinas de tempo utilizando software. Essas rotinas, geralmente, são utilizadas para definir o tempo de acionamento e desativação de determinadas portas de saída (tempo de acionamento e desativação de leds, bips, displays de sete segmentos, acionamento de gates de tiristores etc.)

1 - Utilizando um único registrador ou uma única posição de memória

Para implementar rotinas de tempo por meio de software é necessário escolher um determinado registrador ou uma determinada posição de memória. Depois, é preciso inicializar seu conteúdo com um determinado valor

constante e, finalmente, ficar diminuindo o conteúdo desse registrador ou dessa posição de memória até que o seu valor se torne igual a zero. Um exemplo desse tipo de rotina é dado na Figura 6.5.

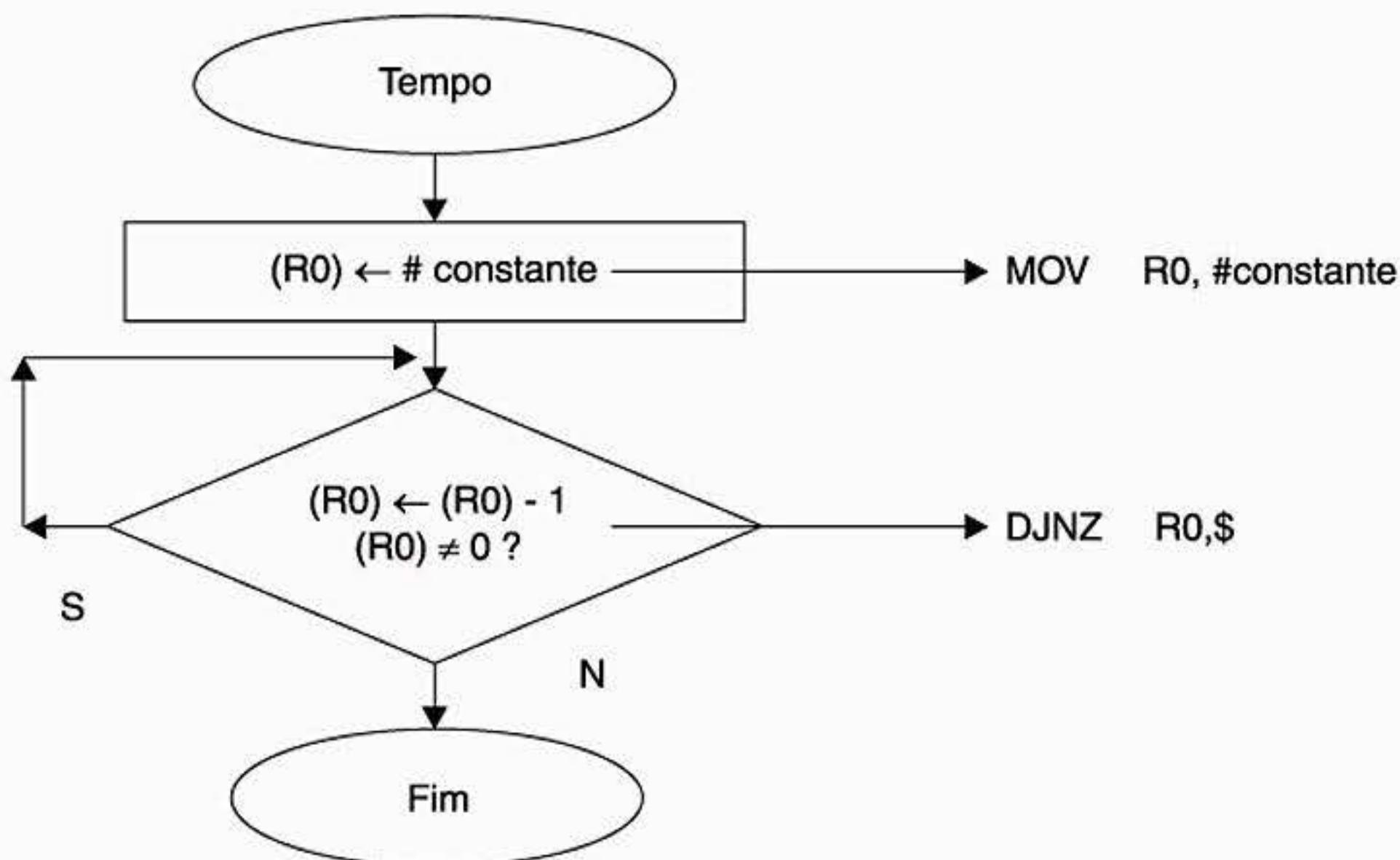


Figura 6.5 Fluxograma e programa-fonte de uma rotina que gera tempo por software.

O tempo gasto para executar esse programa, utilizando-se a Tabela 3.11 do Capítulo 3, é calculado da seguinte maneira:

Instrução	Número de vezes que a instrução é executada	Quantidade de ciclos de máquina por instrução
MOV R0,# constante	1	2
DJNZ R0,\$	constante	3

Assim, a quantidade de ciclos de máquina necessários para executar essa rotina é:

Quantidade total de ciclos de máquina = $1 * 2 + constante * 3 = 2 + 3 * constante$.

Sabendo-se que a freqüência de um ciclo de máquina na família de microcontroladores MCS-51 corresponde a 1/12 da freqüência do cristal, temos:

$$f_{\text{ciclo de máquina}} = (1/12) * f_{\text{cristal}}$$

O período de um ciclo de máquina é:

$$T_{\text{ciclo de máquina}} = 1/f_{\text{ciclo de máquina}} = 12/f_{\text{cristal}}$$

Assim, o tempo total gerado por essa rotina é:

$$\text{Tempo da rotina} = (\text{Quantidade total de ciclos de máquina}) * T_{\text{ciclo de máquina}} = (2+3*\text{constante}) * \frac{12}{f_{\text{cristal}}} \Rightarrow$$

$$\text{Tempo da rotina} = 12 * (2 + 3 * \text{KTE}) / f_{\text{cristal}} \text{ (s)}$$

Considere que um microcontrolador esteja utilizando um cristal de $f_{\text{cristal}} = 12 \text{ MHz}$ e que o valor da constante seja 255, então o tempo total da rotina é:

$$\text{Tempo da rotina} = 12 * (2 + 3 * 255) / (12M) = 0,767 \mu\text{s}$$

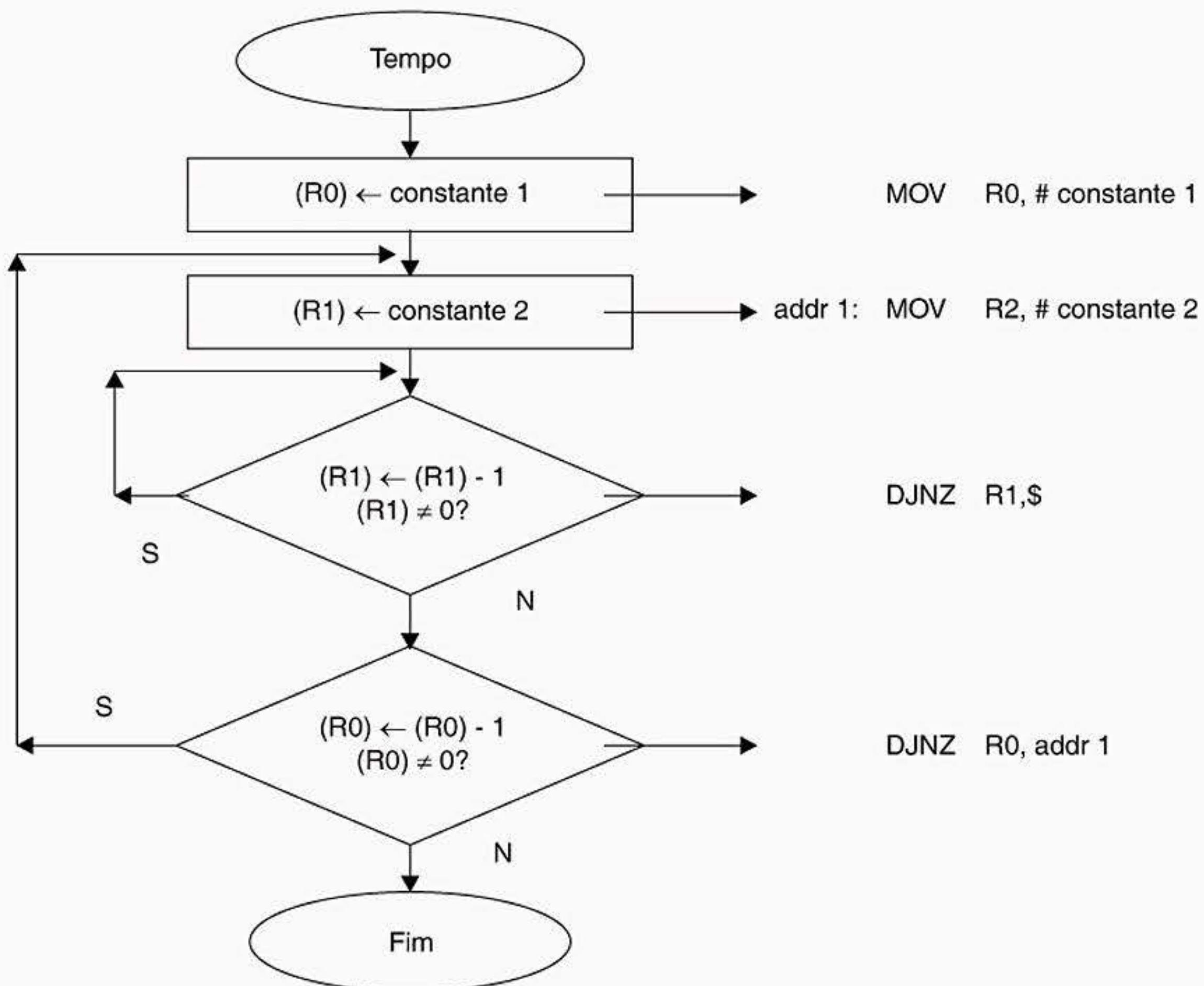


Figura 6.6 Fluxograma e programa-fonte de uma rotina que gera tempo por software utilizando dois contadores.

2 - Utilizando dois registradores ou duas posições de memória

O tempo gasto para executar esse programa, utilizando a Tabela 3.11 do Capítulo 3, é calculado da seguinte maneira:

Instrução	Número de vezes que a instrução é executada	Quantidade de ciclos de máquina por instrução
MOV R0,#constante1	1	2
addr1:MOV R1,#constante2	constante1	2
DJNZ R1,\$	constante1*constante2	3
DJNZ R0, addr1	constante1	3

Assim, a quantidade de ciclos de máquina necessária para executar essa rotina é:

$$\begin{aligned} \text{Quantidade total de ciclos de máquina} &= 1*2 + \text{constante1}*2 + \text{constante1}*\text{constante2}*3 + \text{constante1}*3 = \\ &= 2 + 5*\text{constante1} + 3*\text{constante1}.\text{constante2} \end{aligned}$$

O tempo total gasto para executar essa rotina é:

Sabendo-se que a freqüência de um ciclo de máquina na família de microcontroladores MCS-51 corresponde a 1/12 da freqüência do cristal, temos:

$$f_{\text{ciclo de máquina}} = (1/12) * f_{\text{cristal}}$$

O período de um ciclo de máquina é:

$$T_{\text{ciclo de máquina}} = 1/f_{\text{ciclo de máquina}} = 12/f_{\text{cristal}}$$

Assim, o tempo total gerado por essa rotina é:

$$\begin{aligned} \text{Tempo da rotina} &= (\text{Quantidade total de ciclos de máquina}) * T_{\text{ciclo de máquina}} = \\ &= (2 + 5*\text{constante1} + 3*\text{constante1}.\text{constante2}) * 12/f_{\text{cristal}} \Rightarrow \end{aligned}$$

$$\text{Tempo da rotina} = (2 + 5*\text{constante1} + 3*\text{constante1}.\text{constante2}) * 12/f_{\text{cristal}} \text{ (s)}$$

Considere que um microcontrolador esteja utilizando um cristal de $f_{\text{cristal}} = 12 \text{ MHz}$ e que os valores das constantes sejam iguais a 255, então o tempo total da rotina é:

$$\text{Tempo da rotina} = 12 * (2 + 5 * 255 + 3 * 255 * 255) / (12M) \cong 196 \text{ mseg} \cong 0,196 \text{ seg.}$$

6.2.3 - Eliminando o ruído (bounce) gerado por chaves mecânicas: quando se aciona uma chave mecânica, a mesma gera um ruído transitório, como é mostrado na Figura 6.7.

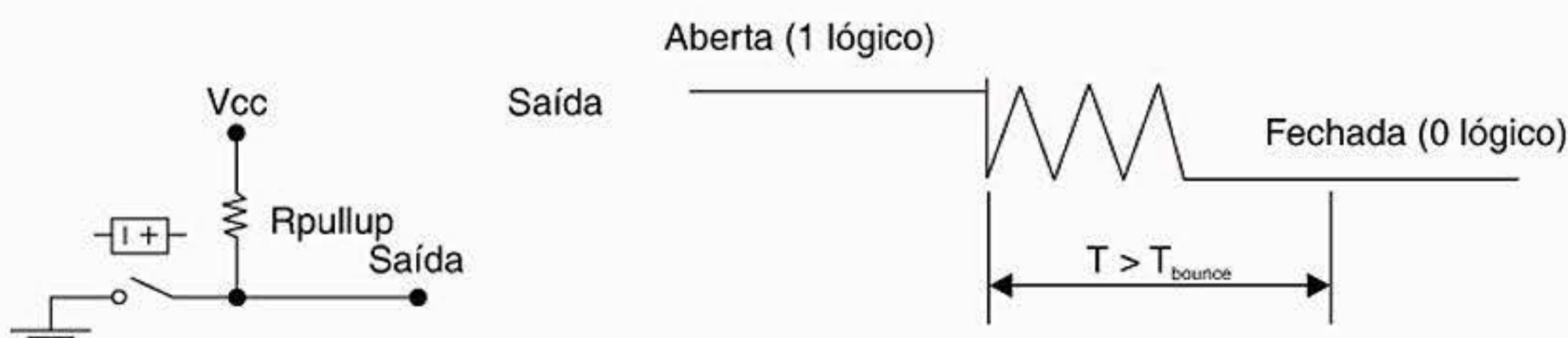


Figura 6.7 Hardware de uma interface de entrada utilizando uma chave mecânica e seu sinal elétrico de saída enfatizando o ruído gerado pelo chaveamento.

O ruído (*bounce*) gerado pelo seu chaveamento interfere diretamente no monitoramento por software do acionamento e do desacionamento dessa chave, quando se utiliza as portas. Assim, esse ruído deve ser levado em consideração, quando da elaboração da rotina de monitoração do acionamento dessa chave mecânica.

Dessa maneira, a estratégia de implementação de uma rotina que monitora o acionamento e o desacionamento de uma chave mecânica é a seguinte:

- verificar continuamente a ocorrência de um acionamento ou de um desacionamento;
- uma vez verificado que ocorreu um acionamento ou um desacionamento, deve-se esperar um tempo T ($T > T_{bounce}$), utilizando-se uma rotina de tempo por software;
- passado o tempo T , deve-se verificar novamente se o acionamento ou o desacionamento realmente ocorreu (muitas vezes, o acionamento é falso), ou seja, é preciso confirmar o acionamento ou o desacionamento. Em caso positivo, a rotina deve deixar o programa prosseguir para iniciar novas atividades. Em caso negativo, isso significa que a chave não foi realmente acionada (essa situação é facilmente simulada por um operador, por meio do posicionamento de uma chave em seu limite de acionamento) e, assim, a rotina deve voltar para o primeiro passo.

Outra forma de eliminar o ruído de uma chave mecânica é por meio da utilização de hardware adicional, por exemplo, utilizando um monoestável ou portas *NAND* configuradas como *flip-flop* do tipo RS, ligados ao pino de saída da interface, como na Figura 6.7 apresentada anteriormente.

Um exemplo de rotina para monitorar o acionamento de uma chave mecânica qualquer, levando-se em consideração o ruído (*bounce*), é apresentado, a seguir, na Figura 6.8, que também mostra o programa-fonte da mesma rotina, porém, já incluindo a rotina de tempo. A constante 1 e a constante 2 podem assumir valores de 00H a FFH.

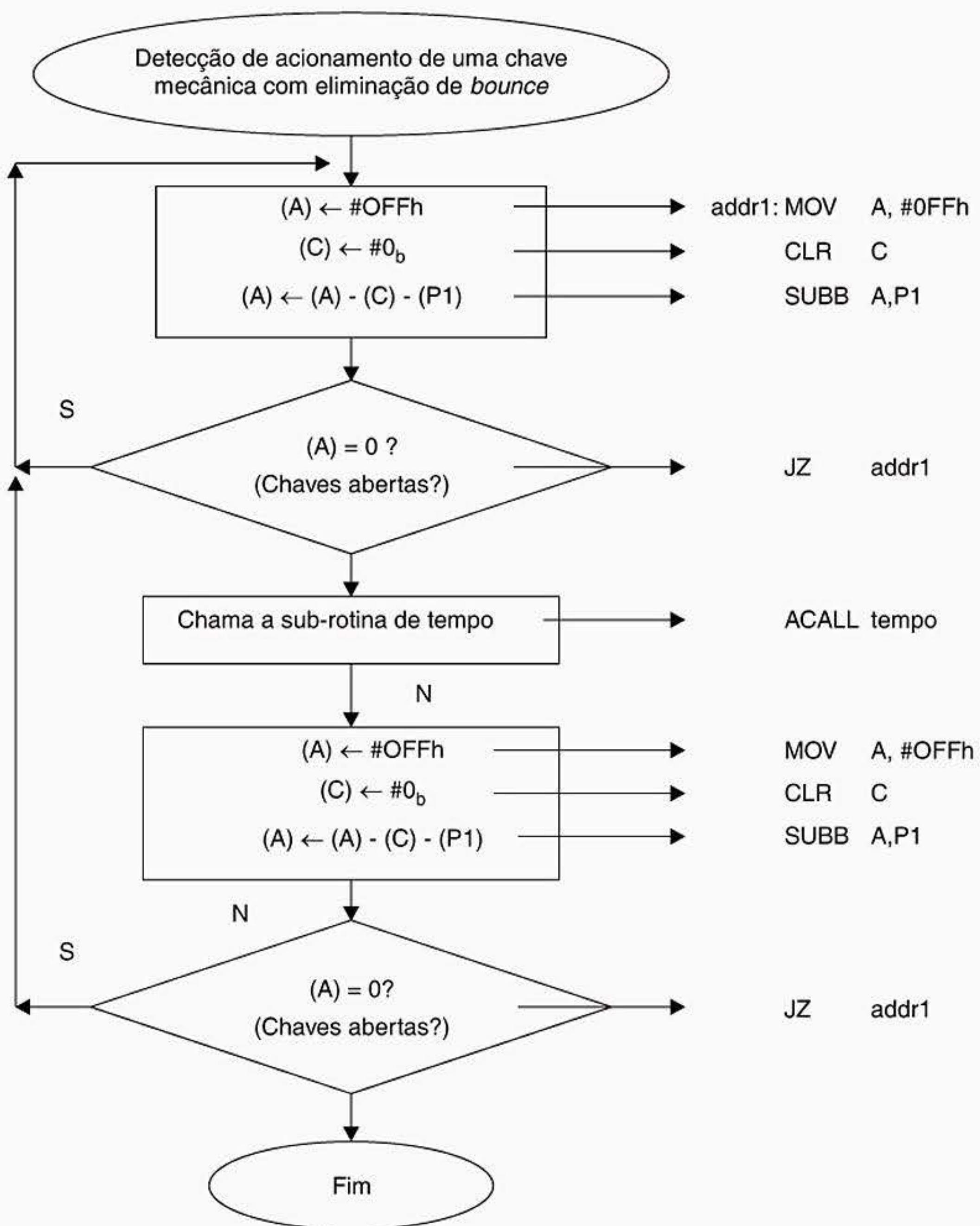


Figura 6.8(a) Fluxograma de uma rotina que monitora o acionamento de uma chave mecânica qualquer com eliminação de ruído (*bounce*).

WAIT:	MOV	A,#0FFh	; Aguarda o acionamento de uma porta
	CLR	C	
	SUBB	A,P1	
	JZ	WAIT	
	MOV	R0,#constante1	; Rotina de atraso de tempo
ATRAS:	MOV	R1,#constante2	
	DJNZ	R1,\$	
	DJNZ	R0, ATRAS	
	MOV	A,#0FFh	; Confirma o fechamento
	CLR	C	
	SUBB	A,P1	
	JZ	WAIT	; Foi algum ruído do acionamento, reinicia o processo ; de detecção de acionamento
	END		; Confirmado o acionamento

Figura 6.8(b) Programa-fonte de uma rotina que monitora o acionamento de uma chave mecânica qualquer com eliminação de ruído (*bounce*).

6.2.4 – Como contar o número de acionamentos e desacionamentos de uma chave qualquer: o programa a seguir conta o número de acionamentos e desacionamentos do bit de uma porta qualquer.

Na Figura 6.9, são apresentados o fluxograma e o programa-fonte da sub-rotina que monitora o desacionamento de uma chave mecânica qualquer.

Na Figura 6.10, é apresentado o programa estruturado (fluxograma e programa-fonte) utilizando-se um membro da família de microcontroladores da Intel, que conta o número de acionamentos e desacionamentos de uma chave mecânica qualquer, com a eliminação do ruído de chaveamento (*bounce*). Repare que foram utilizadas duas sub-rotinas:

- sub-rotina que monitora o acionamento;
- sub-rotina que monitora o desacionamento.

Cada vez que uma chave é acionada e desacionada, é adicionada uma unidade ao conteúdo do registrador R2 (inicialmente, ele vale zero). Repare também que é preciso transformar a rotina de monitoramento de acionamento de uma chave qualquer com eliminação de ruído de chaveamento (Figura 6.8) em sub-rotina. Isso é feito simplesmente trocando-se Fim por RET.

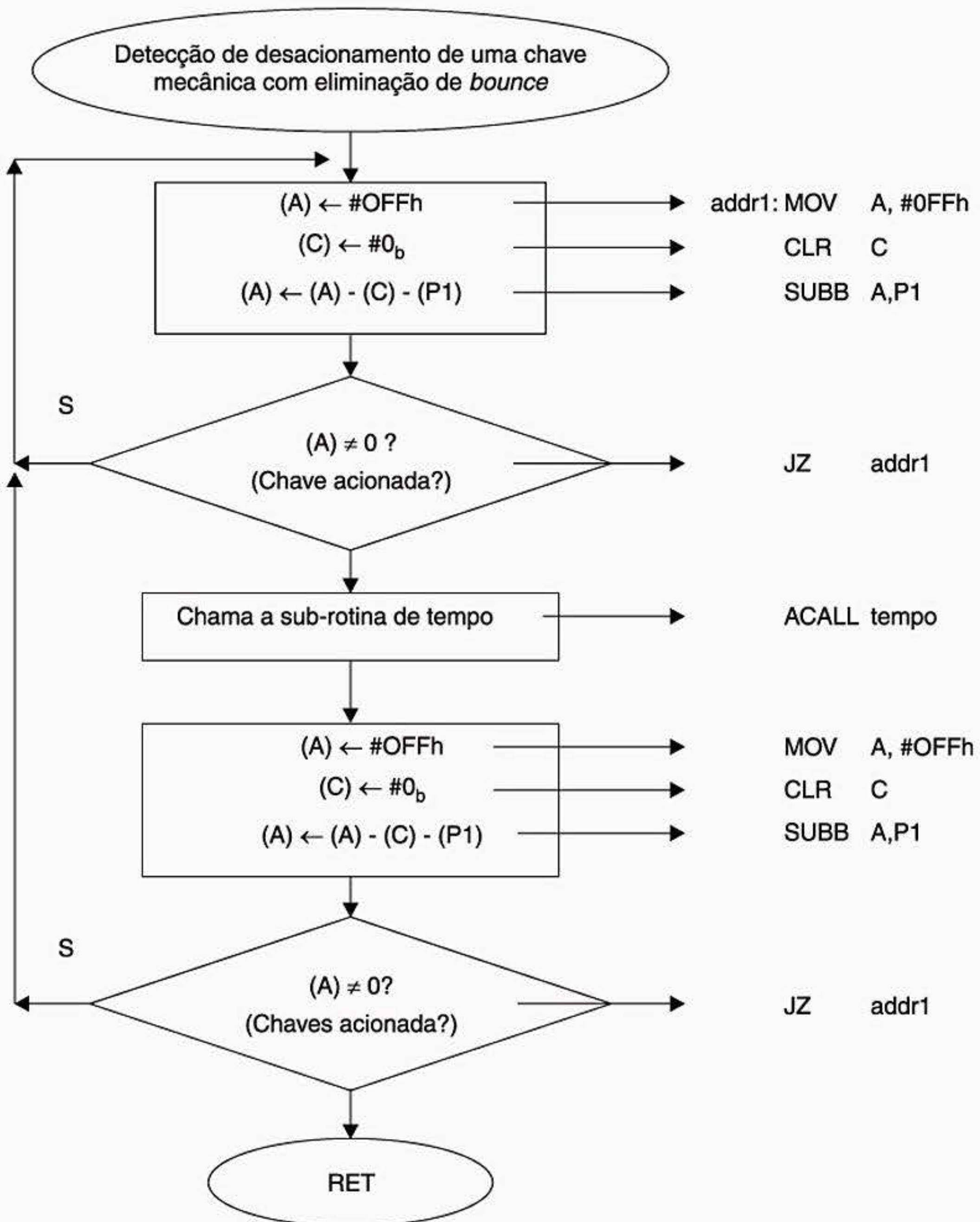


Figura 6.9 Fluxograma e programa-fonte de uma rotina que monitora o desacionamento de uma chave mecânica qualquer com eliminação de ruído (*bounce*).

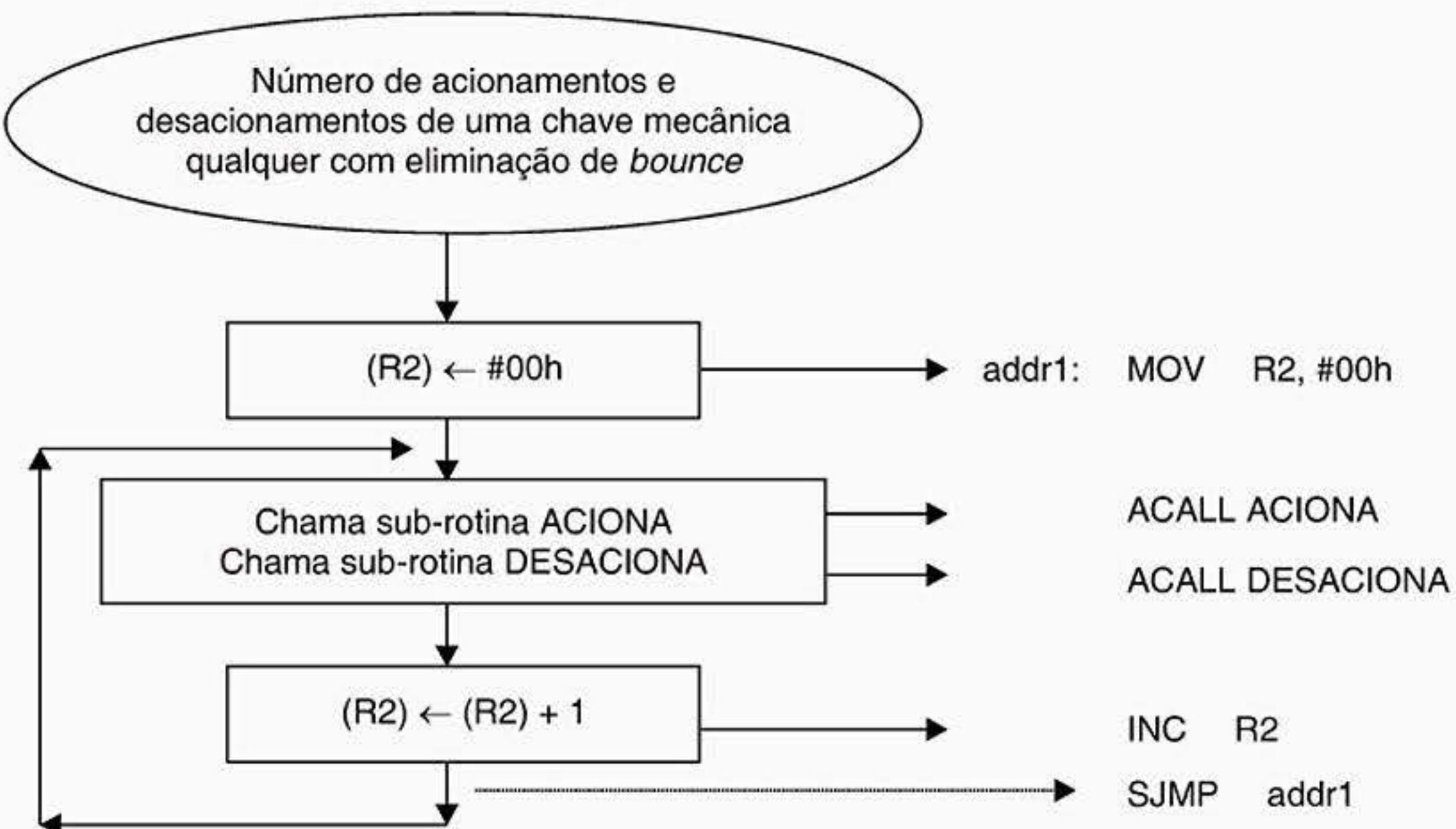


Figura 6.10 Fluxograma e programa-fonte que conta o número de acionamentos e desacionamentos de uma chave mecânica qualquer com eliminação de ruído (*bounce*).

A seguir, é apresentado apenas o programa-fonte que conta o número de acionamentos e desacionamentos sem utilizar sub-rotinas.

WAIT1:	MOV R2,#00h	
	MOV A,#0FFh	; Aguarda o acionamento de uma porta
	CLR C	
	SUBB A,P1	
	JZ WAIT1	
	MOV R0,#constante1	; Rotina de atraso de tempo
ATRAS:	MOV R1,#constante2	
	DJNZ R1,\$	
	DJNZ R0, ATRAS	
	MOV A,#0FFh	; Confirma o acionamento
	CLR C	
	SUBB A,P1	
	JZ WAIT1	
WAIT2:	MOV A,#0FFh	; Aguarda o desacionamento do bit da porta
	CLR C	
	SUBB A,P1	
	JNZ WAIT2	
	MOV R0,#constante1	; Rotina de atraso de tempo
ATRAS:	MOV R1,#constante2	
	DJNZ R1,\$	
	DJNZ R0, ATRAS	
	MOV A,#0FFh	; Confirma o desacionamento
	CLR C	
	SUBB A,P1	
	JNZ WAIT2	
	INC R2	; Conta quantos acionamentos e desacionamentos ocorreram
	SJMP WAIT1	

Exercício resolvido

1 - Apresente apenas o programa-fonte que identifica o número da chave acionada.

Solução:

```

WAIT1:    MOV     A,#0FFh      ; Aguarda o acionamento de uma porta
          CLR     C
          SUBB   A,P1
          JZ      WAIT1
          MOV     R0,#constante1 ; Rotina de tempo
ATRAS:    MOV     R1,# constante2
          DJNZ   R1,$
          DJNZ   R0, ATRAS
MOV       A,#0FFh      ; Confirma acionamento
          CLR     C
          SUBB   A,P1
          JZ      WAIT1
          MOV     R2,#00h      ; Identificação do número do bit acionado (val. inic.=0)
          MOV     A,P1
ROTA:    RRC     A      ; Rotaciona para a direita com flag carry-bit
          JNC     OK      ; Se (C) = 0 (o ‘bit’ foi acionado) => fim, caso contrário
          INC     R2      ; incrementa o número do bit (0 a 7)
          SJMP   ROTA      ; esse bit ainda não foi acionado
OK:      END

```

Exercícios e questões propostos

1 - Considerando o exercício resolvido 1:

- obtenha o fluxograma a partir do programa-fonte;
- faça o fluxograma e o programa-fonte de maneira estruturada (utilizando sub-rotinas).

2 - Faça os seguintes fluxogramas e programas-fonte estruturados, em Assembly, utilizando um dos membros da família de microcontroladores MCS-51, considerando que, à porta 0, está ligada uma interface de entrada com um *dip switch* de oito chaves (1: chave aberta e 0: chave fechada), e que, à porta 1, está ligado um conjunto de oito leds (lógica positiva: 0 lógico desativa led e 1 lógico aciona led).

- 2.1 - Faça um programa estruturado que mostre um contador binário decrescente nos leds, sempre que a chave 2 for acionada (considerar o *bounce*). A informação sobre a contagem deve ser mostrada durante um período de 0,6 segundo no conjunto de leds.
- 2.2 - Faça um programa estruturado que mostre um contador decimal crescente nos leds, sempre que as chaves 2 e 4 forem acionadas (considerar o *bounce*) e desacionadas. A informação da contagem deve ser mostrada durante um período de 0,8 segundo no conjunto de leds em modo piscante.
- 2.3 - Faça um programa estruturado que mostre um contador de números pares crescentes nos leds, sempre que a chave 3 ou 4 for acionada (considerar o *bounce*) e desacionada. A informação sobre a contagem deve ser mostrada durante um período de 0,1 segundo no conjunto de leds.
- 2.4 - Faça um programa estruturado que, sempre que a informação no *dip switch* for igual a AAh, calcule a quantidade de números menores e iguais a 36h do buffer de memória, cujo endereço inicial é 40h e o final é 60h. O resultado deve 'piscar' 10 vezes com uma freqüência de 0,5 segundo no conjunto de leds.
- 2.5 - Calcule a quantidade de números ímpares de um buffer de memória cujo endereço inicial é 30h e o endereço final é 55h, sempre que as chaves ligadas à porta 0 tiverem o padrão 6Dh. A quantidade de números positivos deve ficar piscando (a cada 0,9 segundo) nos leds ligados à porta 1.
- 2.6 - Faça um programa que mostre as seguintes informações no conjunto de leds, por 0,3 segundo, sempre que as chaves 5 e 7 forem acionadas e que a chave 7 estiver desacionada.

led 7	led 6	led 5	led 4	led 3	led 2	led 1	led 0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	1	0	0
0	0	0	0	1	0	0	0
0	0	0	1	0	0	0	0
0	0	1	0	0	0	0	0
0	1	0	0	0	0	0	0
1	0	0	0	0	0	0	0



2.7 – Faça um programa que mostre as seguintes informações no conjunto de leds, por 0,25 segundo, sempre que a chave 3 estiver acionada e a chave 7 estiver desacionada.

led 7	led 6	led 5	led 4	led 3	led 2	led 1	led 0
0	0	0	1	1	0	0	0
0	0	1	0	0	1	0	0
0	1	0	0	0	0	1	0
1	0	0	0	0	0	0	1



capítulo

7

Os TIMERS/CONTADORES

7.1 Objetivos

- ❖ Apresentar as interrupções da família de microcontroladores MCS-51
- ❖ O nível de prioridade das interrupções
- ❖ Os *timers/contadores* da família de microcontroladores MSC-51
- ❖ Os modos de programação dos *timers/contadores*
- ❖ Projetos utilizando os *timers/contadores*

7.2 Introdução teórica

O estudo detalhado do sistema de interrupções da família de microcontroladores MCS-51 se faz necessário antes do estudo dos *timers/contadores*.

7.2.1 - Interrupções: a família de microcontroladores MCS-51 possui cinco fontes de interrupção (seis, no caso dos microcontroladores 8x32/8x52).

Define-se interrupção como uma solicitação ou requisição feita ao microprocessador para o processamento de uma sub-rotina específica adicional, além das sub-rotinas requisitadas pelo programa principal. Essas sub-rotinas devem estar alocadas em posições de memória de programa em endereços específicos, definidos pelo fabricante.

As interrupções, na família de microcontroladores MCS-51, são geradas por sinais elétricos vindos de dispositivos internos (*timers/contadores* ou canal de comunicação serial) ou de dispositivos externos (interfaces eletrônicas que monitoram uma determinada variável de controle, como temperatura, pressão, velocidade etc., e que estão ligadas às entradas de interrupções externas 0 e 1 do microcontrolador). Assim, sempre que um dispositivo necessitar de

um atendimento por parte do microprocessador, ele vai gerar um sinal elétrico digital de interrupção (0 lógico), por meio de hardware, para o microprocessador. Após o microprocessador reconhecer a solicitação de interrupção, ele finalizará a execução da instrução que estava sendo executada; e, se tal interrupção estiver habilitada, o mesmo atenderá a essa solicitação de interrupção. Isso é feito por meio da execução automática, por hardware, de uma instrução de chamada à sub-rotina "*LCALL endereço predefinido pelo fabricante*", pelo microprocessador. A sub-rotina de atendimento a essa interrupção, obviamente, deverá ter sido armazenada no respectivo endereço de memória de programa pelo programador. Cada fonte de interrupção tem um endereço de memória predefinido pelo fabricante e, assim, o microprocessador executará uma sub-rotina de atendimento à interrupção a partir de tal endereço. Toda sub-rotina de atendimento a uma fonte de interrupção deve apresentar, como característica fundamental, terminar com a instrução RETI (*Return Interruption*). Assim, ao executar a instrução RETI, o microprocessador retornará o processamento para uma instrução imediatamente após aquela instrução quando foi detectada a solicitação da interrupção.

Repare que o atendimento a uma fonte de interrupção funciona de maneira similar a uma chamada à sub-rotina. A única diferença é que, para chamar uma sub-rotina por software, a instrução "*LCALL endereço inicial da subrotina*" deve fazer parte do programa-fonte, e, para o microprocessador atender à solicitação de uma fonte de interrupção, é executada, por hardware, uma instrução "*LCALL endereço predefinido pelo fabricante*", que não faz parte do programa-fonte.

A instrução "*LCALL endereço predefinido pelo fabricante*" tem uma característica especial. Ela não calcula o endereço da próxima instrução a ser executada, como a instrução "*LCALL endereço inicial da subrotina*". Essa instrução simplesmente armazena o endereço da próxima instrução a ser executada na pilha quando foi gerada pela instrução em que foi reconhecida e atendida a interrupção. A Tabela 7.1 exemplifica o que foi exposto anteriormente.

Tabela 7.1 Descrição da instrução LCALL vetor de interrupção.

Instrução	Bytes	Ciclos	Codificação	Representação simbólica	Instrução	Representação simbólica
LCALL addr ₁₆ (sub- rotina)	3	2	0001 0010 addr _{15..8} addr _{7..0}	(PC) ← (PC) + 3 (SP) ← (SP) + 1 ((SP)) ← (PC _{7..0}) (SP ← (SP) + 1 ((SP)) ← (PC _{15..8}) (PC) ← addr ₁₆	LCALL endereço predefinido pelo fabricante (interrupção)	(PC) ← (PC) + 3 (SP) ← (SP) + 1 ((SP)) ← (PC _{7..0}) (SP ← (SP) + 1 ((SP)) ← (PC _{15..8}) (PC) ← addr ₁₆

Repare também que, para entender o processo de atendimento a uma fonte de interrupção, é necessário apenas compreender o princípio de funcionamento de uma chamada à sub-rotina, ou seja, mais especificamente, entender como as instruções LCALL e RET funcionam.

Existem duas entradas de interrupção externas (INT0\ e INT1\), que são ativadas por nível ou por borda de descida, dependendo dos valores dos bits IT0 e IT1 do registrador TCON. Os *flags* que geram essas interrupções são os bits IE0 e IE1 de TCON.

Quando uma interrupção externa é gerada, o *flag* que a gerou é zerado por hardware se a sub-rotina de atendimento a essa interrupção é vetorizada (endereçada), 'se e somente se' a interrupção foi programada para ser sensível à borda de descida. Caso a interrupção tenha sido programada para ser sensível ao nível, então a fonte de requisição externa controlará o *flag* requisitante da interrupção e deverá ser zerado da sub-rotina de atendimento à fonte de interrupção.

Existem mais duas fontes de interrupção geradas pelo excesso de contagem crescente (*overflow*) dos registradores de contagem dos *timers/contadores* 1 e 2. TF0 e TF1 são *setados* pelo estouro dos registradores de contagem (111...111 → 000...000), exceto quando o *timer/contador* 0 operar no modo 3.

Quando ocorre uma interrupção de *timer/contador*, o *flag* que a gerou é zerado por hardware se a sub-rotina de atendimento a essa fonte de interrupção for vetorizada (endereçada).

Existe mais uma fonte de interrupção, aquela referente ao canal de comunicação da porta serial, que é gerada pela lógica OR entre RI e TI. Nenhum desses *flags* é zerado por hardware quando a rotina de atendimento a essa fonte de interrupção é vetorizada (endereçada). De fato, a rotina de serviço dessa fonte de interrupção normalmente terá de determinar se foi RI ou TI que gerou a interrupção, e o bit deverá ser zerado por meio de software.

Todos os bits que geram interrupções podem ser *setados* ou *resetados* por software, com o mesmo resultado com que ele é *setado* e *resetado* por hardware. Isso significa que as interrupções podem ser geradas por software ou que as interrupções pendentes também podem ser canceladas por software.

Cada uma dessas fontes de interrupção pode ser individualmente habilitada ou desabilitada por meio de alguns bits do registrador de função especial IE (0: desabilita a fonte de interrupção; 1: habilita a fonte de interrupção). Ele também contém um bit desabilitador global, chamado de EA, que é capaz de desabilitar todas as interrupções de uma só vez.

1 - *Estrutura do nível de prioridade das interrupções:* cada fonte de interrupção pode ser individualmente programada para ter um dos dois níveis de prioridade. Isso é feito por meio do registrador de função especial IP – *Interrupt Priority* (0: prioridade menor; 1: prioridade maior).

(IP) =	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
	–	–	PT2	PS	PT1	PX1	PT0	PX0

Bit de prioridade = 1 \Rightarrow atribui alta prioridade;

Bit de prioridade = 0 \Rightarrow atribui baixa prioridade;

Símbolo	Posição	Função
–	IP.7	Reservada
–	IP.6	Reservada
PT2	IP.5	Bit de prioridade da fonte de interrupção do <i>timer/contador 2</i>
PS	IP.4	Bit de prioridade da fonte de interrupção do canal de comunicação serial
PT1	IP.3	Bit de prioridade da fonte de interrupção do <i>timer/contador 1</i>
PX1	IP.2	Bit de prioridade da fonte de interrupção externa 1
PT0	IP.1	Bit de prioridade da fonte de interrupção do <i>timer/contador 0</i>
PX0	IP.0	Bit de prioridade da fonte de interrupção externa 0

- ◆ uma fonte de interrupção de baixa prioridade pode ser interrompida por uma fonte de interrupção de alta prioridade;
- ◆ uma fonte de interrupção de alta prioridade não pode ser interrompida por uma fonte de interrupção de baixa prioridade;
- ◆ se duas fontes de interrupção com prioridades diferentes se manifestarem ao mesmo tempo, a que tiver maior nível de prioridade será atendida;
- ◆ se duas fontes de interrupção com a mesma prioridade se manifestarem ao mesmo tempo, uma seqüência de varredura (*polling*) de atendimento determinará qual fonte de interrupção será atendida, conforme a Tabela 7.2.

Tabela 7.2 Fonte de interrupção e nível de prioridade.

Item	Fonte de interrupção	Nível dentro da mesma prioridade	
1	IE0	(+++++)	Maior
2	TF0	(++++)	
3	IE1	(+++)	
4	TF1	(++)	
5	RI + TI	(+)	
6	TF2 + EXF2		Menor

2 - *O registrador de habilitador de interrupção IE (endereçável por bit):* se o bit for colocado por software em 0 lógico, a fonte de interrupção correspondente será desabilitada, caso contrário, será habilitada.

(IE) =	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
	EA	----	ET2	ES	ET1	EX1	ET0	EX0

Símbolo	Posição	Função
EA	IE.7	Desabilitador geral de interrupções (0: nenhuma interrupção é reconhecida; 1: cada fonte de interrupção é individualmente habilitada ou desabilitada por <i>setar</i> ou zerar seu bit habilitador)
-	IE.6	Reservada
ET2	IE.5	Habilita/desabilita a fonte de interrupção de <i>overflow</i> ou captura a partir do <i>timer 2</i>
ES	IE.4	Habilita/desabilita a fonte de interrupção do canal de comunicação serial
ET1	IE.3	Habilita/desabilita a fonte de interrupção de <i>overflow</i> do <i>timer 1</i>
EX1	IE.2	Habilita/desabilita a fonte de interrupção externa 1
ET0	IE.1	Habilita/desabilita a fonte de interrupção de <i>overflow</i> do <i>timer 0</i>
EX0	IE.0	Habilita/desabilita a fonte de interrupção externa 0

Observação: não é recomendada a utilização de bits reservados (IE.6), pois novos produtos poderão utilizar tais bits.

3 - *Como as interrupções são manipuladas:* os *flags* relativos a cada fonte de interrupção são mostrados em S5P2 de todo o ciclo de máquina. As amostragens são interpretadas (*polled*) durante o ciclo de máquina seguinte ao qual o sistema de interrupção gera um LCALL por hardware, para executar a rotina de atendimento apropriada à fonte de interrupção.

Qualquer uma das condições a seguir bloqueará a geração do LCALL, gerada por hardware:

- ◆ uma interrupção de nível igual ou superior já está sendo processada;
- ◆ o ciclo de interpretação atual (*polling*) não está no final da execução da instrução que está sendo processada. Isso garante que a instrução que está em progresso seja concluída antes de vетorizar qualquer rotina de atendimento à fonte de interrupção;
- ◆ a instrução que está sendo processada é RETI ou a instrução que escreve no registrador IE ou IP.

O LCALL é gerado por hardware e coloca o conteúdo do registrador *Program Counter* (PC) na pilha (mas não grava o conteúdo do registrador de função especial PSW, isso deve ser feito por software, pela rotina de atendimento da fonte de interrupção), e recarrega o conteúdo do registrador *Program Counter* (PC) com um endereço que depende da fonte de interrupção que está sendo vетorizada de acordo com a Tabela 7.3.

Tabela 7.3 Fonte de interrupção e endereço vetor.

Fonte de interrupção	Nome da fonte de interrupção	Endereço do vetor
RESET	Reset	0000h
IE0	Fonte de interrupção externa 0	0003h
TF0	Fonte de interrupção do <i>timer/contador</i> 0	000Bh
IE1	Fonte de interrupção externa 1	0013h
TF1	Fonte de interrupção do <i>timer/contador</i> 1	001Bh
RI + TI	Fonte de interrupção do canal de comunicação serial	0023h
TF2 + EXF2	Fonte de interrupção do <i>timer/contador</i> 2 + externa 2	002Bh

Quando a fonte de interrupção é vetorizada e atendida, o programa executa a sub-rotina armazenada no endereço da memória de programa, indicado na Tabela 7.3, até que a instrução RETI seja processada.

A instrução RETI informa ao microprocessador que a sub-rotina de atendimento à fonte de interrupção não está mais sendo executada, depois ela lê os dois bytes que estão no topo da pilha e recarrega o conteúdo do registrador *Program Counter* (PC) com o endereço de retorno ao programa principal. Dessa maneira, a execução do programa continua a partir da instrução na qual a interrupção foi atendida.

Para utilizar qualquer interrupção do MCS-51, as seguintes etapas devem ser seguidas:

- ◆ *setar* para 1 lógico o conteúdo do bit EA do registrador *Interrupt Enable IE*;
- ◆ *setar* para 1 lógico o conteúdo dos bits habilitadores de fontes de interrupção, individualmente;
- ◆ iniciar as rotinas de atendimento às fontes de interrupção no endereço correspondente ao vetor da interrupção, conforme tabela anterior;
- ◆ para as interrupções externas, *setar* os pinos INT0/ e INT1/ (P3.2 e P3.3) para 1 lógico e, caso a fonte de interrupção seja ativada por nível ou transição, os bits IT0 e IT1 no registrador TCON podem precisar ser *setados* para 1 lógico ($ITX = 0 \Rightarrow$ ativado por nível; $ITX = 1 \Rightarrow$ ativado por transição).

7.2.2 - Timers/Contadores: o 8051 tem dois registradores dos *timers/contadores* de 16 bits (*timer/contador* 0 e 1). O 8052 tem mais um (*timer/contador* 2). Todos os três podem ser configurados para operar como *timer* ou como contador de eventos.

Na função *timer*, o registrador de contagem é adicionado a todo ciclo de máquina. Assim, pode-se dizer que ele é um contador de ciclos de máquina. Como o ciclo de máquina possui doze períodos de *clock* do oscilador, a razão de contagem é 1/12 da freqüência do cristal.

Na função contador, o registrador é adicionado em resposta a uma transição de 1 para 0 em seu pino de entrada externa correspondente, T0 ou T1 no 8051 e T2 no 8052. Como ela leva dois ciclos de máquina (24 períodos de *clock* do oscilador), a razão de contagem máxima é 1/24 da freqüência do oscilador.

Não existe restrição alguma no período ativo do sinal de entrada externa, mas para assegurar que um determinado nível seja amostrado ao menos uma vez antes de se modificar, ele deveria ser mantido, pelo menos, por um ciclo de máquina completo.

Na seleção de *timer* ou contador, os *timers* 0 e 1 têm quatro modos de operação, enquanto o *timer* 2 tem apenas três modos de operação: captura, recarregamento automático e gerador de *baud rate* (razão de recepção/transmissão de dados na comunicação serial).

- a) *Timer 0 e Timer 1*: ambos são apresentados no 8051 e 8052. A função de *timer* ou contador é selecionada pelo bit de controle C/T_{barra} do registrador de função especial TMOD.

Esses dois *timers*/contadores têm quatro modos de operação selecionados pelos bits (M1,M0) do registrador de função especial TMOD. Os modos 0, 1 e 2 são iguais para ambos os *timers*/contadores. O modo 3 é diferente.

a.1) Registrador de controle de modo do timer/contador:

TMOD =	Timer 1				Timer 0			
	GATE	C/T\	M1	M0	GATE	C/T\	M1	M0

Símbolo	Posição	Função
GATE	IE.7	Controle de acionamento: 0: O <i>timer</i> X é habilitado sempre TRX = 1 (contagem controlada por software); 1: O <i>timer</i> /contador é habilitado somente enquanto o pino INTX = 1 e TRX (em TCON) = 1 (contagem controlada por software e hardware);
C/T _{barra}	IE.6	Seletor de <i>timer</i> ou contador: 0: define operação <i>timer</i> (entrada de <i>clock</i> do sistema interno = $f_{cristal}/12$) 1: define operação contador (entrada de <i>clock</i> por meio do pino externo TX).
M1	IE.5	Bits programadores de modo de operação.
M0	IE.4	Bits programadores de modo de operação.

M1	M0	Modo de operação
0	0	0: timer com 13 bits de contagem ($THX_s = xxxx\ xxxx$ $TLX_s = \overset{7}{***}x\ \overset{4}{***}0\ 0000$) contagem inicial: ($0000\ 0000$ $\overset{3}{***}0\ 0000$) contagem final: ($1111\ 1111$ $\overset{2}{***}1\ 1111$)
0	1	1: <i>timer/contador de 16 bits</i> ($THX_s\ TLX_s$) ₁₆
1	0	2: <i>timer/contador de 8 bits recarregamento automático.</i> A cada interrupção, o registrador de contagem (TLX) é recarregado automaticamente com o valor inicializado no registrador de recarregamento (THX) Registrador de Contagem Registrador de Recarregamento $(TLX_s = xxxx\ xxxx) \leftarrow (THX_s = \text{valor a ser recarregado no registrador de contagem})$
1	1	3: <i>timer 0:</i> - TL0 é um <i>timer/contador de 8 bits</i> , controlado pelos bits de controle do <i>timer 0</i> . - TH0 é simplesmente um <i>timer de 8 bits</i> , controlado pelos bits de controle <i>timer 1</i> . <i>timer 1:</i> parado.

Observação: X = 0 ou 1.

7.3 Modos de operação

Os *timers/contadores* podem trabalhar em quatro modos de operação diferentes (modo 0 a modo 3):

- a) *Modo 0:* é um *timer/contador de 13 bits*, sendo 8 bits do registrador THX e apenas 5 bits do registrador TLX (Figura 7.1).

Quando o registrador de contagem excede sua contagem (em inglês, se diz *overflow*), ou seja, a contagem vai de 1111 1111 xxx1 1111 \Rightarrow 0000 0000 xxx0 0000, o *flag* de interrupção TFX é *setado*. Se a interrupção do *timer/contador* estiver habilitada, a interrupção será reconhecida e gerada uma instrução LCALL *endereço vetor* (*endereço vetor*: 000Bh para o *timer/contador 0* e 001Bh para o *timer/contador 1*).

A contagem é habilitada para o *timer/contador* somente quando TR1 = 1 e o *Gate* = 0 ou INTXbarra = 1.

Quando *Gate* = 1, o *timer/contador* pode ser controlado pela entrada externa INTXbarra. Uma das aplicações desse tipo de uso dos *timers/contadores* serve para a obtenção das medidas de largura de pulso elétrico, geradas por interfaces de leitores de código de barras, código magnético etc.

A operação em modo 0 é a mesma para ambos os *timers/contadores*.

Existem dois bits diferentes de *Gates*, um para o *timer/contador 1* (TMOD.7) e um para o *timer/contador 0* (TMOD.3).

TR1 é um bit de controle de liga e desliga que está contido no registrador de função especial TCON, como é mostrado a seguir.

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
(TCON) =	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

Símbolo	Posição	Nome e significado
TF1	TCON.7	Flag de overflow do timer/contador 1: setado por hardware no overflow do timer/contador 1. Zerado por hardware quando o processador vetoriza (endereço) a rotina de atendimento à fonte de interrupção do timer/contador 1 (001Bh).
TR1	TCON.6	Bit de controle para ligar/desligar o timer/contador 1. Setado/excluído por software para ativar ou desativar o timer/contador 1.
TF0	TCON.5	Flag de overflow do timer/contador 0: setado por hardware no overflow do timer/contador 0. Zerado por hardware quando o processador vetoriza (endereço) para a rotina de atendimento à fonte de interrupção do timer/contador 1 (000Bh).
TR0	TCON.4	Bit de controle para ligar/desligar o timer/contador 0. Setado/zerado por software para ativar ou desativar o timer/contador 0.
IE1	TCON.3	Flag de detecção de borda de interrupção 1. Setado/zерado por hardware quando uma margem da interrupção externa é detectada. Excluído quando a interrupção é processada.
IT1	TCON.2	Bit de controle do tipo da interrupção 1. Setado/zерado por software para especificar o tipo de detecção da interrupção externa 1, se é por borda de descida ou nível baixo.
IE0	TCON.1	Flag de detecção de borda da interrupção 0. Setado/zерado por hardware quando uma margem da interrupção externa é detectada. Zerado quando a interrupção é processada.
IT0	TCON.0	Bit de controle do tipo da interrupção 0. Setado/zерado por software para especificar o tipo de detecção da interrupção externa 0, se é por borda de descida ou nível baixo.

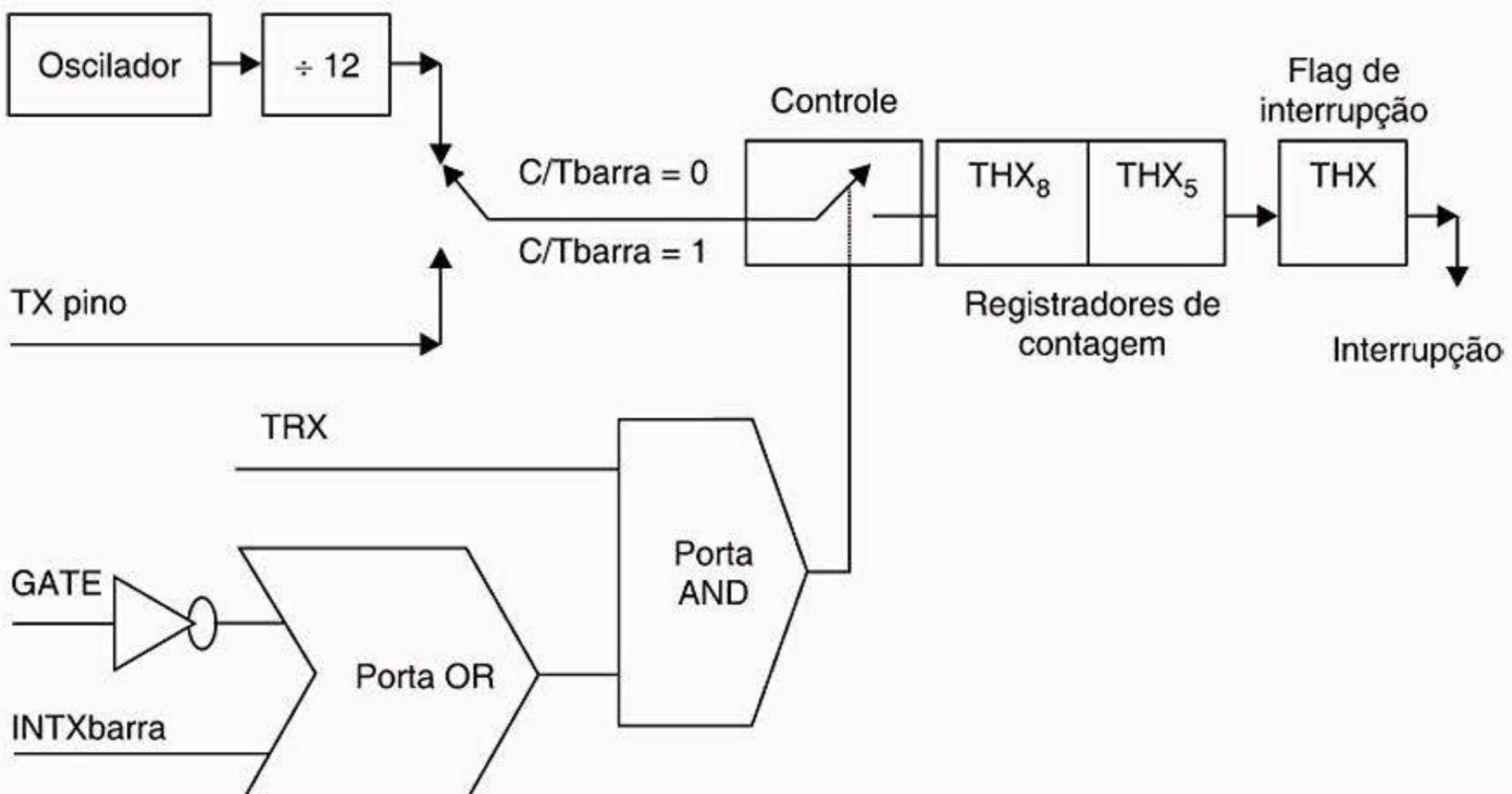


Figura 7.1 Timer/contador no Modo 0 de operação (13 bits de contagem).

- b) *Modo 1:* é um *timer/contador* de 16 bits. Apresenta o mesmo hardware da Figura 7.1, porém o seu registrador de contagem é de 16 bits (8 bits do registrador THX e 8 bits do registrador TLX). Utilizado para gerar tempos maiores.
- c) *Modo 2:* é um contador de 8 bits com recarregamento automático (Figura 7.2).

O *overflow* do registrador de contagem TLX for TFX igual a 1 e também recarrega o registrador de contagem TLX com o conteúdo do registrador de recarregamento automático THX. O registrador de recarregamento automático deve ser definido inicialmente por software com um determinado valor, para gerar a contagem ou o tempo desejado. Quando ocorre o recarregamento, THX não se modifica. O Modo 2 de operação é o mesmo para o *timer/contador* 0 e 1.

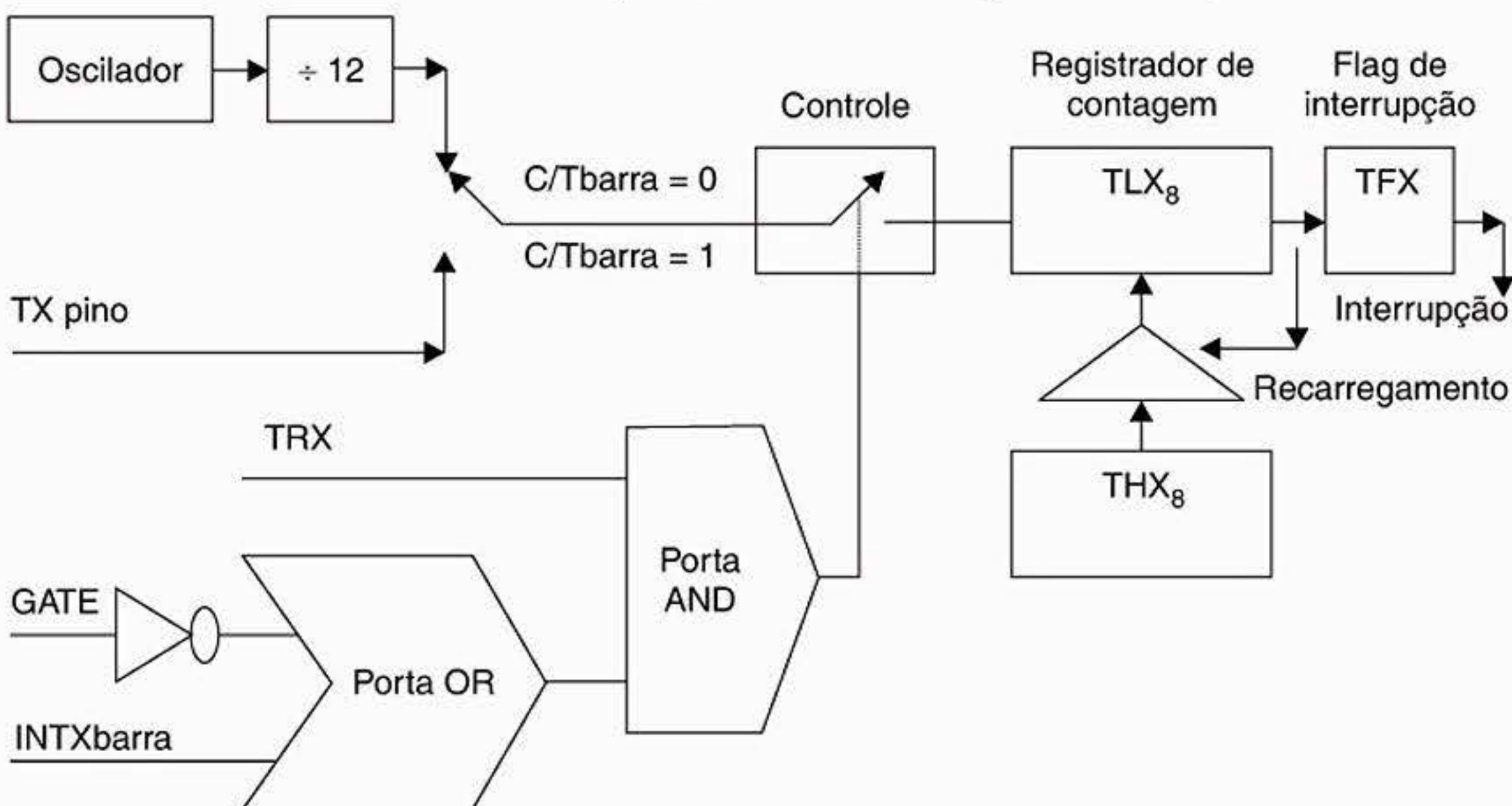


Figura 7.2 *Timer/contador* no Modo 2 de operação (8 bits de contagem).

- d) *Modo 3:* são dois *timers/contadores* de 8 bits, sendo dois registradores de contagem, nos quais o registrador de contagem TL0 é controlado pelos bits de controle do *timer/contador* 0 e o registrador de contagem TH0 é controlado pelo bit de controle do *timer/contador* 1 (TR1) e com freqüência de contagem de $f_{\text{cristal}}/12$ (Figura 7.3).

O *timer/contador* 0, no Modo 3, define TL0 e TH0 como dois registradores de contagem separados.

O *timer/contador* 1, no Modo 3, simplesmente é desabilitado para a contagem. O efeito é o mesmo que fazer TR1 = 0.

O Modo 3 é fornecido para aplicações que exigem um *timer/contador* extra de 8 bits. Com o *timer/contador* 0 no Modo 3, pode parecer que o 8051 tem três *timers/contadores* e que o 8052 tem quatro.

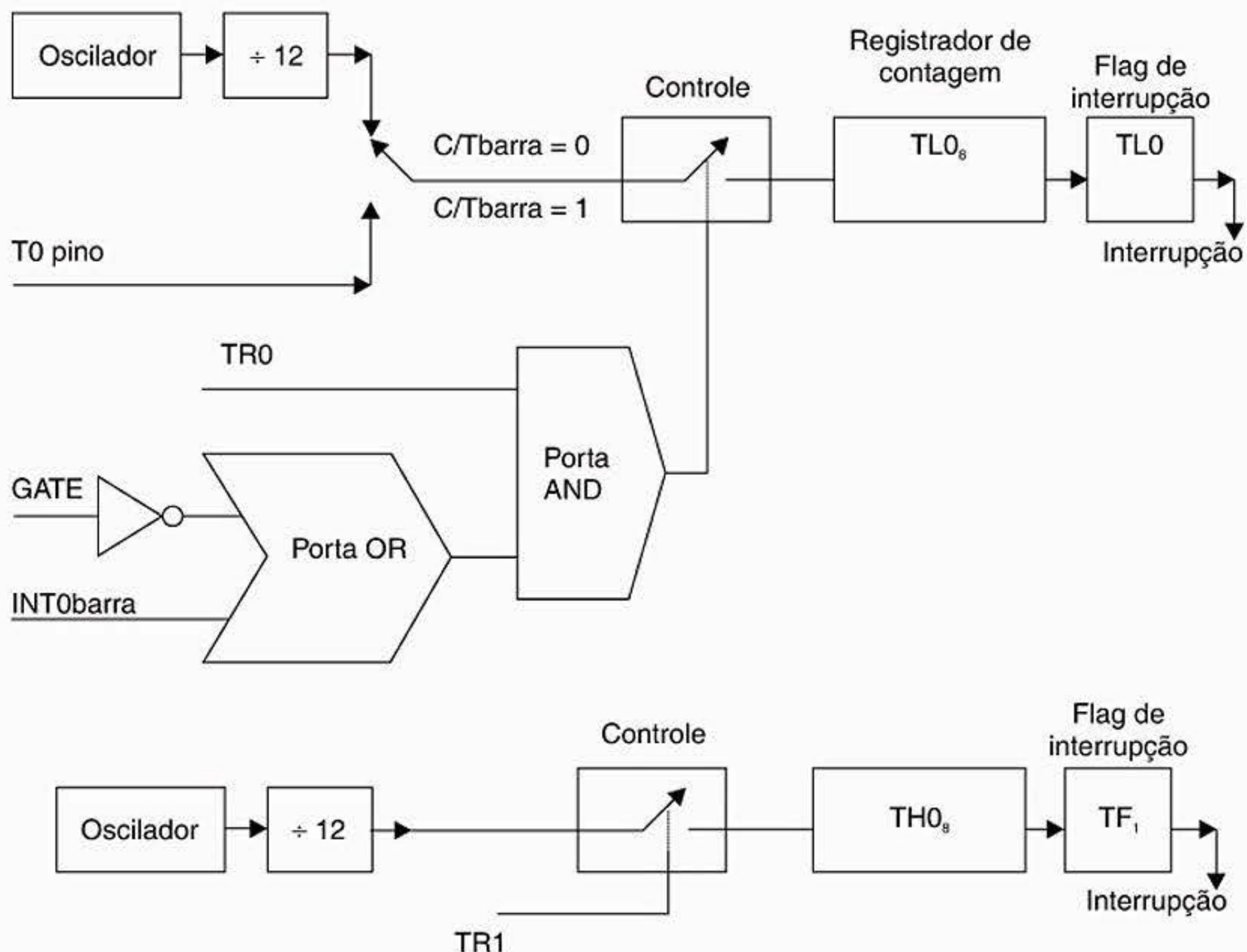


Figura 7.3 Timer/contador no Modo 3 de operação (8 bits de contagem).

7.4 Inicialização (setup) dos *timers/contadores*

As tabelas a seguir fornecem alguns valores para TMOD, sendo que estes podem ser utilizados para inicializar o *timer/contador* 0 em diferentes modos. Considera-se que somente um *timer/contador* está sendo utilizado por vez. Se a idéia for executar os *timers/contadores* 0 e 1 simultaneamente em qualquer modo, o valor em TMOD para o *timer/contador* 0 deve ser calculado por meio da operação lógica OR, com o valor mostrado para o *timer/contador* 1.

Por exemplo, se for desejado executar o *timer/contador* 0 no Modo 1 Gate (controle externo) e o *timer/contador* 1 no Modo 2 como contador, então o valor que deverá ser carregado em TMOD é 69h (09h da primeira tabela OR 60h da última tabela).

Fazendo-se TRX = 1 em TCON os *timers* começam a contar.

a) Timer/contador 0 operando como um *timer*:

TMOD			
Modo	Função do timer/contador 0	Controle interno (observação 1)	Controle externo (observação 2)
0	Timer/contador de 13 bits	00h	08h
1	Timer/contador de 16 bits	01h	09h
2	8 bits - recarregamento automático	02h	0Ah
3	2 timers/contadores de 8 bits	03h	0Bh

b) Timer/contador 0 operando como um contador:

TMOD			
Modo	Função do timer/contador 0	Controle interno (observação 1)	Controle externo (observação 2)
0	Timer/contador de 13 bits	04h	0Ch
1	Timer/contador de 16 bits	05h	0Dh
2	8 bits - recarregamento automático	06h	0Eh
3	2 timers/contadores de 8 bits	07h	0Fh

c) Timer/contador 1 operando como um *timer*:

TMOD			
Modo	Função do timer/contador 0	Controle interno (observação 1)	Controle externo (observação 2)
0	Timer/contador de 13 bits	00h	80h
1	Timer/contador de 16 bits	10h	90h
2	8 bits - recarregamento automático	20h	A0h
3	2 timers/contadores de 8 bits	30h	B0h

d) Timer/contador 1 operando como um contador:

TMOD			
Modo	Função do timer/contador 0	Controle interno (observação 1)	Controle externo (observação 2)
0	Timer/contador de 13 bits	40h	C0h
1	Timer/contador de 16 bits	50h	D0h
2	8 bits - recarregamento automático	60h	E0h
3	2 timers/contadores de 8 bits	70h	F0h

Observação 1: o timer/contador é ligado/desligado por *setar/zerrar* o bit TRX por software.

Observação 2: o timer/contador é ligado/desligado pela transição de 1 para 0 em INTXbarra (P3.2) quando TRX = 1 (controle por hardware).

e) Hardware do timer: esta área mostra os conteúdos dos registradores de cada timer.

Objeto na tela	Símbolo	Tipo de informação	Tamanho
Timer /contador	TH/TL	Hexadecimal	2 bytes
<i>Flags:</i>			
Overflow	TF	Binário	1 bit
Controle para executar	TR	Binário	1 bit
Controle de entrada externa	G	Binário	1 bit
Seletor	T	Binário	1 bit
Controle de modo	M1 M0	Binário	2 bits

f) Hardware de interrupções: esta área mostra o conjunto de hardware de interrupções do foco da CPU:

Objeto na tela	Símbolo	Tipo de informação	Tamanho
Reg. habilitadores:			
Todos	A	Binário	1 bit
Comunicação serial	S	Binário	1 bit
Timers	T0/T1/T2	Binário	1 bit
Interrupções externas	X0/X1	Binário	1 bit
Reg. prioridades			
Comunicação serial	S	Binário	1 bit
Timers	T0/T1/T2	Binário	1 bit
Interrupções externas	X0/X1	Binário	1 bit
<i>Flags int. externas:</i>			
Trigger (acionador)	IT	Binário	1 bit
Borda	IE	Binário	1 bit

Exercícios resolvidos

1 - Faça um programa estruturado (fluxograma e programa-fonte) em Assembly que utilize um dos membros da família de microcontroladores MCS-51 e que seja capaz de executar as seguintes atividades:

1.1 - Programe o *timer*/contador 0 no Modo 0 e no Modo 1. Ambos os *timers*/contadores devem ser inicializados com o valor FF00h.

1.2 - Implemente uma sub-rotina capaz de aumentar em uma unidade o conteúdo da porta 0 e atualize o conteúdo da porta 3 com o complemento um do conteúdo da porta 0. Essa sub-rotina deve ser armazenada no conteúdo do endereço de memória de programa 0080h.

1.3 - Implemente uma rotina de atendimento à fonte de interrupção do *timer*/contador 0, que rotacione o conteúdo da porta 1 um bit para a esquerda. Assim, toda vez que ocorrer uma interrupção gerada pela fonte de interrupção do *timer*/contador 0, o conteúdo da porta 0 será rotacionado em um bit para a esquerda.

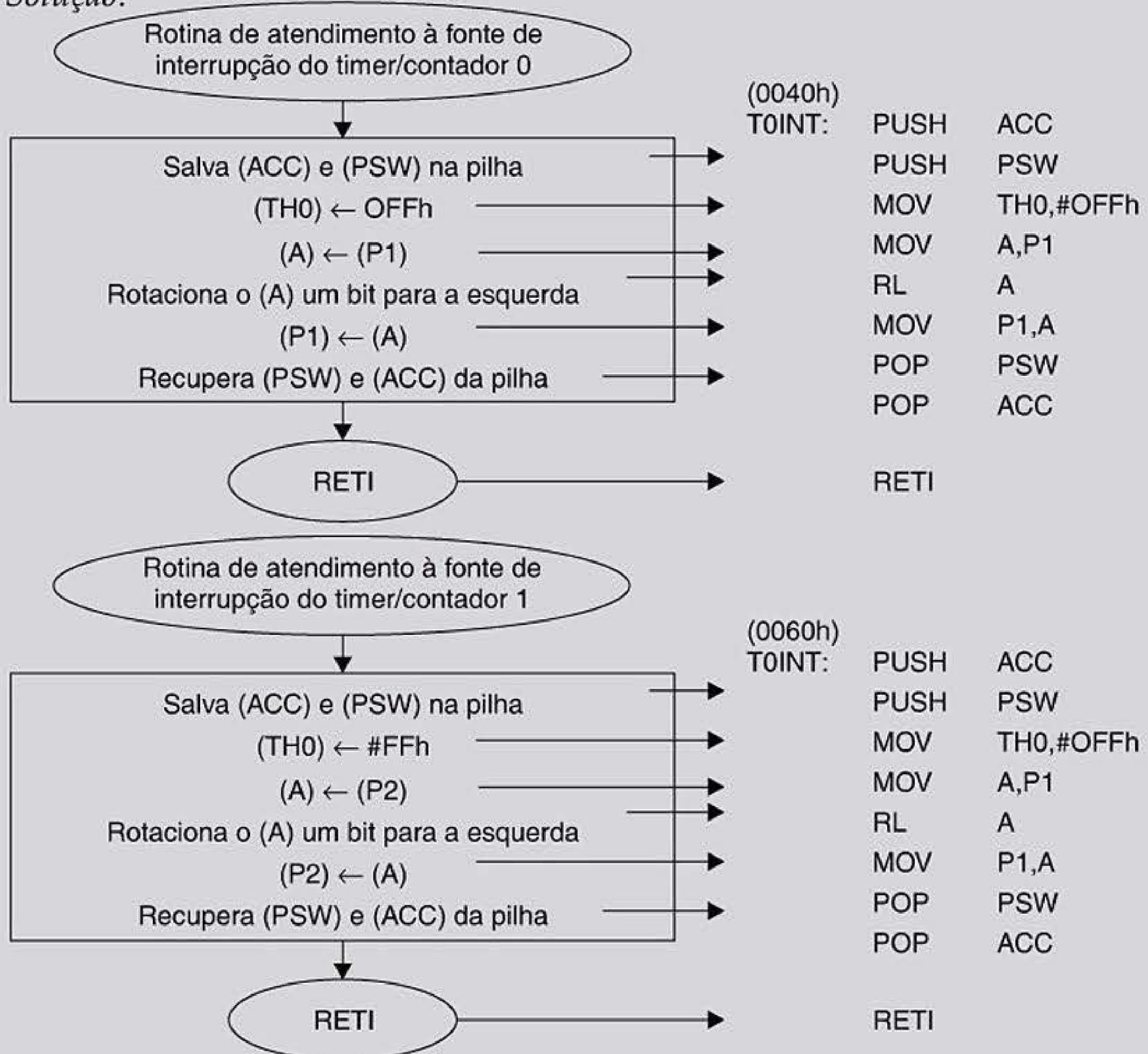
1.4 - Implemente uma rotina de atendimento à fonte de interrupção do *timer/contador 1*, que rotacione o conteúdo da porta 2 um bit para a esquerda. Assim, toda vez que ocorrer uma interrupção gerada pela fonte de interrupção do *timer/contador 1*, o conteúdo da porta 2 será rotacionado um bit para a esquerda.

1.5 - A rotina de atendimento à fonte de interrupção do *timer/contador 0* deve ser escrita no conteúdo do endereço de memória de programa 0040h.

1.6 - A rotina de atendimento à fonte de interrupção do *timer/contador 1* deve ser escrita no endereço de memória de programa 0060h.

1.7 - O programa principal deve ser escrito no conteúdo do endereço de memória de programa 0100h.

Solução:



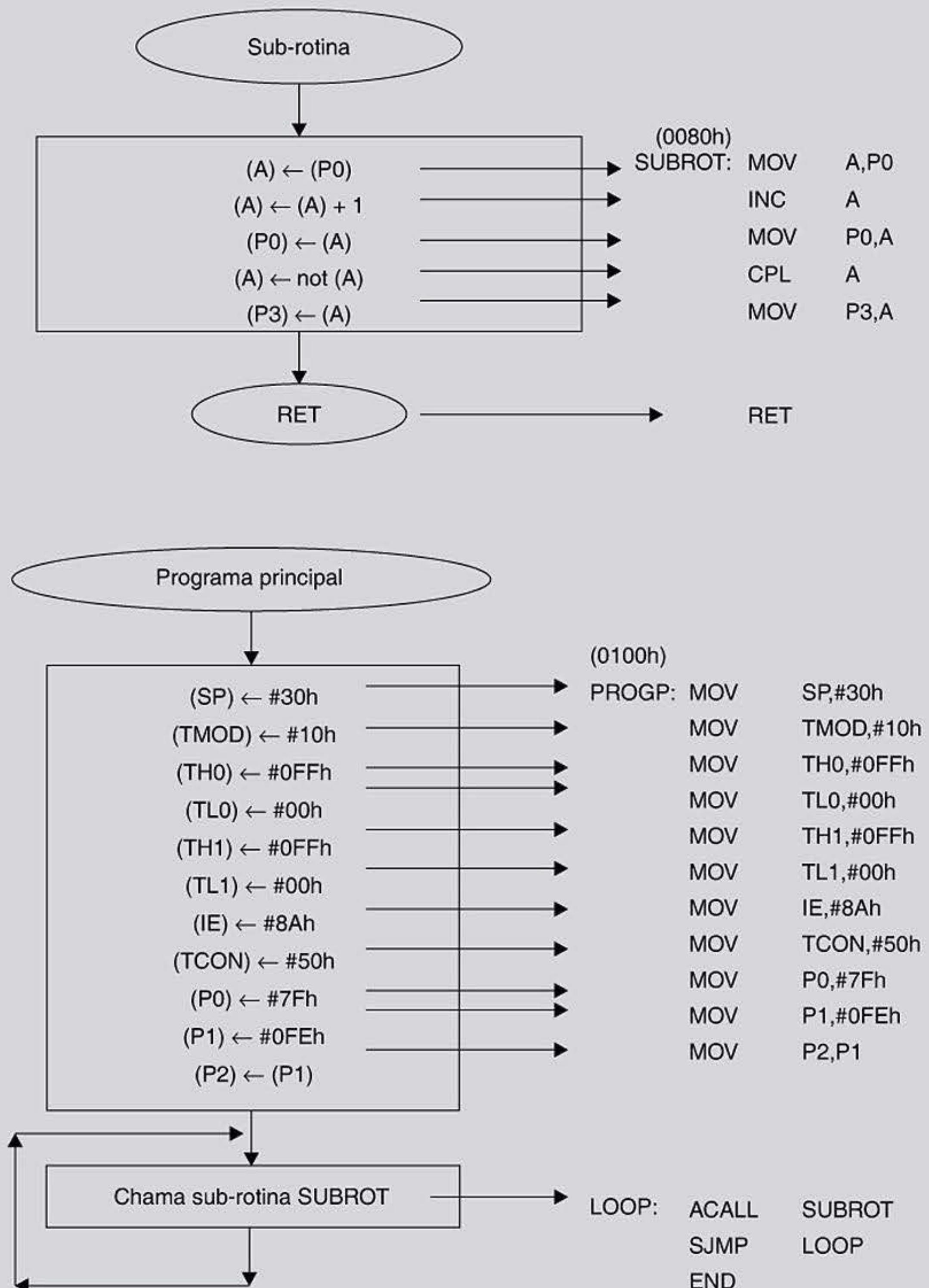


Figura 7.4 Fluxograma e programa-fonte estruturado do exercício resolvido 1.

Exercícios e questões propostos

- 1 - Faça um programa estruturado (fluxograma e programa-fonte), em Assembly, que utilize um dos membros da família de microcontroladores MCS-51 e que seja capaz de executar as seguintes atividades:
- 1.1 - Programar o *timer*/contador 0 no Modo 1.
 - 1.2 - Programar o *timer*/contador 1 no Modo 2.
 - 1.3 - O *timer*/contador 0 deve gerar interrupções a cada 2 mseg.
 - 1.4 - O *timer*/contador 1 deve gerar interrupções a cada 100 mseg.
 - 1.5 - Implementar uma sub-rotina que seja capaz de calcular a quantidade de números diferentes de 7Fh e que apresentam o bit 3 igual a 1 do buffer de memória que vai da posição de memória 53h a 7Ch. O resultado deve ser colocado no conteúdo da porta 3 por 0,5 segundo. Essa sub-rotina deve ser armazenada no endereço de memória de programa 0090h.
 - 1.6 - Implementar uma sub-rotina de atendimento à fonte de interrupção do *timer*/contador 0, que faz a porta 0 ser um contador ascendente binário.
 - 1.7 - Implementar uma sub-rotina de atendimento à fonte de interrupção do *timer*/contador 1, que faz a porta 2 ser um contador descendente hexadecimal.
 - 1.8 - A sub-rotina de atendimento à fonte de interrupção do *timer*/contador 0 deve ser escrita no endereço de memória de programa 0050h.
 - 1.9 - A sub-rotina de atendimento à fonte de interrupção do *timer*/contador 1 deve ser escrita no endereço de memória de programa 0070h.
 - 1.10 - O programa principal deve ser escrito no endereço de memória de programa 0200h.
- 2 - Faça um programa estruturado (fluxograma e programa-fonte), em Assembly, que utilize um dos membros da família de microcontroladores MCS-51 e que seja capaz de executar as seguintes atividades:

- 2.1 - Programar os *timers*/contadores 0 no Modo 3.
 - 2.2 - Um dos *timers*/contadores deve ser inicializado com o valor 33h.
 - 2.3 - O outro *timer*/contador deve ser inicializado com o valor 55h.
 - 2.4 - Implementar uma sub-rotina que seja capaz de calcular a quantidade de números diferentes de 99h e que apresentam o bit 7 igual a 0 do buffer de memória que vai da posição de memória 36h a 61h. O resultado deve ser colocado no conteúdo da porta 2 por 1 mseg de forma piscante. Essa sub-rotina deve ser armazenada no endereço de memória de programa 009Fh.
 - 2.5 - Implementar uma sub-rotina de atendimento à fonte de interrupção do *timer*/contador 0, que faça a porta 3 ser um contador ascendente decimal.
 - 2.6 - Implementar uma sub-rotina de atendimento à fonte de interrupção do *timer*/contador 1, que faça a porta 0 ser um contador decimal de números ímpares.
 - 2.7 - A sub-rotina de atendimento à fonte de interrupção do *timer*/contador 0 deve ser escrita no endereço de memória de programa 0050h.
 - 2.8 - A sub-rotina de atendimento à fonte de interrupção do *timer*/contador 1 deve ser escrita no endereço de memória de programa 0070h.
 - 2.9 - O programa principal deve ser escrito no endereço de memória de programa 0200h.
- 3 - Faça um programa estruturado (fluxograma e programa-fonte) em Assembly que utilize um dos membros da família de micro-controladores MCS-51 e que seja capaz de executar as seguintes atividades:
- 3.1 - Programar o *timer*/contador 0 no Modo 1.
 - 3.2 - Programar o *timer*/contador 1 no Modo 1.
 - 3.3 - O *timer*/contador 0 deve gerar interrupções a cada 50 mseg.
 - 3.4 - O *timer*/contador 1 deve gerar interrupções a cada 20 mseg.

- 3.5 - Implementar uma sub-rotina que seja capaz de calcular a quantidade de números menores e iguais a 66h e que apresentam paridade ímpar do buffer de memória que vai da posição de memória 37h a 6Ah. O resultado deve ser colocado no conteúdo da porta 3, a cada 2 segundos, de forma piscante. Essa sub-rotina deve ser armazenada no endereço de memória de programa 0300h.
- 3.6 - Implementar uma sub-rotina de atendimento à fonte de interrupção do *timer/contador 0*, que torne o conteúdo do bit 3 da porta 0 igual a 1.
- 3.7 - Implementar uma sub-rotina de atendimento à fonte de interrupção do *timer/contador 1*, que torne o conteúdo do bit 3 da porta 0 igual a 0.
- 3.8 - A sub-rotina de atendimento à fonte de interrupção do *timer/contador 0* deve ser escrita no endereço de memória de programa 0066h.
- 3.9 - A sub-rotina de atendimento à fonte de interrupção do *timer/contador 1* deve ser escrita no endereço de memória de programa 0088h.
- 3.10 - O programa principal deve ser escrito no endereço de memória de programa 0500h.
- 4 - Faça um programa estruturado (fluxograma e programa-fonte), em Assembly, que utilize um dos membros da família de microcontroladores MCS-51 e que seja capaz de executar as seguintes atividades:
- 4.1 - Programar o *timer/contador 0* no Modo 1.
 - 4.2 - Programar o *timer/contador 1* no Modo 0.
 - 4.3 - O *timer/contador 0* deve gerar interrupções a cada 65 mseg.
 - 4.4 - O *timer/contador 1* deve gerar interrupções a cada 5 mseg.
 - 4.5 - Implementar uma sub-rotina que seja capaz de calcular a quantidade de números maiores que 57h e que são pares do buffer de memória que vai da posição de memória 47h até a 65h. O resultado deve ser colocado no conteúdo da porta 0. Considere que na porta 0 está ligada uma interface com um display

de sete segmentos. Essa sub-rotina deve ser armazenada no endereço de memória de programa 0096h.

- 4.6 - Implementar uma sub-rotina de atendimento à fonte de interrupção do *timer/contador 0*, que leia o estado das chaves que estão ligadas à porta 3, por meio de uma interface em lógica negativa (chave aberta = 1 lógico, chave fechada = 0 lógico).
- 4.7 - Implementar uma sub-rotina de atendimento à fonte de interrupção do *timer/contador 1*, que mostre o estado das chaves por meio de um conjunto de 8 *leds*, durante 2 segundos, de forma piscante, que estejam ligados à porta 2 por meio de uma interface de saída em lógica positiva.
- 4.8 - A sub-rotina de atendimento à fonte de interrupção do *timer/contador 0* deve ser escrita no endereço de memória de programa 0106h.
- 4.9 - A sub-rotina de atendimento à fonte de interrupção do *timer/contador 1* deve ser escrita no endereço de memória de programa 0168h.
- 4.10 - O programa principal deve ser escrito no endereço de memória de programa 0400h.

PÁGINA EM BRANCO

8.1 Objetivos

- ❖ Apresentação da interface de comunicação serial
- ❖ Os diferentes modos de operação da interface de comunicação serial
- ❖ Comunicação de multiprocessadores
- ❖ Definindo o *baud rate* da interface de comunicação serial
- ❖ Projeto utilizando a interface de comunicação serial

8.2 Introdução teórica

A comunicação entre microcomputadores é de total relevância para o controle inteligente de máquina e de processo. Imagine um determinado sistema de controle de ponto, que tem a finalidade de armazenar dados sobre a entrada e saída dos funcionários de uma empresa. Nesse sistema, o terminal de ponto tem como objetivo coletar e armazenar as informações referentes a cada funcionário em sua memória de dados. O sistema de ponto só se completa se esses dados forem recolhidos por outro microcomputador para, depois, serem processados gerando a folha de pagamento. Geralmente, a captação desses dados nesse sistema é feita por meio de comunicação serial de dados. Essa é apenas uma aplicação entre milhares de outras que se pode exemplificar.

Assim, o estudo da interface de comunicação serial é de fundamental importância para o desenvolvimento de sistemas controlados por microcontroladores.

Com o estudo da interface de comunicação serial, terminam todos os conceitos básicos e fundamentais sobre a família de microcontroladores MCS-51 da Intel.

8.2.1 - A interface de comunicação serial: a interface de comunicação serial é *full duplex*, ou seja, ela pode, ao mesmo tempo, receber e transmitir dados seriais.

Além disso, ela pode receber dados vindos da comunicação serial através de um buffer, ou seja, pode iniciar a recepção de um segundo dado (byte) antes que um dado anterior seja lido a partir do registrador de recebimento. Entretanto, se o primeiro dado ainda não foi lido durante o recebimento do segundo, o primeiro dado será perdido.

Os registradores de recepção e transmissão da interface de comunicação serial são, ambos, acessados pelo registrador de função especial SBUF e são fisicamente separados. Quando o dado é movido *para* o registrador de função especial SBUF, ele vai para o *buffer de transmissão*, onde se dá o início da transmissão. Quando o dado é movido *do* SBUF, ele vem do *buffer de recebimento*, ou seja, é lido um dado da interface de comunicação serial. A recepção é inicializada no Modo 0 pela condição dos bits RI=0 e REN=1.

A recepção é inicializada em outros modos pela chegada do *start bit*, se o bit (REN) do registrador de função especial SCON for igual a 1.

Em todos os modos, a transmissão é inicializada por qualquer instrução que utiliza o SBUF como um registrador de destino.

Após o final da recepção do último bit do dado (bit mais significativo), o bit RI do registrador de função especial SCON é *setado*, sinalizando que a recepção foi finalizada. Caso a interface de comunicação serial esteja habilitada, sua sub-rotina de atendimento à interrupção por hardware será vetorizada (endereçada) por meio da execução de uma instrução LCALL 0023h. No endereço 0023h de memória de programa, deve-se armazenar a rotina de atendimento à fonte de interrupção da interface de comunicação serial, que trata da recepção. É necessário *resetar* o flag RI a fim de liberar o canal de comunicação serial para uma nova recepção de dados e também salvar o conteúdo do registrador SBUF em uma posição de memória para posterior tratamento da informação recebida.

Após o final de transmissão do último bit do dado (bit mais significativo), o bit TI do registrador de função especial SCON é *setado*, sinalizando que a transmissão foi finalizada. Caso o canal de comunicação serial esteja habilitado, a rotina de atendimento à interrupção por hardware será vetorizada (endereçada), por meio da execução de uma instrução LCALL 0023h. No endereço 0023h de memória de programa, deve-se armazenar a sub-rotina de atendimento à fonte de interrupção do canal de comunicação serial, que trata da transmissão. É necessário *resetar* o flag TI a fim de liberar o canal de comunicação serial para uma nova transmissão de dados.

Caso a recepção e a transmissão estejam habilitadas, a sub-rotina de atendimento à fonte de interrupção da interface de comunicação serial deve distinguir qual foi a fonte de interrupção da CPU, ou seja, deve identificar se foi a recepção ou a transmissão que gerou a interrupção. Isso é feito por meio do teste dos *flags* RI e TI, para verificar se estes são iguais a 1.

O registrador de função especial SCON é utilizado para determinar o modo de operação do canal de comunicação serial. Nele, é recebido o nono (9º) bit de dados (RB8), que contém outros *flags* de condição, conforme é mostrado a seguir:

$$(SCON) = \boxed{\text{SM0} \quad \text{SM1} \quad \text{SM2} \quad \text{REN} \quad \text{TB8} \quad \text{RB8} \quad \text{TI} \quad \text{RI}}$$

SM0	SM1	Modo	Descrição	Baud Rate
0	0	0	Registrador de deslocamento	$f_{osc}/12$
0	1	1	UART de 8 bits	variável
1	0	2	UART de 9 bits	$f_{osc}/64$ ou $f_{osc}/32$
1	1	3	UART de 9 bits	variável
Símbolo	Nome e significado			
SM2	Habilita a característica de comunicação de multiprocessadores nos modos 2 e 3. Nesses modos, se SM2 = 1, RI não será ativado, caso o nono bit de dados recebido seja igual a 0. No Modo 1, se SM2 = 1, RI não será ativado, caso um <i>stop bit</i> válido não seja recebido. No Modo 0, SM2 deverá ser 0.			
REN	Bit habitador da recepção serial. <i>Setado</i> /zerado por software para habilitar ou desabilitar a recepção serial.			
TB8	É o nono bit de dados que será transmitido nos modos 2 e 3. <i>Setado</i> ou zerado por software.			
RB8	Nos modos 2 e 3, é o nono bit de dados que foi recebido. No Modo 1, se SM2 = 0, RB8 é o <i>stop bit</i> que foi recebido. No Modo 0, RB8 não é utilizado.			
TI	É o <i>flag</i> de interrupção de transmissão. <i>Setado</i> por hardware no final do tempo do 8º bit no Modo 0 ou no início do <i>stop bit</i> em outros modos, em qualquer transmissão serial. Deverá ser zerado por software.			
RI	É o <i>flag</i> de interrupção de recepção. <i>Setado</i> por hardware no final do tempo do 8º bit no Modo 0 ou na metade do tempo do <i>stop bit</i> em outros modos, em qualquer recepção serial. Deverá ser zerado por software.			

- a) *Modos de operação*: a porta serial pode operar em quatro modos (do Modo 0 ao Modo 3), como se segue.
- a.1) *Modo 0*: dados serializados entram e saem por meio do pino RXD.

O pino TXD é responsável pela transmissão do *clock* de transmissão (*baud rate*).

Oito bits são transmitidos ou recebidos da seguinte maneira: 8 bits de dados, sendo que o primeiro bit transmitido ou recebido é o menos significativo.

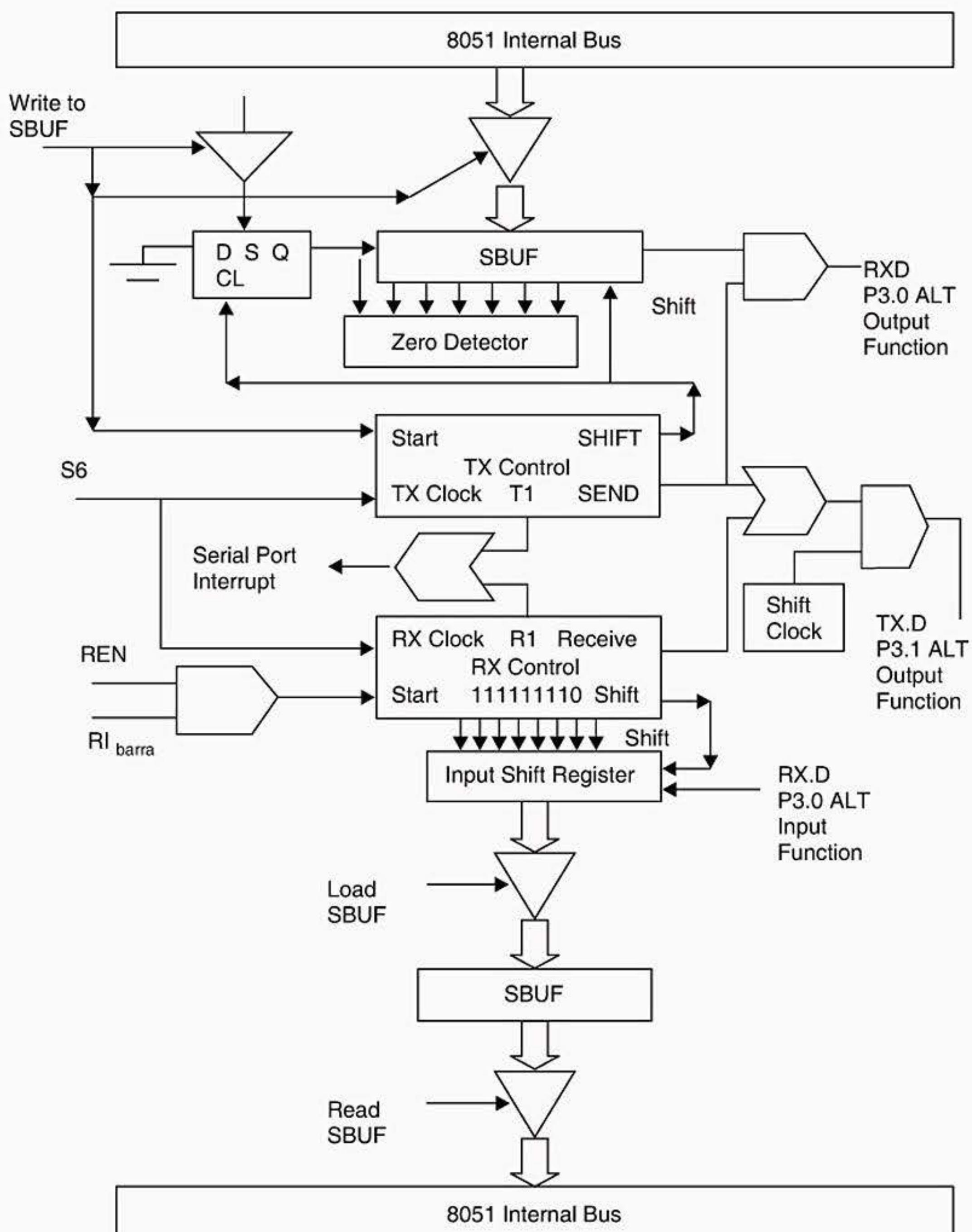


Figura 8.1 A interface de comunicação serial no Modo 0.

O *baud rate* (velocidade de comunicação serial de dados) é fixo em 1/12 da freqüência do oscilador.

A Figura 8.1 mostra o diagrama funcional simplificado da interface de comunicação serial no Modo 0.

Recepção: a recepção será iniciada se a condição dos bits do registrador de função especial SCON tiver $REN = 1$ e $RI = 0$ e se o pino de função alternativa P3.1 estiver habilitado. Em S6P2 do próximo ciclo de máquina, a unidade de controle de RX escreve os bits 11111110 no registrador de deslocamento e, no ciclo seguinte, ativa a função RECEIVE.

A função RECEIVE habilita a saída do registrador de deslocamento para o pino de saída de função alternativa P3.0. *Shift Clock* faz transições em S3P1 e S6P1 de todo o ciclo de máquina. Em S6P2, de todo o ciclo de máquina, no qual RECEIVE está ativa, o conteúdo do registrador de deslocamento de recepção é deslocado um bit para a esquerda. O valor fornecido pela direita é o que foi mostrado em P3.0 no S5P2 do mesmo ciclo de máquina.

Como os bits de dados se deslocam para a direita, números 1 preenchem os bits da esquerda a cada deslocamento. Quando o 0 lógico, que foi inicialmente carregado na posição mais à direita, chega à posição mais à esquerda no registrador de deslocamento, ele sinaliza ao bloco de controle de RX para fazer um último deslocamento e então carregar SBUF. Em S1P1 do décimo ciclo de máquina após a escrita em SCON é zerado RI, RECEIVE é zerado e RI é *setado*.

Quando a recepção de um dado é finalizada, é *setado* o bit RI do registrador de função especial SCON, sinalizando o final da recepção de um dado serial obtido dentro do buffer de recepção do canal de comunicação serial, ou seja, fica armazenado no conteúdo do registrador SBUF.

Transmissão: a transmissão é iniciada por qualquer instrução que utiliza SBUF como um registrador de destino.

Na transmissão, carregar um 1 lógico dentro do nono bit do registrador de deslocamento de transmissão habilitará a saída do registrador de deslocamento para o pino da função alternativa P3.0 e também habilitará o *clock* do deslocamento para o pino de função alternativa P3.1.

O sinal *Write to SBUF* em S6P2 também carrega 1 lógico na nona posição do registrador de deslocamento de transmissão e informa o bloco de controle de TX para que inicie a transmissão. A temporização interna é tal que um ciclo de máquina completo é finalizado entre o sinal *Write to SBUF* e o sinal SEND.

SEND habilita a saída do registrador de deslocamento para o pino de saída de função alternativa P3.0 e também habilita *SHIFT Clock* para o pino de saída de função alternativa P3.1. *SHIFT Clock* é baixo durante as fases S3, S4 e S5 do ciclo de máquina e alto durante as fases S6, S1 e S2. Para S6P2 de todo o ciclo de *clock* em que *SEND* está ativo, o conteúdo do registrador de deslocamento de transmissão é deslocado um bit para a direita.

Como os bits de dados se deslocam para a direita, os zeros preenchem os bits da esquerda a cada deslocamento. Quando o bit mais significativo do dado está na posição para ser transmitido, então o 1 lógico que foi inicialmente carregado no nono bit é o único bit à esquerda do bit mais significativo, e todas as posições estão 'zeradas'. Essa condição sinaliza ao bloco de controle de TX para fazer um último deslocamento e então desativar o sinal de *SEND* e *setar* TI. Ambas as ações ocorrem em S1P1 do décimo ciclo de máquina após o *Write to SBUF*.

- a.2) *Modo 1:* dez bits são transmitidos por meio do pino TXD e recebidos por meio do pino RXD. São eles:

- ◆ um *start bit* (0 lógico);
- ◆ oito bits de dados (primeiro, o bit menos significativo);
- ◆ um *stop bit*.

O *baud rate* é variável. O *timer* 1 ou 2 pode ser utilizado para fornecer o *clock* do canal de comunicação serial para gerar um *baud rate* variável.

A Figura 8.2 mostra o diagrama funcional simplificado da interface de comunicação serial no Modo 1.

Recepção: na recepção, o *stop bit* é recebido por meio do conteúdo do bit RB8 do registrador de função especial SCON.

A recepção é iniciada ao detectar uma borda de descida ($1 \rightarrow 0$) em RXD, e o bit RI do registrador de função especial é *setado* quando a recepção é finalizada, somente se RI = 0.

Transmissão: a transmissão é iniciada por qualquer instrução que utiliza SBUF como um registrador de destino, e a cada excesso do contador do *timer* será transmitido um bit de dados. Após o final da transmissão do último bit de dados (bit mais significativo) o bit TI do registrador de função especial SCON é *setado*.

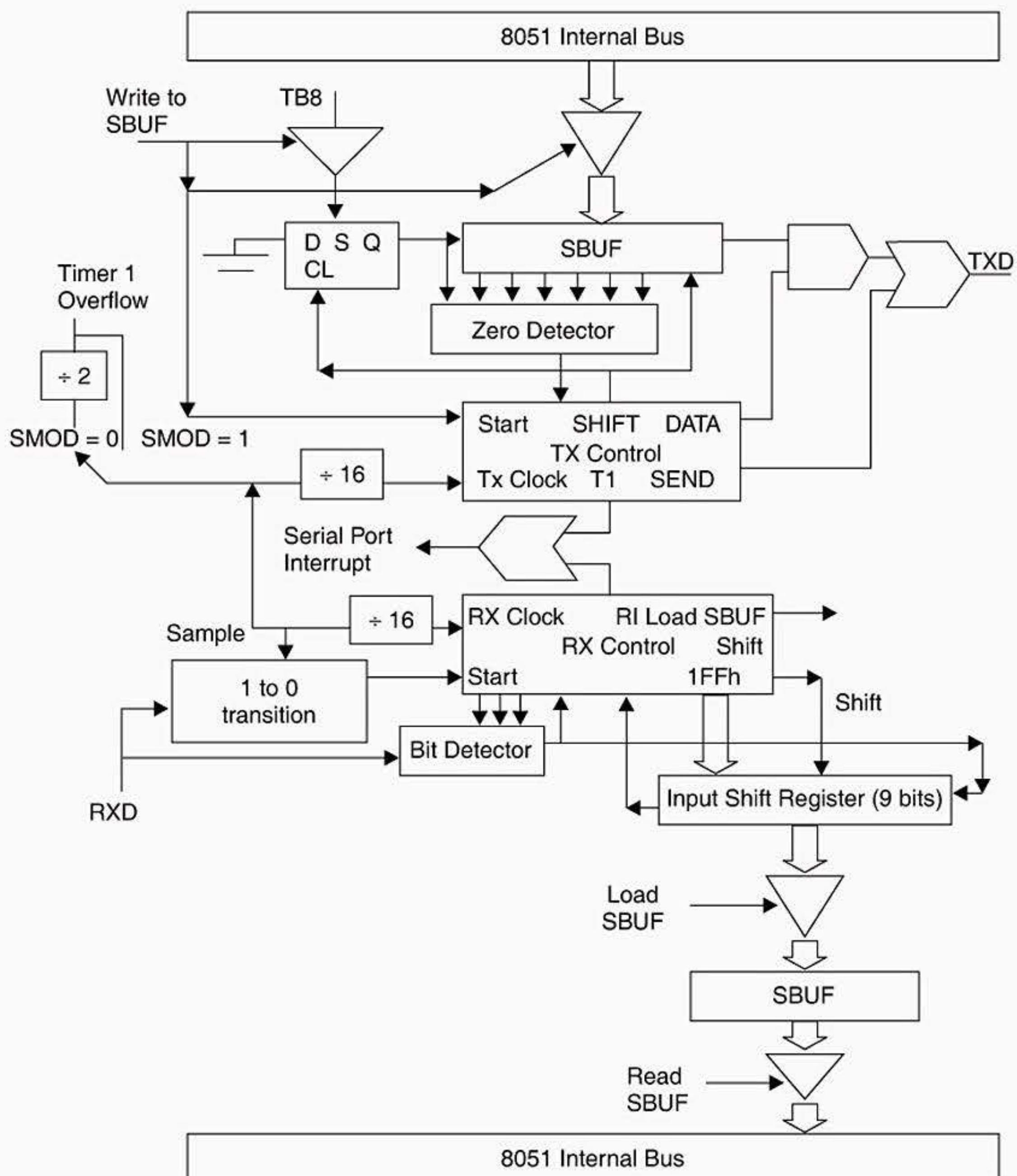


Figura 8.2 A interface de comunicação serial no Modo 1.

- a.3) *Modo 2: onze bits de dados são transmitidos por meio do pino TXD e recebidos por meio do pino RXD. São eles.*
- ◆ 1 *start bit* (0);
 - ◆ 8 bits de dados (primeiro, o bit menos significativo);
 - ◆ 1 nono bit de dados programável;
 - ◆ 1 *stop bit*.

O *baud rate* é programável para ou 1/32 ou 1/64 da freqüência do oscilador.

Recepção: na recepção, o nono bit de dados é recebido por meio do conteúdo do bit RB8 do registrador de função especial SCON, enquanto o *stop bit* é ignorado.

A recepção é iniciada ao detectar uma borda de descida ($1 \rightarrow 0$) no pino RXD; quando a recepção é finalizada, o bit RI do registrador de função especial SCON é *setado*, somente se $(RI) = 0$.

Transmissão: na transmissão, ao nono bit de dados (bit TB8 em SCON) pode ser atribuído um valor de 0 ou 1. Ou, por exemplo, o bit de paridade (P do PSW) poderia ser movido para TB8.

A transmissão é iniciada por qualquer instrução que utilize SBUF como um registrador de destino e finalizada quando TI = 1.

- a.4) *Modo 3:* onze bits de dados são transmitidos por meio do pino TXD e recebidos por meio do pino RXD:

- ◆ 1 *start bit* (0);
- ◆ 8 bits de dados (primeiro, o bit menos significativo);
- ◆ 1 nono bit de dados programável;
- ◆ 1 *stop bit*.

De fato, o Modo 3 é igual ao Modo 2 em todos os aspectos, exceto quanto à *baud rate*, que é variável.

8.2.2 - Comunicação de multiprocessadores: os modos 2 e 3 têm uma característica especial para as comunicações de multiprocessadores.

Nesses modos, 9 bits de dados são recebidos. O nono bit é armazenado no conteúdo do bit RB8 do registrador de função especial SCON e depois é enviado um *stop bit*. O canal de comunicação serial pode ser programado de maneira que quando o *stop bit* é recebido, a interrupção da porta serial seja ativada somente se o bit (RB8) = 1. Essa característica é habilitada ao *setar* o bit SM2 em SCON. Um modo de utilizar essa característica em sistemas a multiprocessadores é o seguinte:

- ◆ quando um processador 'mestre' quer transmitir um bloco de dados para um dos 'escravos', ele primeiro envia um byte de endereço que identifica um dos 'escravos'. Um byte de endereço difere de um byte de dados, sempre que o nono bit for 1 em um byte de endereço e 0 em um byte de dados;

- ◆ quando um $(SM2) = 1$, nenhum 'escravo' será interrompido por um byte de dados. Um byte de endereço interromperá todos os 'escravos', porém cada escravo examinará o byte recebido e verificará se é ele que está sendo endereçado. O 'escravo' endereçado excluirá o conteúdo de seu bit SM2 e vai se preparar para receber os bytes de dados que serão transmitidos pelo mestre;
- ◆ os 'escravos' que não foram endereçados deixam seus bits SM2 *setados* e ignorarão os bytes de dados que serão enviados pelo mestre;
- ◆ SM2 não tem efeito no Modo 0 e no Modo 1 ele pode ser utilizado para verificar a validade do *stop bit*. Na recepção em Modo 1, se $(SM2) = 1$, a interrupção de recepção não será ativada, a menos que um *stop bit* válido seja recebido.

8.2.3 - A freqüência de transmissão/recepção de dados (*baud rate*):

- Modo 0:* o *baud rate* é fixo e vale $f_{osc.}/12$.
- Modo 2:* depende do conteúdo do bit SMOD, que pertence ao registrador de função especial PCON.
Se $(SMOD) = 0$ (o valor é atribuído a esse bit sempre que ocorrer um sinal de *reset*), a *baud rate* será igual a $1/64$ da freqüência do oscilador.
Se $(SMOD) = 1$, o *baud rate* será igual a $1/32$ da freqüência do oscilador.
- Modos 1 e 3:* são determinados pelo *overflow* do *timer 1*.
 - Utilizando o timer 1 para gerar baud rates:* o *baud rate* é calculado por:

$$\text{Baud rate} = 2^{\text{SMOD}(0 \text{ ou } 1)} / 32 * (\text{Rate do overflow do timer 1}) \text{ (bits/seg.)}$$

A interrupção do *timer 1* deveria ser desabilitada para essa aplicação.

Pode ser configurado como *timer* e como contador e em qualquer modo de operação.

Na maioria das aplicações, ele é configurado para operar como *timer* em modo de recarregamento automático (*nibble* superior de TMOD = 0010_b). Nesse caso, o *baud rate* é fornecido pela fórmula:

$$\text{Baud rate} = 2^{\text{SMOD}(0 \text{ ou } 1)} / 32 * (\text{Freqüência do oscilador} / \{12 * [256 - (\text{TH1})]\}) \text{ (bits/seg.)}$$

A tabela a seguir relaciona vários *baud rates* utilizados e a maneira como elas podem ser obtidas utilizando o *timer 1*:

Baud rate (bits/seg.)	Freq. osc. (MHz)	SMOD	Timer 1		
			C/Tbarra	Modo	Valor recar.
Modo 0 Máx: 1MHz	12	X	X	X	X
Modo 2 Máx: 375K	12	1	X	X	X
Modo 1, 3: 62,5K	12	1	0	2	FFh
19,2K	11,059	1	0	2	FDh
9,6K	11,059	0	0	2	FDh
4,8K	11,059	0	0	2	FAh
2,4K	11,059	0	0	2	F4h
1,2K	11,059	0	0	2	E8h
137,5	11,059	0	0	2	1Dh
110	6	0	0	2	72h
110	12	0	0	1	FEEBh

Exercícios resolvidos

1 - Faça um programa estruturado (fluxograma e programa-fonte) em Assembly que utilize um dos membros da família de microcontroladores MCS-51, que seja capaz de executar as seguintes atividades:

- 1.1 - Programar o canal de comunicação serial no Modo 0.
- 1.2 - Transmitir continuamente, por meio do canal de comunicação serial, os valores de 55h e AAh.
- 1.3 - O programa principal deve ser escrito no endereço de memória de programa 0100h.

Solução:



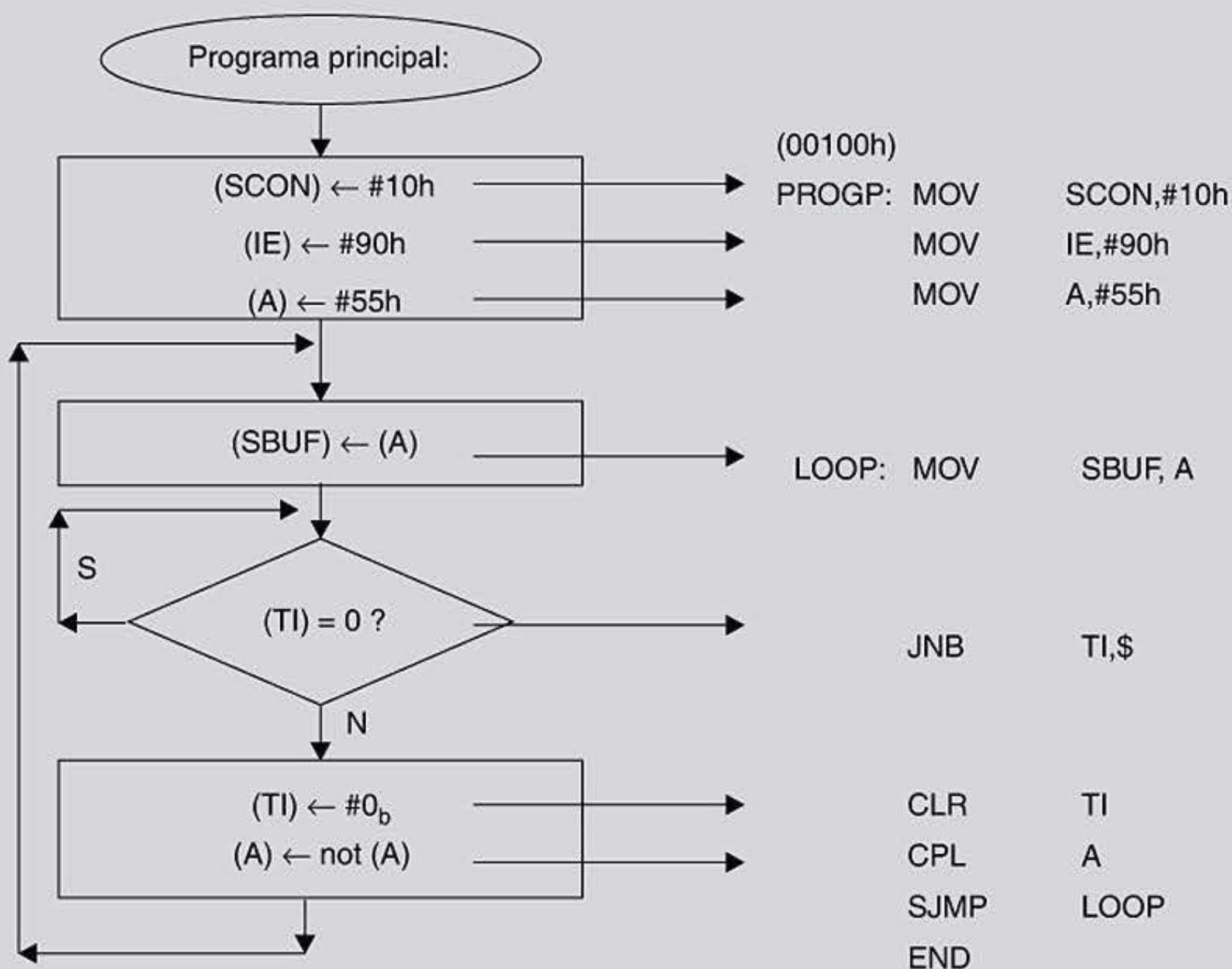


Figura 8.3 Fluxograma e programa-fonte estruturado do exercício resolvido 1.

A seguir, é mostrado o programa-fonte estruturado, preparado para uma simulação, por meio do AVSIM51 da AVOCKET, ou similar, com comentários da explicação e função de cada instrução dentro do programa.

```

DEFSEG EXEM, ABSOLUTE ; Define segmento de programa EXEM
SEG      EXEM

ORG      0000h      ; Após o reset da CPU, o programa prossegue no endereço 0000h
AJMP    0100h      ; (durante o acionamento da chave de reset ou após a energização do
                   ; sistema por meio do circuito de reset formado por um resistor em
                   ; paralelo com um diodo e em série com um capacitor que está ligado ao
                   ; pino de reset da CPU). Como foi pedido que o programa principal
                   ; fosse escrito no endereço de memória de programa 0100h, um salto
                   ; incondicional foi utilizado para tal posição de memória.

ORG      0023h      ; Após uma interrupção do canal serial, o programa prossegue no
RETI           ; endereço '0023h' e retorna da rotina de atendimento à fonte de
                   ; interrupção do canal de comunicação serial.
  
```

```

ORG      0100h    ; Início do programa principal
MAIN: MOV SCON,#10h ;(SCON) ← # 00010000b. Programa o canal serial como registrador de
        ; deslocamento e baud rate de Fosc/12 (modo 0).
        MOV IE,#90h   ;(IE)= 1001 0000b. Faz (EA)= 1 (cada interrupção é habilitada pelo seu
        ; bit habilitador); (ES)= 1 (habilita a interrupção de comunicação serial)
        MOV A,#55h    ;(A) ← #55h (dado a ser transmitido)
LOOP:  MOV SBUF,A  ;(SBUF) ← (A). Carrega o dado a ser transmitido no (SBUF) e é
        ; iniciada a transmissão serial.
        JNB TI,$     ; Se (TI)= 0 (o dado ainda não foi transmitido serialmente, bit a bit) ⇒
        ; (PC) ← $ (salta para o próprio endereço da instrução JNB). Aguarda
        ; ser transmitido. Quando o dado é transmitido, faz com que (TI)= 1 e é
        ; gerada uma interrupção do canal serial (salta para o endereço 0023h -
        ; rotina de serviço de comunicação serial).
        CLR TI       ;(TI) ← 0 (libera canal de comunicação serial para a transmissão de
        ; um novo dado)
        CPL A        ;(A) ← complemento de um do (A)
        SJMP LOOP    ; Loop do programa principal.
END

```

2 - Faça um programa estruturado (fluxograma e programa-fonte) em Assembly que utilize um dos membros da família de micro-controladores MCS-51, que seja capaz de executar as seguintes atividades:

- 2.1 - Programar o canal de comunicação serial no Modo 1 com um *baud rate* de 2,4KHz, considerando que o oscilador a cristal apresenta uma freqüência de $f_{osc} = 11,059\text{MHz}$.
- 2.2 - Obter informações recebidas do canal de comunicação serial e armazená-las no conteúdo das portas 0, 1, 2 e 3. Considere que as portas 0, 1, 2 e 3 estejam ligadas a quatro interfaces que acionam um conjunto de oito leds cada.
- 2.3 - O programa principal deve ser escrito no endereço de memória de programa 0700h.

Solução:

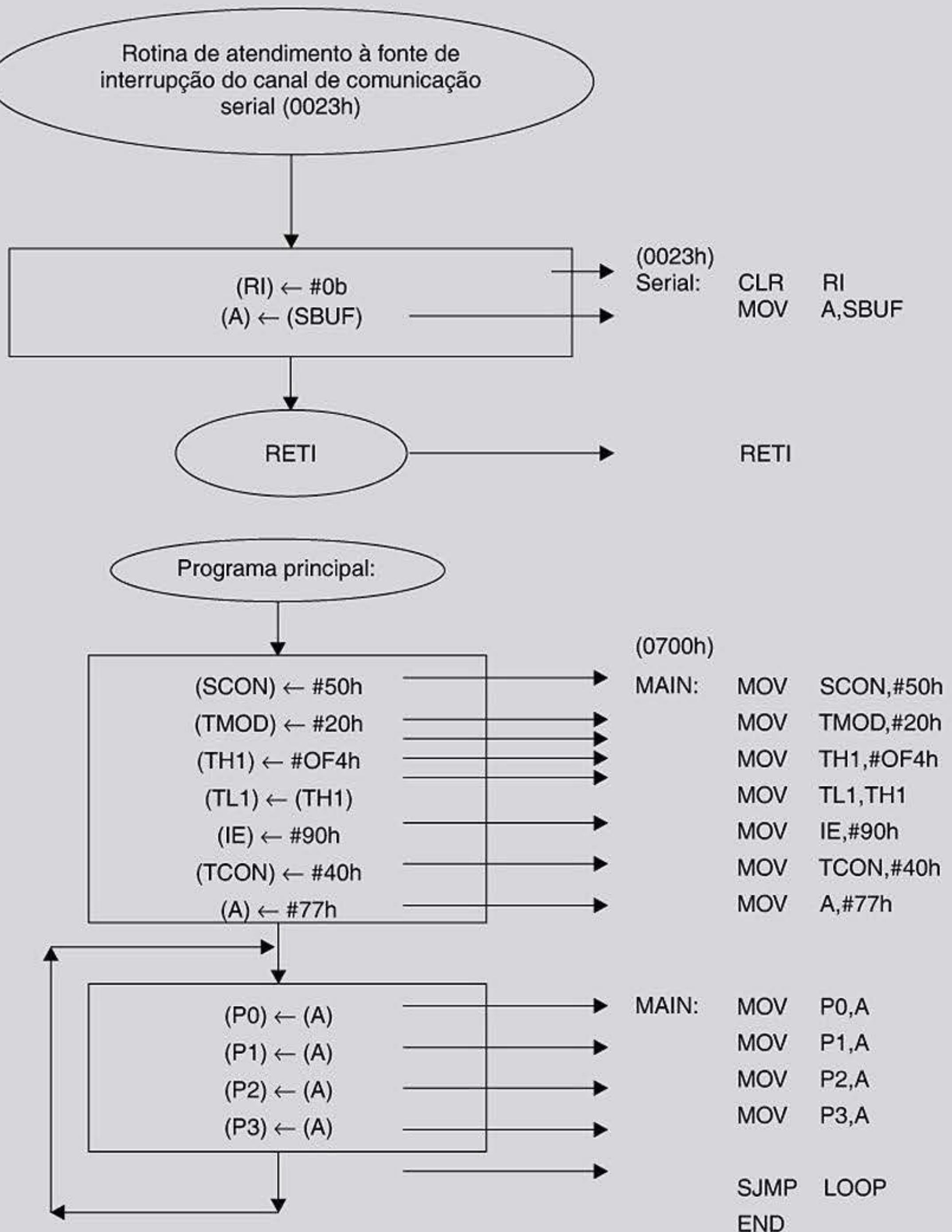


Figura 8.4 Fluxograma e programa-fonte estruturado do Exercício 2.

A seguir, é mostrado o programa-fonte estruturado, preparado para uma simulação por meio do AVSIM51 da AVOCET, ou similar, com comentários da explicação e função de cada instrução dentro do programa.

```

DEFSEG EXEM, ABSOLUTE ; Define segmento de programa EXEM
SEG EXEM
ORG 0000h ; Após o reset da CPU, o programa prossegue no endereço 0000h
AJMP 0100h ; (durante o acionamento da chave de reset ou após a energização do
; sistema, por meio do circuito de reset, formado por um resistor
; em paralelo com um diodo e em série com um capacitor que está
; ligado ao pino de reset da CPU). Como foi pedido que o programa
; principal fosse escrito no endereço de memória de programa 0700h,
; um salto incondicional foi utilizado para tal posição de memória.

ORG 0023h ; Após uma interrupção do canal serial, o programa prossegue no
; endereço '0023H', retorna da interrupção de comunicação serial.
CLR RI ; (RI) ← 0. Exclui o flag de interrupção de comunicação serial
MOV A,SBUF ; (A) ← (SBUF). Obtém o dado.
RETI ; Retorna da rotina de serviço da interrupção serial

ORG 0100H ; Programa principal
MAIN: MOV SCON,#50h ;(SCON) ← # 010 0000b. Programa o canal serial como UART de
; 8 bits, baud rate variável e habilita a recepção serial
MOV TMOD,#20h; (TMOD) ← #0010 0000b. Programa o timer 1 no Modo 2
; (8 bits com recarregamento automático).
MOV TH1,#0F4h ;(TH1) ← F4h. Valor para gerar baud rate de 2,4KHz (recarregador
; automático)
MOV TL1,TH1 ;(TL1) ← (TH1). Valor inicial para um baud rate de 2,4KHz
MOV IE,#90h ;(IE) ← #1001 0000b. Faz (EA) = 1 (cada interrupção é habilitada
; pelo seu bit habilitador); (ES) = 1 (Habilita a interrupção de
; comunicação serial e não habilita a interrupção do timer 1
MOV TCON,#40h ; Liga/faz executar o timer 1
MOV A,#77h ;(A) ← #77h
LOOP: MOV P0,A ;(P0) ← (A)
        MOV P1,A ;(P1) ← (A)
        MOV P2,A ;(P2) ← (A)
        MOV P3,A ;(P3) ← (A)
        SJMP LOOP ; Loop do programa principal
END

```

Exercícios e questões propostos

1 – Faça um programa estruturado (fluxograma e programa-fonte) em Assembly que utilize um dos membros da família de microcontroladores MCS-51 e que seja capaz de executar as seguintes atividades:

1.1 – Programar o canal de comunicação serial no Modo 1 com uma *baud rate* de 9,6KHz, considerando que o oscilador a cristal apresente uma freqüência de $f_{osc} = 11,059\text{MHz}$.

- 1.2 - Transmitir o conteúdo do buffer de memória cujo endereço inicial é 30h e o final é 40h, com a utilização do canal de comunicação serial, cada vez que ocorrer o acionamento e o desacionamento da chave 5. Considere que, à porta 0, está ligada uma interface de entrada com um *dip switch* de oito chaves numeradas de 0 a 7 (desconsidere o *bounce* das chaves do *dip switch*);
- 1.3 - O programa principal deve ser escrito no endereço de memória de programa 0200h.
- 2 - Faça um programa estruturado (fluxograma e programa-fonte) que utilize um dos membros da família de microcontroladores MCS-51 e que seja capaz de executar as seguintes atividades:
- 2.1 - Programar o canal de comunicação serial no Modo 1 com um *baud rate* de 9,6KHz, considerando que o oscilador a cristal apresenta uma freqüência de $f_{osc} = 11,059\text{MHz}$.
- 2.2 - Receber informações por meio do canal de comunicação serial e armazená-las no conteúdo do buffer de memória, cujo endereço inicial é 45h e o final é 61h, cada vez que for definido um byte igual a 3Ah pelo *dip switch* de oito chaves. Considere que, à porta 0, está ligada uma interface de entrada com um *dip switch* de oito chaves (desconsidere o *bounce* das chaves do *dip switch*);
- 2.3 - O programa principal deve ser escrito no endereço de memória de programa 0050h.
- 3 - Faça um programa (fluxograma e programa-fonte) estruturado em Assembly, que utilize um dos membros da família de microcontroladores MCS-51 e que seja capaz de executar as seguintes atividades:
- 3.1 - Programar o canal de comunicação serial no Modo 3 com um *baud rate* de 4,8KHz, considerando que o oscilador a cristal apresente uma freqüência de $f_{osc} = 11,059\text{MHz}$.
- 3.2 - Receber informações por meio do canal de comunicação serial e armazená-las no conteúdo do buffer de memória, cujo endereço inicial é 5Fh e o final é 72h, cada vez que as chaves 3 e 6 forem acionadas e desacionadas pelo *dip switch* de oito chaves. Considere que, à porta 0, está ligada uma interface de entrada com um *dip switch* de oito chaves (desconsidere o *bounce* das chaves do *dip switch*).

3.3 - Transmitir pelo canal de comunicação serial o buffer de memória, cujo endereço inicial é 4Ch e o final é 65h, cada vez que as chaves 1 e 4 forem acionadas e transmitir o buffer de memória, cujo endereço inicial é 66h e o final é 75h, cada vez que as chaves 1 e 4 forem desacionadas.

3.4 - O programa principal deve ser escrito no endereço de memória de programa 0350h.

4 - Considere um microcontrolador 8051 com sua porta P0 ligada a um *dip switch* de oito chaves (considere, inicialmente, todas abertas (nível lógico 1) - P0.0←CH0, ..., P0.7←CH7) e sua porta 2 ligada a um conjunto de oito leds. Foi implementado um programa-fonte estruturado com as seguintes características:

4.1 - Calcular a quantidade de números pares que apresentam o bit 4 igual a 1 e que apresentam paridade ímpar do buffer de memória, cujo endereço inicial é 3Bh e o final é 66h, sempre que as chaves 0, 2 e 7 são acionadas (considerar o *bounce* e utilizar um dos *timers* para gerar a rotina de tempo). O resultado do cálculo, mostrado anteriormente, deve ser colocado no conjunto de leds, de forma pulsante (deve ficar piscando) a cada 0,6 segundo;

4.2 - O resultado também deve ser transmitido pelo canal de comunicação serial com um *baud rate* de 1,2Kbits/seg.

5 - Obtenha o fluxograma a partir do programa-fonte fornecido a seguir e descreva em detalhes o funcionamento desse programa.

```
; Programa de identificação de chaves com eliminação de bounce
; Descrição: essa rotina descreve como fazer uma máquina de estados com a
; utilização dos timers
; Estado 0: aguarda o acionamento das chaves 0, 2, e 7 -
; 01111010 = 7Ah(Registrador R(0) = 00h)
; Estado 1: confirmação do acionamento após o tempo de eliminação do bounce
; (ruído da chave mec.)
; Registrador (R0) = 01H
; Estado 2: aguarda a desativação dessa combinação de chaves (Registrador
; R(0) = 02h)
; Estado 3: confirmação da desativação da combinação dessas chaves (Registrador R(0) = 03h)
; Quando estiver no estado 3 (aguardando a desabilitação do conjunto de chaves):
; 1 - Calcular a quantidade de números pares que apresentam o bit 4 igual a 1 e
; que apresentam paridade ímpar do buffer de memória cujo endereço inicial é 3Bh e o final é 66h.
; 2 - O resultado deve ser colocado no display de sete segmentos (parte menos significativa) em modo
; pulsante (deve ficar piscando a cada 0,6 segundo).
```

; 3 - O resultado deve ser transmitido pelo canal de comunicação serial
; com um *baud rate* de 1,2Kbits/seg.

```

org 0000h ; Após o reset, o programa prossegue
            ; no endereço 0040h
ajmp ProgP

org 000bh
reti

org 001bh
reti

org 0023h
jnb ri,sai ; Acabou de RX um caractere
clr ri       ; Libera o canal de comunicação serial para
              ; uma nova RX
mov r7,sbuf ; Armazena o caractere recebido no ACC
cjne r7,#35h,sai ; O caractere é igual a 55h
setb 20h.5   ; Habilita tempo de 2 segundos
sai: jnb ti,sail
      clr ti
sail: reti

org 0060h ; Tabela de código de sete segmentos
            ; de 0 a F
db 3fh    ; 0
db 06h    ; 1
db 5bh    ; 2
db 4fh    ; 3
db 66h    ; 4
db 6dh    ; 5
db 7dh    ; 6
db 07h    ; 7
db 7fh    ; 8
db 6fh    ; 9
db 77h    ; A
db 7ch    ; B
db 39h    ; C
db 5eh    ; D
db 79h    ; E
db 71h    ; F

org 0100h

VarCH:
      clr 20h.0 ; A cada loop, reseta o flag de
                  ; varredura de teclado
      mov a,th0 ; Lê o valor do timer mais significativo
                  ; (aprox. 5ms)
      clr c     ; Prepara carry bit para não
                  ; influenciar a operação de subtração

      subb a,#28h ; Verifica se chegou a hora de fazer a
                  ; varredura das teclas

```

```

jc      adr1          ; Ainda não chegou a hora de fazer a
                  ; varredura das teclas
setb    20h.0          ; Chegou a hora de fazer a varredura
                  ; das teclas
mov     th0,#00h       ; Inicializa o regulador de contagem do
                  ; timer 0
adr1:  ret

; Início do Estado 0 (aguarda o acionamento das chaves 0, 2 e 4)

DeAcCH:
jnb    20h.0,adr2      ; Salta, se não for o momento de fazer a
                      ; varredura do teclado
cjne   r0,#00h,adr2      ; Salta, se não for Estado 0 (aguarda o
                      ; acionamento das chaves)
mov    a,p0
cjne   a,#7Ah,adr2      ; Salta, se as chaves não estiverem
                      ; acionadas
mov    r0,#01h          ; Define Estado 1: confirmação do
                      ; acionamento com eliminação do bounce
adr2:  ret

; Início do Estado 1 (confirmação do acionamento das teclas 0, 2 e 4)

CoAcCH:
jnb    20h.0,adr3      ; Salta, se não for o momento de fazer a
                      ; varredura do teclado
cjne   r0,#01h,adr3      ; Salta, se não for Estado 1 (aguarda o
                      ; acionamento das chaves)
mov    r0,#00h          ; Alarme falso de acionamento (ruído),
                      ; define Estado 0
mov    a,p0
cjne   a,#7Ah,adr3      ; Salta, se as chaves não estiverem
                      ; acionadas
mov    r0,#02h          ; Define Estado 2: aguarda desativação das
                      ; teclas 0, 2 e 4
clr    20h.0          ; Não é o momento de fazer a varredura do teclado
setb    20h.1          ; Confirmados os acionamentos das chaves 0, 2 e 7
adr3:  ret

; Início do Estado 2 (aguarda a desativação das teclas 0, 2 e 4)

DeDeCH:
jnb    20h.0,adr4      ; Salta, se não for o momento de fazer a
                      ; varredura do teclado
cjne   r0,#02h,adr4      ; Salta, se não for Estado 2 (aguarda o
                      ; acionamento das chaves)
mov    a,p0
clr    c
subb   a,#7Ah          ; Lê o conteúdo do P0 (chaves)
                      ; Zera o carry bit para não influenciar na
                      ; subtração
jz     adr4            ; Compara a condição das chaves com a
                      ; condição de acionamento
                      ; Se for igual, as chaves continuam
                      ; acionadas, continua no Estado 2
mov    r0,#03h          ; Define Estado 3: confirmação do
                      ; desacionamento com eliminação do bounce
clr    20h.0          ; Não é o momento de fazer a varredura do
                      ; teclado

```

```

adr4: ret
      ; Início do Estado 3 (confirmação da desativação das teclas 0, 2 e 4)

CoDeCH:
      jnb    20h.0,adr5   ; Salta, se não for o momento de fazer a varredura do
      ; teclado
      cjne  r0,#03h,adr5   ; Salta, se não for Estado 2 (aguarda o acionamento
      ; das chaves)
      mov    r0,#02h   ; Alarme falso de acionamento (ruído), define
      ; Estado 0
      mov    a,p0   ; Lê o conteúdo do P0 (chaves)
      clr    c   ; Zera o carry bit para não influenciar na subtração
      subb  a,#7Ah   ; Compara a condição das chaves com a condição
      ; de acionamento
      jz     adr5   ; Se for igual, as chaves continuam acionadas,
      ; continua no Estado 2
      mov    r0,#00h   ; Define Estado 3: confirmação do desacionamento com
      ; eliminação do bounce
      mov    r2,#00h   ; Zera contador de cálculo de número do buffer
      clr    20h.0   ; Não é o momento de fazer a varredura do teclado
      clr    20h.1   ; Teclas não são mais acionadas
      mov    p1,#00h   ; Apaga o display de sete segmentos

adr5: ret

calc: jb    20h.5,adr6   ; Acabou de receber um caractere 55h do canal
      ; de comunicação serial
      jnb    20h.1,adr6   ; Chaves não são acionadas
      mov    r1,#3bh   ; Início do buffer de memória
      mov    r2,#00h   ; Quantidade de números pares que apresentam o
      ; bit 4 igual a 1 e que apresentam paridade ímpar
      adr8: mov    a,@r1   ; Traz dado do buffer da memória para o acumulado
      jb    acc.0,adr7   ; Se o número for ímpar, analisa próximo dado do buffer
      ; de memória
      jnb    acc.4,adr7   ; Se o bit 4 for zero, analisa próximo dado do
      ; buffer de memória
      jnb    psw.0,adr7   ; Se a paridade for par, analisa próximo dado do buffer
      ; de memória
      inc    r2   ; Se for par, tiver o bit 4 igual a 1 e a
      ; paridade for ímpar, soma 1 ao conteúdo do registrador
      ; r2.

adr7: inc    r1   ; Aponta para o próximo dado do buffer de memória
      cjne  r1,#67h,adr8   ; Testa se o buffer foi analisado totalmente
      setb    20h.2   ; Sinaliza que já foi calculada a quantidade de números

adr6: ret
      ; Rotina de transformação de código hexadecimal para o código de sete segmentos:

Hex7S:
      jb    20h.5,adr6   ; Acabou de receber um caractere 55h do canal de
      ; comunicação serial
      jnb    20h.2,adr9   ; Endereço inicial da tabela dos códigos de sete
      ; segmentos
      mov    dptr,#0060h
      mov    a,r2   ; Obtém resultado
      anl    a,#0fh   ; Mascara a parte mais significativa

```

```

        movc  a,@a+dptr ; Obtém dado convertido em código de sete
                      ; segmentos
        mov   r2,a      ; Armazena dados em código de 7 segundos em r2.
        setb  20h.3    ; Sinaliza que a quantidade de número está em código de sete
                      ; segmentos
        clr   20h.2    ; Reseta o flag de cálculo do resultado em hexadecimal
adr9:  ret

; Rotina de tempo para o acionamento do display de sete segmentos (0,5 segundo)

TeA05s:
        jnb   20h.0,adr10
        jb    20h.6,adr10 ; Não foi recebido um caractere 55h
                      ; pelo canal de comunicação serial
        jb    20h.4,adr10 ; Display com os leds desacionados
                      ; (apagados)
        cjne r0,#02h,adr10 ; Salta, se não for o Estado 2 (conjunto de chaves
                      ; acionadas)
        inc   r6
        mov   a,r6      ; Calcula o tempo de 50mseg
        clr   c
        subb a,#32h
        jc    adr10    ; Ainda não passou 0,5 segundo
        mov   r6,#00h   ; Reinicializa contador (R6)
        acall calc     ; Calcula quantidade de números do buffer
        acall Hex7S   ; Transforma (R2) de Hexadecimal para sete segmentos
        mov   p1,r2    ; Aciona o display com o valor de (R2) em código
                      ; de sete segmentos
        setb  20h.4    ; Display mostrando o (R2)
adr10: ret

; Rotina de tempo para o acionamento do display de sete segmentos (0,5 segundo)

TeD05s:
        jnb   20h.0,adr11
        jb    20h.6,adr11 ; Não foi recebido um caractere 55h pelo canal de
                      ; comunicação serial
        jnb   20h.4,adr11 ; Display com os leds desacionados (apagados)
        cjne r0,#02h,adr11 ; Salta, se não for o Estado 2 (conjunto de chaves
                      ; acionadas)
        inc   r6
        mov   a,r6      ; Calcula o tempo de 50 mseg.
        clr   c
        subb a,#32h
        jc    adr11    ; Ainda não passou 0,5 segundo
        mov   r6,#00h   ; Reinicializa contador (R2)
        mov   p1,#00h   ; Desativa display com o valor de (R2) em
                      ; código de sete segmentos
        clr   20h.4    ; Display mostrando o (R2)
adr11: ret

; Rotina de transmissão (R2) pelo canal de comunicação serial

Transm:
        jb    20h.5,adr12 ; Não foi recebido um caractere 55h pelo
                      ; canal comunicação serial
        cjne r0,#02,adr12 ; Transmite (R2) pelo canal de
                      ; comunicação serial, se for o Estado 2

```



```

    mov  p1,#00h      ; Display de sete segmentos inicialmente apagado
    mov  20h,#00h      ; 'Zera' todos os flags do programa

LooPP:
    lcall VarCH        ; Chama sub-rotina de varredura de teclas
    lcall DeAcCH        ; Detecta acionamento das chaves 0, 2 e 7
    lcall CoAcCH        ; Confirmação do acionamento das chaves 0, 2 e 7
    lcall DeDeCH        ; Detecta o acionamento das chaves 0, 2 e 7
    lcall CoDeCH        ; Confirmação do acionamento das chaves 0, 2 e 7
    lcall calc          ; Calcula a quantidade de números do buffer
    lcall Hex7S
    lcall TeA05s        ; Chama a rotina de tempo de 0,5 segundo e aciona
                        ; (P1)
    lcall TeD05s        ; Chama a rotina de tempo de 0,5 segundo e
                        ; desativa (P1)
    lcall Transm         ; Transmite (R2) pelo canal de comunicação serial a
                        ; 1,2 Kbits/s
    lcall TeAc2s         ; Temporiza 2 segundos para o caractere "o"
    lcall TeDe2s         ; Final da temporização de 2 segundos

```

EXPERIÊNCIA 01: IMPLEMENTAÇÃO E EXECUÇÃO DE PROGRAMAS EM ASSEMBLY POR MEIO DO SIMULADOR AVSIM51 OU DO KIT DE DESENVOLVIMENTO AES-51

A.1 Objetivos

- ❖ Apresentação das etapas básicas necessárias para a implementação e execução de um programa em Assembly por meio do simulador *8051 Simulator/Debugger* (AVSIM51.exe) da AVOCET, versão 1.6, e do kit de desenvolvimento AES-51
- ❖ Quais são as etapas para o projeto de um programa em Assembly em um ambiente profissional de trabalho?
- ❖ Simulação de programas em Assembly utilizando o simulador 'AVSIM51', da AVOCET, e o Kit de desenvolvimento AES-51

A.2 Introdução teórica

As seguintes etapas são necessárias para a implementação e simulação de um programa em Assembly:

A.2.1 – Edição do programa-fonte em Assembly: o editor de textos a ser utilizado é o EDIT do DOS, que é bastante simples e funciona da mesma maneira que o editor MS-WORD, porém com muito menos recursos.

Observação: se quiser editar um programa em Assembly, utilizando um microcomputador (PC), proceda da seguinte maneira:

- a) clique em **Iniciar, > Programas e Prompt do MS-DOS;**
- b) escolha o subdiretório ou a pasta em que você armazenará o seu programa em Assembly, por meio do comando *CD..* e *CD nome do subdiretório* (p. ex.: *c:\windows>cd..*, *c:\>cd eletrica*, *c:\eletrica>*). Caso o leitor

queira criar algum diretório dentro do DOS, utilize o comando: *MD nome do novo diretório;*

- c) digite **EDIT**.

É possível estruturar a edição de um programa-fonte, em Assembly, em quatro partes importantes:

1 - *Colunas de 1 a 6:* reservadas, exclusivamente, para definir os endereços da memória de programa das instruções em Assembly. Esses endereços são representados por palavras seguidas de dois pontos, que representam números de posições de memória de programa em hexadecimal. Exemplos: ADR1:, LOOP: etc. Não é obrigatório defini-los em todas as linhas de programa. Geralmente, são definidos no programa, quando:

- ◆ é necessário destacar determinados endereços iniciais de trechos de programa de importância relevada. Por exemplo: PROGPRIN: (endereço inicial de um **Programa principal**);
- ◆ são endereços iniciais de sub-rotinas. Por exemplo: ACIONA (endereço inicial da sub-rotina de nome ACIONA);
- ◆ são endereços que definem os locais de desvios condicionais ou incondicionais, utilizados nas instruções de saltos incondicionais e condicionais. Por exemplo: LOOP (endereço de desvio incondicional, utilizado em uma instrução, por exemplo, LJMP LOOP).

2 - *Colunas de 7 a 10:* reservadas para os mnemônicos das instruções em Assembly. Não há obrigatoriedade de utilizar todas essas colunas. Por exemplo: (MOV, ADD etc.).

3 - *Colunas de 11 a 18:* reservadas para os argumentos dos mnemônicos das instruções em Assembly (MOV A,R2, ADD 30H,@R1 etc.). Não é obrigatório utilizar essas colunas.

4 - *Coluna 19 em diante:* geralmente, utilizada para fazer comentários explicativos em frente às instruções do programa. Esses comentários têm a finalidade de fazer a documentação do programa. Todo comentário se inicia com o símbolo ';'. Não é obrigatório utilizar essas colunas. O comentário também pode ser feito a partir de qualquer coluna.

Veja a seguir um exemplo de um programa-fonte em Assembly, editado utilizando a estruturação mostrada anteriormente.

Endereço de memória da ROM	Mnemônico da instrução	Argumentos da instrução	Comentários (tudo que estiver após o símbolo ";").
ADR1:	MOV	A,R0	; Representação simbólica da instrução: (A)←(R0) ; O conteúdo do registrador acumulador será armazenado ; com o conteúdo do registrador R0.
	ADD	R1,A	; Representação simbólica da instrução: (A)←(A) + (R0) ; No conteúdo do registrador acumulador, será ; armazenado o resultado da operação de adição entre ; o conteúdo do registrador acumulador A e o conteúdo ; do registrador R1
LOOP:	MOV	@R0,30h	; Representação simbólica da instrução: ((R0))←(30h) ; move para o conteúdo da posição de memória, cujo ; endereço é dado pelo conteúdo do registrador R0, o ; conteúdo da posição de memória cujo endereço é 30h.
	END		; Diretriz do compilador Assembly ; fim do programa.

Antes de editar o programa, crie a pasta *c:\eletrica* utilizando o comando *MD eletrica*. Crie uma outra pasta, dentro da pasta *eletrica*, chamada *avc51*.

Acesse a pasta *c:\eletrica\avc51*. Utilize os comandos de *CD eletrica* e *CD abc51*. Para voltar à pasta anterior, utilize o comando *CD...*

Assim, utilizando o editor de textos *EDIT* do DOS, edite o programa. Salve o programa editado no microcomputador, com o nome de *exemplo1.asm*, na pasta *c:\eletrica\avc51*.

A.2.2 – Compilação do programa em Assembly: define-se compilador como um programa aplicativo que transforma um arquivo constituído por códigos ASCII (caracteres que podem ser impressos), gerados normalmente por um editor de textos, em um arquivo binário que contém os bytes correspondentes às instruções (códigos de máquina) do microprocessador do microcontrolador, que foram digitadas por meio do editor de textos.

Para compilar um arquivo de texto, crie uma pasta de nome *eletrica*, e dentro dela, crie outra pasta chamada *avc51* (comando *MD nome da pasta*, para criar a pasta, *CD nome da pasta* para acessar a pasta e *CD..* para sair dessa pasta e voltar para a anterior). Copie todos os arquivos do disco da AVOCET correspondentes aos programas de compilação da família MCS-51 da Intel.

Acesse na pasta *c:\eletrica\avc51* e digite:

```
c:\eletrica\avc51>ava51 exemplo1.asm ALLPUBLIC (essa última palavra deve estar em letras maiúsculas)
```

Utilizando um compilador da família de microcontroladores MCS-51, no caso, o simulador *AVA51.exe* da AVOCET, após a execução da compilação de um arquivo de texto, serão gerados dois arquivos, que são:

- ◆ um arquivo de mesmo nome, porém com extensão *.obj*, que corresponde ao arquivo compilado contendo os códigos-objeto, ou os códigos de máquina, que correspondem às instruções contidas no arquivo de texto digitado. Por exemplo, ao compilar um arquivo *Exemplo1.asm*, será gerado o arquivo *Exemplo1.obj*;
- ◆ um outro arquivo de mesmo nome, porém com a extensão *.prn*, que corresponde a um arquivo de texto que mostra o resultado da compilação, contendo, para cada linha de programa, o código de máquina relativo a cada instrução. Caso exista um erro de compilação, esse arquivo mostra qual é a linha do programa em que ocorreu o erro e também o tipo de erro. Assim, sempre que houver um erro de compilação, o leitor deve abrir esse arquivo de extensão *.prn*, por meio do editor de textos, procurar onde ocorreu o erro, reabrir o arquivo original de extensão *.asm*, corrigir o erro e novamente executar o processo de compilação. Esse processo é interativo, e só é finalizado quando não existirem mais erros de compilação.

Observação: a fase de compilação não mostra erros lógicos de programação.

A.2.3 – Linkagem do programa-objeto '*.obj*': essa etapa tem a finalidade de reunir, em um único arquivo, vários programas feitos em arquivos diferentes (método recomendado para executar projetos de software). Caso só exista um arquivo compilado, essa fase também é obrigatória.

Para linkar um único programa compilado *.obj*, digite:

```
c:\eletrica\avc51>avr51 exemplo = exemplo1.obj -SYMBOLS (em letras maiúsculas)
```

Para linkar vários programas compilados *.obj*, digitar:

```
c:\eletrica\avc51>avr51 exemplo = exemplo1.obj, exemplo2.obj,..., exemplon.obj -SYMBOLS
```

Após a linkagem, serão gerados vários arquivos, dos quais destacaremos alguns de fundamental relevância:

- ◆ arquivo de mesmo nome, sem extensão: arquivo gerado como resultado da fase de linkagem dos programas que compõem o projeto. Esse arquivo será utilizado na *simulação*;
- ◆ arquivo de mesmo nome, com a extensão *.hex*: arquivo para gravar a memória EPROM do microcontrolador, por meio de um programador de memórias EPROM.

A.2.4 – Simulação do programa: o objetivo é a visualização da condição dos registradores e posições de memória por meio da execução e do funcionamento passo a passo das instruções do programa projetado.

Durante a simulação, são corrigidos os erros de lógica do projeto do programa (de software), se existirem.

O programa de simulação que será utilizado é o *AVSIM51.exe*.

a) *Vantagens da simulação:*

- ◆ teste da funcionalidade do programa sem a necessidade do hardware do produto final;
- ◆ depuração (correção dos erros) do programa antes de sua implementação no produto final;
- ◆ redução de erros de programa quando ocorrer a implementação do programa no produto final.

Para simular o programa editado, compilado e linkado, crie a pasta chamada *avsim51*, na pasta *eletrica* (*CD eletrica*), utilizando o comando *MD avsim51*.

Para simular um programa digite:

```
c:\eletrica\avsim51>avsim51 - C1 (C1 em letras maiúsculas)
```

Digite A, que corresponde à utilização do modelo 8051, da família de microcontroladores da Intel, constituída de uma área de ROM/EPROM, RAM, E/Ss e registradores internos, como é mostrado na Figura A.1.



Figura A.1 Simulador do 8051 da AVOCET (AVSIM51).

Para carregar o programa editado, compilado e linkado para o simulador, proceda da seguinte maneira:

- ◆ digite na linha de comando: L (*Load*), A (*Avocet*) e *c:\eletrica\avc51\nome do programa* linkado (por exemplo, exemplo1 sem extensão);
- ◆ digite *ESCAPE* (*ESC*) para chavear o cursor da área de comandos do simulador AVSIM51 para a área interna do microcontrolador 8051, e vice-versa, quando o leitor pode modificar as condições dos registradores de funções especiais e conteúdos das posições de memória RAM;
- ◆ para que o programa apareça na área de memória ROM/EPROM do microcontrolador, acesse a área interna, utilizando a tecla *ESC*, e digite no campo do registrador de função especial PC (*Program Counter*) responsável por definir o endereço da próxima instrução a ser executada o mesmo valor que foi utilizado na instrução *ORG* do programa-fonte. Assim, o programa, além de ser visualizado na tela do simulador, também estará pronto para ser simulado.

Observação:

1 - *ORG* é uma diretriz do compilador Assembly, utilizada para definir o endereço inicial de memória de programa, em que um programa ou um trecho de programa será armazenado.

2 - Utilize os comandos do DOS para mudar de subdiretório:

- a) *CD nome do subdiretório*: vai para o subdiretório *c:\nome do subdiretório*.

Ex1: *c:>CD eletrica*: vai para o subdiretório (pasta) *eletrica*, ou seja, *c:\eletrica>*

Ex2: *c:\eletrica>CD avc51*: vai para o subdiretório *eletrica*, *avc51*, ou seja, *c:\eletrica\avc51*

- b) *CD..* : volta ao subdiretório anterior.

Ex.: *c:\eletrica\avc51>CD..* : volta para o subdiretório anterior, chamado *eletrica*, ou seja, *c:\eletrica*.

A.3 Procedimento experimental

O microcomputador em que o leitor executará as experiências pode ser dividido em subdiretórios da seguinte maneira:

C:\eletrica>: crie os seguintes subdiretórios/pastas: AVC51, AVEDIT, AVSIM51 e AVSIM85.

No subdiretório *c:\eletrica\avc51>* insira os seguintes programas: AVA51.exe (o programa compilador Assembly) e o AVL51.exe (programa Linkador Assembly) a partir do disquete fonte AVSIM51.

No subdiretório *c:\eletrica\avsim51>* insira o programa AVSIM51.exe (simulador da família de microcontroladores 80X51) a partir do disquete fonte do AVSIM51.

Procedimento para editar, compilar, linkar e simular um programa em Assembly, com base na família do 8051/32:

- o AVSIM51.exe não é executado no ambiente Windows. Assim, é necessário acessar o ambiente DOS. Para fazer isso, dê um clique em **Iniciar > Programas e Prompt do MS-DOS**;
- acesse o subdiretório *c:\eletrica\avc51>*;
- edite o programa a seguir a partir da sétima coluna, utilizando o editor de textos EDIT:

; Exemplo 1 – Microcontroladores – Simulação		
DEFSEG	exemplo1, absolute	; Define o segmento de exemplo1 como Absolute (área de memória de programa).
SEG	exemplo1	; Define o inicio do endereço de memória de programa do segmento exemplo1
ORG	0100h	; O programa será carregado a partir da posição de memória de programa 0100h
CLR	A	; (A) ← #00h [O conteúdo do acumulador (A) será armazenado com a constante #00h]. () = conteúdo e #=constante.
MOV	R2,A	; (R2) ← (A) [o conteúdo do registrador R2 será armazenado com o conteúdo do registrador acumulador A].
MOV	B,R2	; (B) ← (R2) [o conteúdo do registrador B será armazenado com o conteúdo do registrador R2].
END		; Diretriz do Assembly para indicar fim do programa (obrigatório).

- grave esse arquivo com o nome *exemplo1.asm* (a extensão .asm significa um programa escrito em Assembly) no subdiretório *c:\eletrica\avc51>*. Utilize o comando **Arquivo, Salvar como**, do editor de textos EDIT;
- compile o programa da seguinte maneira:

No subdiretório *c:\eletrica\avc51>*, digite *avc51 exemplo1.asm ALLPUBLIC* (ALLPUBLIC deve estar em letras maiúsculas).

A compilação gera dois arquivos. Veja e anote os nomes desses arquivos (comando *dir*) e observe o seu conteúdo (comando *type*). O conteúdo desses dois arquivos deve constar no relatório, e eles devem ser explicados.

- f) linke o programa como a seguir:

No subdiretório *c:\eletrica\avc51>* digite *avr51 exemplo1 = exemplo1.obj -SYMBOLS* (SYMBOLS deve ser escrito em letras maiúsculas).

Dessa linkagem, serão gerados mais três arquivos. Veja e anote os nomes desses arquivos (utilize o comando *dir*) e veja o seu conteúdo (comando *type 'nome do arquivo'*). O conteúdo desses três arquivos deve constar no relatório, descrevendo seu conteúdo e inter-relacionando com os outros dois arquivos gerados na fase de compilação.

A simulação do program a já está pronta para ser executada.

- g) faça uma simulação do programa:

- g.1) acesse o subdiretório *c:\eletrica\avsim51>* (utilize os comandos *CD..* e *CD\avsim51*) e digite

avsim51 -C1

Dessa maneira, o simulador de programas *AVSIM51*, da família de microcontroladores MCS-51 é acessado. Digite *A* para selecionar o tipo de microcontrolador que será utilizado. Nesse caso, trabalharemos com o simulador do microcontrolador 8051.

- g.2) carregue o programa para o simulador como a seguir:

Na área de comandos do simulador, digite *L (LOAD)*, *A (AVOCET)* e escreva o nome do programa linkado sem extensão e o caminho (*path*) no qual ele se encontra *c:\eletrica\avc51\exemplo1*.

Utilizando a tecla *ESC*, novamente, mude o conteúdo do registrador *Program Counter (PC)* para 0100_h . Assim, você visualizará o programa na área de memória de programa do simulador (ROM/EPROM).

- g.3) faça a simulação

A tecla *F10* realiza a simulação passo a passo, ou seja, executa instrução por instrução.

A próxima instrução a ser executada na memória de programa será sempre a que estiver indicada pelo endereço armazenado no conteúdo do registrador PC e visualizada em azul na instrução da memória do programa do microcontrolador.

A tecla *F9* volta para a condição anterior, após a execução de uma instrução, efetuada por meio da tecla *F10*.

A tecla *F1* executa o programa por inteiro.

No relatório, deve ser descrito o funcionamento de cada instrução que foi executada passo a passo. Para fazer isso, anote os valores dos conteúdos iniciais e finais dos seguintes registradores ((A), (B), (R2) e (PC)), cada vez que uma instrução for executada.

No relatório, devem ser descritas todas as etapas necessárias para executar a simulação de um determinado programa e responder os exercícios propostos, indicados a seguir.

Exercícios e questões propostos

- 1 - Quais são as etapas necessárias para a execução de um programa em Assembly? Explique cada uma delas.
- 2 - Como deve ser o formato de edição de um programa em Assembly? Dê um exemplo.
- 3 - O que é compilar um programa? Dê um exemplo de um comando para compilar um programa em Assembly.
- 4 - O que é linkar um programa? Dê um exemplo de um comando para linkar um programa em Assembly.
- 5 - Qual a finalidade principal da simulação de um programa? Quais são as vantagens da simulação? Qual é o comando empregado para carregar um programa em Assembly utilizando o simulador AVSIM51?
- 6 - Qual é a função das teclas *F1*, *F9* e *F10*, do simulador AVSIM51?

Kit Didático AES-51

I – Executando um programa em código de máquina no AES-51:

- 1 - Ligue a placa do AES-51.
- 2 - Pressione a segunda tecla (*2nd*) para entrar no programa monitor.
- 3 - Pressione novamente a tecla PROG (a mesma da segunda tecla *2nd*) para selecionar o modo de programa. No visor, aparecerá a seguinte informação:

*	PROG	7000	28672
	FF	255	11111111

(7000 = endereço inicial da memória RAM em hexadecimal e decimal)
 (conteúdo da posição de memória cujo endereço é 7000h em hexadecimal, decimal e binário)

- 4 - Caso o endereço não seja 7000h, pressione a tecla GOTO, digite 7000 e pressione a tecla ENTER. O endereço 7000h é o endereço inicial da RAM externa, em que se armazenam os programas a serem executados no kit de desenvolvimento AES-51.
- 5 - Pressione a segunda tecla para acessar o modo de entrada de dados (o caractere * desaparece do display de cristal líquido). Você pode utilizar as setas (\leftarrow , \rightarrow) para mover o cursor para o campo de dados binário, decimal e hexadecimal.
- 6 - Insira os dados mostrados na tabela a seguir. Você precisa concluir cada inserção de dados com a tecla ENTER. O endereço avança automaticamente após cada inserção de dados. Quando você quiser verificar outra posição de memória, simplesmente pressione a segunda tecla (2nd) e, utilizando as teclas de seta para cima (\uparrow) e para baixo (\downarrow), as outras posições de memória e seus respectivos conteúdos. Quando quiser alterar uma determinada posição, basta acionar a segunda tecla (2nd) novamente, para voltar para o modo de entrada de dados. Para finalizar a entrada de dados, pressione a segunda tecla (2nd) para voltar para o modo de programa.

Endereço	Código de máquina	Endereço	Mnemônicos
7000	75 90 0F	LTX:	MOV P1,#0FH
7003	11 0C		ACALL DELAY
7005	75 90 FF		MOV P1,#0FFH
7008	11 0C		ACALL DELAY
700A	80 F4		SJMP LTX
700C	78 00	DELAY:	MOV R0,#0
700E	79 00		MOV R1,#0
7010	7A 0B		MOV R2,#11
7012	C2 A8		CLR EX0
7014	D8 FE		DJNZ R0,\$
7016	D9 FC		DJNZ R1,\$ - 2
7018	DA FA		DJNZ R2,\$ - 4
701A	D2 A8		SETB EX0
701C	22		RET

Esse programa faz piscar quatro leds ligados aos quatro bits mais significativos da porta P1 do kit de desenvolvimento.

- 7 - Para fazer a verificação do que foi digitado, utilize as teclas de seta para cima (\uparrow) e para baixo (\downarrow).
- 8 - Pressione a tecla RUN. Digite 7000, se o endereço *from* não for 7000. Digite 701C, se o endereço *to* não for 701C (endereço inicial e final do programa digitado). Pressione a tecla RUN (ou qualquer outra tecla, exceto BREAK, IRAM e STEP) para executar o programa. Verifique se os leds estão piscando. Se não estiverem, repita as etapas 2, 3 e 6 para verificar os códigos de máquina em cada endereço e faça as correções necessárias.
- 9 - Pressione a tecla STEP para executar o programa passo a passo. Pressione a tecla ENTER duas vezes para inserir os endereços *from* e *to* e, em seguida, pressione STEP, até que o endereço inicial 7000 apareça.
- 10 - Pressione a tecla RUN para voltar para o modo de execução. Você pode repetir as etapas 8 e 9 quantas vezes desejar.
- 11 - Pressione a tecla BREAK para sair do monitor e voltar para o BASIC-52.
- 12 - Pressione a tecla RESET para iniciar o hardware.

II – Utilizando o PC para executar o programa por meio de uma transferência por download:

- 1 - Edite, compile e linke o programa anterior e denomine-o como *lights.asm*.
- 2 - Conecte o cabo no canal de comunicação serial do PC e no kit de desenvolvimento AES-51.
- 3 - Acesse o subdiretório (pasta AES-51), execute o programa *TE.exe*, digitando.
- 4 - Configure o emulador de terminal TE:
 - 4.1 - acesse o DOS (por meio dos comandos: **Iniciar > Programas > Prompt do DOS**). Acesse o subdiretório (pasta) AES-51 utilizando os comandos *CD..* e *CD AES-51*. Execute o programa *TE.exe*, digitando *TE*. Pressione *Alt-c* para exibir a configuração do programa TE. Em seguida, forneça as seguintes definições:

```
COM1, 9600, N, 8, 1, N, N, Y, Y, >
```

Finalizada a configuração, é gerado um arquivo chamado *TEPARAM.CFG*, que contém as informações de configuração.

- 4.2 - pressione *ESC* para sair da configuração;
- 4.3 - pressione *Alt-x* para sair do *TE*.
- 5 - Execute o AES-51 utilizando o PC como um computador central.

- 6 - No DOS, execute *TE*. Energize o AES-51 e pressione a barra de espaços do PC. O AES-51 responderá com a seguinte mensagem na tela do computador:

* MCS-51 (tm) Basic V1.1b*

Ready

>

- 7 - Faça o download do programa lights e execute um arquivo hexadecimal:

- 7.1 - digite RX, CR (tecla *ENTER*) e *Alt-S*. Digite o nome do arquivo *light.hex*. Isso carregará o programa no conteúdo do endereço 7000h da memória RAM do AES-51;
- 7.2 - digite *call 7000h*. O programa será executado por meio do kit de desenvolvimento AES-51.

III – Arquitetura da memória do AES-51:

- ◆ 32K bytes de memória ROM (0000h a 03FFFh);
- ◆ 32K bytes de memória RAM (4000h a 7FFFh). Utilize a memória RAM, a partir da posição de memória 5100h;

IV – Endereçamento dos dispositivos de entrada e saída (E/S mapeados em memória):

- ◆ display de cristal líquido: FFF3h;
- ◆ teclado: FFF4h;
- ◆ segunda porta serial: FFF5h;
- ◆ primeira porta serial padrão: FFF6h;
- ◆ conversor analógico/digital: FFF7h.

V – Modo monitor:

- ◆ acione a segunda tecla para acessar o modo monitor;
- ◆ a função *IRAM* verifica as 256 posições de memória internas (00h a FFh) do processador 8032. Utilize as teclas de seta para cima e para baixo para verificar as posições de memória. Os SFRs são verificados, observando-se as seguintes posições de memória:

- ◆ (D0h) → (A)
- ◆ (D1h) → (B)
- ◆ (D2h) → (DPL)

- ◆ (D3h) → (DPH)
 - ◆ (D4h) → (SP)
 - ◆ (D5h) → (P1)
 - ◆ (D6h) → (P3)
 - ◆ (D7h) → (PSW)
 - ◆ (D8h) → (IE)
 - ◆ (D9h) → (IP)
 - ◆ (DAh) → (SCON)
 - ◆ (DBh) → (SBUF)
 - ◆ (DCh) → (PCON)
 - ◆ (DDh) → (TCON)
 - ◆ (DEh) → (TMOD)
 - ◆ (DFh) → (TL0)
 - ◆ (E0h) → (TH0)
 - ◆ (E1h) → (TL1)
 - ◆ (E2h) → (TH1)
 - ◆ (E3h) → (T2COM)
 - ◆ (E4h) → (T2MOD)
 - ◆ (E5h) → (TL2)
 - ◆ (E6h) → (TH2)
 - ◆ (E7h) → (RCAP2L)
 - ◆ (E8h) → (RCAP2H)
 - ◆ (E9h) → (P0)
 - ◆ (EAh) → (P2)
-
- ◆ a função *RUN* executa um conjunto de códigos de programa pré-carregados. Ela pode ser selecionada a partir do monitor, *STEP* e *PROG*, acionando-se a tecla *RUN*. Para sair, pressione a tecla *BREAK*;
 - ◆ a função *STEP* executa passo a passo um conjunto de códigos de programa pré-carregados por acionar *STEP*. Selecione *STEP* do Monitor, *RUN* e *PROG* pressionando a tecla *STEP*. Para sair pressione a tecla *BREAK*;
 - ◆ a função *PROG* (programa) serve para editar e ler a memória de programa. Utilize a segunda tecla para alternar entre a escrita e a leitura. *PROG* pode ser selecionada a partir do monitor ao acionar *PROG*. Você pode

selecionar RUN, STEP e IRAM durante a leitura. Para sair, pressione a tecla BREAK.

VI – Descrição do hardware:

São oito blocos funcionais:

- 1 - Microprocessador e memória
- 2 - Lógica de controle do sistema
- 3 - Controle de entrada de teclado
- 4 - Display de cristal líquido
- 5 - Fonte de energia
- 6 - Entrada digital
- 7 - E/S analógica
- 8 - E/S de dados seriais

VII – Descrição dos blocos funcionais:

- 1 - 80C32 (U1) - microcontrolador: 16 bits de endereços capaz de endereçar 64 Kbytes de memória. O barramento dos dados e dos 8 bits de endereço são multiplexados no tempo AD0-7. Sua freqüência de operação é de 11.0592 Mhz.
- 2 - 75HC373 (U2) - *Latch tri-state*.
- 3 - 27C256 - EPROM CMOS de 32Kx2.
- 4 - 62256 (U4) - RAM Estática CMOS de 32Kx8.

Executando um programa com o kit didático AES-51, utilizando o compilador (AVA51.exe) e o linkador (AVL51.exe) da AVOCET

O programa a seguir tem a finalidade de calcular a quantidade de números iguais a 0FFh do buffer de memória que vai do endereço de memória 60h até 64h. O resultado é colocado na posição de memória 65h. Esse programa utiliza o conceito de sub-rotina.

```

defseg    exemplo8,absolute
seg      exemplo8
;
org      70ffh
sub1:   mov      r0,#60h
        mov      r1,#05h
        mov      65h,#00h
adr2:   clr      c
        mov      a,@r0
        subb   a,#0ffh
        jnz     adr1
        inc      65h
adr1:   inc      r0
        djnz   r1,adr2
        ret
;
org      7000h
acall   sub1
mov      a,65h
ret
end

```

- 1 - Edite, compile e linke o programa mostrado anteriormente.
- 1.1 - encerre o Windows e acesse o DOS: Iniciar, Prompt do DOS;
 - 1.2 - com os comandos *CD..* , *CD AES-51*, execute o programa terminal *Emulator* (Emulador), digitando TE (Enter), a partir do kit didático AES-51;
 - 1.3 - desligue e ligue o kit didático AES-51, pressione duas vezes a barra de espaços do PC. A seguinte mensagem aparecerá em seu monitor de vídeo:

TERMINAL EMULATOR (c) 1993
 MCS-51(tm)BASIC V1.1b
 READY

AES SYSTEMS, INC.

Digite *RX* e pressione a tecla *ENTER*:

>RX

- 1.4 - pressione *ALT-S*: na tela do PC, será aberta uma janela solicitando o nome do arquivo *.hexa*, que foi compilado e linkado. Digite *c:\eletrica\avc51\exemplo.hex*.

Na tela do PC, será mostrado o conteúdo do arquivo *exemplo.hex*, que foi carregado no kit didático, como é mostrado a seguir:

```
#>:1370FF0078607905756500C3E694FF7002056508D9F5223E
#>:057000001FFE565220F
#>:000000001FF
READY
```

1.5 - com o comando *CI* (*Change Internal RAM* – que modifica o conteúdo da RAM interna), digite *CI 60H* e pressione *Enter*:

```
>CI 60H _
```

Na tela do PC, é mostrado o conteúdo atual da posição de memória 60h. Mude para 0FFH e pressione *Enter*.

Após isso, será mostrado o conteúdo da próxima posição de memória. Modifique essas posições de memória com os valores dados a seguir até a posição de memória 64h.

Para finalizar o processo de alteração do conteúdo da posição de memória, pressione *Ctrl-C*.

```
60H: 00H 0FFH
61H: 00H 0F4H
62H: 00H 0FFH
63H: 00H 0ABH
64H: 00H 0FFH
READY
```

1.6 - para executar o programa no AES-51, usando o terminal de emulação, digite *CALL 7000H* e pressione *Enter*;

```
>CALL 7000H
```

1.7 - para verificar o resultado gerado pelo programa correspondente à quantidade de números iguais a 0FFh, que foi armazenado na posição de memória 65H, digite *DI 60H,65H* e pressione ENTER.

```
>DI 60H,65H
60H: FFH F4H FFH ABH 00H 03H 00H 00H 00H 00H 00H 00H 00H 00H 00H
READY
>
F1 = Help | COM2 9600,N,8,1 | Echo: N | Add LF: N | Strip LF: Y | Prompt Line: Y
```

EXPERIÊNCIA 02: MODOS DE ENDEREÇAMENTO DA FAMÍLIA DE MICROCONTROLADORES MCS-51 DA INTEL

B.1 Objetivo

- ❖ Simulação dos diferentes tipos de endereçamento da família de microcontroladores MCS-51

B.2 Introdução teórica

Veja o Capítulo 3.

B.3 Procedimento experimental

Edita, compile, linke e simule o programa a seguir. Execute as instruções passo a passo mostrando as condições iniciais e finais dos registradores e as posições de memória que são afetadas pelas instruções.

Para auxiliá-lo na execução passo a passo das instruções, utilize as tabelas do Capítulo 3 para verificar quais registradores e posições de memória são afetados.

Observação: no relatório, deverão constar todos os registradores e as posições de memória que são alterados durante a execução do programa. Não se esqueça de anotar o valor do conteúdo do registrador *Program Counter* (PC).

No relatório, deverão constar as respostas às perguntas efetuadas no decorrer deste Apêndice.

Incialize os registradores e as posições de memória, digitando as condições iniciais, mostradas a seguir.

Registradores internos à CPU:

(A)= 8Ch (B)=0ACh (PSW)=32h (PC)=0100h (SP)=20h (P0)=0AAh (P1)=55h (P2)=28h (P3)=0AEh.

Posições de memória:

(00h) = 21h (01h) = 25h (02h) = 0E2h (03h) = 6Dh (04h) = 1Ch (05h) = 0F2h (06h) = 9Ah (07h) = 18h
 (08h) = 30h (09h) = 32h (0Ah) = 0Fh (0Bh) = 2Ah (0Ch) = 4Eh (0Dh) = 0F2h (0Eh) = 0F1h (0Fh) = 09h
 (10h) = 43h (11h) = 44h (12h) = 0E2h (13h) = 6Dh (14h) = 1Ch (15h) = 0F2h (16h) = 9Ah (17h) = 18h
 (18h) = 56h (19h) = 57h (1Ah) = 0Fh (1Bh) = 2Ah (1Ch) = 4Eh (1Dh) = 0F2h (1Eh) = 0F1h (1Fh) = 09h
 (20h) = 2Eh (21h) = 2Bh (22h) = 2Ah (23h) = 2Dh (24h) = 23h (25h) = 22h (26h) = 2Ah (27h) = 91h
 (28h) = 0Ah (29h) = 0Bh (2Ah) = 0CFh (2Bh) = 0DAh (2Ch) = 0EEh (2Dh) = 0FFh (2Eh) = 6Fh (2Fh) = 49h
 (30h) = 77h (31h) = 0Ah (32h) = 0Bh (33h) = 0Fh (34h) = 2Ah (35h) = 4Eh (36h) = 0F2h (37h) = 0F1h
 (38h) = 09h (39h) = 2Eh (3Ah) = 0ABh (3Bh) = 5Eh (3Ch) = 4Dh (3Dh) = 6Ch (3Eh) = 72h (3Fh) = 8Ah
 (40h) = 1Ah (41h) = 8Dh (42h) = 0Eh (43h) = 6Dh (44h) = 1Ch (45h) = 02h (46h) = 9Ah (47h) = 18h
 (48h) = 0Ah (49h) = 0Bh (4Ah) = 0Fh (4Bh) = 2Ah (4Ch) = 4Eh (4Dh) = 0F2h (4Eh) = 0F1h (4Fh) = 09h
 (50h) = 1Ah (51h) = 99h (52h) = 0E2h (53h) = 6Dh (54h) = 1Ch (55h) = 0F2h (56h) = 9Ah (57h) = 18h
 (58h) = 0Ah (59h) = 0Bh (5Ah) = 0Fh (5Bh) = 2Ah (5Ch) = 4Eh (5Dh) = 0F2h (5Eh) = 0F1h (5Fh) = 09h
 (60h) = 2Eh (61h) = 0ABh (62h) = 5Eh (63h) = 4Dh (64h) = 6Ch (65h) = 72h (66h) = 8Ah (67h) = 91h
 (68h) = 0Ah (69h) = 0Bh (6Ah) = 0CFh (6Bh) = 0DAh (6Ch) = 0EEh (6Dh) = 0FFh (6Eh) = 6Fh (6Fh) = 49h
 (70h) = 77h (71h) = 0Ah (72h) = 0Bh (73h) = 0Fh (74h) = 2Ah (75h) = 4Eh (76h) = 0F2h (77h) = 11h
 (78h) = 19h (79h) = 2Eh (7Ah) = 0Bh (7Bh) = 5Eh (7Ch) = 4Dh (7Dh) = 6Ch (7Eh) = 72h (7Fh) = 8Ah

DEFSEG EXEMPLO2, ABSOLUTE

SEG	EXEMPLO2	
ORG	0100h	; Define o endereço de memória de programa, em que o programa ; será armazenado (ROM/EPROM), ou seja, a instrução MOV PSW, #00h ; será armazenada no endereço de memória 0100h (ORG é uma diretriz do ; Assembly).
MOV	PSW,#00h	; (PSW) ← #00h = 0000 0000 _b . O conteúdo do registrador PSW é ; armazenado com a constante (#) 00h. Define: CY = 0, AC = 0, F0 = 0, RS1 = 0, ; RS0 = 0, OV = 0 e P = 0; ; Perguntas: qual banco foi selecionado? Quais registradores foram ; selecionados?
MOV	R7,A	; Represente simbolicamente essa instrução. Descreva em palavras essa ; instrução. Quais são os registradores afetados por essa instrução? Qual é ; o modo de endereçamento dessa instrução?
MOV	R6,3Fh	; Idem ao anterior.
MOV	R5,#6Dh	; Idem ao anterior.
MOV	55h,R3	; Idem ao anterior.
MOV	61h,75h	; Idem ao anterior.
MOV	@R0,#3Ah	; Idem ao anterior.
MOV	@R1,71h	; Idem ao anterior.
MOV	0E0h,B	; Como ficou representada essa instrução no simulador ; AVSIM51? ; Idem ao anterior. ; Idem ao anterior.
ADD	A,R4	; Represente simbolicamente essa instrução. Descreva em palavras essa ; instrução. Quais são os registradores afetados por essa instrução? ; Justifique o resultado. Qual é o modo de endereçamento dessa instrução?
ADDC	A,38h	; Idem ao anterior.
SUBB	A,@R1	; Idem ao anterior.
XRL	A,#55h	; Idem ao anterior.

ANL	32h,A	; Idem ao anterior.
ORL	44h,A	; Idem ao anterior.
MOV	07h,C	; (07h) = (20h.7) ← (C). O conteúdo do bit cujo endereço é 07h, que é o ; conteúdo do bit 7 da posição de memória 20h, será armazenado com o ; conteúdo do <i>carry bit</i> . Pergunta: por que (07h) = (20h.7)? Justifique. ; Qual o modo de endereçamento dessa instrução?
MOV	C, 00h	; (C) ← (00h). ; Descreva detalhadamente em palavras essa instrução. Em que bit e ; posição de memória está localizado o bit de endereço 00h? Represente o ; bit 00h na notação de endereço de memória e posição do bit 2qh.r. ; Qual é o modo de endereçamento dessa instrução?
MOV	6Dh, C	; Pergunta: qual é a simbologia dessa função? Qual é a descrição sucinta, ; em palavras, dessa instrução? Que endereço de memória e que bit dessa ; posição de memória estão sendo acessados? Represente isso no formato ; 2qh.r. ; Qual é o modo de endereçamento dessa instrução?
END		

PÁGINA EM BRANCO

EXPERIÊNCIA 03:

FLUXOGRAMA E PROGRAMAÇÃO EM ASSEMBLY APLICADOS À FAMÍLIA DE MICROCONTROLADORES MCS-51 DA INTEL

C.1 Objetivos

- ❖ Entendimento dos fluxogramas
- ❖ Simulação de exemplos de programas simples e com loop em Assembly, utilizando a família de microcontroladores MCS-51

C.2 Introdução teórica

Veja o Capítulo 4.

C.3 Procedimento experimental

- ◆ edite (EDIT do DOS), compile (AVA51.exe) e linke (AVL51.exe) o programa a seguir;
- ◆ carregue no simulador o programa compilado e linkado (L, A, nome do programa linkado sem extensão);
- ◆ antes de simular o programa, inicialize os registradores e as posições de memória com as condições iniciais fornecidas a seguir;
- ◆ execute as instruções passo a passo.

Observação: no relatório deverão constar todos os registradores e as posições de memória alterados durante a execução do programa. Não se esqueça de anotar o valor do conteúdo do registrador PC. Responda às perguntas, quando for solicitado.

Considere as seguintes condições iniciais:

Registradores internos à CPU:

(A) = 8Ch (B) = 0ACh (PSW) = 32h (PC) = 0100h (SP) = 20h (P0) = 0AAh (P1) = 55h (P2) = 28h (P3) = 0AEh.

Posições de memória:

(00h) = 21h (01h) = 25h (02h) = 0E2h (03h) = 6Dh (04h) = 1Ch (05h) = 0F2h (06h) = 9Ah (07h) = 18h
 (08h) = 30h (09h) = 32h (0Ah) = 0Fh (0Bh) = 2Ah (0Ch) = 4Eh (0Dh) = 0F2h (0Eh) = 0F1h (0Fh) = 09h
 (10h) = 43h (11h) = 44h (12h) = 0E2h (13h) = 6Dh (14h) = 1Ch (15h) = 0F2h (16h) = 9Ah (17h) = 18h
 (18h) = 56h (19h) = 57h (1Ah) = 0Fh (1Bh) = 2Ah (1Ch) = 4Eh (1Dh) = 0F2h (1Eh) = 0F1h (1Fh) = 09h
 (20h) = 2Eh (21h) = 2Bh (22h) = 5Eh (23h) = 2Dh (24h) = 2Ch (25h) = 22h (26h) = 2Ah (27h) = 29h
 (28h) = 0AAh (29h) = 0BCh (2Ah) = 0CFh (2Bh) = 0DAh (2Ch) = 0EEh (2Dh) = 0FFh (2Eh) = 6Fh (2Fh) = 49h
 (30h) = 77h (31h) = 0Ah (32h) = 0Bh (33h) = 0Fh (34h) = 2Ah (35h) = 4Eh (36h) = 0F2h (37h) = 11h
 (38h) = 09h (39h) = 2Eh (3Ah) = 0ABh (3Bh) = 5Eh (3Ch) = 4Dh (3Dh) = 6Ch (3Eh) = 72h (3Fh) = 8Ah
 (40h) = 1Ah (41h) = 8Dh (42h) = 0E2h (43h) = 6Dh (44h) = 1Ch (45h) = 0F2h (46h) = 9Ah (47h) = 18h
 (48h) = 0Ah (49h) = 0Bh (4Ah) = 0Fh (4Bh) = 2Ah (4Ch) = 4Eh (4Dh) = 0F2h (4Eh) = 0F1h (4Fh) = 09h
 (50h) = 1Ah (51h) = 99h (52h) = 0E2h (53h) = 6Dh (54h) = 1Ch (55h) = 0F2h (56h) = 9Ah (57h) = 18h
 (58h) = 0Ah (59h) = 0Bh (5Ah) = 0Fh (5Bh) = 2Ah (5Ch) = 4Eh (5Dh) = 0F2h (5Eh) = 0F1h (5Fh) = 09h
 (60h) = 2Eh (61h) = 0ABh (62h) = 5Eh (63h) = 4Dh (64h) = 6Ch (65h) = 72h (66h) = 8Ah (67h) = 91h
 (68h) = 0AAh (69h) = 0BCh (6Ah) = 0CFh (6Bh) = 0DAh (6Ch) = 0EEh (6Dh) = 0FFh (6Eh) = 6Fh (6Fh) = 49h
 (70h) = 77h (71h) = 0Ah (72h) = 0Bh (73h) = 0Fh (74h) = 2Ah (75h) = 4Eh (76h) = 0F2h (77h) = 0F1h
 (78h) = 09h (79h) = 2Eh (7Ah) = 0ABh (7Bh) = 5Eh (7Ch) = 4Dh (7Dh) = 6Ch (7Eh) = 72h (7Fh) = 8Ah

- 1 - Faça um programa (fluxograma e programa-fonte) que efetua a operação de adição entre 2 bytes localizados nos conteúdos das posições de memória, cujos endereços são 45h e 5Ah. O resultado deve ser armazenado no conteúdo da posição de memória cujo endereço é 6Dh.

Solução:

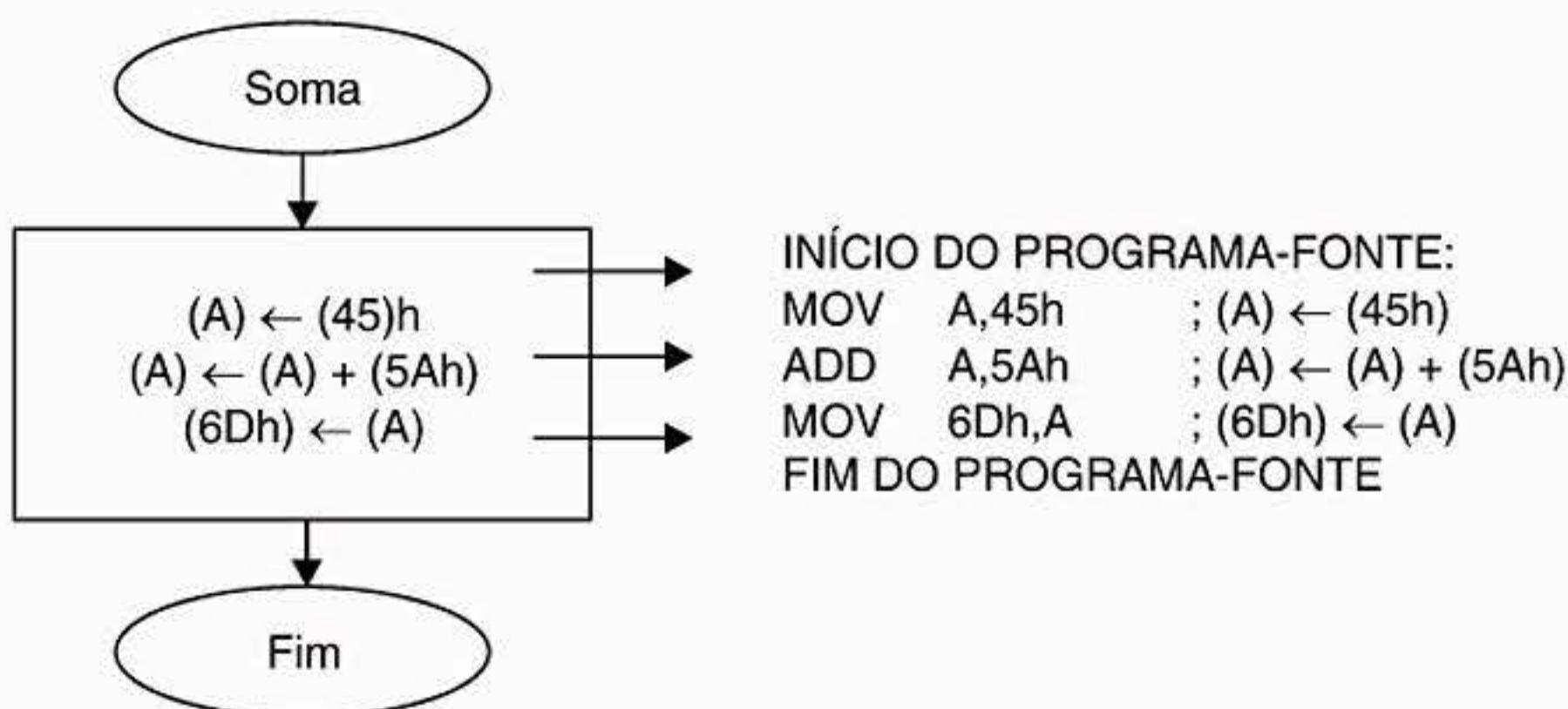


Figura C.2 Fluxograma de um programa que efetua a edição entre 2 bytes.

- 2 - O segundo programa calcula a quantidade de números menores que 38h de uma faixa de memória que vai da posição de memória cujo endereço inicial é 60h e o final é 7Ah. O resultado é colocado no conteúdo da posição de memória cujo endereço é 7Bh.

Solução:

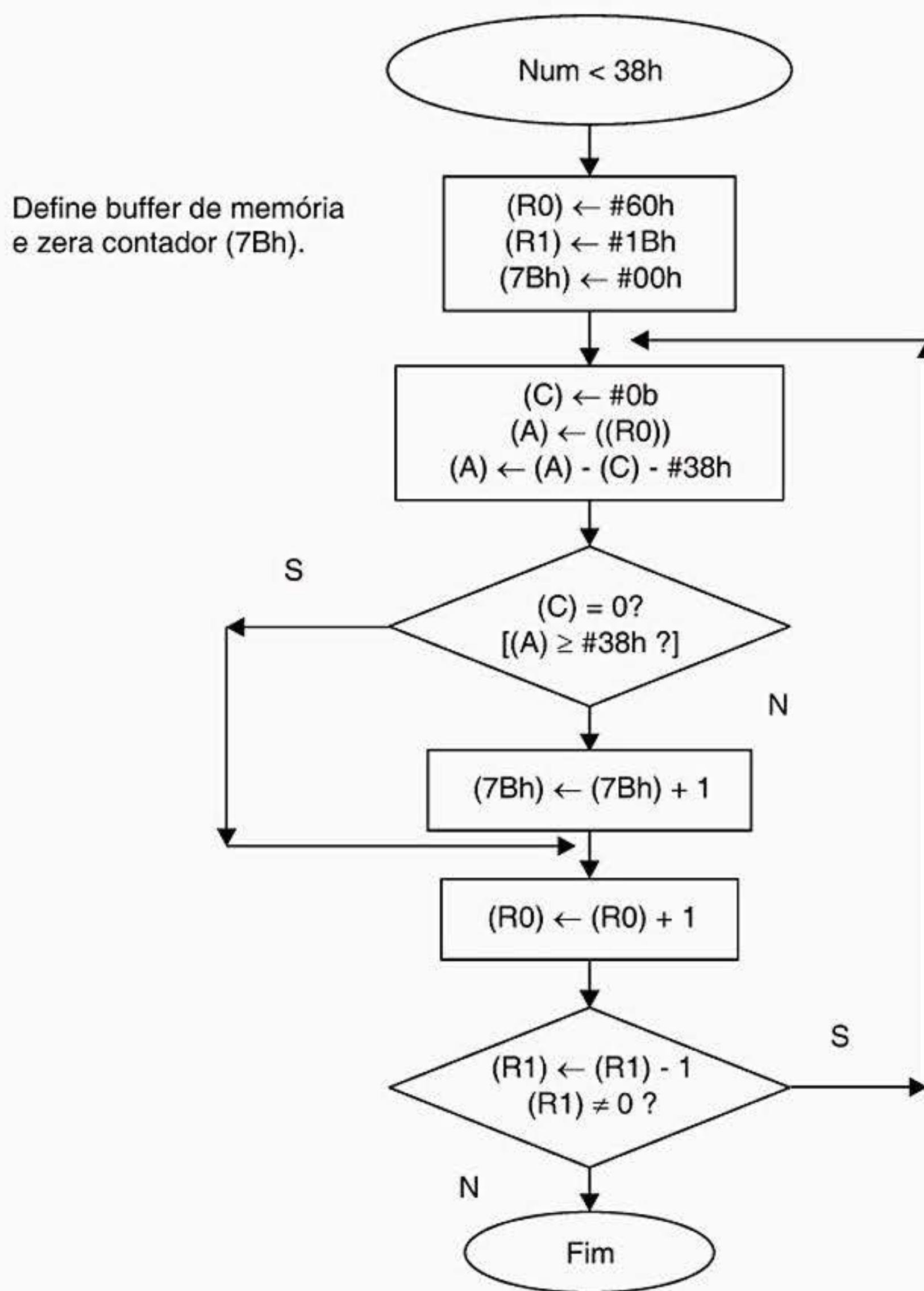


Figura C.3 Fluxograma de um programa que calcula a quantidade de números menores que $38h$.

Programa-fonte:

;Início do programa que calcula a quantidade de elementos < $38h$

```

MOV R0,#60h      ; Posição inicial do buffer
MOV R1,#1Bh      ; Quantidade de elementos do buffer
MOV 7Bh,#00      ; 'Zera' a posição de memória que contém a quantidade de números < #38h
ADR2: CLR C      ; 'Zera' o carry bit para não influenciar na operação de subtração
    MOV A,@R0      ; Coloca no acumulador o conteúdo da posição de memória do buffer
    SUBB A,#38h    ; Subtrai o (A) com a constante #38h para definir o flag (C)
    JNC ADR1       ; Se (C) = 0 ( $A \geq #38h$ ) => (PC) = ADR1 (não adiciona um ao (7Ah))
    INC 7Bh        ; Se (C) = 1 ( $A < #38h$ ), adiciona um ao (7Bh)
ADR1: INC R0      ; Aponta para a próxima posição de memória a ser analisada
    DJNZ R1,ADR2   ; Se a quantidade de elementos a serem analisados for '0',
                    ; então salta para ADR2
END              ; Termina o programa, se (R1) = 0.
    
```

A simulação deve consistir em:

- ◆ teste de três números: um menor que 38h, um igual a 38h e um maior que 38h;
- ◆ teste do processamento (fazer, teoricamente, a operação de subtração e obter o resultado dos *flags* de sinalização no PSW para cada uma das condições de teste (menor, igual e maior que 38h). Comparar com os resultados obtidos na simulação;
- ◆ teste das variáveis de controle do buffer de memória por meio dos registradores R0 e R1. Verificando todas as vezes que o programa desvia, até não desviar mais para o início do programa e até que ocorra a finalização do mesmo.

C.4 Para o relatório

- 1 - Descreva com suas palavras o fluxograma da solução de cada um dos programas.
- 2 - Descreva com suas palavras o programa-fonte.
- 3 - Apresente o resultado da simulação passo a passo, mostrando os registradores de funções especiais e conteúdos de posições alterados durante a execução do programa. Não esquecer o conteúdo do PC e do PSW e compará-los com os resultados obtidos na simulação.
- 4 - Descreva sucintamente como o programa resolveu o problema por meio do fluxograma.
- 5 - Responda às questões teóricas do Capítulo 4 do livro.

EXPERIÊNCIA 04: SUB-ROTINA E ESTRUTURAÇÃO DA LINGUAGEM ASSEMBLY APLICADOS À FAMÍLIA DE MICROCONTROLADORES MCS-51 DA INTEL

D.1 Objetivo

- ❖ Simulação de programas estruturados em Assembly, utilizando sub-rotinas para a família de microcontroladores MCS-51 da Intel

D.2 Introdução teórica

Veja o Capítulo 5.

D.3 Procedimento experimental

- ◆ edite (EDIT do DOS), compile (AVA51.exe) e linke (AVL51.exe) o programa que se segue;
- ◆ carregue o programa compilado e linkado no simulador AVSIM51.exe (L, A, nome do programa linkado sem extensão);
- ◆ antes de simular o programa, inicialize os registradores e as posições de memória com as condições iniciais fornecidas a seguir;
- ◆ execute as instruções passo a passo. Enfatize a descrição detalhada do funcionamento das instruções ACALL, LCALL e RET.

Observação: no relatório, deverão constar todos os registradores e as posições de memória que são alterados durante a execução do programa. Não se esqueça de anotar o valor do conteúdo do registrador PC (PC). Responder às perguntas, quando for solicitado.

Considere as seguintes condições iniciais:

Registradores internos à CPU:

(A) = 8CH (B) = 0ACH (PSW) = 32H (PC) = 0100H (SP) = 20H (P0) = 0AAH (P1) = 55H (P2) = 28H (P3) = 0AEH.

Posições de memória:

(00H) = 21H (01H) = 25H (02H) = 0E2H (03H) = 6DH (04H) = 1CH (05H) = 0F2H (06H) = 9AH (07H) = 18H
 (08H) = 30H (09H) = 32H (0AH) = 0FH (0BH) = 2AH (0CH) = 4EH (0DH) = 0F2H (0EH) = 0F1H (0FH) = 09H
 (10H) = 43H (11H) = 44H (12H) = 0E2H (13H) = 6DH (14H) = 1CH (15H) = 0F2H (16H) = 9AH (17H) = 18H
 (18H) = 56H (19H) = 57H (1AH) = 0FH (1BH) = 2AH (1CH) = 4EH (1DH) = 0F2H (1EH) = 0F1H (1FH) = 09H
 (20H) = 2EH (21H) = 0ABH (22H) = 5EH (23H) = 4DH (24H) = 6CH (25H) = 72H (26H) = 8AH (27H) = 91H
 (28H) = 0AH (29H) = 0CH (2AH) = 0CFH (2BH) = 0DAH (2CH) = 0EEH (2DH) = 0FFH (2EH) = 6FH (2FH) = 49H
 (30H) = 77H (31H) = 0AH (32H) = 0BH (33H) = 0FH (34H) = 2AH (35H) = 4EH (36H) = 0F2H (37H) = 0F1H
 (38H) = 09H (39H) = 2EH (3AH) = 0ABH (3BH) = 5EH (3CH) = 4DH (3DH) = 6CH (3EH) = 72H (3FH) = 8AH
 (40H) = 1AH (41H) = 8DH (42H) = 0E2H (43H) = 6DH (44H) = 1CH (45H) = 0F2H (46H) = 9AH (47H) = 18H
 (48H) = 0AH (49H) = 0BH (4AH) = 0FH (4BH) = 2AH (4CH) = 4EH (4DH) = 0F2H (4EH) = 0F1H (4FH) = 09H
 (50H) = 1AH (51H) = 99H (52H) = 0E2H (53H) = 6DH (54H) = 1CH (55H) = 0F2H (56H) = 9AH (57H) = 18H
 (58H) = 0AH (59H) = 0BH (5AH) = 0FH (5BH) = 2AH (5CH) = 4EH (5DH) = 0F2H (5EH) = 0F1H (5FH) = 09H
 (60H) = 2EH (61H) = 0ABH (62H) = 5EH (63H) = 4DH (64H) = 6CH (65H) = 72H (66H) = 8AH (67H) = 91H
 (68H) = 0AH (69H) = 0BH (6AH) = 0CFH (6BH) = 0DAH (6CH) = 0EEH (6DH) = 0FFH (6EH) = 6FH (6FH) = 49H
 (70H) = 77H (71H) = 0AH (72H) = 0BH (73H) = 0FH (74H) = 2AH (75H) = 4EH (76H) = 0F2H (77H) = 0F1H
 (78H) = 09H (79H) = 2EH (7AH) = 0ABH (7BH) = 5EH (7CH) = 4DH (7DH) = 6CH (7EH) = 72H (7FH) = 8AH

1 - A seguir, são apresentados de forma estruturada o programa-fonte e o fluxograma, que calculam a quantidade de números iguais a OFFh de uma faixa de memória que vai da posição de memória 60h a 64h. O resultado é colocado no conteúdo da posição de memória cujo endereço é 65h. Esse programa utiliza sub-rotinas e estruturou a linguagem Assembly para a linguagem de programação estruturada.

```

DEFSEGEXAMPLE,      ABSOLUTE
SEG               EXEMPLO
ORG              0100h

; Sub-rotina que calcula a quantidade de números = OFFh
SUB1: MOV     R0,#60h    ; Posição inicial do buffer
      MOV     R1,#05h    ; Quantidades de elementos do buffer
      MOV     65h,#00    ; 'Zera' o contador de números = OFFh
ADR2: CLR     C         ; 'Zera' o (C): não influenciar a operação de subtração
      MOV     A,@R0      ; Carrega (A) com o conteúdo do buffer de memória
      SUBB   A,#0FFh    ; Subtrai o (A) da constante OFFh
      JNZ    ADR1       ; Se (A) ≠ 0 (A ≠ #0FFh) => (PC) = ADR1 [não adiciona um ao (65h)]
      INC    65h        ; Se (A) = 0 (A = #0FFh) adiciona um ao (65h)
ADR1: INC    R0         ; Aponta para a próxima posição de memória a ser analisada
      DJNZ   R1,ADR2    ; Existem mais dados a serem analisados? Se houver, salta para ADR2
      RET                ; Se não houver, finaliza a sub-rotina se (R1) = 0 e retorna.

; Início do programa principal
PROGP: MOV    PSW,#00h   ; Faz (PSW) ← #00h.
      MOV    SP,#70h    ; Define (SP) = 70H (início da Pilha)
      ACALL  SUB1      ; Chama a sub-rotina SUB1.
      END               ; Fim do programa principal

```

Figura D.1(a) Programa-fonte e fluxograma de uma rotina que calcula a quantidade de números iguais a OFFh.

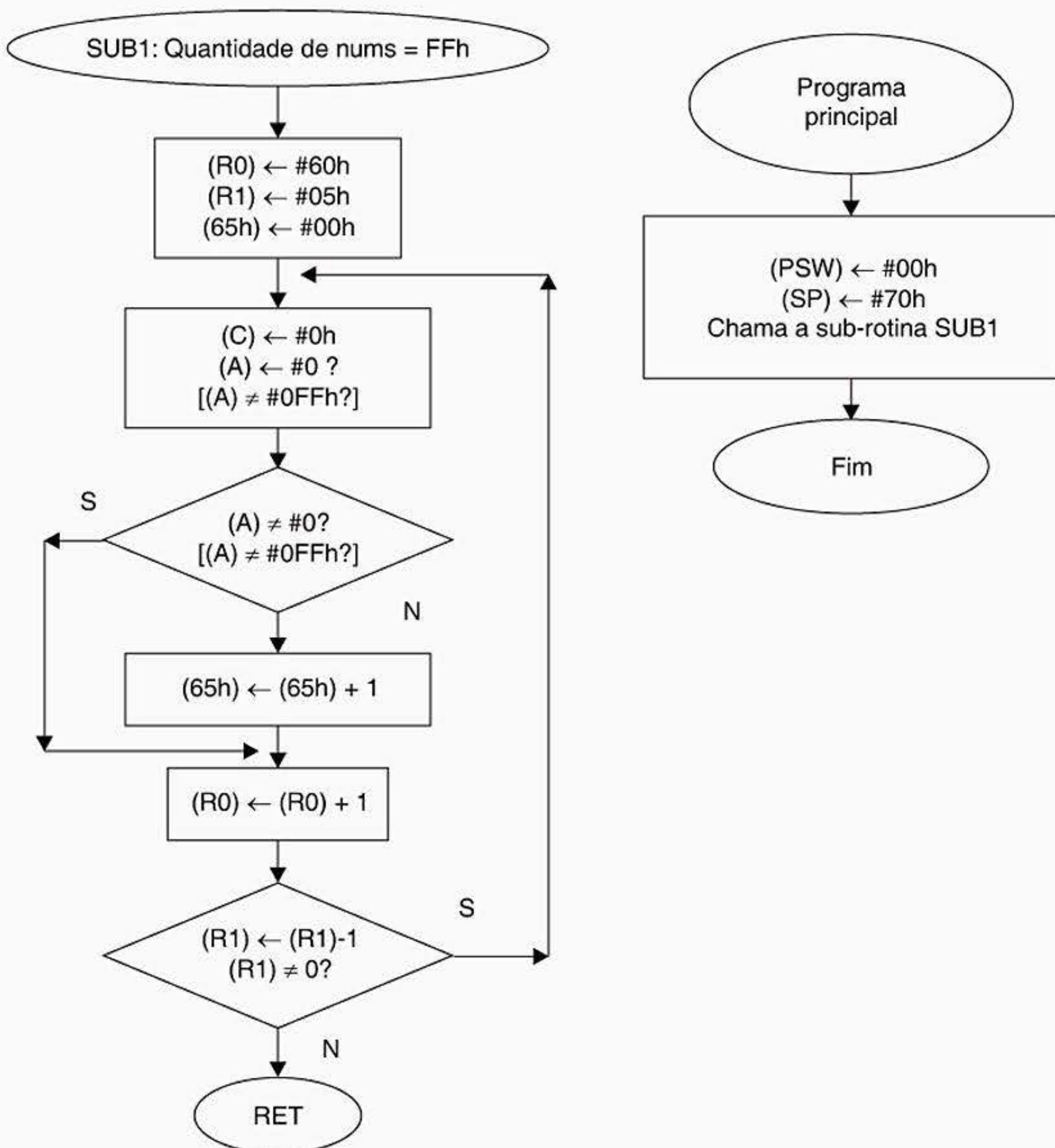


Figura D.1(b) Fluxograma de uma rotina que calcula a quantidade de números iguais a OFFh.

2 - A seguir, são apresentados de forma estruturada o fluxograma e o programa-fonte, que fazem a adição dos bytes do conteúdo das posições de memória cujos endereços inicial e final são 40h a 50h. Considere que o resultado dessa adição pode ser maior que 255_{10} . Dessa maneira, a parte menos significativa do resultado deve ser armazenada no conteúdo da posição de memória cujo endereço é 51h, e a parte mais significativa do resultado deve ser armazenada no conteúdo da posição de memória cujo endereço é 52h.

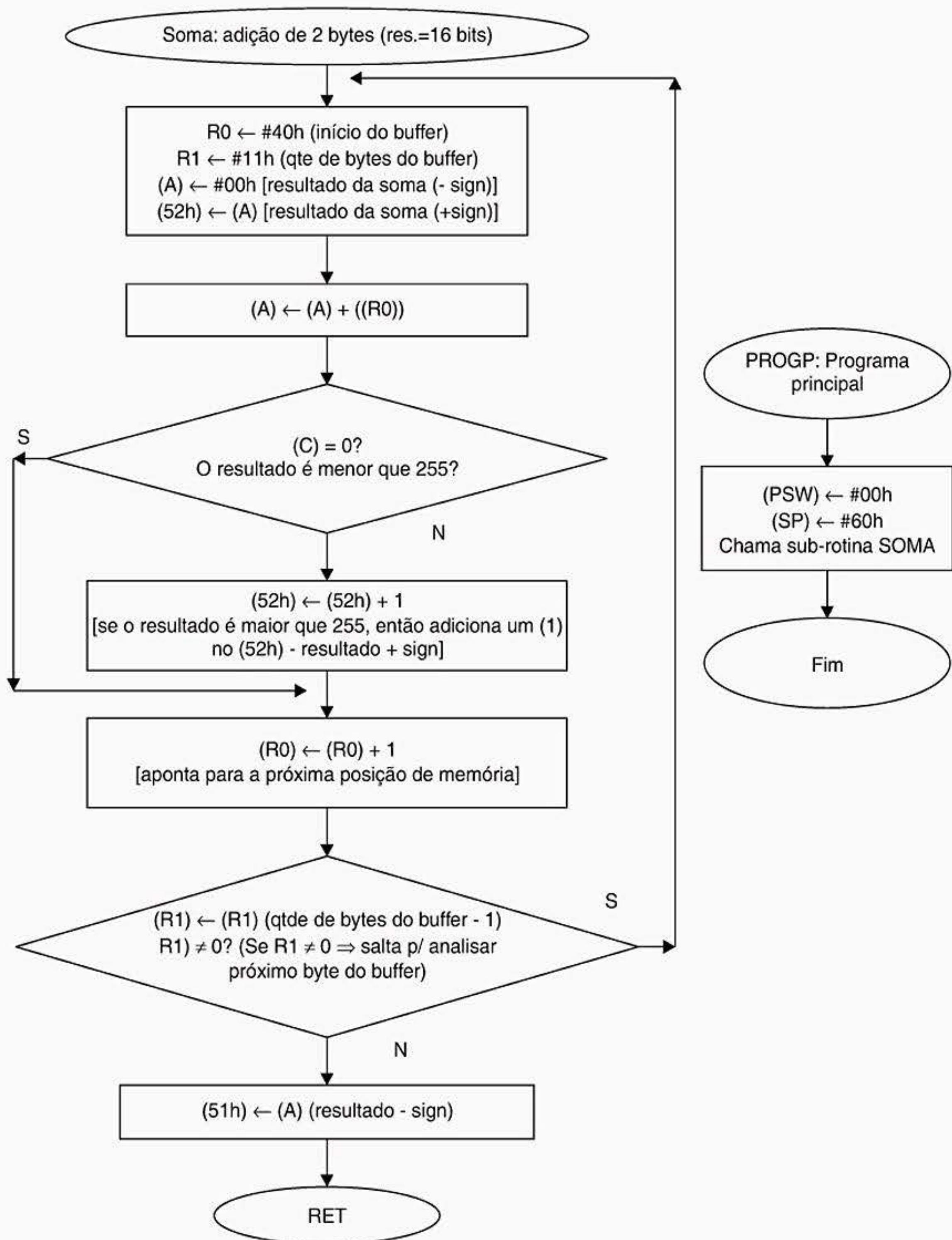


Figura D.2(a) Fluxograma de uma rotina que faz a adição dos bytes das posições de memórias cujos endereços inicial e final são 40h e 50h.

```

DEFSEG EXEMPLO, ABSOLUTE
SEG      EXEMPLO
ORG     0100h ; O programa será armazenado a partir do conteúdo da posição de memória
               ; de programa (EPROM).
SOMA:   MOV R0,#40h ; (R0) é o ponteiro do buffer. Valor inicial igual a 40h (início da posição
               ; de memória do buffer).
         MOV R1,#11h ; (R1) é um contador descendente que contém a quantidade de bytes do
               ; buffer a serem adicionados (50h - 40h + 1).
         MOV A,#00h ; 'Zera' o conteúdo do acumulador que conterá a parte menos significativa
               ; do resultado de 16 bits.
         MOV 52h,A ; 'Zera' o conteúdo da posição de memória cujo endereço é 52h, que
               ; será responsável por armazenar a parte mais significativa do resultado de
               ; 16 bits.

ADR2:   ADD A,@R0 ; Adiciona o resultado - sign ao conteúdo do byte da memória
         JNC ADR1 ; Se o resultado da soma for menor que 255, salta (não adiciona 1
               ; ao conteúdo da posição de memória cujo endereço é 52h (resultado +
               ; sign)).

         INC 52h ; Se o resultado da soma for maior que 255, adiciona 1 ao conteúdo da
               ; posição de memória cujo endereço é 52h (resultado + sign).

ADR1:   INC R0 ; Aponta para o próximo byte da memória a ser adicionado.
         DJNZ R1,ADR2 ; Diminui uma unidade do conteúdo do registrador R1 e salta, se ele
               ; for diferente de zero, isto é, há mais dados a serem adicionados do buffer
               ; e memória.

         MOV 51h,A ; Armazena a parte - sign no conteúdo da posição de memória cujo
               ; endereço é 51h.

         RET ; Retorno da sub-rotina. Simular/explicar detalhadamente o funcionamento
               ; dessa instrução, mostrando os conteúdos dos registradores PC,
               ; SP e da pilha.

PROGP:  MOV PSW,#00h ; Faz (PSW) ← #00h. Define RS1 = 0 e RS0 = 0 no (PSW), ou seja, define
               ; banco 0 (R0 até R7 – 00h até 07h).
         MOV SP,#60h ; Define (SP) = 60h (início da pilha)
         LCALL SOMA ; Simular/explicar detalhadamente o funcionamento dessa instrução
               ; mostrando os conteúdos dos registradores PC, SP e da pilha.

         END ; Fim do programa principal.

```

Figura D.2(b) Programa-fonte de uma rotina que faz a adição dos bytes das posições de memórias cujos endereços inicial e final são 40h e 50h.

A simulação deve consistir em:

- ◆ inicializar os conteúdos das posições de memória com valores definidos pelo grupo. Reportar tais valores em relatório;
- ◆ testar o processamento (fazer, teoricamente, as operações de subtração e adição obter o resultado dos *flags* de sinalização em PSW, para cada uma das condições de teste. Comparar com os resultados obtidos na simulação;
- ◆ executar passo a passo, teoricamente, as instruções ACALL, LCALL e RET. Comparar com os resultados obtidos na simulação;

- ◆ testar as variáveis de controle do buffer de memória por meio dos registradores R0 e R1. Verifique todas as vezes que o programa desviar, até não desviar mais para o início do programa e até que ocorra a finalização do mesmo.

D.4 Para o relatório

- 1 - Explique a execução das instruções passo a passo realizadas pela simulação, considerando os dados que o grupo adotou para ambos os exercícios (quantidade = #0FFh e soma).
- 2 - Compare o funcionamento teórico das instruções ACALL, LCALL e RET com o funcionamento simulado, ou seja, mostre o resultado do conteúdo dos registradores PC, SP e da pilha.
- 3 - Que modificação será necessária no programa de adição se quisermos fazer:
 - a) a operação de subtração entre os dados da memória;
 - b) a operação de "OR" entre os dados da memória;
 - c) a operação de "AND" entre os dados da memória;
 - d) a operação de "OR-EX" entre os dados da memória.
- 4 - Utilizando sub-rotinas, faça um programa estruturado (fluxograma e programa-fonte) que calcule a quantidade de números que apresentam paridade ímpar e que são maiores que 55h do buffer de memória cujo endereço inicial é 38h e o final é 55h. Essa quantidade deve ser armazenada no conteúdo do registrador R5 do segundo banco de registradores. Esse programa também deve adicionar dois buffers de memória: o primeiro buffer de memória inicia no endereço 56h e tem 10 elementos; o segundo buffer de memória inicia no endereço 6Ah. O resultado da adição desses dois buffers de memória deve ser armazenado a partir do endereço de memória 7Bh.
- 5 - Utilizando sub-rotinas, faça um programa estruturado (fluxograma e programa-fonte) que calcule o valor máximo do buffer de memória, cujo endereço inicial é 3Dh e o final é 52h. O resultado deve ser armazenado no conteúdo de posição de memória 21h.
- 6 - Utilizando sub-rotinas, faça um programa estruturado (fluxograma e programa-fonte) que calcule o valor mínimo do buffer de memória, cujo endereço inicial é 4Ah e o final é 57h. O resultado deve ser armazenado no conteúdo de posição de memória, cujo endereço é 41h.
- 7 - Entregue os exercícios propostos do Capítulo 5.

EXPERIÊNCIA 05:

AS PORTAS DE ENTRADA E SAÍDA DA FAMÍLIA DE MICROCONTROLADORES MCS-51 DA INTEL E SUAS APLICAÇÕES NO CONTROLE DIGITAL DE MÁQUINA E DE PROCESSO

E.1 Objetivo

- ❖ Simulação de programas em Assembly da família de microcontroladores MCS-51 da Intel utilizando os portes de entrada e saída para o controle digital de processos

E.2 Introdução teórica

Veja o Capítulo 6.

E.3 Procedimento experimental

O campo das portas de entrada e saída do simulador AVSIM51 está situado no canto inferior direito da tela do simulador AVSIM51, como está indicado na Figura E.1. Para cada porta da família de microcontroladores MCS-51, existem duas informações. A primeira informação diz respeito ao conteúdo dos *latches*, e a segunda informação reporta o nível lógico de cada pino físico.

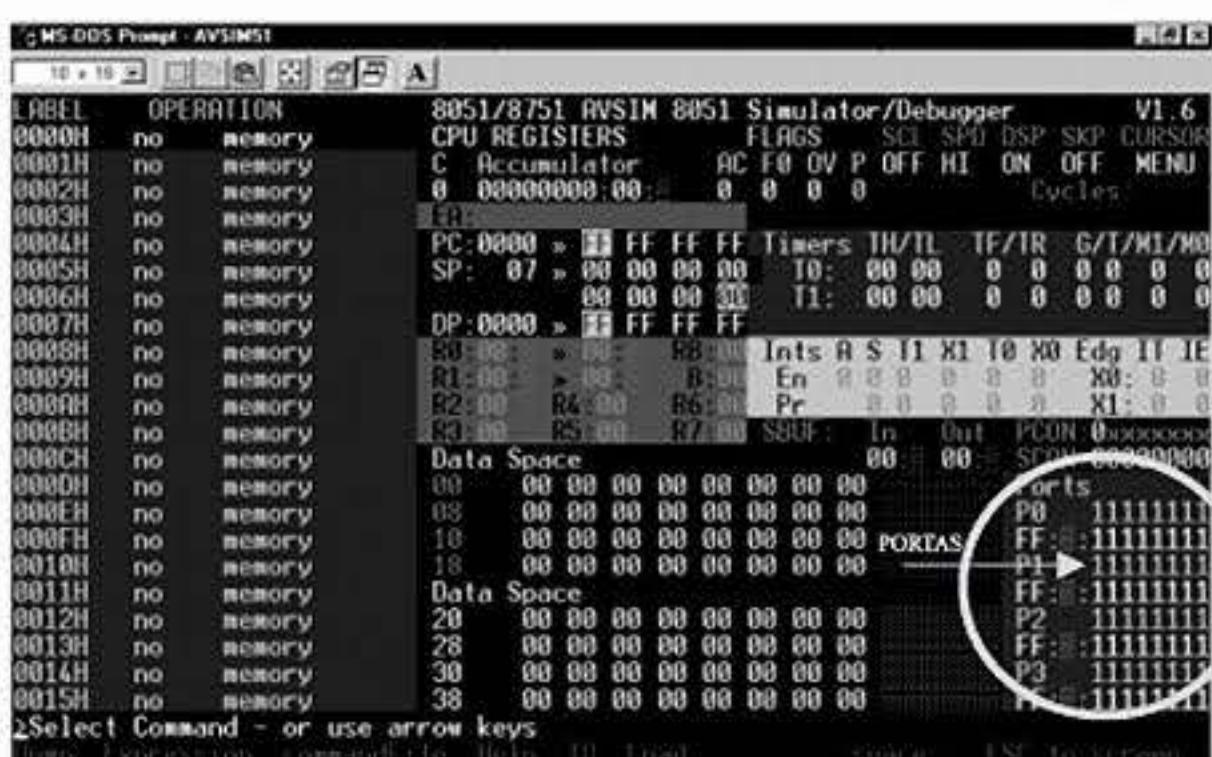


Figura E.1 O local dos registradores de funções especiais de controle das portas de entrada e saída.

Características das informações referentes às portas P0, P1, P2 e P3 no simulador AVSIM51.

- ◆ Dos conteúdos dos *latches* de cada porta:
 - ◆ só existe a representação binária;
 - ◆ seus valores podem ser alterados somente quando são feitas operações de escrita na porta;
 - ◆ seus valores podem ser lidos somente quando são feitas operações de leitura.
- ◆ Dos níveis lógicos de cada pino das portas:
 - ◆ existem as representações hexadecimal, ASCII e binária. Esses valores podem ser alterados manualmente e por meio das operações de escrita nas portas.

1 - Utilizando os comandos *Patch* (P) e *Patch Code* (P), que fazem parte da área de comandos do simulador AVSIM51, armazene no simulador as instruções fornecidas a seguir.

; 1º conjunto de instruções – dê sua representação simbólica e descreva sucintamente o que ele faz.

```
MOV      A,#00h      ;
MOV      P0,A       ;
ANL      P0,#0F0h    ;
```

; Tente inserir um dado qualquer em P0, por digitação, usando o simulador. O que aconteceu?

; Justifique. Agora, execute a instrução a seguir e observe o que acontece.

```
MOV      A, P0       ;
```

; 2º conjunto de instruções – dê sua representação simbólica e descreva sucintamente o que ele faz.

```
MOV      A,#0FFh     ;
MOV      P0,A       ;
MOV      P1,P0      ;
ANL      P2,#0Fh     ;
MOV      A,P0       ;
```

; 3º conjunto de instruções – dê sua representação simbólica e descreva sucintamente o que ele faz.

```
MOV      A,0F0h     ;
MOV      A,P3       ;
ORL      P3,#A6h    ;
```

; 4º conjunto de instruções – dê sua representação simbólica e descreva sucintamente o que ele faz.

```
JB      P1.0,$      ;
JNB     P1.0,$      ;
```

; 4º conjunto de instruções – dê sua representação simbólica e descreva sucintamente o que ele faz.

```
MOV      A,#01111110b ;
CJNE    A,P1,$      ;
```

2 - Simule (execute passo a passo) cada conjunto de instruções (do 1º ao 4º), mostrando sua representação simbólica, e descreva sucintamente qual sua função dentro do contexto do programa.

- 3 - Faça o fluxograma e o programa-fonte de uma rotina de tempo que utilize três contadores. Considere que constante 1 = 60h, constante 2 = AAh e constante 3 = C7h.

Dica: Veja como foi feita a rotina de tempo que utiliza dois contadores, fornecida no Capítulo 6.

- 4 - Simule a rotina de identificação do acionamento de uma chave mecânica, com a eliminação do ruído (*bounce*), fornecida a seguir. Adote constante 1 = 0FFh, constante 2 = 0FFh.

- 4.1 - Faça a simulação de um acionamento sem a sua confirmação (acionamento falso) e veja como a rotina funciona.

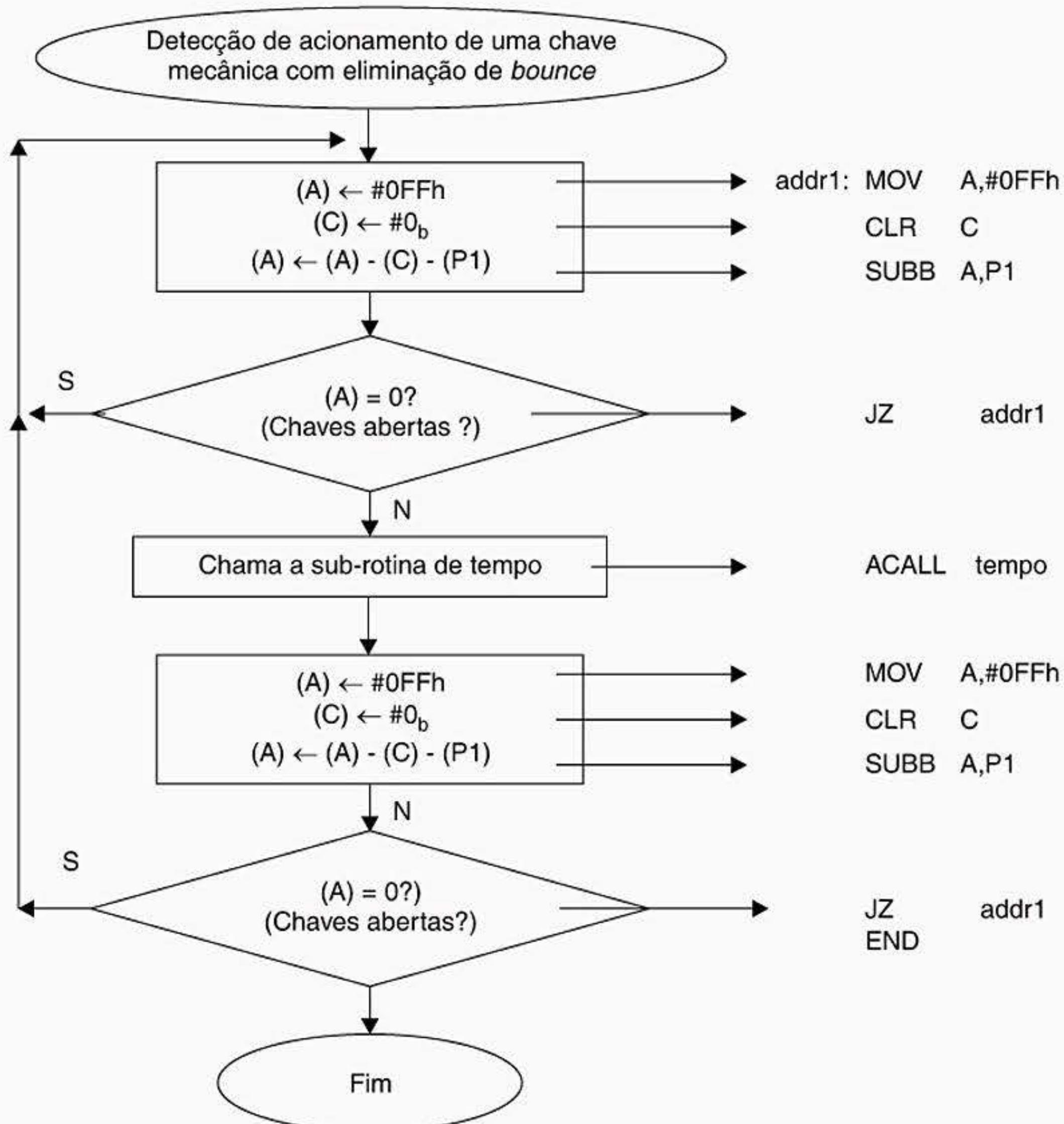


Figura E.2(a) Fluxograma de uma rotina que monitora o acionamento de uma chave mecânica qualquer com eliminação de ruído (*bounce*).

WAIT:	MOV A,#0FFh	; Aguarda o acionamento de uma porta
	CLR C	
	SUBB A,P1	
	JZ WAIT	
	MOV R0,#constante1	; Rotina de atraso de tempo
ATRAS:	MOV R1,#constante2	
	DJNZ R1,\$	
	DJNZ R0, ATRAS	
	MOV A,#0FFh	; Confirma o fechamento
	CLR C	
	SUBB A,P1	
	JZ WAIT	; Foi algum ruído do acionamento, reinicia o processo de ; detecção de acionamento
	END	; Confirmado o acionamento

Figura E.2(b) Propaganda-fonte de uma rotina que monitora o acionamento de uma chave mecânica qualquer com eliminação de ruído (*bounce*).

5 – Simule a rotina que conta o número de acionamentos e desacionamentos de uma chave mecânica qualquer.

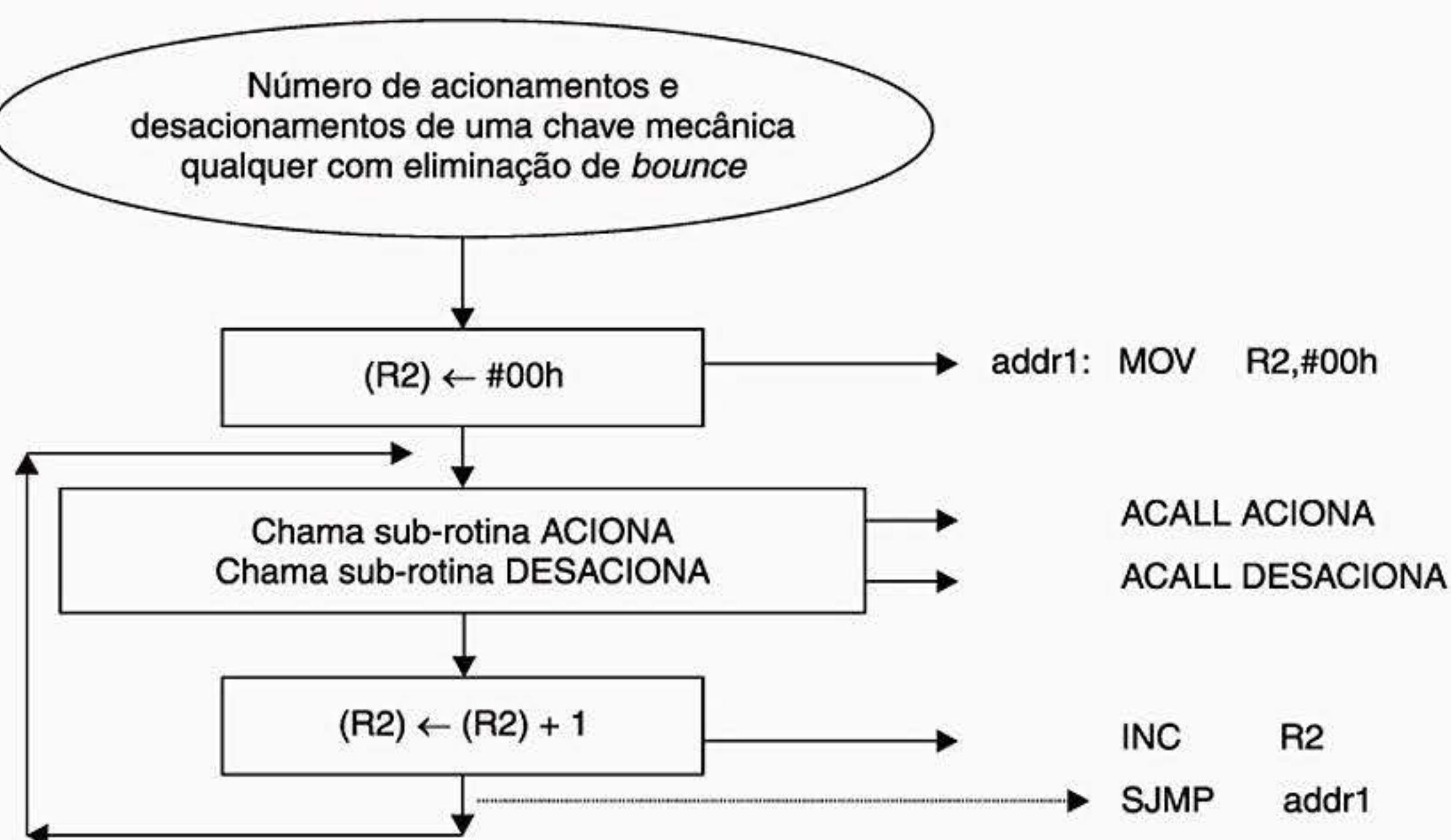


Figura E.3(a) Fluxograma que conta o número de acionamentos e desacionamentos de uma chave mecânica qualquer com eliminação de ruído (*bounce*).

```

MOV    R2,#00h
WAIT1: MOV    A,#0FFh      ; Aguarda o acionamento de uma porta
       CLR    C
       SUBB   A,P1
       JZ     WAIT1
       MOV    R0,#constante1 ; Rotina de atraso de tempo
ATRAS: MOV    R1,#constante2
       DJNZ   R1,$
       DJNZ   R0, ATRAS
       MOV    A,#0FFh      ; Confirma o acionamento
       CLR    C
       SUBB   A,P1
       JZ     WAIT1
WAIT2: MOV    A,#0FFh      ; Aguarda o desacionamento do bit da porta
       CLR    C
       SUBB   A,P1
       JNZ    WAIT2
       MOV    R0,#constante1 ; Rotina de atraso de tempo
ATRAS: MOV    R1,#constante2
       DJNZ   R1,$
       DJNZ   R0, ATRAS
       MOV    A,#0FFh      ; Confirma o desacionamento
       CLR    C
       SUBB   A,P1
       JNZ    WAIT2
       INC    R2          ; Conta quantas vezes ocorre o acionamento e o desacionamento
       SJMP   WAIT1

```

Figura E.3(b) Propaganda-fonte que conta o número de acionamentos e desacionamentos de uma chave mecânica qualquer com eliminação de ruído (*bounce*).

6 – Simule a rotina que identifica o número da chave ligada a um determinado bit da porta P0. Descreva sucintamente como o programa funciona.

```

WAIT1: MOV    A,#0FFh      ; Aguarda o acionamento de uma porta
       CLR    C
       SUBB   A,P1
       JZ     WAIT1
       MOV    R0,#constante1 ; Rotina de tempo
ATRAS: MOV    R1,# constante2
       DJNZ   R1,$
       DJNZ   R0, ATRAS
       MOV    A,#0FFh      ; Confirma o acionamento
       CLR    C
       SUBB   A,P1
       JZ     WAIT1
       MOV    R2,#00h      ; Identificação do número do bit acionado (val. inic.= 0)
       MOV    A,P1
ROTA:  RRC    A          ; Rotaciona para a direita com flag carry bit
       JNC    OK         ; Se (C) = 0 (o bit foi acionado) => fim, caso contrário
       INC    R2         ; aumenta o número do bit (0 a 7)
       SJMP   ROTA       ; esse bit ainda não foi acionado
OK:    END

```

E.4 Para o relatório

- 1 - As perguntas são solicitadas em cada item do procedimento experimental e para cada simulação; descreva sucintamente o funcionamento de cada programa.
- 2 - Utilizando sub-rotinas, faça um programa estruturado (fluxograma e programa-fonte) que faça o bit 4 da porta 3 piscar a cada 1 segundo.

Dica:

- ◆ acione o conteúdo do bit 4 da porta 3;
 - ◆ execute uma rotina de tempo de 1 segundo (devem ser calculados os valores da constante₁, constante₂, ..., constante_N).
 - ◆ desacione o conteúdo do bit 4 da porta 3;
 - ◆ execute uma rotina de tempo de 1 segundo;
 - ◆ volte incondicionalmente para o início do programa (*looping* infinito nesse processo, gerando um 'pisca-pisca').
- 3 - Entregue, anexos ao relatório, os exercícios propostos referentes ao Capítulo 6.

EXPERIÊNCIA 06: Os TIMERS/CONTADORES DA FAMÍLIA DE MICROCONTROLADORES MCS-51 DA INTEL

F.1 Objetivo

- ❖ Simulação de programas estruturados em Assembly, que utilizam os *timers*/contadores da família de microcontroladores MCS-51 para o controle de processos.

F.2 Introdução teórica

Veja o Capítulo 7.

F.3 Procedimento experimental

O campo dos registradores de funções especiais de controle dos *timers*/contadores do simulador AVSIM51 está situado no canto superior direito da tela do simulador AVSIM51, como está indicado na Figura F.1. Para cada *timer*/contador da família de microcontroladores MCS-51, existem as seguintes informações:

- ◆ os registradores de contagem TH/TL;
- ◆ os *flags* das fontes de interrupção TF;
- ◆ os *flags* de acionamento/desacionamento dos *timers*/contadores;
- ◆ os *flags* G (Gate);
- ◆ T (*timer*/contador) e
- ◆ os *flags* selecionadores de modo de funcionamento M1 e M0.

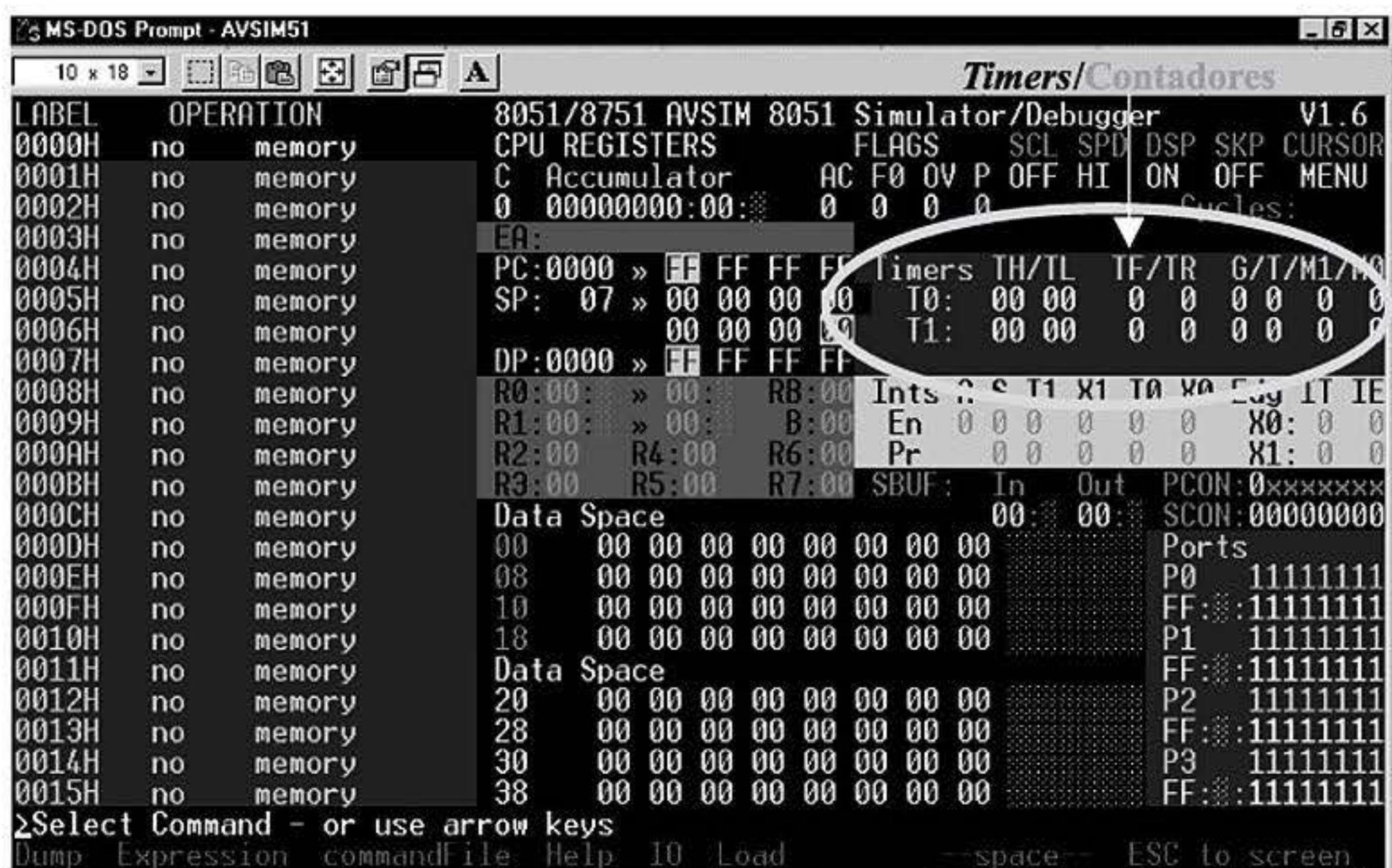


Figura F.1 O local dos registradores de funções especiais de controle dos *timers/contadores*.

1 - Utilizando os comandos *Patch* (P) e *Patch Code* (P), que fazem parte da área de comandos do simulador AVSIM51, armazene o programa da Figura F.2 no simulador.

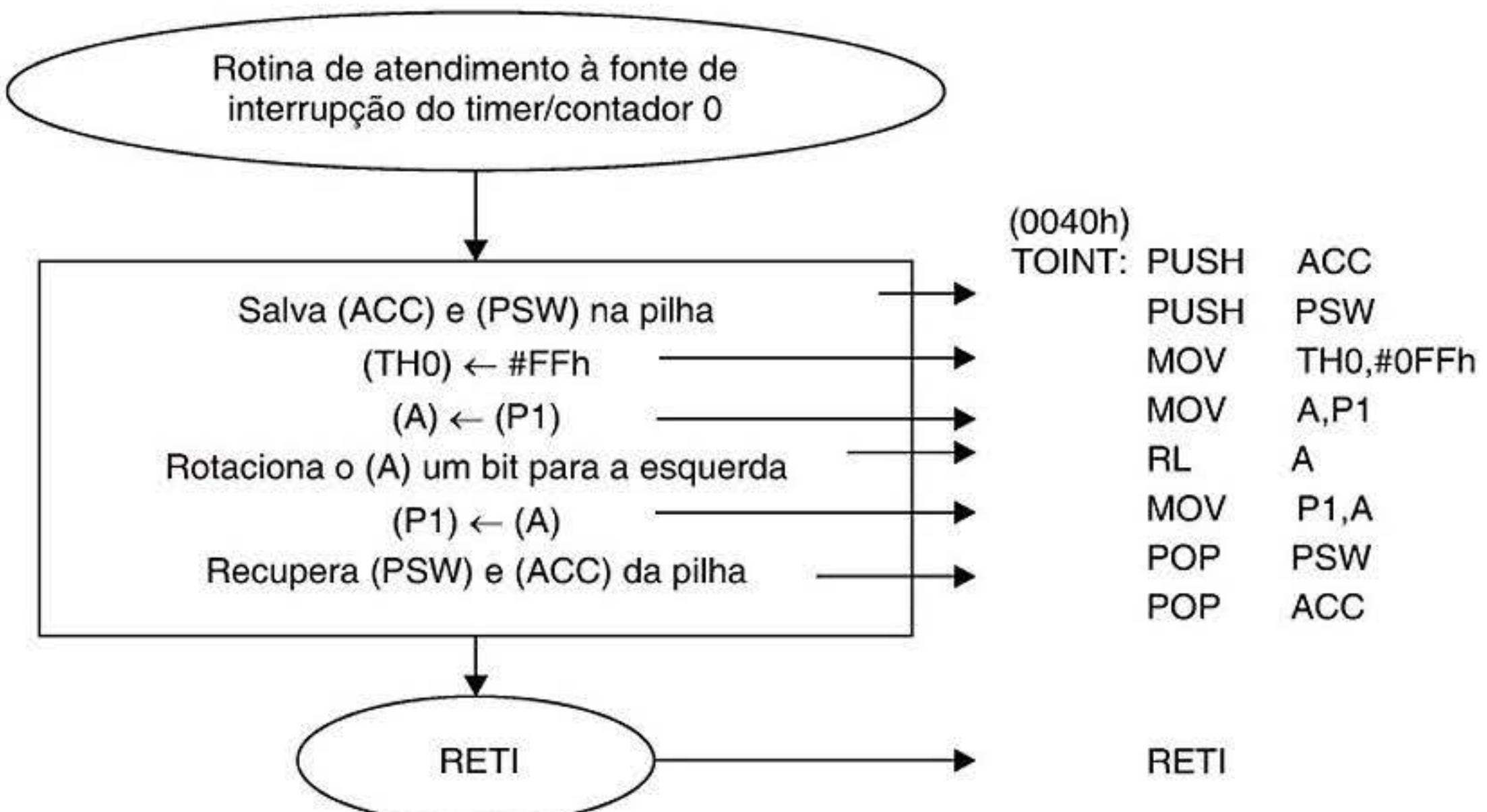


Figura F.2(a) Fluxograma estruturado que utiliza os *timers/contadores* nos modos 0 e 1.

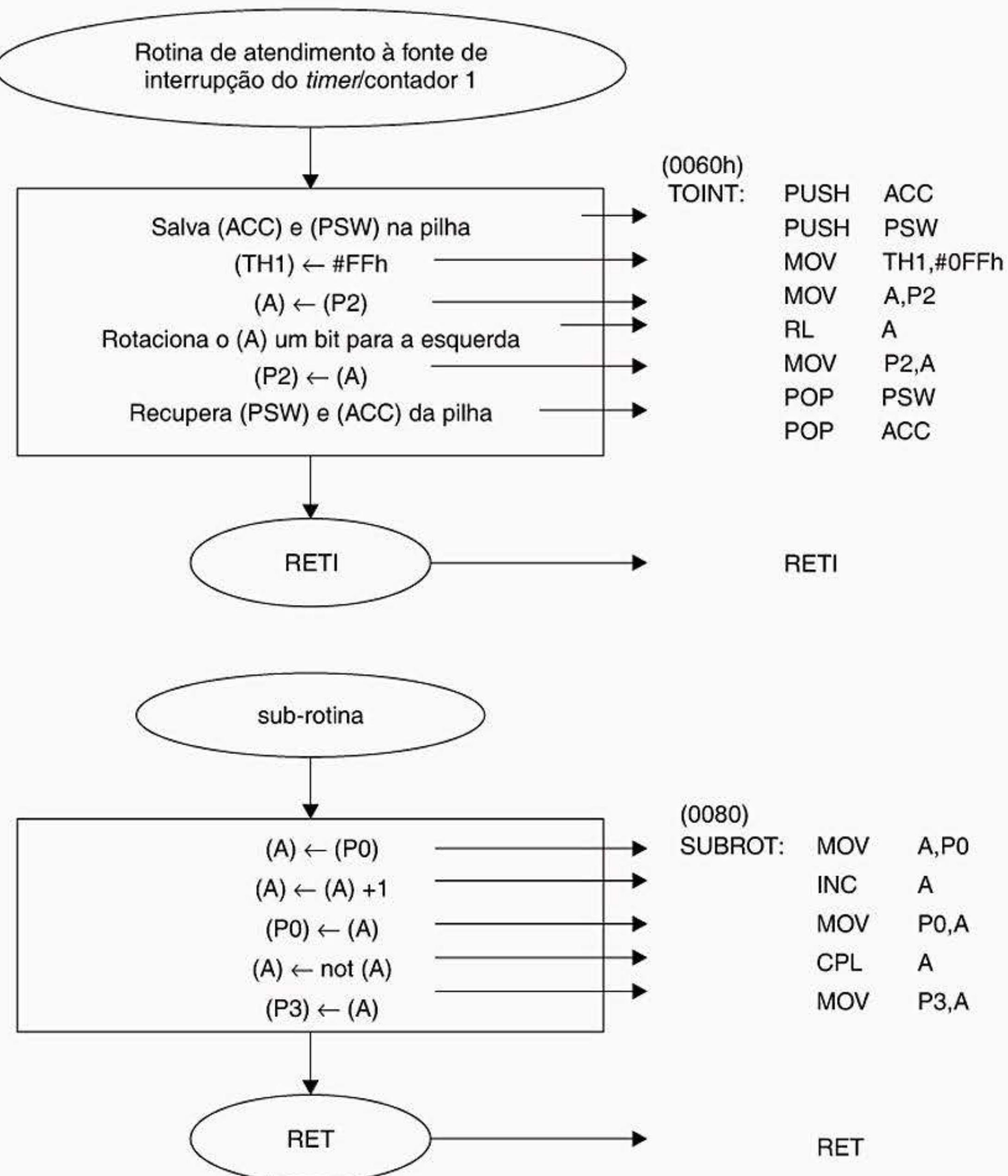


Figura F.2(a) Fluxograma estruturado que utiliza os timers/contadores nos modos 0 e 1 (Continuação).

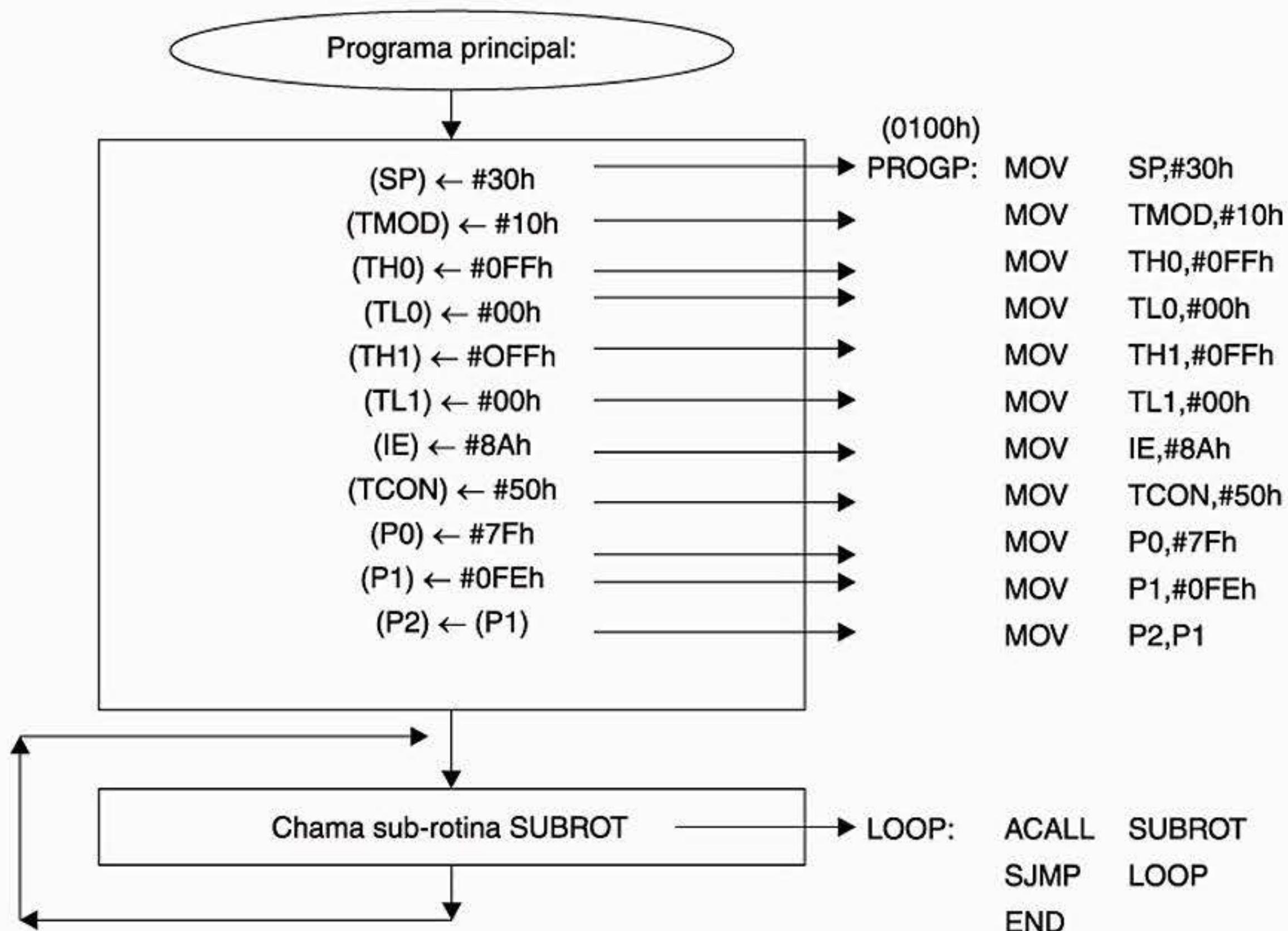


Figura F.2(a) Fluxograma estruturado que utiliza os *timers*/contadores nos modos 0 e 1 (*Continuação*).

A seguir, está representado o programa completo que deve ser armazenado em AVSIM51.

```

; Esse programa é um exemplo da utilização dos timers/contadores no Modo 0 (T/C 0) e 1 (T/C 1)
DEFSEG EXEM, ABSOLUTE ; Define o segmento EXEM na memória de programa
SEG      EXEM      ; Segmento EXEM
ORG      0000h     ; Após o reset da CPU, o registrador de função especial Program Counter (PC) é inicializado com o valor 0000h. Dessa maneira, a CPU inicia a execução do programa a partir do endereço 0000h. Com a utilização dos timers/contadores é necessário escrever suas rotinas de atendimento às fontes de interrupção a partir dos endereços 000Bh e 001Bh; o programa principal não poderá utilizar esse endereçamento. Assim, utiliza-se uma instrução de salto incondicional para o endereço inicial do programa principal 0100h, em que ela é escrita.
; Endereço inicial da rotina de atendimento à fonte de interrupção do timer/contador 0
ORG      000Bh     ; Após uma interrupção gerada pelo timer/contador 0, o programa é desviado para o endereço 000Bh. Como a rotina de atendimento à interrupção do timer/contador 0 gastará mais de 8 bytes, é utilizada uma instrução de salto incondicional para a posição de memória 0040h, em que é escrita a rotina de atendimento à fonte de interrupção do timer/contador 0.
AJMP    0040h

```

```

; Endereço inicial da rotina de atendimento à fonte de interrupção do timer/contador 1
ORG 001Bh ; Após uma interrupção gerada pelo timer/contador 1, o programa é
AJMP 0060h ; desviado para o endereço 000Bh. Como a rotina de atendimento à
; interrupção do timer/contador 1 gastará mais de 8 bytes, é
; utilizada uma instrução de salto incondicional para a posição de
; memória 0060h, em que é escrita a rotina de atendimento à fonte
; de interrupção do timer/contador 1.

; Rotina de atendimento à fonte de interrupção do timer/contador 0
ORG 0040h
T0INT: PUSH ACC ; Salva o estado do conteúdo do acumulador e do conteúdo do PSW
        PUSH PSW ; na pilha para preservar a execução do programa principal após executar
; a rotina de atendimento à fonte de interrupção do timer/contador 0.
        MOV TH0,#0FFh ; Reinicializa o conteúdo de TH0 com o valor previamente definido
        MOV A,P1 ; (A) ← (P1)
        RL A ; Rotaciona o (A) um bit para a esquerda
        MOV P1,A ; (P1) ← (A)
        POP PSW ; Restaura o (PSW) e (A) da pilha, para preservar a execução do programa
        POP ACC ; principal.
        RETI ; Retorna a partir da rotina de atendimento à fonte de interrupção do
; timer/contador 0 (Observação: execute passo a passo essa
; instrução, anotando os valores iniciais e finais dos conteúdos dos
; registradores e das posições de memória afetados pela mesma.)

; Rotina de atendimento à fonte de interrupção do timer/contador 1
ORG 0060h
T1INT: PUSH ACC ; Salva o estado do conteúdo do acumulador e o conteúdo do PSW
        PUSH PSW ; na pilha para preservar a execução do programa principal após executar
; a rotina de atendimento à fonte de interrupção do timer/contador 1.
        MOV TH0,#0FFh ; Reinicializa o conteúdo de TH0 com o valor previamente definido
        MOV A,P2 ; (A) ← (P2)
        RL A ; Rotaciona o (A) um bit para a esquerda
        MOV P2,A ; (P2) ← (A)
        POP PSW ; Restaura o (PSW) e (A) da pilha para preservar a execução do programa
        POP ACC ; principal.
        RETI ; Retorna a partir da rotina de atendimento à fonte de interrupção do
; timer/contador 1 (Observação: execute passo a passo essa
; instrução, anotando os valores iniciais e finais dos conteúdos dos
; registradores e das posições de memória afetados pela mesma.)

; Endereço inicial da sub-rotina SUBROT
ORG 0080h
SUBROT: MOV A,P0 ; Lê (P0)
        INC A ; Aumenta uma unidade o (P0)
        MOV P0,A ; Atualiza (P0)
        CPL A ; Complementa o (A)
        MOV P3,A ; Atualiza (P3)
        RET ; Retorna a partir da sub-rotina (Observação: execute passo a passo
; essa instrução, anotando os valores iniciais e finais dos conteúdos dos
; registradores e das posições de memória afetados pela mesma).

; Início do programa principal
ORG 0100h ; Programa principal
MAIN: MOV SP,#30h ; O (SP) foi mudado de sua condição inicial 07h (após um reset)
; para 30h.

```

```

MOV TMOD,#10h ; (TMOD) = 0001 0000b faz o timer/contador 1 no Modo 1 (timer de
; 16 bits de contagem) e faz o timer/contador 0 no Modo 0 (timer de
; 13 bits de contagem).
MOV TH0,#0FFh ; Inicializa o timer/contador 0 para iniciar a contagem a partir de FF00h.
MOV TL0,#00h ; O timer/contador 0 fica: 11111111 xxx00000 (contador de 13 bits)
MOV TH1,#0FFh ; Inicializa o timer/contador 0 para iniciar a contagem a partir de FF00h.
MOV TL1,#00h ; O timer/contador 1 fica: 11111111 00000000 (contador de 16 bits)
MOV IE,#8Ah ; (IE) = 1000 1010b. Faz (EA) = 1 (cada interrupção é habilitada pelo seu
; bit habilitador), (ET1) = 1 (habilita o reconhecimento da fonte de
; interrupção do timer/contador 1) e (ET0) = 1 (habilita o
; reconhecimento da fonte de interrupção do timer/contador 0)
MOV TCON,#50h ; (TCON) = 0101 0000b. Faz (TR1) = 1 (liga/executa o
; timer/contador 1) e (TR0) = 1 (liga/executa o timer/contador 0).

MOV P0,#7Fh ; (P0) = 0111 1111b. Inicializa o (P0)
MOV P1,#0FEh ; (P1) = 1111 1110b. Inicializa o (P1)
MOV P2,P1 ; (P2) ← (P1)
LOOP: ACALL SUBROT ; Chama a sub-rotina SUBROT (Observação: execute passo a passo
; essa instrução, anotando os valores iniciais e finais dos conteúdos dos
; registradores e das posições de memória afetados pela mesma).
SJMP LOOP ; Loop do programa principal
END

```

2 - Simule (execute passo a passo) cada instrução do programa, mostrando sua representação simbólica e as condições iniciais e finais dos conteúdos dos registradores e das posições de memória afetados. Descreva também, sucintamente, qual é a função de cada bloco de programa no contexto total do programa.

3 - Verifique o funcionamento do *timer*/contador 0 no Modo 2:

3.1 - utilizando o comando *Patch* e *Patch Code* do simulador AVSIM51, altere o programa anterior da seguinte maneira:

- ◆ na rotina de atendimento à interrupção do *timer*/contador 0 (T0INT:):
 - ◆ MOV TH0,#0FFh → três instruções NOP (no operation)
- ◆ na rotina do programa principal (MAIN:):
 - ◆ MOV TMOD,#10h → MOV TMOD,#02h
 - ◆ MOV TH0,#0FFh → MOV TH0,#23h
 - ◆ MOV TL0,#00h → MOV TL0,#11h
 - ◆ MOV IE,#8Ah → MOV IE,#82h
 - ◆ MOV TCON,#50h → MOV TCON,#10h

Observação: dê a representação simbólica, descreva a função de cada instrução alterada, simule o novo programa e entenda o funcionamento do *timer*/contador 0 no Modo 2.

4 - Verifique o funcionamento dos *timers*/contadores 0 e 1 no Modo 3:

4.1 - utilizando o comando *Patch* e *Patch Code* do simulador AVSIM51, altere o programa anterior da seguinte maneira:

- ◆ nas rotinas de atendimento às fontes de interrupção do *timer*/contador 0 (T0INT:) e 1 (T1INT:)

T0INT:	PUSH ACC	; Salva o estado do conteúdo do acumulador e do conteúdo do PSW na
	PUSH PSW	; Pilha para preservar a execução do programa principal após executar a
		; rotina de atendimento à fonte de interrupção do timer/contador 0.
MOV	A,TL0	; Reinicializa o valor de TH0 com o valor previamente
	ADD A,#12H	; definido. Evita atrasos de contagens.
	MOV TL0,A	;
	MOV A,P1	; (A) ← (P1)
	RL A	; Rotaciona o (A) um bit para a esquerda
	MOV P1,A	; (P1) ← (A)
	POP PSW	; Restaura o (PSW) e (A) da pilha para preservar a execução do
	POP ACC	; programa principal.
	RETI	; Retorna a partir da rotina de atendimento à fonte de interrupção do
		; timer/contador 0 (Observação: execute passo a passo essa
		; instrução, anotando os valores iniciais e finais dos conteúdos dos
		; registradores e das posições de memória afetados pela mesma.)
; Rotina de atendimento à fonte de interrupção do timer/contador 1		
T1INT:	PUSH ACC	; Salva o estado do conteúdo do acumulador e do conteúdo do PSW na
	PUSH PSW	; Pilha para preservar a execução do programa principal após executar a
		; rotina de atendimento à fonte de interrupção do timer/contador 1.
MOV	A,TH0	; Reinicializa o valor de TH0 com o valor previamente
	ADD A,#12h	; definido. Evita atrasos de contagens.
	MOV TH0,A	;
	MOV A,P2	; (A) ← (P2)
	RR A	; Rotaciona o (A) um bit para a direita
	MOV P2,A	; (P2) ← (A)
	POP PSW	; Restaura o (PSW) e (A) da pilha para preservar a execução do
	POP ACC	; programa principal.
	RETI	; Retorna a partir da rotina de atendimento à fonte de interrupção do
		; timer/contador 1 (Observação: execute passo a passo essa
		; instrução, anotando os valores iniciais e finais dos conteúdos dos
		; registradores e das posições de memória afetados pela mesma.)

- ◆ na rotina do programa principal (MAIN:):

- ◆ MOV TMOD,#33H
- ◆ MOV TH0,#11H
- ◆ MOV TL0,#11H
- ◆ MOV IE,#8AH
- ◆ MOV TCON,#50H

Observação: dê a representação simbólica, descreva a função de cada instrução alterada, simule o novo programa e entenda o funcionamento dos *timers*/contadores 0 e 1, no Modo 3.

F.4 Para o relatório

- 1 - Dê uma breve descrição do funcionamento geral do uso dos *timers*/contadores em um programa estruturado, desde o momento em que ocorre a interrupção, de quando a rotina de atendimento à fonte de interrupção é atendida e se há o retorno ao programa principal. Descreva uma condição de simulação, mostrando esse processo por meio dos conteúdos dos registradores e das posições de memórias que foram afetados.
- 2 - Descreva objetivamente o que esse programa faz.
- 3 - Modifique o programa anterior para que ele tenha as seguintes características:
 - 3.1 - *timer/contador 0*:
 - ◆ programado como contador;
 - ◆ no Modo 0;
 - ◆ com sua contagem sendo controlada pela interrupção externa INT0\;
 - ◆ com o valor inicial de contagem igual a 6493h.
 - 3.2 - *timer/contador 1*:
 - ◆ programado como *timer*;
 - ◆ no Modo 2;
 - ◆ para que a cada 100 milissegundos seja gerada uma interrupção TF1;
 - ◆ valor inicial da contagem 88 h.
- 4 - Elabore um programa estruturado (fluxograma e programa-fonte) que faça o bit 4 da porta 3 piscar a cada 1 segundo, utilizando o *timer/contador 0* para gerar a rotina de tempo de 1 segundo.
- 5 - Faça um programa estruturado (fluxograma e programa-fonte) de um relógio digital, utilizando *timer/contadores*. Considere que às portas de 0 a 3 estão ligados quatro displays de sete segmentos ($D7S_0$, $D7D_1$, $D7S_2$ e $D7S_3$). Coloque a indicação dos minutos nos displays $D7S_1$ e $D7S_0$, e a indicação das horas nos displays $D7S_3$ e $D7S_2$.
- 6 - Resolva os exercícios propostos do Capítulo 7.

EXPERIÊNCIA 07:

A INTERFACE DE COMUNICAÇÃO SERIAL DA FAMÍLIA DE MICROCONTROLADORES MCS-51 DA INTEL

G.1 Objetivo

- ❖ Simulação de programas estruturados em Assembly, que utilizam a interface de comunicação serial da família de microcontroladores MCS-51

G.2 Introdução teórica

Veja o Capítulo 8.

G.3 Procedimento experimental

O campo dos registradores de funções especiais de controle da interface de comunicação serial do simulador AVSIM51 está situado na altura média do lado direito da tela do simulador AVSIM51, como está indicado na Figura G.1.

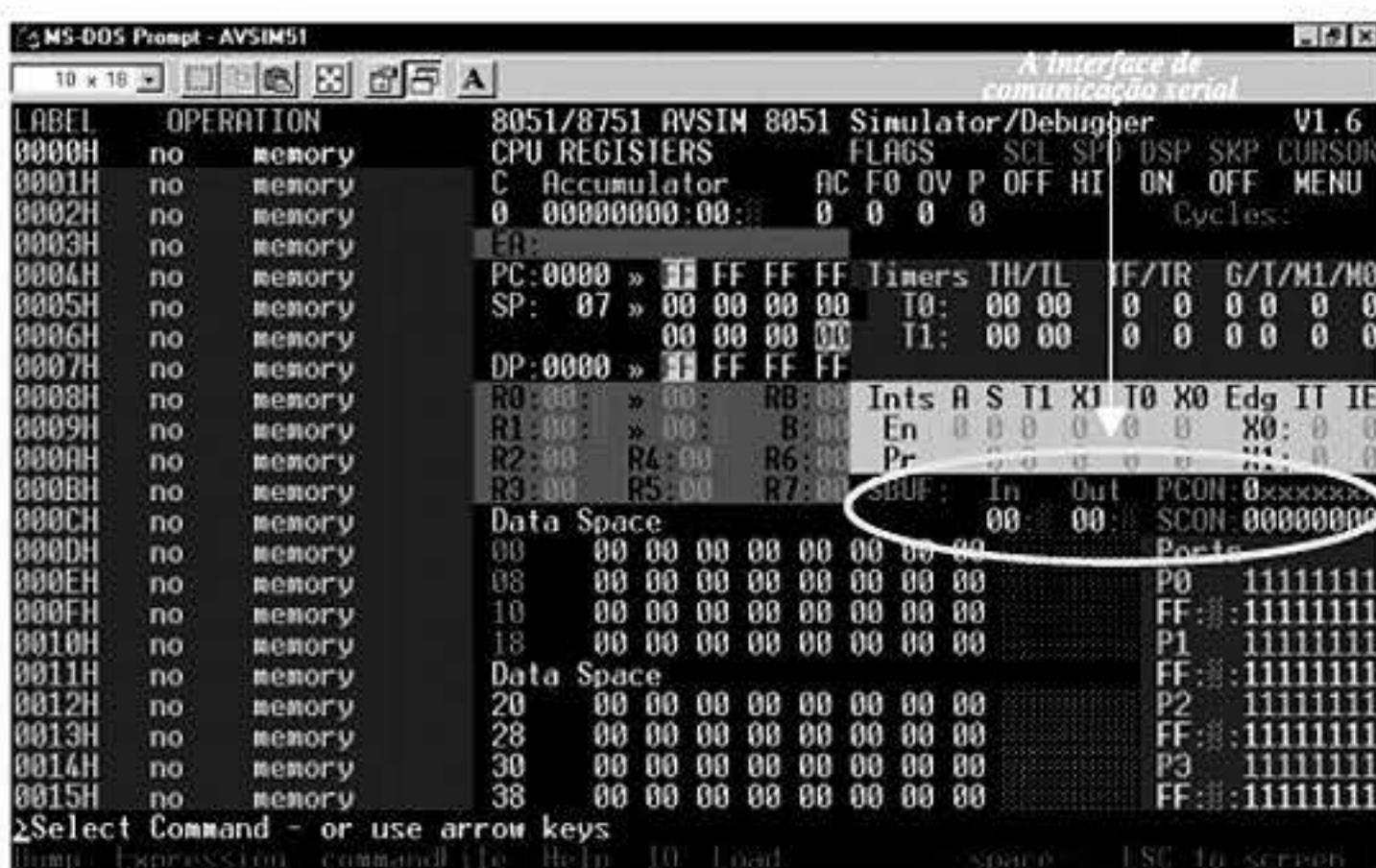


Figura G.1 O local dos registradores de funções especiais de controle da interface de comunicação serial.

Nessa região, são apresentados os registradores de funções especiais do buffer da interface de comunicação serial SBUF, do registrador de controle da interface de comunicação serial SCON e do registrador de prioridade PCON.

- 1 - Utilizando os comandos *Patch* (P) e *Patch Code* (P), que fazem parte da área de comandos do simulador AVSIM51, armazene o programa da Figura G.2 no simulador.

```

DEFSEG EXEM, ABSOLUTE ; Define o segmento de programa EXEM
SEG      EXEM

ORG      0000h ; Após o reset da CPU, o programa prossegue no endereço 0000h
AJMP    0100h ; (durante o acionamento da chave de reset ou após a 'energização' do
               ; sistema, por meio do circuito de reset formado por um resistor
               ; em paralelo com um diodo e em série com um capacitor que está
               ; ligado ao pino de Reset da CPU). Como foi pedido que o programa
               ; principal fosse escrito no endereço de memória de programa 0100h,
               ; um salto incondicional foi utilizado para tal posição de memória.

ORG      0023h ; Após uma interrupção do canal serial, o programa prossegue no
RETI           ; endereço '0023h' e retorna a partir rotina de atendimento à fonte
               ; de interrupção do canal de comunicação serial.

MAIN:   ORG      0100h ; Início do programa principal
        MOV      SCON,#10h ;(SCON) ← # 00010000b. Programa o canal serial como registrador de
                           ; deslocamento e baud rate de fosc/12 (modo 0).
        MOV      IE,#90h ;(IE) = 1001 0000b. Faz (EA) = 1 (cada interrupção é habilitada pelo seu
                           ; bit habilitador); (ES) = 1 (habilita a interrupção de comunicação serial)
        MOV      A,#55h ;(A) ← #55h (dado a ser transmitido)
LOOP:   MOV      SBUF,A ;(SBUF) ← (A). Carrega o dado a ser transmitido no (SBUF), e é iniciada
               ; a transmissão serial
        JNB      TI,$ ; Se (TI) ≠ 0 (o dado ainda não foi transmitido serialmente, bit a bit) ⇒
                   ; (PC) ← $ (salta para o próprio endereço da instrução JNB). Aguarda ser
                   ; transmitido. Quando o dado é transmitido, faz (TI) = 1, e é gerada uma
                   ; interrupção do canal serial (salta para o endereço 0023h - rotina de
                   ; serviço de comunicação serial).
        CLR      TI ;(TI) ← 0 (libera canal de comunicação serial para a transmissão de um
                   ; novo dado)
        CPL      A ;(A) ← complemento de um do (A)
        SJMP    LOOP ; loop do programa principal
END

```

Figura G.2(a) Programa-fonte estruturado de comunicação serial.

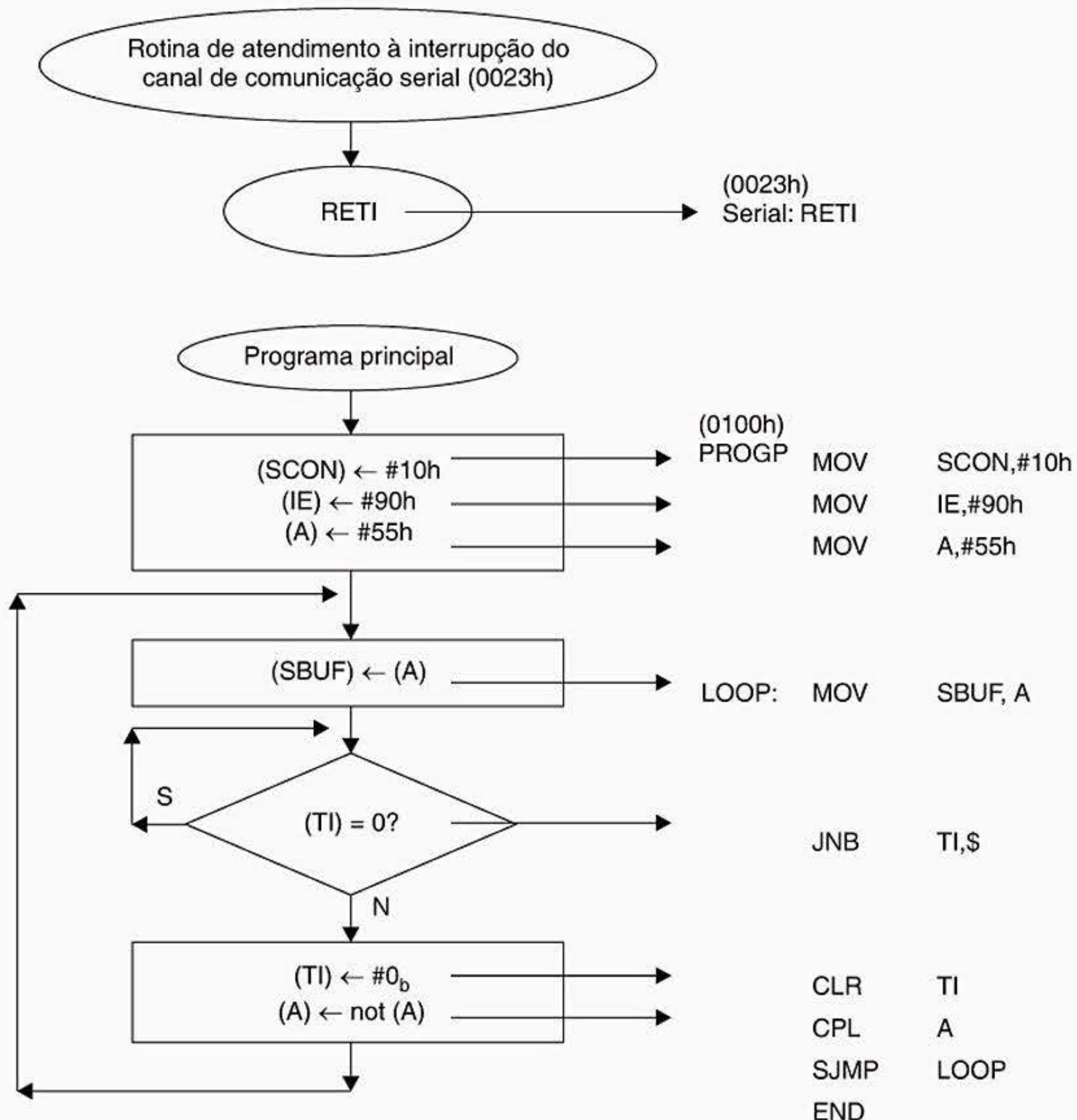


Figura G.2(b) Fluxograma estruturados de comunicação serial.

Esse programa faz a transmissão serial dos bytes 55h e Aah, no Modo 0, continuamente.

- 2 - Simule (execute passo a passo) cada instrução do programa, mostrando sua representação simbólica e as condições iniciais e finais dos conteúdos dos registradores e das posições de memória afetados. Descreva também, sucintamente, qual é a função de cada bloco de programa no contexto total do programa.
- 3 - Programa de transmissão serial no Modo 1: esse programa transmite, serialmente, 55h e Aah, considerando uma *baud rate* = 9,6KHz, que a $f_{osc} = 11,059\text{MHz}$, o *timer 1*, no Modo 2 de operação (valor de

recarregamento automático = FDh). Assim, mude o programa anterior, na parte do programa principal, da seguinte maneira:

MAIN:	MOV SCON,#40h	; (SCON) \leftarrow #0010 0000 _b . Programa o canal serial como UART de ; 8 bits e <i>baud rate</i> variável.
	MOV TMOD,#20h	; (TMOD) \leftarrow #0010 0000 _b , Programa o <i>timer 1</i> no Modo 2 ; (8 bits com recarregamento automático).
	MOV TH1,#0FDh	; (TH1) \leftarrow FDh. Valor para gerar um <i>baud rate</i> de 9,6KHz ; (Recarregador automático.)
	MOV TL1,TH1	; (TL1) \leftarrow (TH1). Valor inicial para um <i>baud rate</i> de 9,6KHz
	MOV IE,#90h	; (IE) \leftarrow #10010000 _b . Faz (EA) = 1 (cada interrupção é habilitada pelo ; seu bit habilitador); (ES) = 1 (Habilita a interrupção de comunicação ; serial) e não habilita a interrupção do <i>timer 1</i> .
	MOV TCON,#40h	; Liga/executa o <i>timer 1</i> .
	MOV A,#55h	; (A) \leftarrow #55h (byte a ser transmitido).
LOOP:	MOV SBUF,A	; (SBUF) \leftarrow (A). Carrega o dado a ser transmitido em SBUF e inicia a ; transmissão serial.
	JNB TI,\$; Se (TI) \neq 0 (o dado ainda não foi transmitido serialmente, bit a bit) \Rightarrow ; (PC) \leftarrow \$ (próprio endereço da instrução JNB), aguarda ser transmitido. ; Quando o dado é transmitido, faz (TI) = 1, e é gerada uma interrupção da ; fonte de interrupção da interface do canal de comunicação serial (salta ; para o endereço 0023h – rotina de atendimento à comunicação serial).
	CLR TI	; (TI) \leftarrow 0 (libera canal de comunicação serial para a transmissão de ; um novo dado).
	CPL A	; (A) \leftarrow complemento de um do (A).
	SJMP LOOP	; loop do programa principal.
	END	

4 - Simule (execute passo a passo) cada instrução do programa, mostrando sua representação simbólica e as condições iniciais e finais dos conteúdos dos registradores e das posições de memória afetados. Descreva também, sucintamente, qual é a função de cada bloco de programa no contexto total do programa.

5 - Simule o programa-fonte estruturado que utiliza um dos membros da família de microcontroladores MCS-51, que seja capaz de executar as seguintes atividades:

- programar o canal de comunicação serial no Modo 1 com *baud rate* de 2,4KHz, considerando que o oscilador a cristal apresente uma freqüência de $f_{osc} = 11,059\text{MHz}$;
- obter as informações recebidas do canal de comunicação serial e armazená-las no conteúdo das portas 0, 1, 2 e 3. Considere que as portas 0, 1, 2 e 3 estejam ligadas a quatro interfaces que acionam um conjunto de oito leds cada uma;
- o programa principal deve ser escrito no endereço de memória de programa 0700h.

Solução:

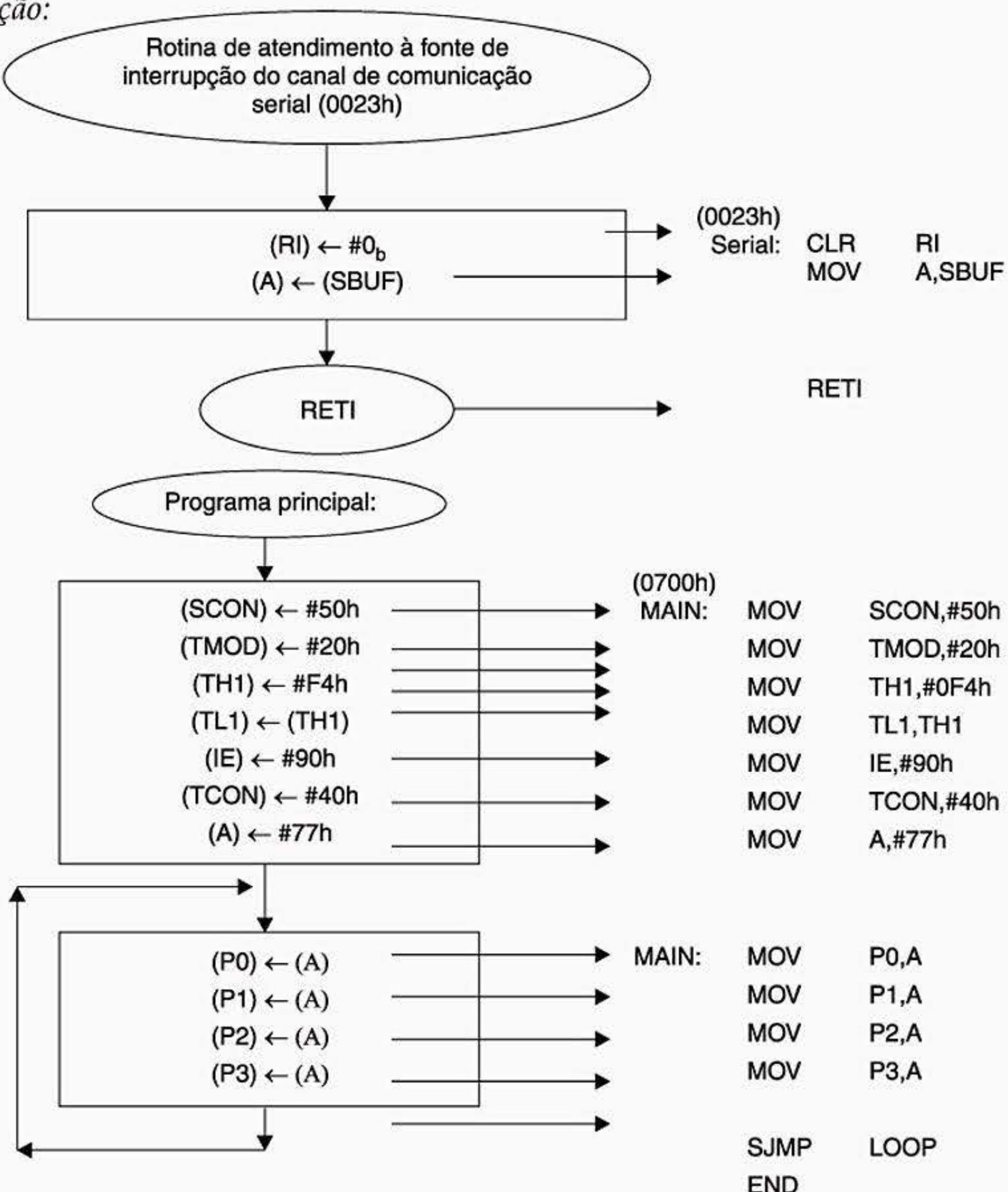


Figura G.3(a) Fluxograma da solução do Exercício 5.

```

DEFSEG EXEM, ABSOLUTE ; Define o segmento de programa EXEM
SEG EXEM
ORG 0000h ; Após o reset da CPU, o programa prossegue no endereço 0000h
AJMP 0700h ; (durante o acionamento da chave de reset ou após a 'energização' do
            ; sistema, por meio do circuito de reset formado por um resistor
            ; em paralelo com um diodo e em série com um capacitor que está
            ; ligado ao pino de reset da CPU). Como foi pedido que o programa
            ; principal fosse escrito no endereço de memória de programa 0700h,
            ; um salto incondicional foi utilizado para tal posição de memória.

ORG 0023h ; Após uma interrupção do canal serial, o programa prossegue no
            ; endereço '0023H', retorna a partir da interrupção de comunicação serial.
    
```

CLR	RI	; (RI) \leftarrow 0. Limpa o <i>flag</i> de interrupção de comunicação serial
MOV	A,SBUF	; (A) \leftarrow (SBUF). Obtém o dado.
RETI		; Retorna a partir da rotina de serviço da interrupção serial
ORG	0100H	; Programa principal
MAIN: MOV	SCON,#50h	; (SCON) \leftarrow # 010 0000 _b . Programa o canal serial como UART ; de 8 bits, baud rate variável e habilita a recepção serial.
MOV	TMOD,#20h	; (TMOD) \leftarrow #0010 0000 _b . Programa o <i>timer</i> 1 no Modo 2 ; (8 bits, com recarregamento automático).
MOV	TH1,#0F4h	; (TH1) \leftarrow F4h. Valor para gerar baud rate de 2,4KHz (recarregador ; automático).
MOV	TL1,TH1	; (TL1) \leftarrow (TH1). Valor inicial para um <i>baud rate</i> de 2,4KHz
MOV	IE,#90h	; (IE) \leftarrow #1001 0000 _b . Faz (EA) = 1 (cada interrupção é habilitada pelo seu ; bit habilitador); (ES) = 1 (Habilita a interrupção de comunicação serial e ; não habilita a interrupção do <i>timer</i> 1).
MOV	TCON,#40h	; Liga/executa o <i>timer</i> 1.
MOV	A,#77h	; (A) \leftarrow #77h
LOOP: MOV	P0,A	; (P0) \leftarrow (A)
MOV	P1,A	; (P1) \leftarrow (A)
MOV	P2,A	; (P2) \leftarrow (A)
MOV	P3,A	; (P3) \leftarrow (A)
SJMP	LOOP	; loop do programa principal.
END		

Figura G.3(b) Programa-fonte da solução do Exercício 5.

6 - Simule (execute passo a passo) cada instrução do programa, mostrando sua representação simbólica e as condições iniciais e finais dos conteúdos dos registradores e das posições de memória afetados. Descreva também, sucintamente, qual a função de cada bloco de programa no contexto total do programa.

7 - Programa de recepção serial no Modo 1: esse programa transmite, serialmente, 55h e Aah, considerando uma *baud rate* = 2,4 KHz, que a $f_{osc} = 11,059$ MHz, o *timer* 1 no Modo 2 de operação (valor de recarregamento automático = F4h). Dessa forma, mude o programa anterior, na parte do programa principal, da seguinte maneira:

DEFSEG	EXEM, ABSOLUTE	; Define o segmento de programa EXEM
SEG	EXEM	
ORG	0000h	; Após o <i>reset</i> da CPU, o programa prossegue no endereço 0000h
AJMP	0100h	; (durante o acionamento da chave de <i>reset</i> ou após a 'energização' do ; mesmo, por meio do circuito de <i>reset</i> formado por um resistor ; em paralelo com um diodo e em série com um capacitor que está ligado ; ao pino de <i>reset</i> da CPU). Como foi pedido que o programa principal ; fosse escrito no endereço de memória de programa 0700h, foi utilizado ; um salto incondicional para tal posição de memória.
ORG	0023h	; Após uma interrupção do canal serial, o programa prossegue no ; endereço 0023h e retorna a partir da interrupção de comunicação serial.

CLR	RI	; (RI) \leftarrow 0. Exclui o <i>flag</i> de comunicação serial.
MOV	A,SBUF	; (A) \leftarrow (SBUF). Obtém o dado.
RETI		; Retorna a partir da rotina de serviço da interrupção serial.
ORG	0100H	; Programa principal.
MAIN: MOV	SCON,#50h	; (SCON) \leftarrow # 010 0000 _b . Programa o canal serial como UART ; de 8 bits, <i>baud rate</i> variável, e habilita a recepção serial.
MOV	TMOD,#20h	; (TMOD) \leftarrow #0010 0000 _b . Programa o <i>timer 1</i> no Modo 2 ; (8 bits, com recarregamento automático).
MOV	TH1,#0F4h	; (TH1) \leftarrow FDh. Valor para gerar <i>baud rate</i> de 2,4KHz (recarregamento ; automático).
MOV	TL1,TH1	; (TL1) \leftarrow (TH1). Valor inicial para um <i>baud rate</i> de 2,4KHz
MOV	IE,#90h	; (IE) \leftarrow #1001 0000 _b . Faz (EA) = 1 (cada interrupção é habilitada pelo seu ; bit habilitador); (ES) = 1 (Habilita a interrupção de comunicação serial) ; e não habilita a interrupção do <i>timer 1</i> .
MOV	TCON,#40h	; Liga/executa o <i>timer 1</i> .
MOV	A,#77h	; (A) \leftarrow #77h
LOOP: MOV	P0,A	; (P0) \leftarrow (A)
MOV	P1A	; (P1) \leftarrow (A)
MOV	P2A	; (P2) \leftarrow (A)
MOV	P3,A	; (P3) \leftarrow (A)
SJMP	LOOP	; loop do programa principal.
END		

G.4 Para o relatório

1 - Mude o programa número 1 para atender às seguintes condições:

- ◆ operar no Modo 2;
- ◆ $baud\ rate = f_{osc}/32$;
- ◆ definir o nono bit dos dados a serem transmitidos, iguais ao valor do bit de paridade.

2 - Mude o programa número 1 para transmitir com $baud\ rate = 2,4\text{KHz}$, considerando $f_{osc} = 11,059\text{MHz}$.

3 - Mude o programa anterior para que a transmissão opere no Modo 3 e que o nono bit seja igual a 1.

4 - Faça um programa estruturado (fluxograma e programa-fonte), que transmita serialmente uma *string* de caracteres (bytes em posições de memória subsequentes), localizada no buffer de memória cujo endereço inicial é 30h e o endereço final é 50h. A transmissão deve ser com UART de 8 bits com freqüência de comunicação de 1,2KHz. Quando finalizada a comunicação serial de toda a *string* de caracteres, armazene no conteúdo da posição de memória 51h o dado CFh e armazene no conteúdo da porta 1 o dado AAh.

- 5 - Faça um programa estruturado (fluxograma e programa-fonte), que receba 20 caracteres do canal de comunicação serial. Esses dados devem ser armazenados na memória RAM interna a partir do conteúdo da posição de memória 40h.
- 6 - Faça um programa que após transmitir um byte no Modo 0 e aguarde receber o byte 7Fh. Quando acontecer isso, escreva no conteúdo da porta 1 o byte 34h.
- 7 - Resolva os exercícios do Capítulo 8.

ÍNDICE REMISSIVO

A

Ação de tomada de decisões, 5
Acesso, tempo de, 23
Acionamento
 de bips, 10
 de Gates de tiristores, 135
 de um bip, 6
 de um relé, 6
 de uma chave mecânica, 139
 monitoração do, 140
Acionamentos
 de interfaces por entrada, 133
 monitorando, 133
Acumulador, 51, 134
 conteúdo do, 87
Adição, operação de, 50
Alto-falantes, 10
Analógicos/digitais, conversores, 4
Aquivo-binário, 80
Área de armazenamento de informações, 9
Arithmetic Logic Unit, 7. *Veja também ULA*
Armazenamento
 da informação, 83
 de dados, memória de, 5

de informação, unidade básica de, 3
Armazenamento de informações
 memória de, 9
Armazenamento de programa,
 memória de, 5, 8
Armazenamento temporário de
 informações, 7
Arquitetura
 básica do microcontrolador 8051,
 18
 de sistemas microprocessados/
 microcontrolados, 11
Arquivo
 binário, 191
 em código ASCII, 80
Assembly, 116
Atendimento
 à fonte de interrupção, rotina de,
 24
 rotinas de, 26
 sub-rotina de, 44
Automação
 embarcada, 4
 industrial, 4
 predial, 4
 residencial, 4

B

Baixa potência
modo de, 18
operação de, 44

Baixo, linguagem de nível, 77

Banco de registradores, 108

Bancos de registradores, 30

Bandeiras de sinalização, 49. *Veja também flags*

Barramento, 10
bidirecional, 10
de dados, 10, 43, 130
de endereços, 10, 11, 27
de temporização de controle, 10
interno, 126
unidirecional, 10

Baud rate, 45, 153, 174

Binary digit, 1

Bip, acionamento de, 6

Bips, acionamento de, 10

Bit a bit, manipulação, 78

Bit, 1
de paridade, 174
desabilitador global, 149

Bits
de entrada de informação, 131
de sinalização, 85
sinalizadores, 7

Bloco de temporização e controle, 6, 8, 11

Blocos
básicos de microcomputadores, 4, 5
diagramas de, 79

Booleanas, instruções, 129

Bounce, 139

Buffer
de entrada, 125
de memória de dados, 83
de memória, 90
de memória, controles do, 91
de recebimento, 168

de transmissão, 168
informação do, 85

Busca
das instruções na memória de programa interna, 26
ciclo de, 13

Byte, 1
de código, 27

C

Caixa eletrônico de banco, 6

Caminho de processamento da informação, 85

Canal de comunicação serial, 5, 10, 25, 41, 104, 147, 153, 169, 171

Capacitor, 40

Característica de operação, 43

Carry bit flag, 50

Carry bit, 56

Cartão magnético, 6

Central Processing Unit, 3

Cérebro, 3

Chave aberta, 130

Chave mecânica, 139
acionamento de uma, 139

Chaveamento, 139
ruído de, 141

Chaves mecânicas, eliminando o ruído gerado por, 138

Chip enable, 39

Chip select, 39

Ciclo
de busca, 13
de execução, 13, 14
de instrução, 13
de máquina, 42, 43, 151, 153
quantidade total de, 136

Circuito
de reset, 12
integrado, 3, 39
pseudobidirecional, 128
esquema elétrico de um, 48

- Circuitos
 integrados, 10
 lógicos, 3
 variedade de, 3
- Clock, 42
 de transmissão, 169
 do deslocamento, 171
 do oscilador, 153
 fonte de, 41
- Cobre, trilhas de, 10
- Código
 byte de, 27
 da instrução, 109
 de barra, leitores de, 154
 de máquina, programa em, 78
 magnético, leitores de, 154
- Código-objeto, 78
- Códigos
 ASCII, 191
 de máquina, 102, 191
- Compilação, 78
 do programa em Assembly, 191
- Complemento de dois, 51
- Computação, conceitos básicos da, 1
- Computador, transmissão de dados para outro, 6
- Computadores de grande porte, 4
- Comunicação
 de multiprocessadores, 174
 paralela, portas de, 4
- Comunicação serial, 107, 114
 assíncrona, 54
 de dados, 167
 velocidade de, 171
- interface de, 168
 por buffer, 168
 canal de, 10
 portas de, 4
- Conceitos básicos da computação, 1
- Condição dos registradores, 193
- Condições de estresse, 80
- Confiabilidade, 80, 118
- Conflito de sinais, 35
- Conhecimento técnico especializado, 77
- Conjunto
 de instruções do dispositivo, 79
 de instruções, 2, 59
- Contador
 de eventos, 152
 de programa, 12
 ascendente binário, 162
 ascendente decimal, 163
 decimal de números ímpares, 163
 descendente haxadecimal, 162
- Contadores de display de cristal líquido, 39
- Conteúdo
 do acumulador, 87
 do registrador, 87, 106, 109
- Controle
 de leitura, sinal de, 26
 de ponto, sistema de, 167
 digital de máquina e de processo, 125-146
 do fluxo de informações, 8
 do loop, 85, 92
 gerenciamento de, 23
- Controles do buffer de memória, 91
- Conversores
 analógicos/digitais, 4
 digitais/analógicos, 4
- CPU desenergizada, 7
- Cristal, freqüência do, 136
- Curto-circuito, 14
- Custo/benefício, 89
- D**
- Dados internos, memória de, 28
- Dados
 barramento de, 10, 43, 130
 comunicação serial de, 167
 memória de, 4, 27-28
 operações de movimentação de, 81

- transferência de, 6
 - Data Pointer**, 23
 - registrator, 56
 - Decodificação**, 6
 - da senha, 6
 - dos endereços, 36
 - Decodificador**, saída do, 39
 - Definição** de sub-rotina, 102
 - Descrição** da pinagem do 8051/31, 21-23
 - Desenho em forma de rascunho, 86, 90
 - Deslocamento**
 - clock do, 171
 - de recepção, 171
 - registrator de, 171
 - Destino**, registrator de, 171
 - Desvios condicionais ou incondicionais, 190
 - Diagrama**
 - de tempo, 48
 - funcional do latch, 126
 - funcional simplificado, 172
 - Diagramas**
 - de blocos, 79
 - de fluxos de dados, 79
 - Digitais/analógicos**, conversores, 4
 - Direção** do fluxo de informações, 10
 - Display** de cristal líquido, 39
 - Displays**, 10
 - Dispositivo**
 - de entrada e saída, 37
 - externo, 129
 - latch, 27
 - semicondutor, 3, 4
 - Dispositivos**
 - CHMOS, 18
 - de entrada e saída, 36
 - de entrada, 5
 - canal de comunicação serial, 5
 - teclado, 5
 - de saída, 6
 - acionamento de um bip, 6
 - acionamento de um relé, 6
 - escrita na memória de vídeo, 6
 - transmissão de dados para outro microcomputador, 6
 - Doctor Norton Utilities**, 3
 - Documentação** do programa, 81
 - DPTR**, registrator, 23
- E**
- E/S externa**, uso de, 39
 - E/S**, mapeamento de, 39
 - Edição** do programa-fonte em Assembly, 189
 - Editores** de texto, 3, 191
 - Elaboração**
 - de programas simples, 77
 - de projetos de equipamentos inteligentes, 78
 - Elétrica**, energia, 8
 - Empilhamento** físico de objetos, 106
 - Emulação** do programa, 80
 - Endereçamento**
 - combinado, 56
 - da memória de programa, 27
 - da RAM, 28
 - direto, 30, 57
 - espaço de, 36
 - imediato, 58
 - indexado por registrator, 91
 - por registrator-base mais o registrator indexado, 58
 - indireto, 23, 58
 - misto, 56
 - por registrator, 56
 - faixa de, 37-39
 - modos de, 55-56
 - Endereço**
 - da instrução, 133
 - de destino, 63
 - de memória, 11, 163