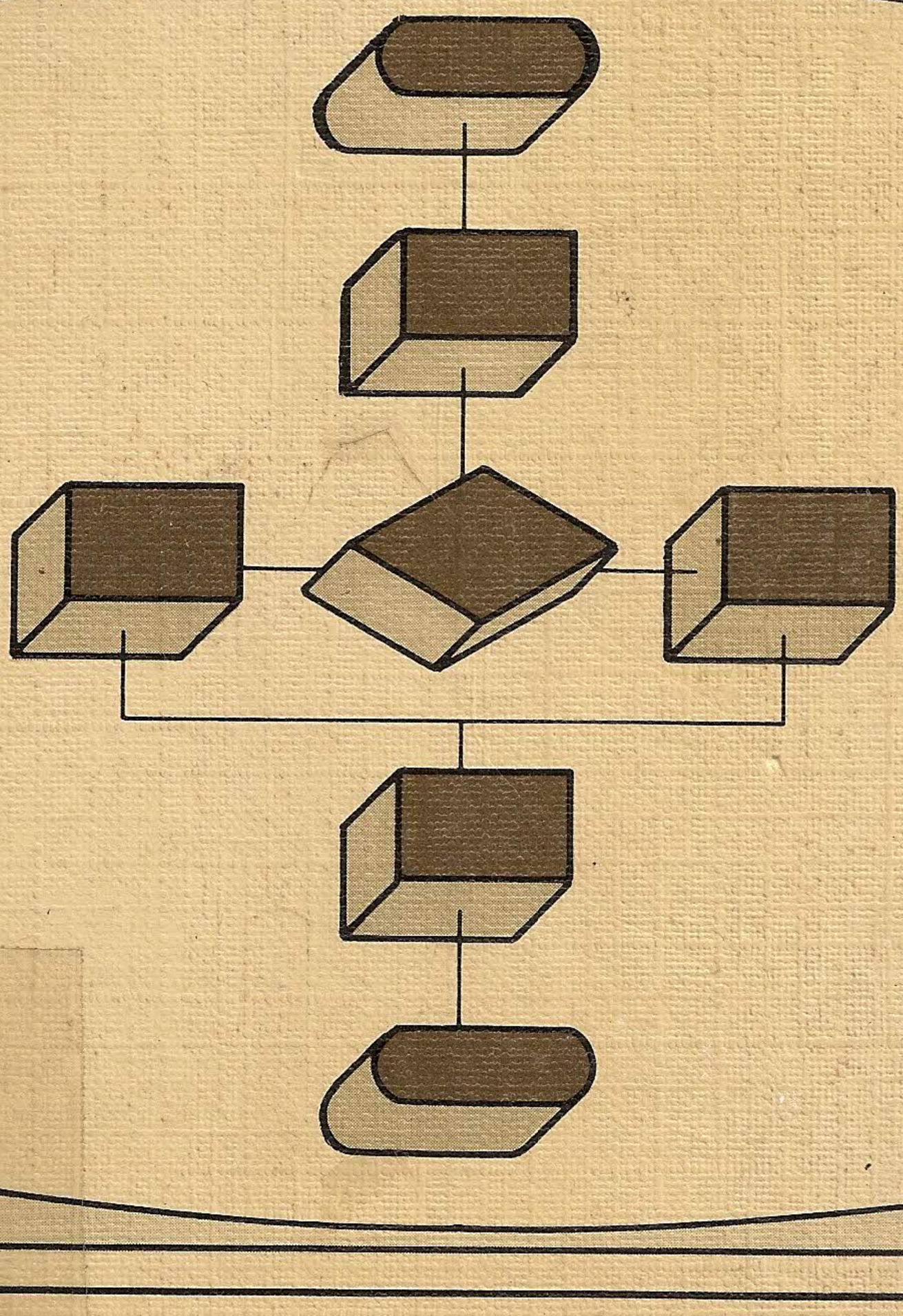
ENG. ANTONIO CARLOS JOSÉ FRANCESCHINI VISCONTI

MICROPROCESSADORES

8080 e 8085

SOFTWARE

5.ª EDIÇÃO



VOLUME 2

ÉRICA

PUBLICAÇÕES EM ELETRÔNICA

TTL/CMOS Circuitos Integrados - Vol. 1 e 2

Eletrônica Digital com circuitos integrados das famílias TTL e CMOS, com características e aplicações abrangendo circuitos combinacionais e seqüencias, com exemplos, projetos e detalhes prático quanto à implementação. 2.ª Edição, 408 páginas.
Autor: **João Batista de Azevedo Júnior**

PROBASIC - Programação em Basic

O livro se distina ao público de uma maneira geral interessado no estudo da linguagem BASIC e, em particular à didática da mesma.

Contém Instruções, Comandos e Funções usados no BASIC apresentadas numa forma gradativa com exemplos e programas. 3.ª Edição, 172 páginas.
Autor: **Ferdinando Natale**

TELECOMUNICAÇÕES

Modulação em Amplitude e em Freqüência — Sistemas Pulsados PAM, TWM, PPM, PCM — Formulário de Trigonometria, Filtros, Osciladores, Propagação de Ondas, Linha de Transmissão, Antenas, Distribuição do Espectro de Freqüência. 3.ª Edição, 460 páginas.
Autor: **Eng.º Alcides Tadeu Gomes.**

Teoria e Desenvolvimento de Projetos de Circuitos Eletrônicos

Diodos, Transistores de Junção, FET, MOS, UJT, LDR, NTC, PTC, SCR, Transformadores, Amplificadores Operacionais e suas aplicações em Projetos de Fontes de Alimentação, Amplificadores, Osciladores, Osciladores de Relaxação e outras. 11.ª Edição, 580 páginas.
Autores: **Eng.ºs. Cipelli/Sandrini**

LIVROS ÉRICA EDITORA LTDA.

MICROPROCESSADORES
8080
e
8085

CIP – Brasil. Catalogação-na-Fonte
Câmara Brasileira do Livro, SP

Visconti, Antônio Carlos José Franceschini, 1952-
V814m Microprocessadores 8080 e 8085 / Antonio Carlos
v.2- José Franceschini Visconti. -- São Paulo : Érica,
1981-

Bibliografia.
Conteúdo: SOFTWARE

1. Microprocessadores - Programação I. Título.

17. CDD-651.8
81-1217 18. -001.642

Índices para catálogo sistemático:

1. Microprocessadores 8080 : Programação : Processamento de dados 651.8 (17.) 001.642 (18.)
2. Microprocessadores 8085 : Programação : Processamento de dados 651.8 (17.) 001.642 (18.)
3. Programação : Microprocessadores 8080 : Processamento de dados 651.8 (17.) 001.642 (18.)
4. Programação : Microprocessadores 8085 : Processamento de dados 651.8 (17.) 001.642 (18.)

ENG. ANTONIO CARLOS JOSÉ FRANCESCHINI VISCONTI

MICROPROCESSADORES 8080 e 8085

SOFTWARE

VOLUME 2

1986
5.ª EDIÇÃO
LIVROS ÉRICA EDITORA LTDA.

Produção Editorial:
PAULO ROBERTO ALVES

Composição dos textos:
ROSELI BARBARESCO DE OLIVEIRA

Desenhos:
JAYME VIEIRA JÚNIOR

Revisão:
GUARACIABA MICHELETTI

LIVROS ÉRICA EDITORA LTDA.
Rua Jarinu, 594 - Tatuapé - São Paulo
Fone: 294-8686 - C.G.C. 50.268.838/0001-39
Caixa Postal 15.617

Dedico à:

ROSA MARIA,
MARIA CRISTINA,
ROSA CRISTINA e
CRISTINA MARIA.

Em especial a:

LEA e ARTHUR.

PREFÁCIO

Este Volume 2 completa o nosso estudo básico de microprocessadores, iniciado no Volume 1, onde foi apresentada a matéria referente ao hardware de sistema, sendo utilizado para base dos estudos os microprocessadores 8080 e 8085.

Neste Volume 2, baseado nos mesmos princípios do Volume anterior, desenvolvemos a matéria referente ao software.

O estudo do software abrange desde a apresentação, conceituação e ilustração de programa, até a análise das técnicas de desenvolvimento. A maior ênfase é dada à programação a nível de máquina o que proporciona os conhecimentos necessários tanto para o projetista de sistema como para o projetista de programa.

A estrutura e técnica para programação de sistemas de grande porte para processamento de dados dependem mais da estrutura das linguagens de alto nível, e cada uma delas requer um estudo separado, o que não é o nosso objetivo. Entretanto, o conhecimento básico necessário como pré-requisito para estudos destas linguagens é inteiramente abordado pelo conteúdo desta obra.

No decorrer dos capítulos, procuramos analisar detalhadamente o conjunto completo de instruções, alguns programas especialmente selecionados e as principais técnicas de programação, sem nos aprofundarmos em embasamentos teóricos, facilitando o entendimento e demostrando as aplicações e os modos de operação dos microprocessadores.

A divisão da obra em dois Volumes foi feita para que cada parte da matéria fosse apresentada de uma maneira independente e completa, facilitando o estudo.

Nosso objetivo foi suprir tanto as necessidades de material didático para aqueles que se iniciam na área dos microprocessadores como as necessidades de material de consulta para aqueles que fazem uso das aplicações práticas.

A todos aqueles que nos ajudaram para que isto fosse possível, o nosso muito obrigado.

O autor.

SUMARIO

Cap. I PROGRAMAÇÃO

I.1 INTRODUÇÃO

O microcomputador é um circuito eletrônico digital programável, capaz de executar tarefas desde as mais simples, como operações aritméticas até os mais sofisticados cálculos, e exercer controle automático de equipamentos, como um semáforo de trânsito ou um torno na fabricação de peças.

Apesar de ser ao todo um sistema complexo, a sua utilização é, basicamente, simples e os campos de aplicação quase que ilimitados, devido ao fato de ser um sistema programável.

O aspecto físico de sistemas com microprocessador, isto é, os circuitos integrados de que é constituído, as suas interligações e suas funções são apresentados no Volume I desta obra, que aborda a parte do hardware de microprocessadores.

A parte de software que corresponde a programação do sistema á apresentada neste Volume. O objetivo nosso é de mostrarmos as técnicas de programação, particularizando o estudo nos microprocessadores 8080 e 8085, seguindo a linha do Volume I.

I.2 INSTRUÇÃO

De uma maneira geral, a programação é a seqüência de operações que o sistema deve executar para que a tarefa determinada seja realizada. Cada operação corresponde a uma instrução que pode ser interpretada e executada pelo microprocessador. As instruções são constituídas por uma série de bits. Esses bits são decodificados e acionam as variáveis de controle internas ao sistema para que a operação correspondente à instrução seja realizada. Por exemplo: a instrução de transferência de um dado contido na memória para um registrador, coloca no bus de endereço o endereço da posição de memória que contém o dado, aciona as váriáveis de controle para leitura da memória e as de escrita no registrador. Desta maneira, a leitura da memória no endereço se

lecionado fará com que o seu conteúdo seja colocado no bus de dados e, neste instante, o registrador lerá a informação deste bus, realizando a transferência do dado da memória para o registrador.

I.3 PROGRAMAÇÃO

A programação de um sistema é a construção correta da seqüência de instruções que o sistema deve realizar para conduzir a solução de um problema. Esta seqüência de instruções é variável, pois a programação possui uma ampla variedade de alternativas e opções que dependem apenas do programador que, através de várias maneiras, atinge os mesmos resultados.

Para ter-se uma idéia mais clara de programa, pode-se fazer uma analogia com uma máquina de calcular. Essa máquina possui, por exemplo, dois registradores (R_1 e R_2) onde são armazenados os dois dados que serão utilizados na operação de cálculo; um outro registrador (R_3) para armazenamento do resultado e as teclas para as operações (+, -, x, ÷).

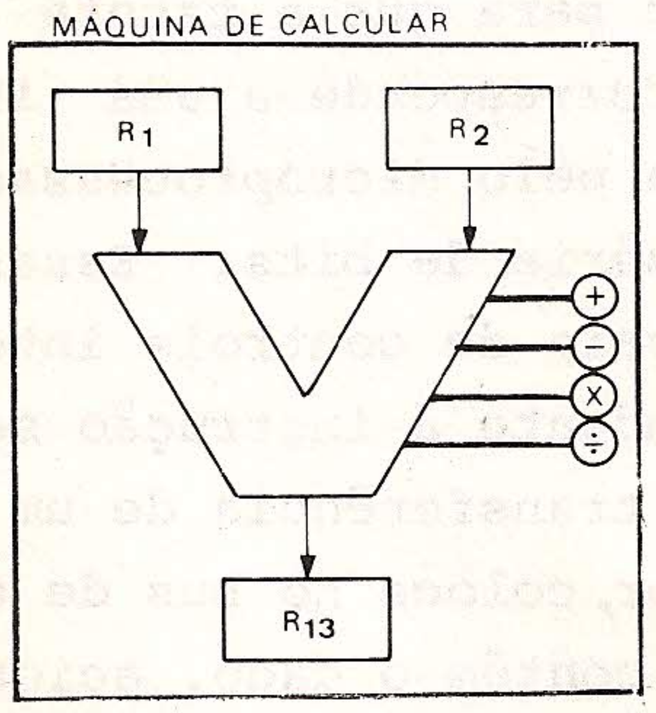
Para calcular a soma de dois números (x e y), uma pessoa pode executar as seguintes operações:

a.) Colocar o número x no registrador R_1.

b.) Colocar o número y no registrador R_2.

c.) Executar a operação de soma do conteúdo dos registradores.

A figura I.1 ilustra este exemplo:



SEQÜÊNCIA DE OPERAÇÕES

1 - Colocar x em R_1.

2 - Colocar y em R_2.

3 - Acionar a tecla +.

Fig. I.1 - Soma de dois números por um operador.

Essas operações que foram realizadas na máquina de calcular podem ter uma instrução equivalente a cada uma delas, que, logicamente, possuem uma configuração de bits diferentes em cada microprocessador. As instruções podem ser representadas por símbolos que podem indicar suas funções, como por exemplo:

$$R_1 \leftarrow [M_1]$$

transfere o conteúdo da memória de endereço M_1 para o registrador R_1.

$$R_2 \leftarrow [M_2]$$

transfere o conteúdo da memória de endereço M_2 para o registrador R_2.

$$R_3 \leftarrow [R_1] + [R_2]$$

soma o conteúdo dos registradores R_1 e R_2 e coloca o resultado em R_3.

Essas instruções correspondem as operações que devem ser executadas para que a tarefa de soma de dois números seja realizada. As instruções a serem executadas devem ser colocadas em seqüência na memória de programa, assim como os dados devem ser colocados na memória de dados e um decodificador de instrução faz a geração das variáveis de controle necessários para a execução das instruções.

A figura I.2 ilustra o programa correspondente as operações que foram executadas para a realização da soma de dois números, carregado devidamente em um sistema simplificado.

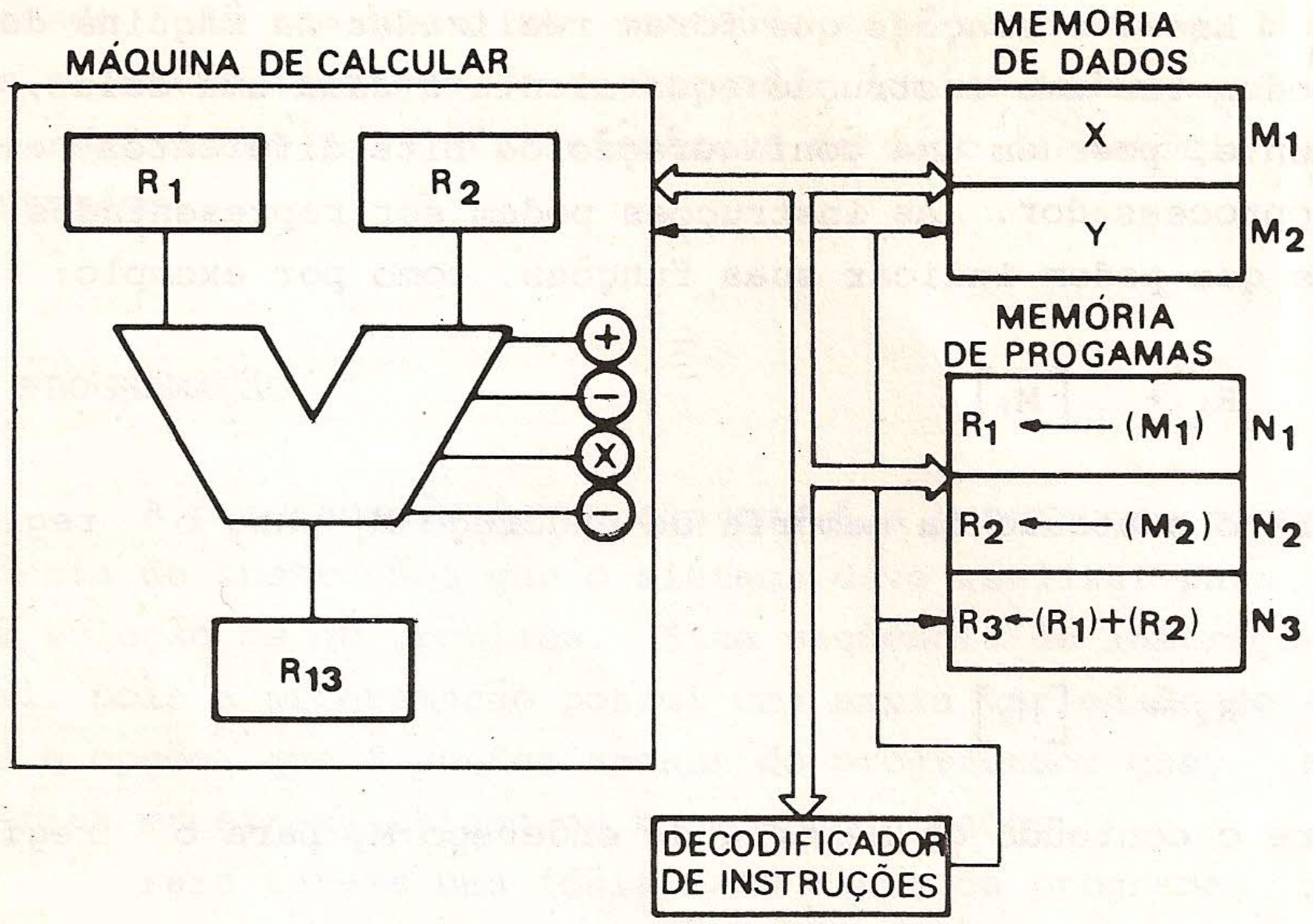


Fig. I.2 - Soma de dois números programados.

I.4 PROCESSAMENTO

O processador ou unidade central de processamento é a parte do sistema que faz o processamento das informações para que as tarefas especificadas sejam executadas.

As tarefas a serem executadas são as instruções que devem estar armazenadas na memória de programa, dispostas na seqüência pela qual devem ser executadas, formando o programa.

A unidade central de processamento possui um registrador que é designado por contador de programa, que contém o endereço da próxima instrução que deve ser executada. Toda vez que uma instrução é retirada da memória pela unidade central de processamento, automaticamente, o contador de programa é incrementado, para que após o processamento desta instrução, quando a unidade central de processamento for buscar a próxima instrução, bastará usar o endereço contido no contador de programa.

Toda vez que um microprocessador é ligado ou ressetado, automaticamente o seu contador de programa é zerado. Desta maneira, a primeira tarefa que a unidade central de processamento irá realizar é a execução da instrução contida na posição de me

mória de endereço "0000".

Cada instrução possui duas fases distintas: o ciclo de busca e o ciclo de execução.

Durante o ciclo de busca de uma instrução, a unidade central de processamento faz com que o conteúdo do contador de programa seja colocado no bus de endereço, endereçando, desta maneira, a posição de memória que contém a instrução que deve ser executada. Esta memória é então lida e o seu conteúdo transferido para um registrador interno da unidade central de processamento, que é o registrador de instrução. Após isto, ainda dentro do ciclo de busca, o contador de programa é incrementado para que este contenha agora o próximo endereço da memória de programa.

Durante o ciclo de execução, a instrução que após o ciclo de busca está no registrador de instrução, é decodificada e enviada à unidade de controle que aciona as variáveis de controle necessárias para a execução das atividades desta instrução.

Após o término do ciclo de execução, a unidade central de processamento inicia o ciclo de busca da instrução seguinte, executa esta e assim sucessivamente, processando todo o programa numa seqüência ordenada.

Para a análise do programa visto anteriormente, da soma de dois números, vamos supor que este programa será processado no sistema apresentado na figura I.3.

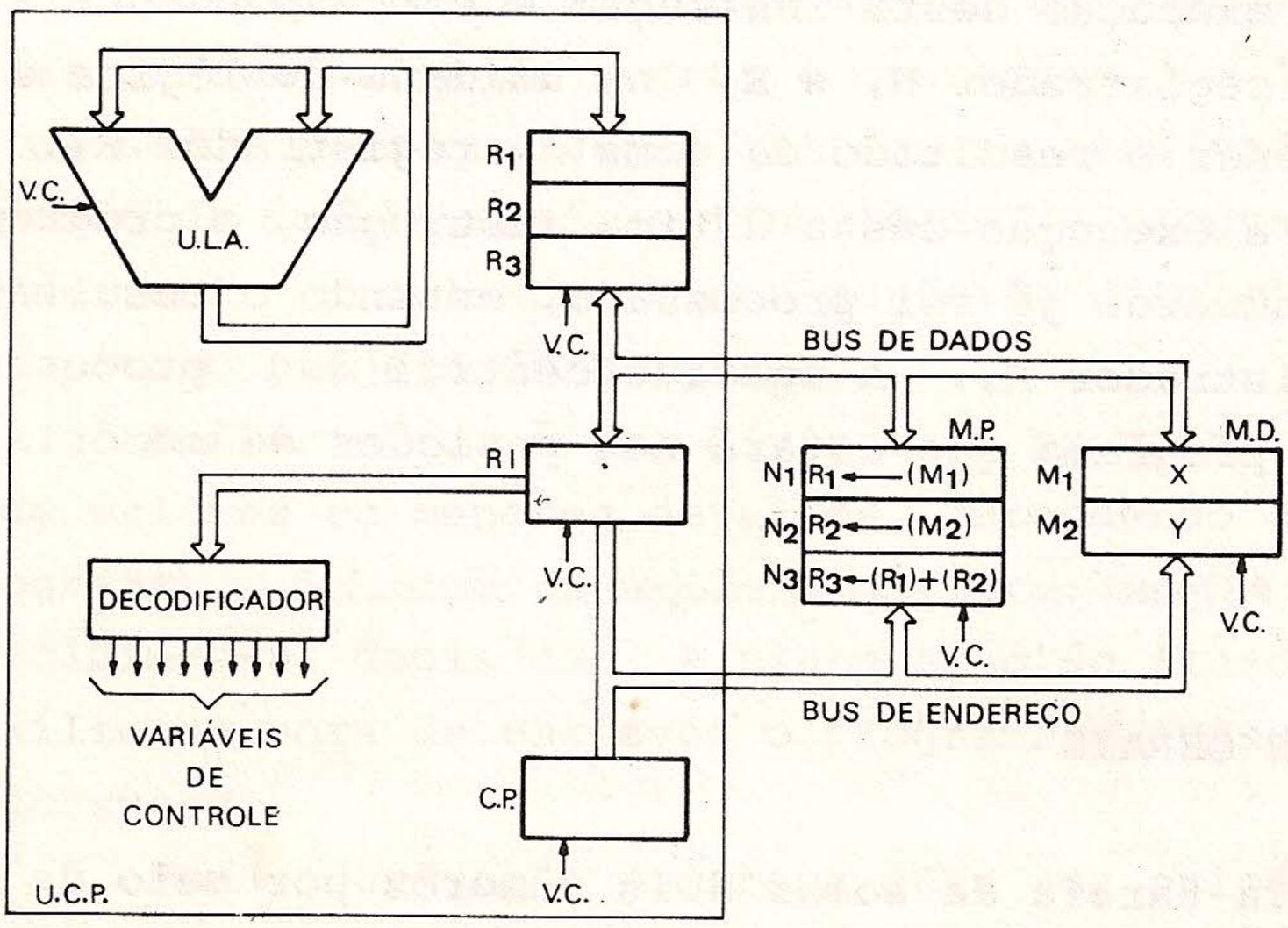


Fig. I.3 - Sistema para a soma de dois números.

A unidade central de processamento (U.C.P.) é composta por: três registradores gerais (R_1, R_2, R_3), um registrador de instrução (R.I.), um contador de programa (C.P.), uma unidade de lógica e aritmética (U.L.A.) e um decodificador que acionará as variáveis de controle internas e externas.

Interligadas à unidade central de processamento por meio do bus de dados e de endereço, estão as memórias de programa e as memórias de dados.

Inicialmente, o contador de programa deverá estar com o endereço de memória N_1. Inicia-se o ciclo de busca desta instrução. A instrução que está contida no endereço N_1 é transferida ao registrador de instrução e o contador de programa é incrementado. Durante o ciclo de execução, a instrução é decodificada e, pelo acionamento das variáveis de controle, o conteúdo da posição de memória de endereço M_1, que o número x, é transferido para o registrador R_1. Inicia-se, então, o ciclo de busca da instrução seguinte de endereço N_2, transferindo o conteúdo desta memória para o registrador de instrução e incrementando novamente o contador de programa. Durante o ciclo de execução desta instrução o conteúdo da posição de memória de endereço M_2, que é o número y, é transferido para o registrador R_2. Na busca da próxima instrução, o conteúdo da memória de endereço M_3 é transferida para o registrador de instrução e o contador de programa é incrementado. A execução desta instrução é correspondente a soma do conteúdo do registrador R_1 e R_2 na unidade de lógica e aritmética e armazenar o resultado da soma no registrador R_3.

Após a execução desta última instrução, o programa da somá dos dois números já foi processado, estando o resultado disponível no registrador R_3. A unidade central de processamento irá executar o programa que estará nas posições de memória consecutivas.

I.5 CONCEITOS GERAIS

Aquela tarefa de somar dois números por meio de um operador, como mostrado na fig. I.1, está agora programada e carregada em um sistema capaz de executar a mesma tarefa automatica

mente. Este programa foi bastante simples, por isto foi possível escrevê-lo diretamente, porém, em um caso real, os programas são mais complicados e maiores do que este, exigindo uma metodologia de trabalho, sem o que, torna-se impossível fazer uma programação correta.

Essencialmente, o projeto de um programa a ser processado por um computador deve ter os seguintes procedimentos:

- análise do problema.
- determinação do algoritmo.
- elaboração do fluxograma.
- escrever o programa em linguagem simbólica.
- traduzir o programa para uma linguagem de máquina.
- testar e corrigir o programa

Na análise do problema deve ser determinado de maneira bem clara quais os objetivos que devem ser alcançados, exatamente que tarefa deve ser realizada. Se este programa for processado por um equipamento já disponível, verificar sua capacidade para execução do programa, sua capacidade de memória, seus periféricos, sua unidade central de processamento. Caso este programa seja para um equipamento que está sendo desenvolvido, definir todo o hardware deste equipamento. Também deve ficar bem claro quais as saídas que devemos ter para cada entrada, e que dispositivos de entrada e saída fornecerão estas informações.

O algoritmo a ser determinado deve estabelecer quais equações ou conjuntos de regras e operações que devem ser submetidos os dados para que sejam obtidos os resultados esperados. Dependendo da complexidade e finalidade do programa os algoritmos podem ser mais complexos e sua perfeita determinação é fundamental para o projeto do programa.

O fluxograma é uma divisão do programa em tarefas (em níveis de maiores ou menores detalhes, dependendo da complexidade do programa) e definição da sequência que estas tarefas devem ser executadas incluindo-se decisões. A elaboração do fluxograma é de grande auxílio na hora de escrever o programa e quase fundamental para a correção.

Para escrever o programa deve ser primeiramente determinado que tipo de linguagem será utilizada, se uma linguagem de baixo nível ou alto nível. Isto depende de ter-se disponível

um sistema de desenvolvimento ou não e que tipos de linguagem este sistema tem condição de interpretar. Quanto mais complicado for o programa mais interessante será a utilização de linguagem de alto nível, pois esta aproxima-se mais do fluxograma, e também torna o programa mais fácil de ser entendido por possuir uma sintaxe mais clara.

A tradução do programa pode ser feita manualmente em programas escritos em linguagem de baixo nível, devido ao fato de que cada instrução possui um código de máquina correspondente, apesar de ser este um processo lento e suscetível de erros. Existem programas especiais para a tradução de programas até em linguagem de alto nível, facilitando muito este procedimento.

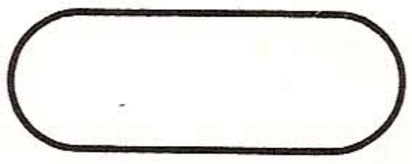
Para testar e corrigir o programa existe uma série de artifícios que auxiliam este procedimento, desde programas especiais até a partição do programa em pedaços para testes e correções separadas.

Esses procedimentos serão analisados melhor no decorrer dos capítulos seguintes; serão acompanhados através de exemplos, e são essenciais para bons e rápidos projetos de programas.

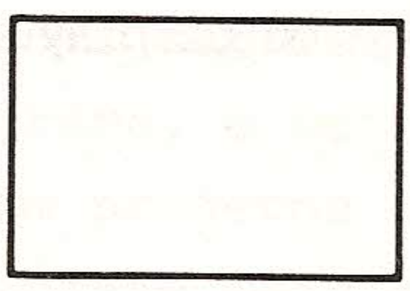
Cap. II FLUXOGRAMA

II.1 SÍMBOLOS

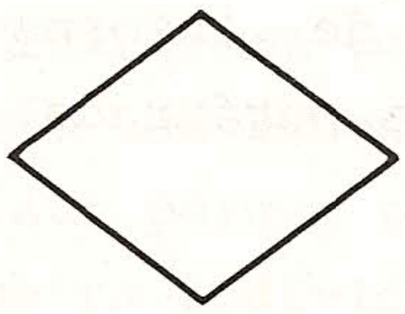
Os principais símbolos utilizados em fluxograma são:



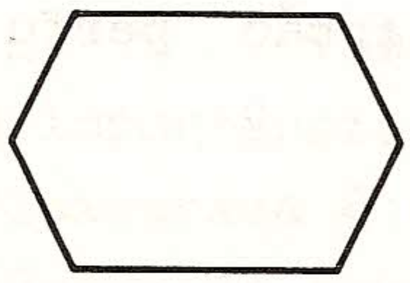
TERMINAL: Início, término ou interrupção de um programa.



PROCESSAMENTO: Uma ação que deve ser tomada.

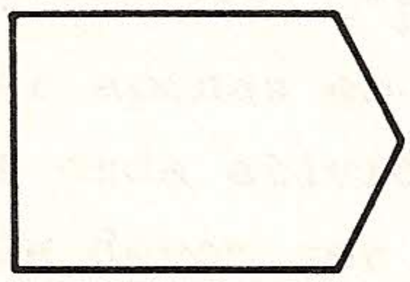


DECISÃO: Desvio para diversos outros pontos do programa, de acordo com uma situação testada.

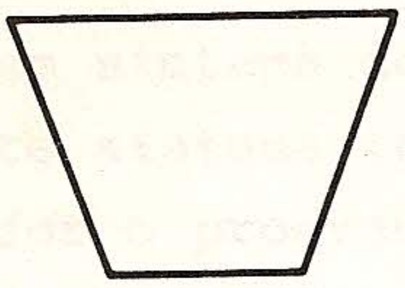


SUB-ROTINA: Um grupo de operações separadas do fluxo do programa.

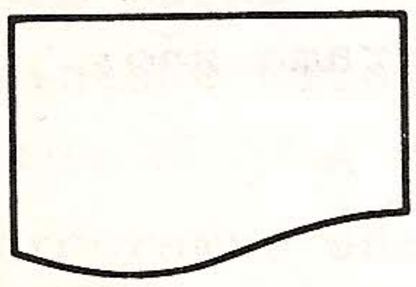
CONEXÃO: Indica a rota de prosseguimento do fluxograma.



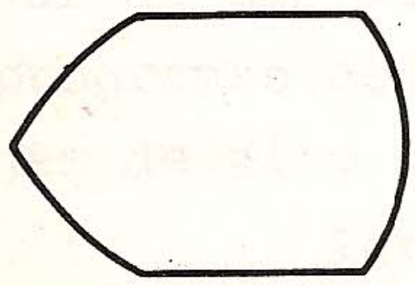
MODIFICAÇÃO DE PROGRAMA: Qualquer função que altera o próprio programa.



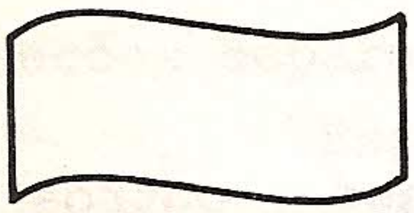
ENTRADA/SAÍDA: Qualquer função relacionada com dispositivos de entrada ou saída genéricos.



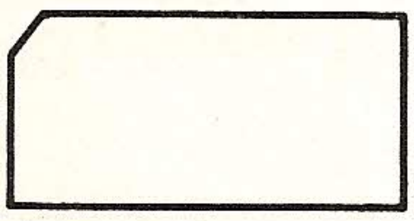
IMPRESSORA: Saída de informação através de impressora.



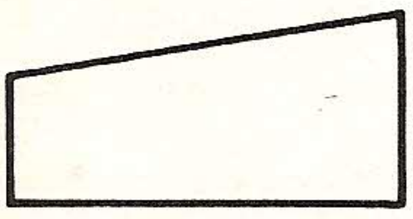
VISOR: Saída de informação através de um terminal de video ou display.



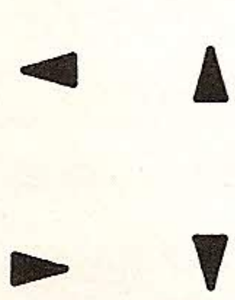
FITA PERFURADA: Entrada ou saída de informação através de fita perfurada.



CARTÃO PERFURADO: Entrada ou saída de informação através de cartão perfurado.



TECLADO: Entrada de informação através de teclado.



FLUXO: Direção de fluxo de processamento.

II.2 DEFINIÇÕES

O fluxograma é uma representação gráfica das tarefas de um programa, por meio de símbolos que fornecem uma visualização imediata do significado da tarefa.

O fluxograma ajuda a organizar o programa, permitindo o seu delineamento e possibilitando seguir, um a um, todos os passos, que serão executados em um programa.

Existem estruturas, definições rigorosas das regras de um fluxograma e uma variedade de símbolos para a construção de um fluxograma. Isto é interessante de ser seguido quando se trata de projetos de programas muito complicados que utilizam pessoas diferentes para a construção de partes diferentes do fluxograma, e outras pessoas para escrever o programa e corrigi-lo. Em projetos de programas para microcomputadores é preferível deixar-se mais em função da arte do programador a estrutura do seu fluxograma para que possa criar um estilo mais individual, pois não convém estipular princípios fixos ou normas de funcionamento. Isto porque os programas para microcomputadores são feitos de maneira individual, e o fluxograma depende do sistema, do tipo de equipamento para o desenvolvimento do programa e do tipo de linguagem a serem utilizados.

Vale salientar que nem por isso está-se negligenciando a importância de um fluxograma. Na prática, percebe-se que o fluxograma é fundamental para a economia de tempo e redução do número de erros de programação.

Um fluxograma não é um elemento indispensável ao desenvolvimento de um programa, porém, sem ele, será necessário um grande esforço para qualquer alteração ou correção. O mais simples, é ter-se uma estruturação visual do programa, economizando-se tempo, evitando-se uma série de tentativas e visualizando-se onde essas alterações irão interferir em outras partes do programa.

O primeiro fluxograma que se faz de um programa deve ser apenas em nível de conceitos, não deve ter muitos detalhes de cada atividade, indicando em linhas gerais quais as atividades que devem ser executadas, assim como a seqüência destas. Após a análise geral deste fluxograma, examina-se se o processamento é correto para a solução do problema a que ele se propõe a resol

ver, deve-se passar a construção de um novo fluxograma mais detalhado, chegando estes detalhes ao nível de instrução do programa. Dependendo da complexidade do programa, o fluxograma a nível de instrução pode ser dividido em diversos outros, fazendo uma partição do fluxograma a nível de conceitos.

II.3 APRESENTAÇÃO

Vamos examinar alguns exemplos simples apenas para termos uma visualização de fluxograma.

Primeiramente, vamos examinar um fluxograma genérico que represente os passos do programa para calcular as raízes da equação $ax^2 + bx + c = 0$.

O algoritmo utilizado, logicamente será:

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

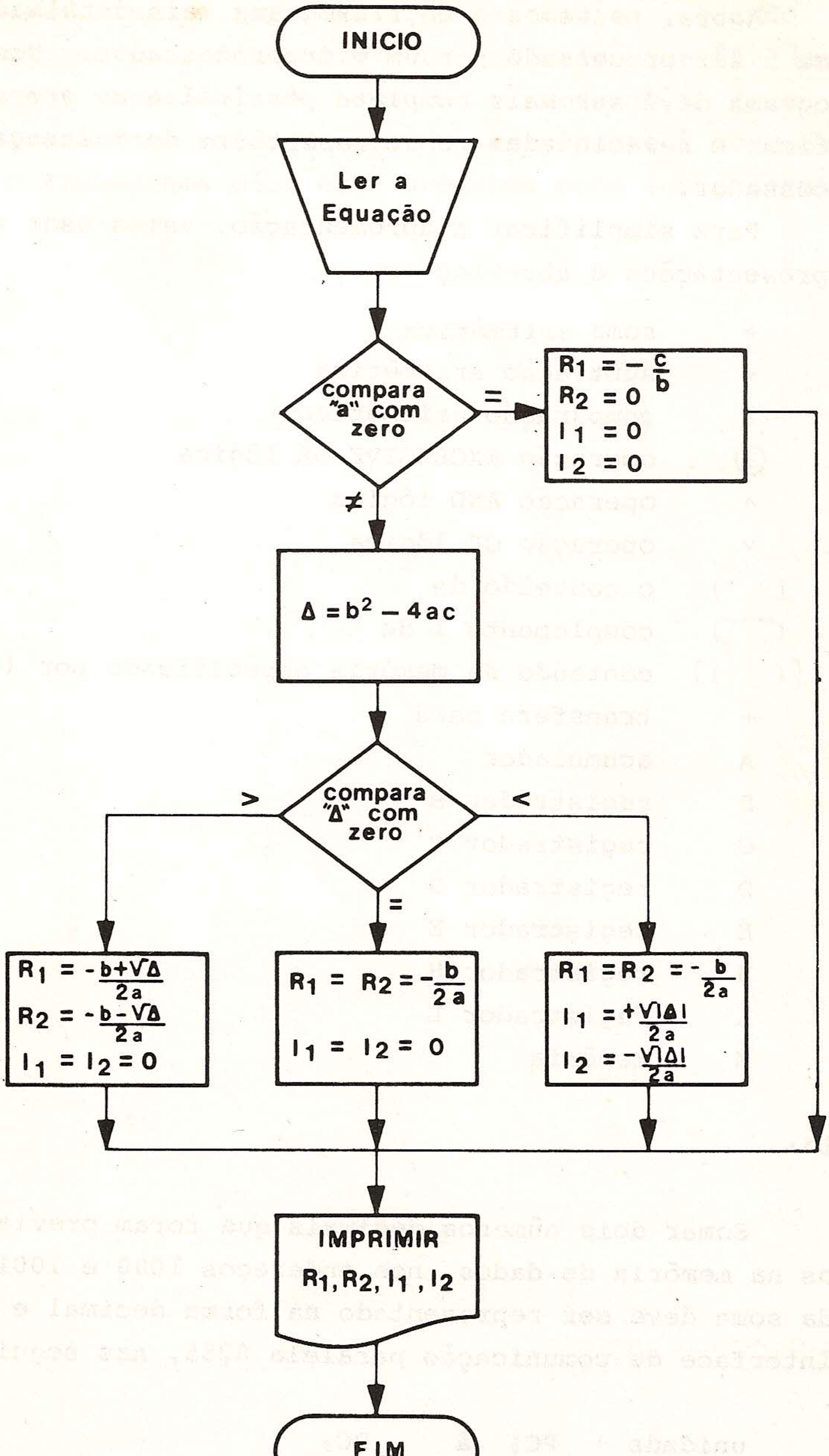
o que pode ser representado por:

$$x_1 = R_1 + j\ I_1$$

$$x_2 = R_2 + j\ I_2$$

o problema, então, é calcular R_1, R_2, I_1 e I_2.

A solução deste problema pode ser representada, como no fluxograma a seguir:


INICIO
Ler a Equação
compara "a" com zero
=
R1 = - c/b
R2 = 0
I1 = 0
I2 = 0
≠
Δ = b2 – 4ac
compara "Δ" com zero
>
<
=
R1 = - (b+√Δ)/2a
R2 = - (b-√Δ)/2a
I1 = I2 = 0
R1 = R2 = - b/2a
I1 = I2 = 0
R1 = R2 = - b/2a
I1 = + √|Δ| /2a
I2 = - √|Δ| /2a
IMPRIMIR
R1, R2, I1, I2
FIM

Agora, passemos a um fluxograma mais detalhado de um problema a ser processado por um microprocessador. Neste caso o fluxograma deve ser mais completo possível e as operações mais específicas e relacionadas com o repertório de instrução do microprocessador.

Para simplificar a apresentação, vamos usar as seguintes representações e abreviações:

+	soma aritmética
-	subtração aritmética
:	comparação aritmética
$\oplus$	operação EXCLUSIVE OR lógica
$\wedge$	operação AND lógica
$\vee$	operação OR lógica
( )	o conteúdo de
$(\overline{\ \ })$	complemento 1 de
[( )]	conteúdo de memória especificado por ( )
$\leftarrow$	transfere para
A	acumulador
B	registrador B
C	registrador C
D	registrador D
E	registrador E
H	registrador H
L	registrador L
M	memória

Exemplo:

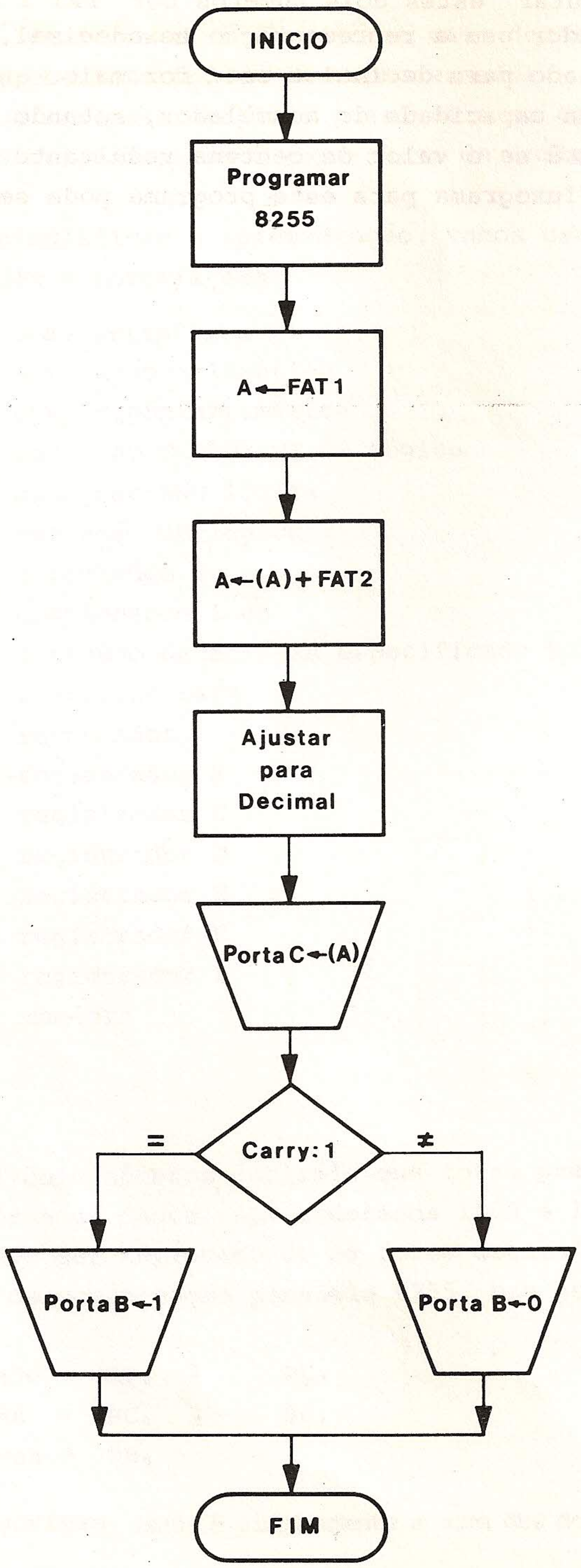
Somar dois números decimais que foram previamente armazenados na memória de dados, nos endereços 1000 e 1001. O resultado da soma deve ser representado na forma decimal e enviado para a interface de comunicação paralela 8255, nas seguintes portas:

unidade → PC_0 à PC_3
dezena → PC_4 à PC_7
centena → PB_0

O algoritmo usado é simplesmente a soma dos dois números.

Vamos representar estes dois números por FAT 1 e FAT 2. Como o microprocessador usa a representação hexadecimal, quando convertido o resultado para decimal e este for maior que "99" haverá um estouro da sua capacidade do acumulador, setando a flag carry, a qual indicará se o valor de centena resultante é zero ou um.

O fluxograma para este programa pode ser o seguinte:


INICIO
Programar
8255
A←FAT 1
A←(A) + FAT2
Ajustar
para
Decimal
Porta C←(A)
Carry: 1
=
≠
Porta B←1
Porta B←0
FIM

Cap. III CARACTERÍSTICAS DAS INSTRUÇÕES DOS MICROPROCESSADORES 8080 E 8085.

III.1 CICLO DE INSTRUÇÕES

O microprocessador é um circuito dinâmico e para o seu funcionamento necessita de um sinal de relógio para a sincronização de suas operações. Este sinal de relógio ou sinal de clock, como é comumente designado, é gerado por um oscilador eletrônico que fornece uma seqüência ininterrupta de pulsos com períodos constantes.

No caso do microprocessador 8080, o sinal de clock, como mostrado na figura III.1, é um sinal resultante da combinação de dois sinais de fases diferentes provenientes do gerador de clock 8224 (capítulo VII - Volume I).

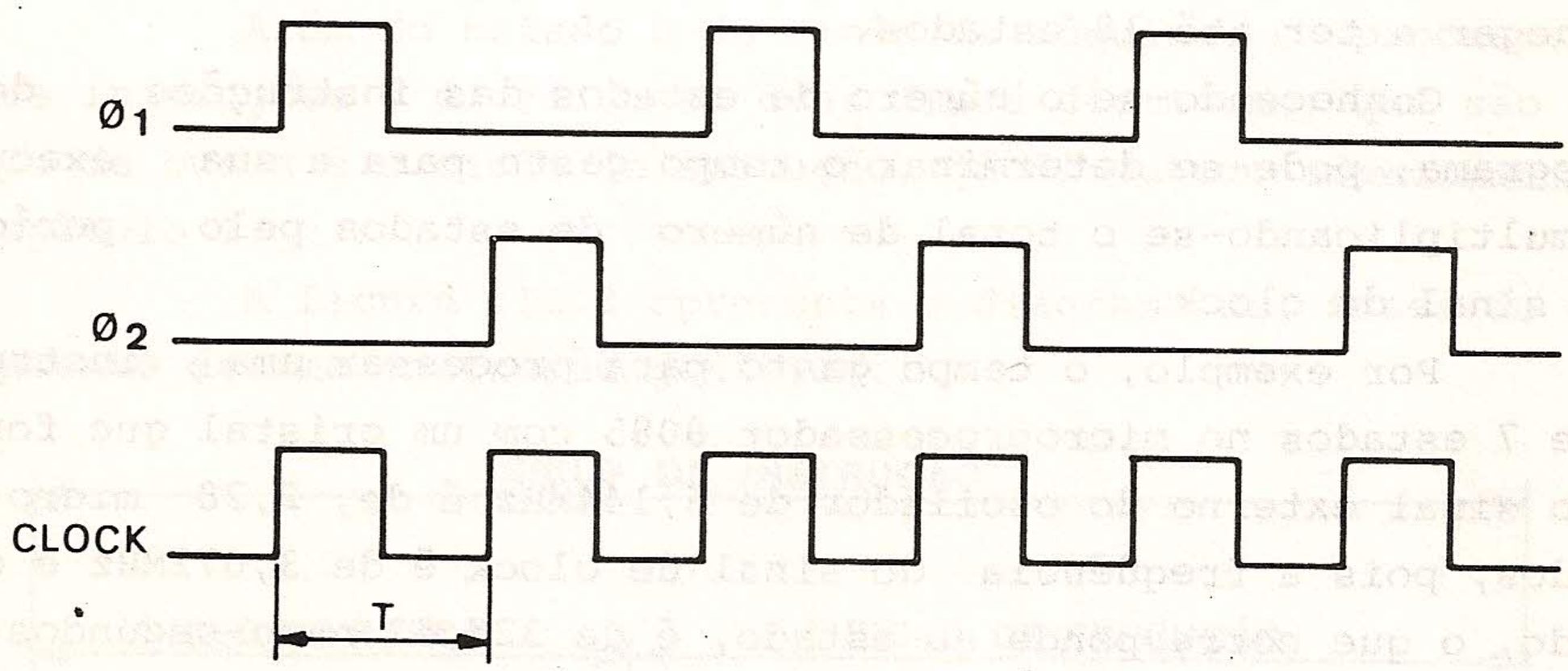


Fig. III.1 - Sinal de clock do microprocessador 8080.

Para o microprocessador 8085, o sinal de clock, como mostrado na figura III.2 é gerado pela divisão por 2 da freqüência do sinal externo do oscilador.

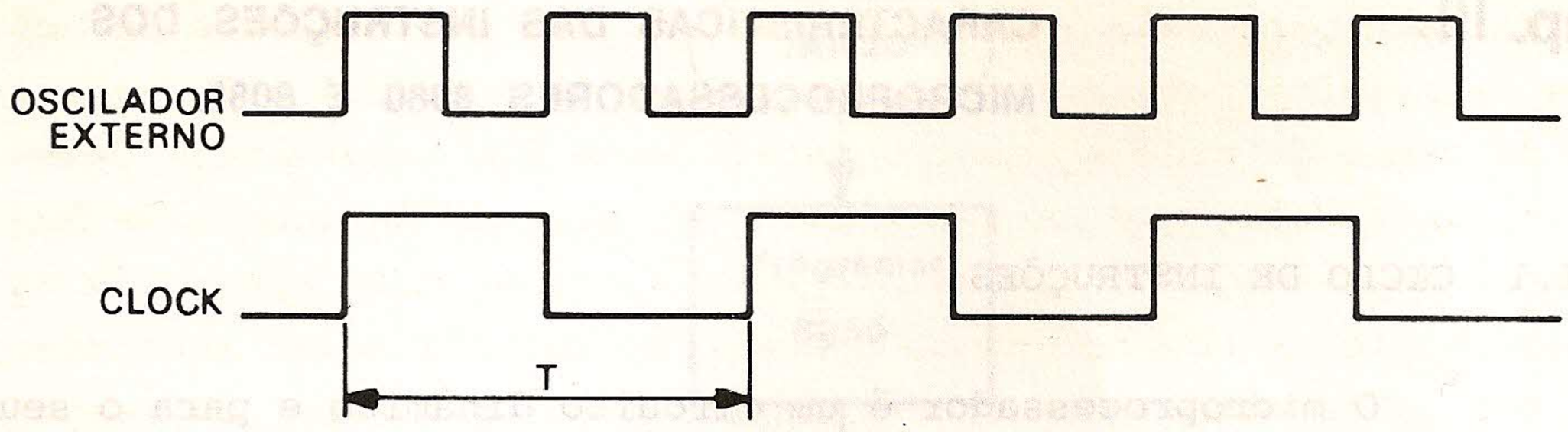


Fig. III.2 - Sinal de clock do microprocessador 8085.

A cada intervalo de tempo T, que corresponde a um período do sinal de clock, da-se o nome de estado. O estado é a unidade básica de tempo do microprocessador.

Cada instrução demora um número inteiro de estado para a sua completa execução. O tempo de processamento é diferente de uma instrução para outra. Algumas instruções podem ter 4 estados, isto é, demoram 4 períodos do sinal de clock, outras podem chegar a ter até 18 estados.

Conhecendo-se o número de estados das instruções de um programa, pode-se determinar o tempo gasto para a sua execução, multiplicando-se o total de número de estados pelo período do sinal de clock.

Por exemplo, o tempo gasto para processar uma instrução de 7 estados no microprocessador 8085 com um cristal que fornece o sinal externo do oscilador de 6,144MHz é de, 2,28 micro-segundos, pois a freqüência do sinal de clock é de 3,072MHz e o período, o que corresponde ao estado, é de 325,52 nano-segundos.

Na determinação do tempo de processamento em sistemas que possuam memórias ou periféricos lentos, deve-se levar em conta que o processador, quando acessa estas memórias ou periféricos pode ficar em estado de espera até que os dados sejam estabilizados.

Para estes casos é importante saber-se quantas vezes, cada instrução acessou uma memória ou um periférico. Toda vez que o processador fizer uma leitura ou escrita em uma memória ou periférico dizemos que foi executado um ciclo de máquina.

Durante a descrição das instruções no capítulo IV, se

rá indicado para cada instrução o seu número de estados e de ciclos de máquina para que seja perfeitamente possível a determinação do tempo de processamento, quando este for necessário. Deve-se notar também, que certas instruções possuem indicação de dois números de estados ou ciclos. Nestes casos o menor número de estados ou ciclos ocorre quando a condição testada pela instrução não for satisfeita. Consequentemente, o maior número, corresponde ao número de estados ou ciclos da instrução, quando a condição testada for satisfeita. Por exemplo, a instrução "JP" tem 7 ou 10 estados e 2 ou 3 ciclos. Esta instrução faz com que o fluxo do programa salte para o endereço especificado na instrução se a flag de sinal indicar que o resultado da operação aritmética, anterior a esta instrução, seja positivo. Neste caso esta instrução possui 10 estados e 3 ciclos.

Se o resultado da operação aritmética não for positivo, o programa prossegue na instrução seguinte, não executando o salto. Caso isto ocorra a instrução terá 7 estados e 2 ciclos.

Além do estado e do ciclo de máquina, o processamento das instruções é dividido em duas fases distintas, que são o ciclo de busca e o ciclo de execução, já discutido anteriormente no capítulo I.

A figura III.3 apresenta o diagrama das fases do processamento de uma instrução genérica.

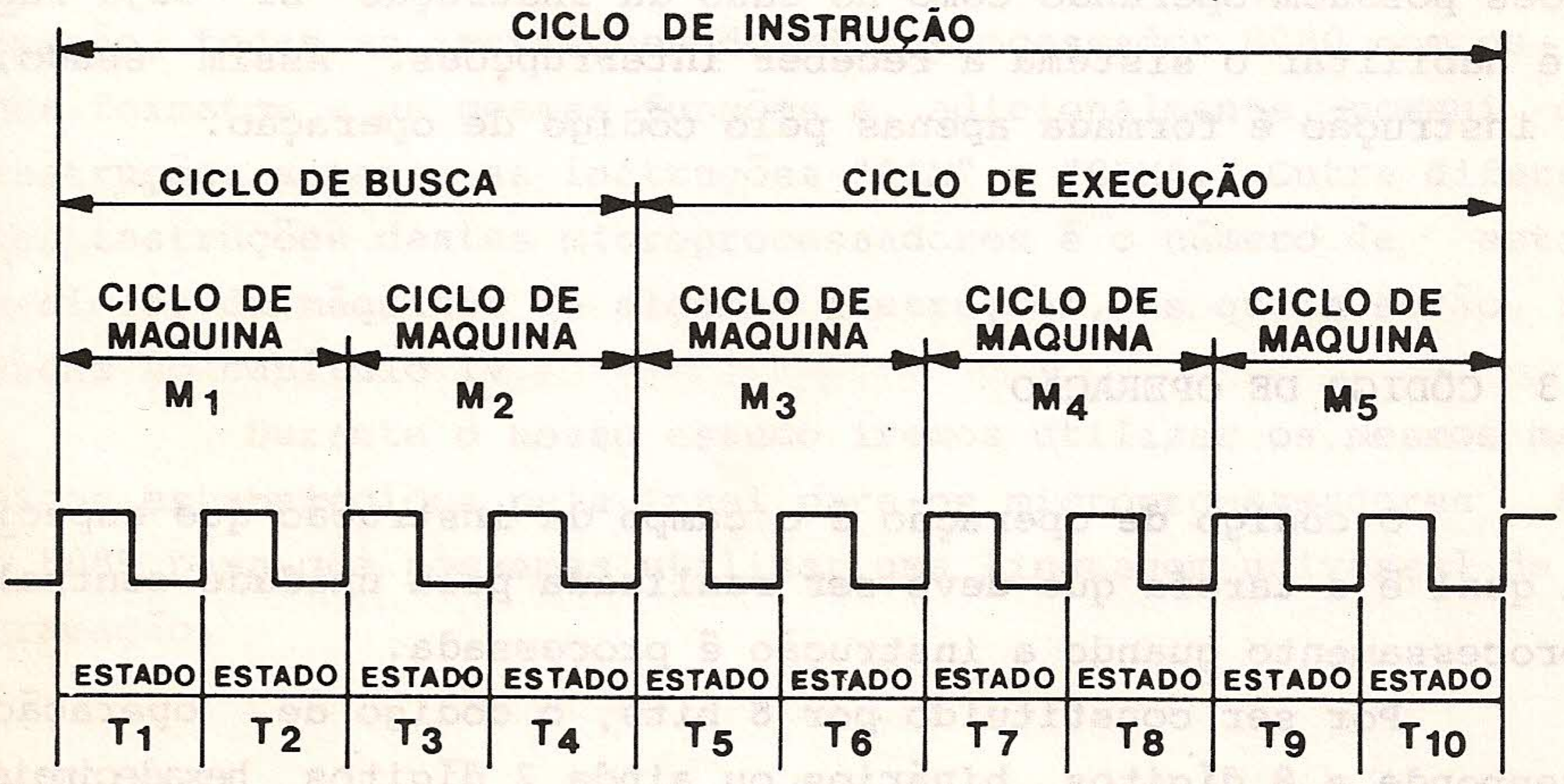


Fig. III.3 - Diagrama de fases de uma instrução genérica.

III.2 FORMATO DAS INSTRUÇÕES

As instruções são formadas por um conjunto de bits que são decodificados pelo microprocessador fazendo com que este execute uma operação bem definida.

A quantidade de bits pode variar de uma instrução para outra. No caso particular dos microprocessadores 8080 e 8085 existem três tamanhos diferentes de instrução:

- instrução de 1 byte ou 8 bits.
- instrução de 2 bytes ou 16 bits.
- instrução de 3 bytes ou 24 bits.

Como as memórias utilizadas em sistemas com o microprocessador 8080 ou 8085 são memórias com 8 bits de informação por endereço, cada conjunto de 8 bits ou um byte é armazenado em um endereço de memória. As instruções de mais de um byte devem ser armazenadas em posições consecutivas de endereço da memória.

As instruções, de uma maneira geral, são formadas por dois campos distintos:

- código de operação.
- operando.

O código de operação é constituído por 8 bits e é sempre o primeiro byte da instrução; os demais formam o operando da instrução e podem conter ou dado ou endereço. Nem todas as instruções possuem operando como no caso da instrução "EI" cuja função é habilitar o sistema a receber interrupções. Assim sendo, esta instrução é formada apenas pelo código de operação.

III.3 CÓDIGO DE OPERAÇÃO

O código de operação é o campo da instrução que especifica qual é a tarefa que deve ser realizada pela unidade central de processamento quando a instrução é processada.

Por ser constituído por 8 bits, o código de operação corresponde a 8 dígitos binários ou ainda 2 dígitos hexadecimais que corresponde exatamente aos níveis lógicos que são interpretados pela unidade de controle. Isto não é problema para um cir

cuito elétrico, porém para o programador, que deve escrever uma instrução após a outra, ler o programa para analisá-lo, corrigi-lo etc, um agrupamento de números torna-se de difícil manipulação, principalmente quando o número de instruções diferentes é muito grande. Como o microprocessador 8080 possui um repertório de 78 instruções básicas e o 8085 possui 80 instruções básicas é quase impossível decorar-se o código de cada uma delas. Este problema é contornado, usando-se um nome ou designação especial para indicar-se o código de operação de cada instrução. O nome do código de operação das instruções é chamado de "mnemônico" da instrução. O mnemônico é uma abreviação da descrição em inglês da função da instrução. Por exemplo:

JMP → salte (jump)
MOV → move (move)
CMA → complemente acumulador (complement accumulator)
HLT → pare (halt)

Normalmente, quando o mnemônico de uma instrução terminar com a letra "x" indica que esta instrução utiliza um par de registradores. Por exemplo, a instrução "INR" incrementa ou um registrador ou uma posição de memória especificado no operando, por outro lado, a instrução "INX" incrementa o par de registradores especificado no operando.

O microprocessador 8085 tem em seu repertório de instrução, todas as instruções do microprocessador 8080 com os mesmos formatos e as mesmas funções e, adicionalmente, possui duas instruções a mais, as instruções "RIM" e "SIM". Outra diferença nas instruções destes microprocessadores é o número de estados e ciclos de máquinas de algumas instruções, as quais serão indicadas no capítulo IV.

Durante o nosso estudo iremos utilizar os mesmos mnemônicos estabelecidos pela Intel para os microprocessadores 8080 e 8085 para que possamos utilizar uma linguagem universal de programação.

III.4 OPERANDO

O outro campo da instrução é o operando, o qual designa ou um dado ou um endereço de um dado que será utilizado pela instrução.

As instruções podem possuir nenhum, um ou dois operandos. Como regra geral, as instruções com dois operandos como as aritméticas ou de transferência, estes são separadas por uma vírgula sendo que o primeiro operando é o destino do dado e o segundo é a fonte do dado. Por exemplo: a instrução MOV A,C transfere o conteúdo do registrador C para o acumulador.

Existem quatro tipos de informações diferentes que podem estar contidas no operando das instruções.

- endereço.
- registrador.
- par de registradores.
- dado imediato.

Endereço:

O endereço pode ser uma informação de 16 bits correspondendo ao endereço de uma posição de memória, ou uma informação de 8 bits correspondendo ao endereço de uma porta de entrada e saída, de um contador ou outro circuito periférico qualquer do sistema.

Registrador:

O registrador usado como operando das instruções pode ser um dos seguintes registradores internos da unidade central de processamento.

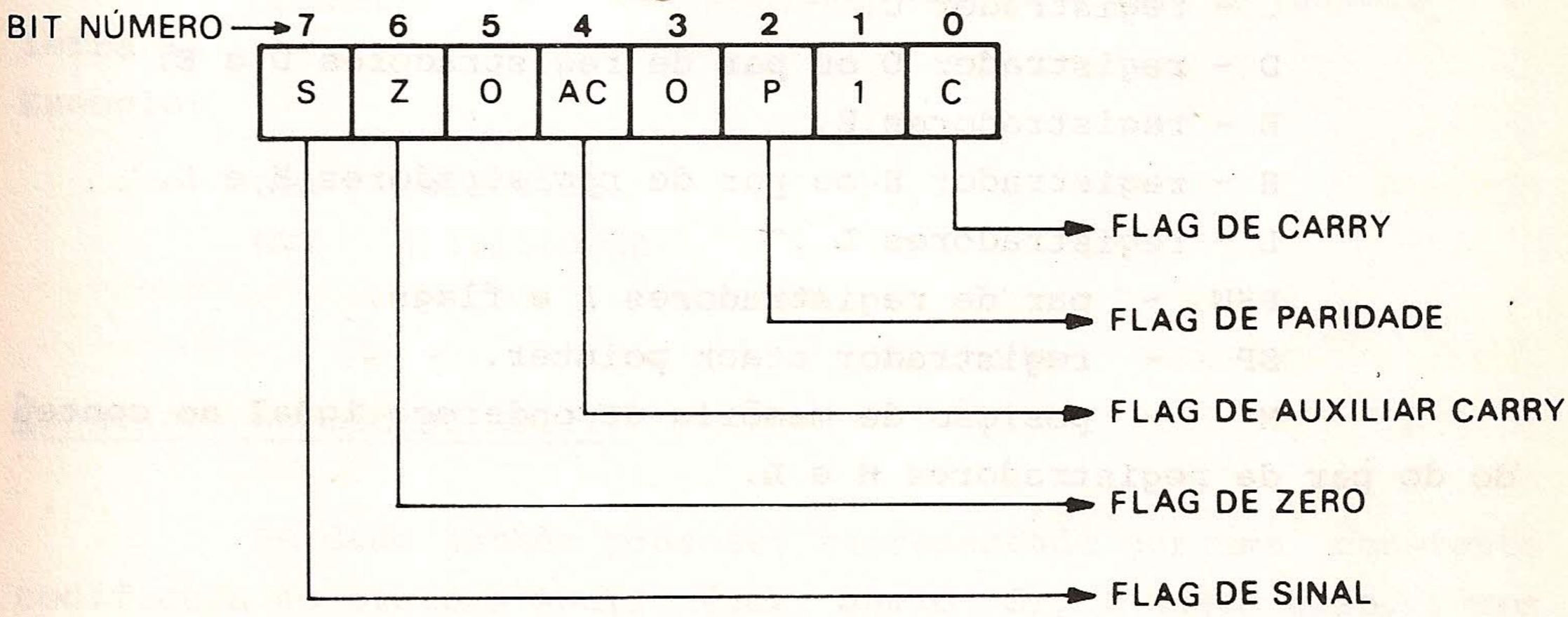
A (acumulador)
B
C
D
E
F
H
L

Par de registradores:

Certas instruções tem como operando um par de registradores. Os pares de registradores são: os registradores de 16 bits stack pointer (indicado por SP) além das seguintes combinações de registradores:

Par de registrador	Formado pelos registradores
B	B e C
D	D e E
H	H e L
PSW	A e Flags

O registrador flags é formado pelos bits de flag e tem o seguinte formato:



Os bits 1, 3 e 5 não são utilizados. Eles têm o valor indicado acima no microprocessador 8080 e valor indefinido no microprocessador 8085.

Dado Imediato:

O dado imediato é uma informação de 8 ou 16 bits, e é o próprio dado que será utilizado pela instrução.

Assim como o código de operação é representado por um mnemônico o operando pode ser especificado por símbolos ao invés de 8 ou 16 bits que o forma.

O operando pode ser especificado das seguintes maneiras:

- letra identificadora do registrador.
- dado no sistema de numeração hexadecimal, decimal, octal ou binário.
- constante no código ASCII.
- label.
- contador de localização.
- expressão.

Letras identificadoras do registrador:

As letras identificadoras dos registradores são:

A - registrador acumulador.
B - registrador B ou par de registradores B e C.
C - registrador C.
D - registrador D ou par de registradores D e E.
E - registradores E.
H - registrador H ou par de registradores H e L.
L - registradores L .
PSW - par de registradores A e flags.
SP - registrador stack pointer.
M - posição de memória de endereço igual ao conteúdo do par de registradores H e L.

Dado:

Quando o operando da instrução é um dado, as seguintes regras devem ser seguidas para identificar o sistema de numeração usado:

Hexadecimal → deve iniciar com um dígito numérico (0 a 9) e deve ser acrescentado ao final do número a letra H.

Exemplo:

```
MVI  A,27H
CALL 1000H
CPI  0A2H
JUMP 0E22AH
```

Decimal → deve ser acrescentado ao final do número a letra D, porém, quando não houver identificação do sistema de numeração, será assumido que o número está representado no sistema decimal.

Exemplo:

```
ADI  25D
JNZ  322D
MVI  B,14
```

Octal → deve ser acrescentado ao final do número a letra Q.

Exemplo:

```
MVI  M,20Q
SUI  37Q
```

Binário → deve ser acrescentado ao final do número a letra B.

Exemplo:

```
ORI  10010011B
MVI  A,11110000B
```

Constante no código ASCII:

Um dado também pode ser representado por uma constante codificada no sistema ASCII (ver anexo C), neste caso, a constante deve estar entre aspas simples:

Exemplo:

```
MVI  B,'@'
CPI  'CR'
```

Label:

O label é um rótulo que se dá a um dado ou a um endereço. O label pode representar:

- uma constante de valor previamente definido por instruções especiais.

- um endereço previamente definido por instruções especiais.

- o endereço da posição de memória do primeiro byte de uma instrução. Neste caso, o código de operação da instrução de endereço especificado, deve ser precedido deste label seguido por dois pontos.

O label pode conter de um a seis caracteres alfanuméricos, porém o primeiro deve ser um caracter alfabético.

Exemplo:

```
        JUMP    INICIO
ROT:    OUT     PORTA3
```

Para não existir confusão de identificação o label deve ser definido uma única vez, isto é, não pode haver duas posições de memória designadas pelo mesmo label ou qualquer outra definição dupla de um mesmo label.

Contador de localização:

O operando de uma instrução pode ser o indicador de um endereço de memória localizado um certo número de endereços atráz ou à frente do primeiro byte desta instrução. Neste caso, o operando deve iniciar com o caracter "$" seguido pelo sinal e o número de bytes a ser incrementado ou decrementado do endereço do primeiro byte da instrução.

Exemplo:

```
JMP  $+5
```

Expressão:

As expressões são combinações aritméticas, lógicas ou especiais entre as maneiras anteriormente vistas de especificar-se o operando. As combinações mais usadas são:

soma	-	indicada por: +
subtração	-	indicada por: -
multiplicação	-	indicada por: *
divisão	-	indicada por: /

AND lógico - indicado por: AND
OR lógico - indicado por: OR
EXCLUSIVE OR Lógico - indicado por: XOR
Complemento 1 - indicado por: NOT

Exemplo:

```
MVI    B,25H*3H
CALL   INICIO + lAH
CPI    7H  0R  12H
```

Além destas combinações descritas, existe uma série de outras que podem ser usadas para a formação de uma expressão, que não serão analisadas por não serem muito necessárias.

O objetivo das expressões é de simplificar e de facilitar o programa.

Para que isto seja possível as expressões devem ser simples e óbvias. Não existe nenhuma vantagem em criar-se expressões complexas ou extensas demais. Deve ter-se em mente que as expressões são utilizadas para especificar um endereço ou um dado a ser utilizado pela instrução e devem ser claras para fácil compreensão do programa.

III.5 TIPOS DE INSTRUÇÕES

Existem diversas divisões diferentes do conjunto de instruções dos microprocessadores 8080 e 8085, classificando as instruções por critérios diferentes. Um critério mais lógico de classificar as instruções destes microprocessadores é por grupo de instruções de função semelhante. Isto porque o repertório de instruções é relativamente grande e decorar-se todas elas não é muito fácil, nem necessário. O importante é ter-se uma idéia das principais operações imediatas que podem ser executadas pelas instruções dos microprocessadores.

Nestas condições os principais tipos de instruções são:

- aritméticas;
- lógicas;

- transferência de dados;
- mudança de seqüência do programa;
- controle;

Instruções aritméticas:

São as instruções que provocam a realização das operações aritméticas: soma, subtração ou ajuste decimal.

Instruções lógicas:

São as instruções que provocam a realização das operações: AND, OR EXCLUSIVO, complemento ou rotação do conteúdo dos registradores.

Instruções de transferência de dados:

São as instruções que provocam a transferência de dados entre registradores, memórias ou periféricos.

Instruções de mudança de seqüência do programa:

São as instruções que alteram a seqüência de execução do programa por meio de salto para um determinado endereço, chamada a uma sub-rotina ou retorno de uma sub-rotina.

Instruções de controle:

São as instruções de controle da unidade central de processamento tais como: habilita interrupção, desabilita interrupção, entre em estado de retenção, etc.

A descrição em detalhes de cada uma das instruções dos microprocessadores 8080 e 8085 é feita no capítulo seguinte.

Cap. IV

CONJUNTO DE INSTRUÇÕES DOS MICROPROCESSADORES 8080 E 8085

Com exceção de duas instruções (RIM e SIM), que são exclusivas apenas do microprocessador 8085, o conjunto de instruções dos microprocessadores 8080 e 8085 são idênticos, sendo que para ambos os microprocessadores cada instrução possui o mesmo formato e a mesma função.

Para facilitar a consulta, as instruções serão descritas a seguir, em ordem alfabética.

1.) ACI — ADD IMMEDIATE WITH CARRY

Descrição:

Soma imediata com carry.

Formato:

Código de operação	Operando
ACI	dado

Função:

Esta instrução somará ao acumulador o dado imediato do segundo byte da instrução mais o carry bit.

Parâmetros:

Mnemônico	Hexa	Flags Afetadas	Ciclos	Estados
ACI	CE	Z,S,P,C, AC	2	7

Exemplo: ACI 14H

$$\begin{aligned} \text{Acumulador} &= 21_{16} = 0010\ 0001 \\ \text{Dado} &= 14_{16} = 0001\ 0100 \quad + \\ \text{Carry bit} &= 1_{16} = 0000\ 0001 \quad + \\ &= 36_{16} = 0011\ 0110 \end{aligned}$$

Z = 0, S = 0, P = 1, C = 0, AC = 0

2.) ADC — ADD WITH CARRY

Descrição:

Soma com carry.

Formato:

Código de Operação	Operando
ADC	R ou M

Função:

Esta instrução possui dois formatos:

a.) ADC R → Somará o conteúdo de um dado registrador mais o "carry bit" ao acumulador.

b.) ADC M → Somará o conteúdo da posição de memória endereçada pelo par de registradores HL mais o carry bit ao acumulador.

Parâmetros:

Mnemônico	Hexa	Flags Afetadas	Ciclos	Estados
ADC (R)	8x	Z,S,P,C,AC	1	4
ADC (M)	8E	Z,S,P,C,AC	2	7

Obs.:

"x" assumirá os valores 8, 9, A, B, C, D ou F correspondendo respectivamente a: (ADC B), (ADC C), (ADC D), (ADC E), (ADC H), (ADC L) ou (ADC A).

Exemplo: ADC B

Acumulador	= 15_{16} =	0001 0101	+
Registrador B	= 8_{16} =	0000 1000	+
Carry bit	= 0_{16} =	0	
	$1D_{16}$	0001 1101	

Z = 0, S = 0, P = 1, C = 0, AC = 0

3.) ADD — ADD

Descrição: Soma

Formato:

Código de operação	Operando
ADD	R ou M

Função:

Esta instrução possui dois formatos:

a.) ADD R → Somará o conteúdo de um dado registrador ao acumulador.

b.) ADD M → Somará o conteúdo da posição de memória endereçada pelo par de registradores "HL" ao conteúdo do acumulador.

Parâmetros:

Mnemônico	Hexa	Flags Afetadas	Ciclos	Estados
ADD (R)	8x	Z,S,P,C,AC	1	4
ADD (M)	86	Z,S,P,C,AC	2	7

Obs.:

"x" assumirá os valores 0, 1, 2, 3, 4, 5 ou 7 correspondendo respectivamente a: (ADD B), (ADD C), (ADD D), (ADD E),(ADD H), (ADD L) ou (ADD A).

Exemplo: ADD C

Acumulador = $3F_{16}$ = 0011 1111 +

Registrador C= $2E_{16}$ = 0010 1110

$6D_{16}$ = 0110 1101

Z = 0, S = 0, P = 1, C = 0, AC = 1

4.) ADI — ADD IMMEDIATE

Descrição:

Soma Imediata.

Formato:

Código de Operação	Operando
ADI	dado

Função:

Esta instrução somará ao acumulador o dado imediato do segundo byte da instrução.

Parâmetros:

Mnemônico	Hexa	Flags Afetadas	Ciclos	Estados
ADI	C6	Z,S,P,C,AC	2	7

Exemplo: ADI 30H

Acumulador = 20_{16} = 0010 0000 +

Dado Imediato = $\underline{30_{16}}$ = $\underline{0011\ 0000}$

50_{16} = 0101 0000

Z = 0, S = 0, P = 1, C = 0, AC = 0

5.) ANA — LOGICAL AND WITH ACCUMULATOR

Descrição:

And lógico com o acumulador.

Formato:

Código de operação	Operando
ANA	R ou M

Função:

Esta instrução possui dois formatos:

a.) ANA (R) → Registrador - Neste caso, a operação lógica "AND" será realizada entre o conteúdo do acumulador e de um dado registrador, e armazenará o resultado no acumulador.

b.) ANA (M) → Memória - Neste caso, a operação lógica "AND" será realizada entre o conteúdo do acumulador e o conteúdo da posição de memória, dado pelo par de registradores HL, e armazenará o resultado no acumulador.

Obs.:

No 8080 as flags C e AC são zeradas, no 8085 a flag C é zerada e a AC é setada.

Parâmetros:

Mnemônico		Hexa	Flags Afetadas	Ciclos	Estados
ANA	(R)	Ax	Z,S,P,C,AC	1	4
ANA	(M)	A6	Z,S,P,C,AC	1	7

Obs.:

"x" assumirá os valores 0, 1, 2, 3, 4, 5 ou 7 correspondendo respectivamente a: (ANA B), (ANA C), (ANA D), (ANA E),(ANA H), (ANA L) ou (ANA A).

Exemplo: ANA D

$$\text{Acumulador} = EF_{16} = 1110\ 1111$$
$$\text{Registrador D} = \underline{2A_{16} = 0010\ 1010}$$
$$2A_{16} = 0010\ 1010$$

Z = 0, S = 0, P = 0, C = 0, AC = 0 ou 1

6.) ANI — AND IMMEDIATE WITH ACCUMULATOR

Descrição:

AND imediato com o acumulador.

Formato:

Código do Operando	Operando
ANI	dado

Função:

Corresponde a uma combinação lógica (AND) entre o acumulador e o dado imediato do segundo byte da instrução.

Obs.:

No 8080 as flags C e AC são zeradas, no 8085 a flag C é zerada e a AC é setada.

Parâmetros:

Mnemônico	Hexa	Flags Afetadas	Ciclos	Estados
ANI	E6	Z,S,P,C,AC	2	7

Exemplo: ANI 31H

$$\text{Acumulador} = AA_{16} = 1010\ 1010$$
$$\text{Dado Imediato} = \underline{31_{16} = 0011\ 0001}$$
$$20_{16} = 0010\ 0000$$

Z = 0, S = 0, P = 0, C = 0, AC = 0 ou 1

7.) CALL — CALL

Descrição:

Chama sub-rotina.

Formato:

Código de operação	Operando
CALL	endereço

Função:

Provocar uma quebra na seqüência normal do programa para a execução de um grupo de instruções separado, chamado de sub-rotina (ver capítulo VI). Após o término da sub-rotina, o fluxo de processamento continuará na instrução seguinte à instrução CALL.

Esta instrução provocará as seguintes ações:

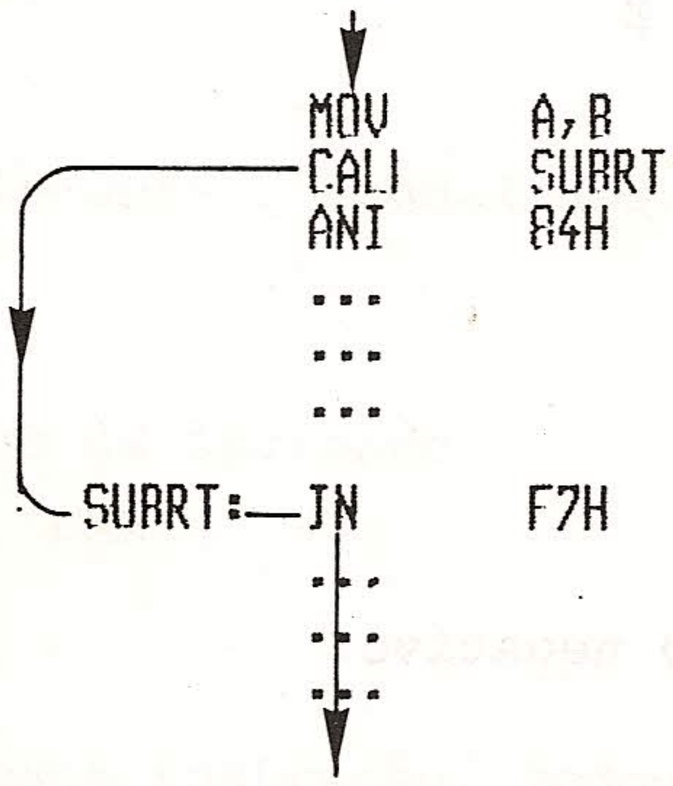
- o endereço da instrução seguinte será armazenado na memória de dados, cujo endereço será indicado pelo registrador stack pointer.
- será subtraído 2 do conteúdo do registrador stack pointer para

que venha a conter o novo endereço disponível do stack.
- o operando da instrução que corresponde ao endereço inicial da sub-rotina, será armazenado no registrador contador de programa para que a execução do programa continue neste novo endereço.

Parâmetros:

Mnemônico	Hexa	Flags Afetadas	Ciclos	Estados
CALL	CD	_	5	17(no 8080) 18(no 8085)

Exemplo: CALL SUBRT



8.) CC — CALL IF CARRY

Descrição:

Chama sub-rotina, se houver carry.

Formato:

Código de Operação	Operando
CC	endereço

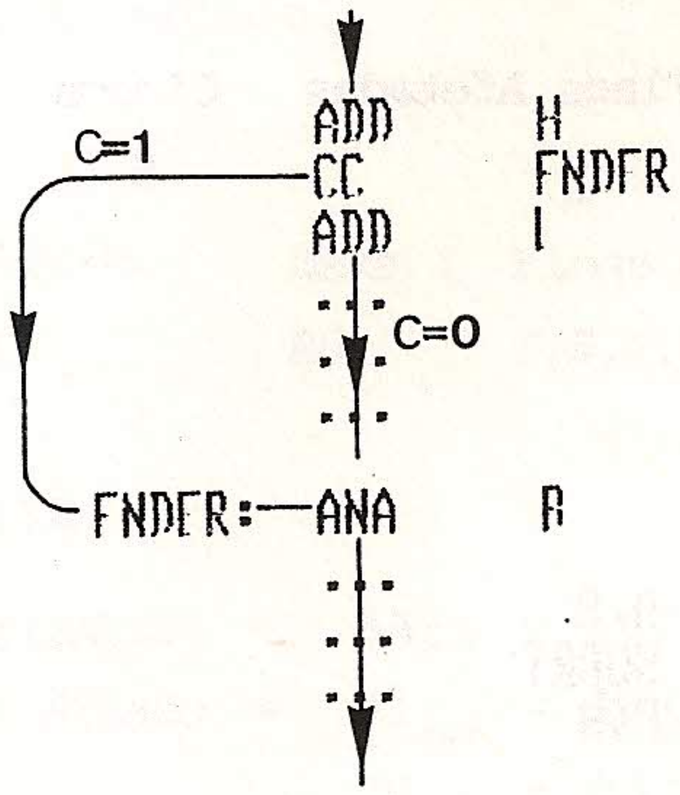
Função:

Esta instrução testará se a flag carry foi setada ou não. Se a mesma for "1" ele funcionará como uma instrução "CALL" e o programa saltará para a sub-rotina de endereço especificado na instrução. Caso contrário, se for "0", a instrução após "CC" será executada e o programa prosseguirá normalmente.

Parâmetros:

Mnemônico	Hexa	Flags Afetadas	Ciclos	Estados
CC	DC	-	3 ou 5(no 8080)	11 ou 17(no 8080)
			2 ou 5(no 8085)	9 ou 18(no 8085)

Exemplo: CC ENDER



9.) CM — CALL IF MINUS

Descrição:

Chama sub-rotina, se negativo.

Formato:

Código de Operação	Operando
CM	endereço

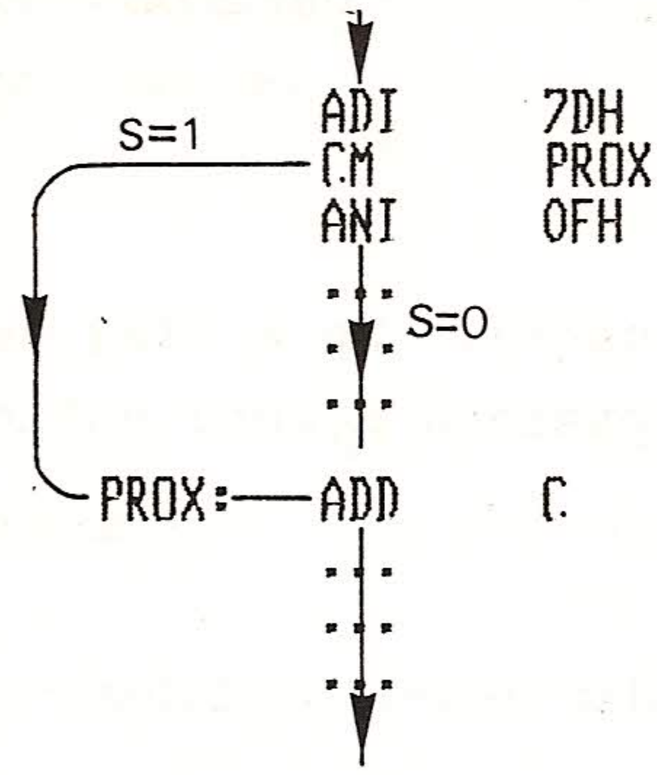
Função:

Esta instrução testará se a flag sinal foi setada ou não. Se a mesma for "1" (o que significa que o resultado da operação anterior foi negativo), ela funcionará como uma instrução "CALL" e o programa saltará para a sub-rotina de endereço especificado na instrução. Caso contrário, se for "0", a instrução após "CM" será executada e o programa prosseguirá normalmente.

Parâmetros:

Mnemônico	Hexa	Flags Afetadas	Ciclos	Estados
CM	FC	-	3 ou 5(no 8080)	11 ou 17(no 8080)
			2 ou 5(no 8085)	9 ou 18(no 8085)

Exemplo: CM PROX



10.) CMA — COMPLEMENT ACCUMULATOR

Descrição:

Complementa o acumulador.

Formato:

Código de Operação	Operando
CMA	-

Função:

Com esta instrução, todos os bits do acumulador serão complementados.

Parâmetros:

Mnemônico	Hexa	Flags Afetadas	Ciclos	Estados
CMA	2F	-	1	4

Exemplo: CMA

Acumulador = $3F_{16}$ = 0011 1111
Ac.Comple. = $C0_{16}$ = 1100 0000

11.) CMC — COMPLEMENT CARRY

Descrição:

Complementa a flag carry.

Formato:

Código de operação	Operando
CMC	-

Função:

Complementar a flag carry. Se a flag de carry estiver setado, após a instrução CMC, passará a zero e vice-versa.

Parâmetros:

Mnemônico	Hexa	Flags Afetadas	Ciclos	Estados
CMC	3F	C	1	4

12.) CMP — COMPARE WITH ACCUMULATOR

Descrição:

Compara com o acumulador.

Formato:

Código de Operação	Operando
CMP	R ou M

Função:

O conteúdo do acumulador será comparado com o conteúdo da memória ou com do registrador especificado sem que estes sejam alterados. Apenas as flags serão modificadas. Esta instrução possui dois formatos:

a.) CMP R → O conteúdo do acumulador será comparado com o conteúdo de um dado registrador.

b.) CMP M → O conteúdo do acumulador será comparado com o conteúdo da posição de memória endereçada pelo par de registradores HL.

Parâmetros:

Mnemônico		Hexa	Flags Afetadas	Ciclos	Estados
CMP	(R)	Bx	Z,S,P,C,AC	1	4
CMP	(M)	BE	Z,S,P,C,AC	2	7

Obs.:

"x" assumirá os valores 8, 9, A, B, D ou F, correspondendo respectivamente a: (CMP B), (CMP C), (CMP E), (CMP H),(CMP L) ou (CMP A).

Exemplo: CMP B

Acumulador = 14_{16} = 0001 0100
Registrador B = 8_{16} = 0000 1000

a.) Será executado o complemento 2 de B.

```
0000 1000  =  conteúdo de B
1111 0111  =  complemento do conteúdo de B
        1 +
-------------
1111 1000  =  complemento 2 do conteúdo de B
```

b.) O conteúdo do acumulador será somado ao complemento 2 do conteúdo de B.

```
0001 0100
1111 1000 +
-----------
0000 1100
```

Z = 0, S = 0, P = 1, C = 1, AC = 0

Após as operações anteriores que corresponderão a subtrair o conteúdo do registrador B do conteúdo do acumulador, teremos:

C.1 - Se a flag carry for igual a "0", significará que o conteúdo do acumulador será maior que o do registrador ou, igual ao mesmo, condição indicada também pela flag zero.

C.2 - Se houver carry, significará que o conteúdo do acumulador será menor que o do registrador em questão.

C.3 - A flag zero indicará a condição de igualdade.

Deste modo a comparação será efetuada sem destruir os conteúdos de ambos os registradores e, através das flags de zero e carry, saberemos qual o resultado da comparação.

13.) CNC — CALL IF NO CARRY

Descrição:

Chama sub-rotina, se não houver carry.

Formato:

Código de operação	Operando
CNC	endereço

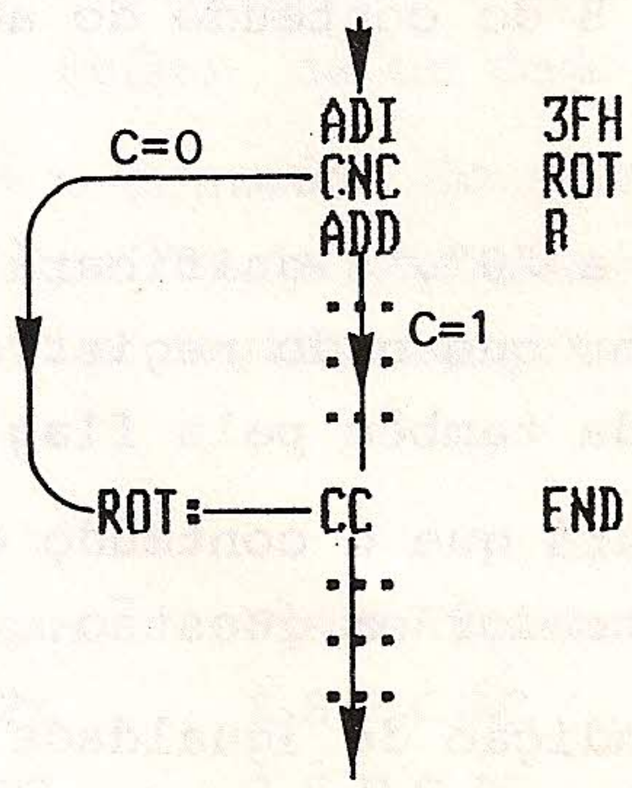
Função:

Esta instrução testará se a flag carry foi setada ou não. Se a mesma for "0", ela funcionará como uma instrução "CALL" e o programa saltará para a sub-rotina de endereço especificado na instrução. Caso contrário, se for "1", a instrução após "CNC" será executada e o programa prosseguirá normalmente.

Parâmetros:

Mnemônico	Hexa	Flags Afetadas	Ciclos	Estados
CNC	D4	-	3 ou 5(no 8080)	11 ou 17(no 8080)
			2 ou 5(no 8085)	9 ou 18(no 8085)

Exemplo: CNC ROT



14.) CNZ — CALL IF NOT ZERO

Descrição:

Chama sub-rotina, se não zero.

Formato:

Código de Operação	Operando
CNZ	endereço

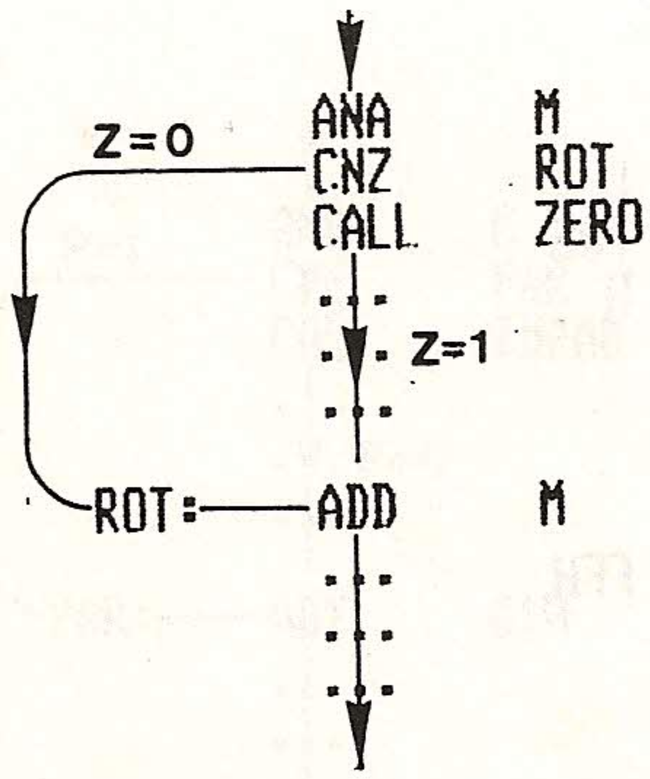
Função:

Esta instrução testará se a flag zero foi setada ou não. Se a mesma for "0" (o que significa que o resultado da operação anterior foi diferente de zero), ela funcionará como uma instrução "CALL" e o programa saltará para a sub-rotina de endereço especificado na instrução. Caso contrário, se for "1", a instrução após "CNZ" será executada e o programa prosseguirá normalmente.

Parâmetros:

Mnemônico	Hexa	Flags Afetadas	Ciclos	Estados
CNZ	C4	-	3 ou 5 (no 8080)	11 ou 17 (no 8080)
			2 ou 5 (no 8085)	9 ou 18 (no 8085)

Exemplo: CNZ ROT



15.) CP — CALL IF POSITIVE

Descrição:

Chama sub-rotina, se positivo.

Formato:

Código de operação	Operando
CP	endereço

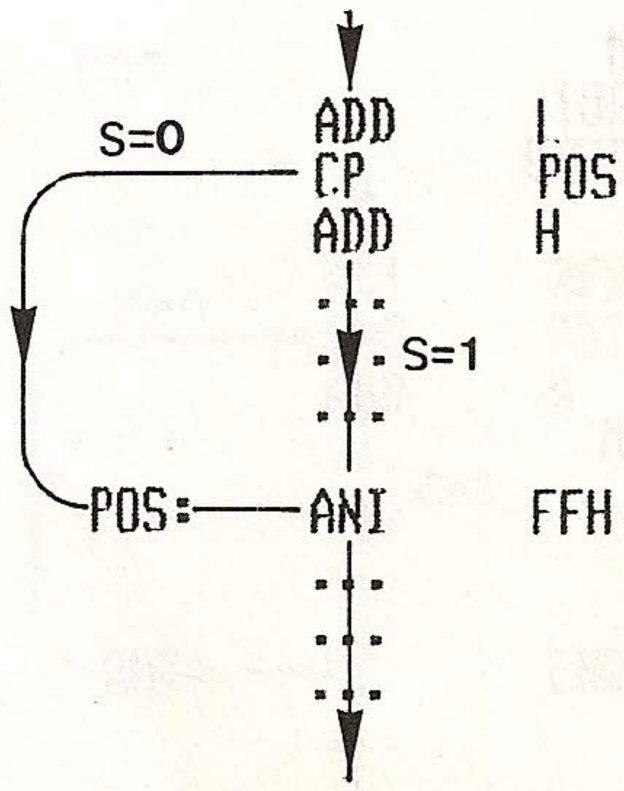
Função:

Esta instrução testará se a flag sinal foi setada ou não. Se a mesma for "0" (o que significa que o resultado da operação anterior foi positivo), ela funciona como uma instrução "CALL" e o programa salta para a sub-rotina de endereço especificado na instrução. Caso contrário, se for "1", a próxima instrução após "CP" será executada e o programa prosseguirá normalmente.

Parâmetros:

Mnemônico	Hexa	Flags Afetadas	Ciclos	Estados
CP	F4	-	3 ou 5(no 8080)	11 ou 17(no 8080)
			2 ou 5(no 8085)	9 ou 18(no 8085)

Exemplo: CP POS



16.) CPE — CALL IF PARITY EVEN

Descrição:

Chama sub-rotina, se paridade for par.

Formato:

Código de Operação	Operando
CPE	endereço

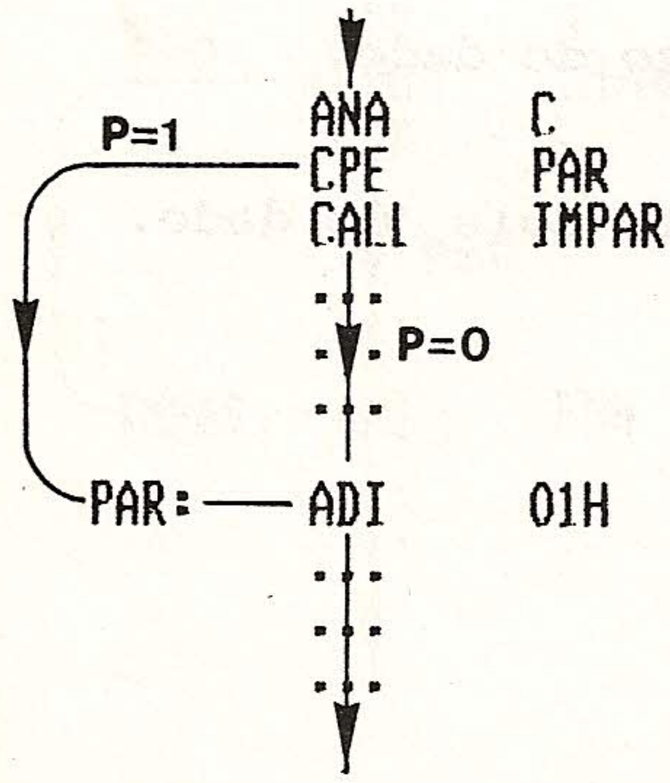
Função:

Esta instrução testa se a flag paridade foi setada ou não. Se a mesma for "1" (o que significa que a paridade do resultado da operação anterior foi par), ela funcionará como uma instrução "CALL" e o programa saltará para a sub-rotina de endereço especificado na instrução. Caso contrário, se for "0", a instrução após "CPE" será executada e o programa prosseguirá normalmente.

Parâmetros:

Mnemônicos	Hexa	Flags Afetadas	Ciclos	Estados
CPE	EC	-	3 ou 5 (no 8080)	11 ou 17 (no 8080)
			2 ou 5 (no 8085)	9 ou 18 (no 8085)

Exemplo: CPE PAR



17.) CPI — COMPARE IMMEDIATE

Descrição:

Compara um dado imediato.

Formato:

Código de Operação	Operando
CPI	dado

Função:

Comparar o conteúdo do acumulador com o dado especificado no operando da instrução. O processo, através do qual a comparação será efetuada, é o mesmo já descrito na instrução "CMP" com a diferença de que neste caso, o dado a ser comparado será o segundo byte da instrução.

Parâmetros:

Mnemônico	Hexa	Flags Afetadas	Ciclos	Estados
CPI	FE	Z,S,P,C,AC	2	7

Exemplo: CPI 7CH

Acumulador = 53_{16} = 0101 0011

Dado = $7C_{16}$ = 0111 1100

a.) 0111 1100 = dado.

1000 0011 = complemento do dado.

1 +

1000 0100 = complemento dois do dado.

b.) 0101 0011 +

1000 0100

1101 0111

Z = 0, S = 1, P = 0, C = 0, AC = 0

18.) CPO — CALL IF PARITY ODD

Descrição:

Chama sub-rotina, se paridade for impar.

Formato:

Código de Operação	Operando
CPO	endereço

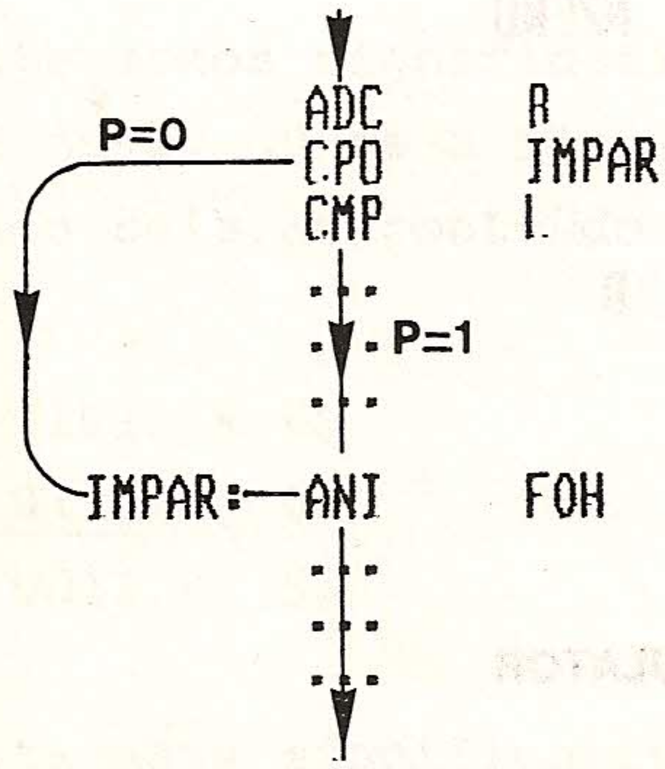
Função:

Esta instrução testará se a flag paridade foi setada ou não. Se a mesma for "0" (o que significa que a paridade do resultado da operação anterior foi impar), ela funcionará como uma instrução "CALL" e o programa saltará para a sub-rotina de endereço especificado na instrução. Caso contrário, se for "1", a instrução após "CPO" será executada e o programa prosseguirá normalmente.

Parâmetros:

Mnemônico	Hexa	Flags Afetadas	Ciclos	Estados
CPO	E4	-	3 ou 5 (no 8080)	11 ou 17 (no 8080)
			2 ou 5 (no 8085)	9 ou 18 (no 8085)

Exemplo: CPO IMPAR



19.) CZ — CALL IF ZERO

Descrição:

Chama sub-rotina, se zero.

Formato:

Código de Operação	Operando
CZ	endereço

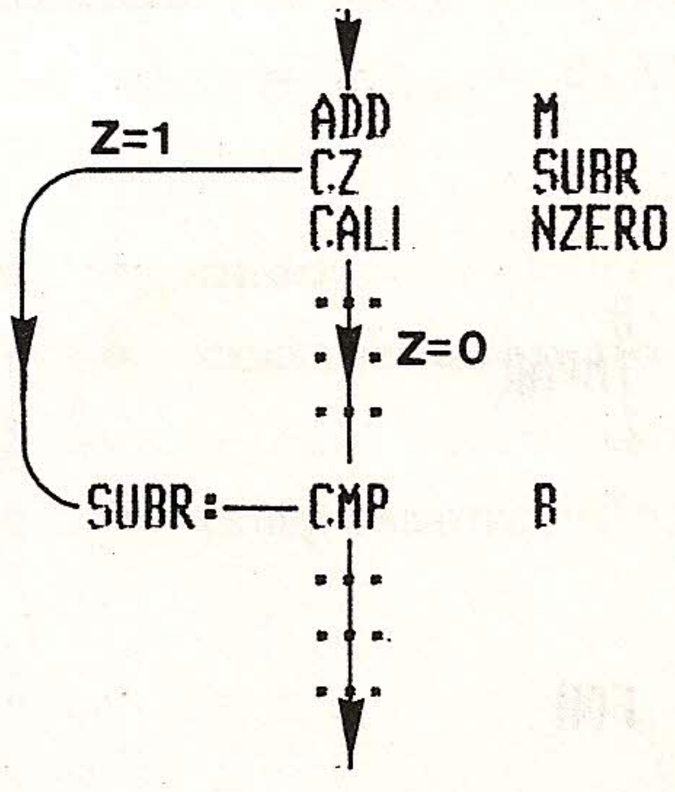
Função:

Esta instrução testará se a flag zero foi setada ou não. Se a mesma for "1" (o que significará que o resultado da operação anterior foi zero), ela funcionará como uma instrução "CALL" e o programa saltará para a sub-rotina de endereço especificado na instrução. Caso contrário, se for "0", a instrução após "CZ" será executada e o programa prosseguirá normalmente.

Parâmetros:

Mnemônico	Hexa	Flags Afetadas	Ciclos	Estados
CZ	CC	-	3 ou 5(no 8080)	11 ou 17(no 8080)
			2 ou 5(no 8085)	9 ou 18(no 8085)

Exemplo: CZ SUBR



20.) DAA — DECIMAL ADJUST ACCUMULATOR

Descrição:

Ajuste decimal do acumulador.

Formato:

Código de operação	Operando
DAA	-

Função:

A instrução de ajuste decimal converterá o conteúdo do acumulador em dois dígitos decimais, codificados em dois grupos de quatro bits binários. É utilizado normalmente após a adição de dois números decimais.

Parâmetros:

Mnemônico	Hexa	Flags Afetadas	Ciclos	Estados
DAA	27	Z,S,P,C,AC	1	4

Exemplo: DAA

Vamos supor que antes da instrução de ajuste decimal tenha sido realizada a seguinte soma:

```
07  =  0000  0111  +
46  =  0100  0110
4D  =  0100  1101
```

Como se trata da soma de dois números decimais, o resultado correto será 53 e não 4D. Logo, utilizando a instrução DAA poderemos ajustar o resultado obtido na operação de adição, da seguinte forma:

a.) Se os 4 bits menos significativos do acumulador tiverem um valor maior que nove, ou se a flag auxiliar carry estiver setada, será somado seis ao conteúdo do acumulador (4 bits menos significativos).

```
0100 1101  = 4D
0000 0110  = 06  +
0101 0011    53
```

b.) Se os 4 bits mais significativos do acumulador tiverem um valor maior que nove, ou se a flag carry estiver setada, o DAA somará seis a estes 4 bits.

Exemplo:

```
15 = 0001  0101 +
87 = 1000  0111
9C   1001  1100
```

Como os quatro bits menos significativos apresentam um valor maior que nove, o DAA somará seis ao acumulador.

```
1001  1100  = 9C
0000  0110  = 06 +
1010  0010    A2
```

Como os quatro bits mais significativos apresentam um valor maior que nove, o DAA somará seis a este valor.

```
1010  0010  =  A2
0110  0000  =  60 +
0000  0010  =  02
```

No final desta segunda parte, a flag de carry será "1", indicando que o resultado da operação será 102.

21.) DAD — DOUBLE REGISTER ADD

Descrição:

Soma o registrador duplo.

Formato:

Código de Operação	Operando
DAD	R

Função:

A instrução "DAD" somará o conteúdo de um dado par de registradores (16 bits) ao conteúdo do par de registradores HL, mantendo o resultado em HL.

Parâmetros:

Mnemônico	Hexa	Flags Afetadas	Ciclos	Estados
DAD	x9	C	3	10

Obs.:

"x" assumirá os seguintes valores 0, 1, 2 ou 3 o que corresponderá respectivamente à: (DAD B), (DAD D), (DAD H), ou (DAD SP).

Exemplo: DAD B

Se assumirmos que o par BC contém o valor 3121_{16} e que o par HL contém o valor $105A_{16}$ após DAD B, teremos como resultado em HL o valor $417B_{16}$.

$$\begin{array}{lll} 0011\ 0001\ 0010\ 0001 & = & 3121_{16} \\ \underline{0001\ 0000\ 0101\ 1010} & = & \underline{105A_{16}} \\ 0100\ 0001\ 0111\ 1011 & = & 417B_{16} \end{array}$$

C = 0 ←┘

22.) DCR — DECREMENT

Descrição:

Decremento.

Formato:

Código de operação	Operando
DCR	M ou R

Função:

Esta instrução possui dois formatos:

a.) DCR (R) → Subtrairá uma unidade do conteúdo de um dado registrador.

b.) DCR (M) → Subtrairá uma unidade do conteúdo da posição de memória endereçada pelo par HL. Neste caso, o operando será a memória.

Parâmetros:

Mnemônico	Hexa	Flags Afetadas	Ciclos	Estados
DCR (R)	xx	Z,S,P,AC	1	5 (no 8080)
				4 (no 8085)
DCR (M)	35	Z,S,P,AC	3	10

Obs.:

"xx" corresponderá respectivamente aos valores 05(DCR B), 0D(DCR C), 15(DCR D), 1D(DCR E), 25(DCR H), 2D(DCR L), e 3D (DCR A).

Exemplo: DCR B

Consideremos que o conteúdo do registrador B seja inicialmente 5_{16}. Após a instrução DCR B, teremos como resultado $B = 4_{16}$.

23.) DCX — DECREMENT REGISTER PAIR

Descrição:

Decrementa par de registradores.

Formato:

Código de operação	Operando
DCX	R

Função:

Decrementa de uma unidade o conteúdo de um especifica do par de registradores.

Parâmetros:

Mnemônico	Hexa	Flags Afetadas	Ciclos	Estados
DCX	xB	-	1	5 (no 8080)
				6 (no 8085)

Obs.:

"x" poderá assumir os valores 0, 1, 2 e 3, o que corresponderá aos pares BC, DE, HL e SP.

Exemplo: DCX B

Consideremos que o conteúdo dos pares de registradores seja inicialmente $54FF_{16}$. Após a instrução DCX B, teremos como

resultado BC = $54FE_{16}$.

24.) DI — DISABLE INTERRUPTS

Descrição:

Desabilita Interrupções.

Formato:

Código de operação	Operando
DI	-

Função:

Esta instrução desabilitará a unidade central de processamento de receber interrupção, isto é, após ser executada esta instrução, um sinal de interrupção externo não será reconhecido pela unidade central de processamento.

A única exceção será a entrada de interrupção TRAP no microprocessador 8085 que não poderá ser desabilitada, pois é uma interrupção prevista para ser acionada por problemas sérios que necessitem ação urgente da unidade central de processamento.

No microprocessador 8080, esta instrução forçará a saída INTE deste microprocessador a um nível "0", indicando, enquanto este nível permanecer, que interrupções não serão habilitadas.

Parâmetros:

Mnemônico	Hexa	Flags Afetadas	Ciclos	Estados
DI	F3	-	1	4

25.) EI — ENABLE INTERRUPTS

Descrição:

Habilita Interrupções.

Formato:

Código de operação	Operando
EI	-

Função:

Esta instrução habilitará a unidade central de processamento de receber uma interrupção.

As interrupções serão desabilitadas ou por uma instrução "DI", ou por um sinal de reset, ou toda vez que uma interrupção seja processada.

Caso um destes fatos tenha ocorrido, a unidade central de processamento só aceitará uma interrupção após a instrução "EI".

Parâmetros:

Mnemônico	Hexa	Flags Afetadas	Ciclos	Estados
EI	FB	-	1	4

26.) HLT — HALT

Descrição:

Halt (retenção).

Formato:

Código de operação	Operando
HLT	-

Função:

Quando a instrução "HLT" for executada, a execução do programa será interrompida. Apenas o registrador contador de programa será incrementado, indicando o endereço da próxima instrução. Os demais registradores não serão afetados. A unidade central de processamento ficará em estado de retenção e nenhum processamento ocorrerá até que esta receba um sinal externo de interrupção (caso esta tenha sido habilitada anteriormente à instrução "HLT") ou de um sinal externo de RESET.

Parâmetros:

Mnemônico	Hexa	Flags Afetadas	Ciclos	Estados
HLT	76	-	1	7(no 8080) 5(no 8085)

27.) IN — INPUT FROM PORT

Descrição:

Entrada através de Porta.

Formato:

Código de operação	Operando
IN	endereço

Função:

Ler o conteúdo da porta de entrada endereçado pelo operando e carregar o acumulador com este dado.

Parâmetros:

Mnemônico	Hexa	Flags Afetadas	Ciclos	Estados
IN	DB	-	3	10

28.) INR — INCREMENT

Descrição:

Incrementa.

Formato:

Código de operação	Operando
INR	M ou R

Função:

Esta instrução possui dois formatos:

a.) INR (R) → Incrementa um dado registrador de uma unidade.

b.) INR (M) → Incrementa de uma unidade a posição de memória endereçada pelo par HL. Neste caso, o operando será a memória.

Parâmetros:

Mnemônico	Hexa	Flags Afetadas	Ciclos	Estados
INR (R)	xx	Z,S,P,AC	1	5(no 8080) 4(no 8085)
INR (M)	34	Z,S,P,AC	3	10

Obs.:

"xx" corresponderá respectivamente aos valores 04(INR B), 0C(INR C), 14(INR D), 1C(INR E), 24(INR H), 2C(INR L) e 3C (INR A).

Exemplo: INR B

Consideremos que o conteúdo do registrador B seja inicialmente 81_{16}. Após a instrução INR B, teremos como resultado B = 82_{16}.

29.) INX — INCREMENT REGISTER PAIR

Descrição:

Incrementa par de registradores.

Formato:

Código de operação	Operando
INX	R

Função:

Incrementar de uma unidade o conteúdo de um especifica do par de registradores.

Parâmetros:

Mnemônico	Hexa	Flags Afetadas	Ciclos	Estados
INX	x3	-	1	5(no 8080) 6(no 8085)

Obs.:

"x" poderá assumir os valores 0, 1, 2, 3, correspondendo sucessivamente aos pares BC; DE; HL e SP.

Exemplo: INX B

Consideremos que o conteúdo do par de registradores seja inicialmente $01EE_{16}$. Após a instrução INX B, teremos como resultado BC = $01EF_{16}$.

30.) JC — JUMP IF CARRY

Descrição:

Salta se houver carry.

Formato:

Código de operação	Operando
JC	endereço

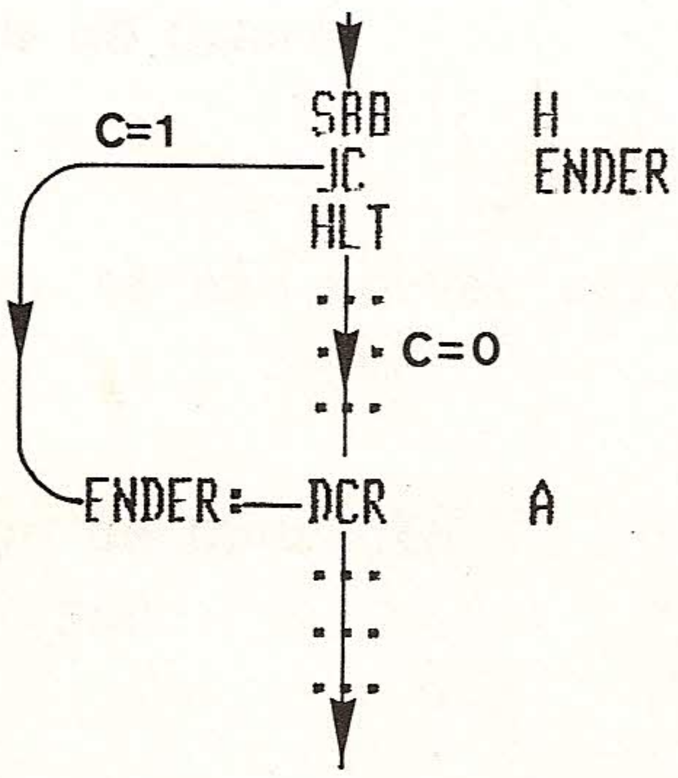
Função:

Esta instrução testará se a flag carry foi setada em "1" ou não. Se a mesma for "1" o programa saltará para o endereço especificado na instrução. Caso contrário, se for "0", a instrução após "JC" será executada e o programa prosseguirá normalmente.

Parâmetros:

Mnemônico	Hexa	Flags Afetadas	Ciclos	Estados
JC	DA	-	3 (no 8080)	10 (no 8080)
			2 ou 3 (no 8085)	7 ou 10 (no 8085)

Exemplo: JC ENDER



31.) JM — JMP IF MINUS

Descrição:

Salta se negativo.

Formato:

Código de Operação	Operando
JM	endereço

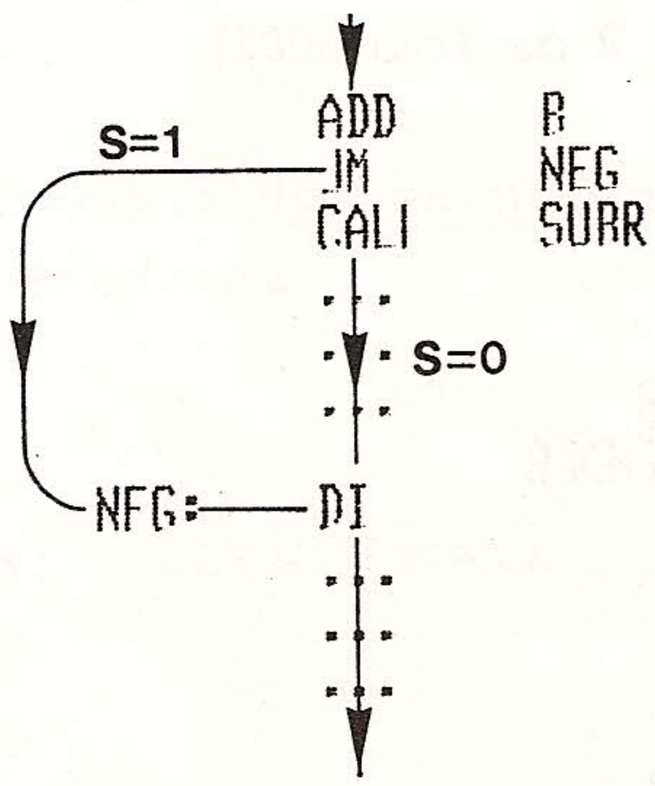
Função:

Esta instrução testará se a flag sinal foi setada em "1" ou não. Se a mesma for "1" (o que significa que o resultado da operação anterior foi negativo) o programa saltará para o endereço especificado na instrução. Caso contrário, se for "0", a instrução após "JM" será executada e o programa prosseguirá normalmente.

Parâmetros:

Mnemônico	Hexa	Flags Afetadas	Ciclos	Estados
JM	FA	-	3(no 8080)	10(no 8080)
			2 ou 3(no 8085)	7 ou 10(no 8085)

Exemplo: JM NEG



32.) JMP — JUMP

Descrição:

Salta.

Formato:

Código de Operação	Operando
JMP	endereço

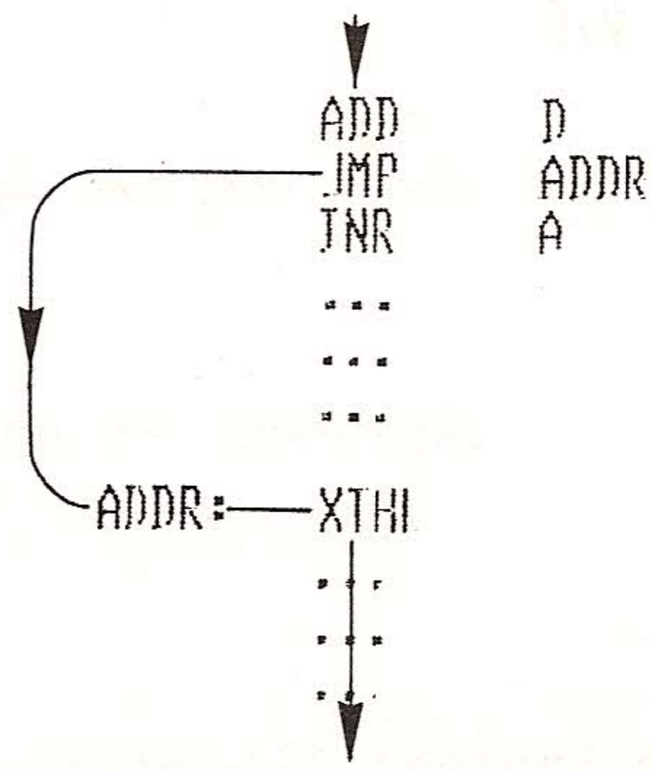
Função:

Esta instrução modificará a seqüência do programa fazendo com que após a execução da instrução "JMP", ao invés de executar a instrução seguinte do programa, fará com que seja executada a instrução endereçada pela instrução "JMP".

Parâmetros:

Mnemônico	Hexa	Flags Afetadas	Ciclos	Estados
JMP	C3	-	3	10

Exemplo: JMP ADDR



33.) JNC — JUMP IF NO CARRY

Descrição:

Salta, se não houver carry.

Formato:

Código de Operação	Operando
JNC	endereço

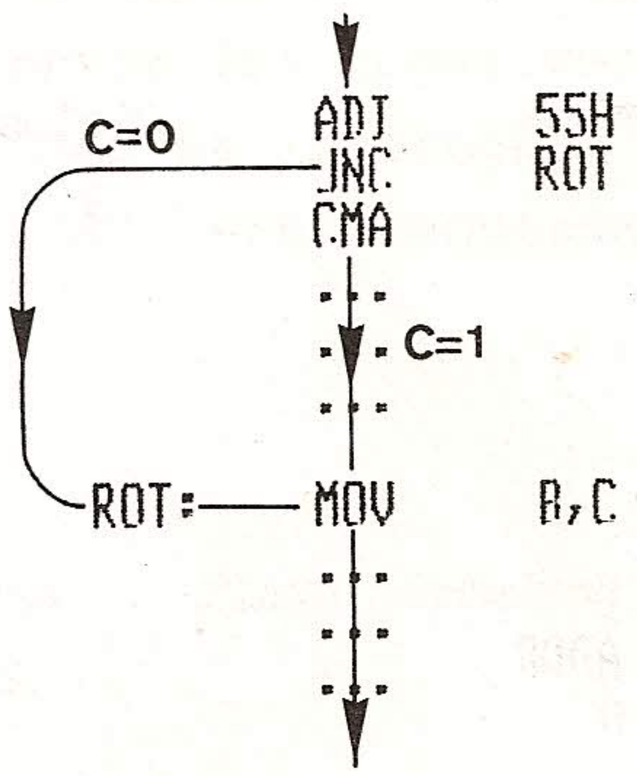
Função:

Esta instrução testará se a flag carry foi setada em "1" ou não. Se a mesma for "0" o programa saltará para o endereço especificado na instrução. Caso contrário, se for "1", a instrução após "JNC" será executada e o programa prosseguirá normalmente.

Parâmetros:

Mnemônico	Hexa	Flags Afetadas	Ciclos	Estados
JNC	D2	-	3(no 8080)	10(no 8080)
			2 ou 3(no 8085)	7 ou 10(no 8085)

Exemplo: JNC ROTA



34.) JNZ — JUMP IF NOT ZERO

Descrição:

Salta, se não zero.

Formato:

Código de Operação	Operando
JNZ	endereço

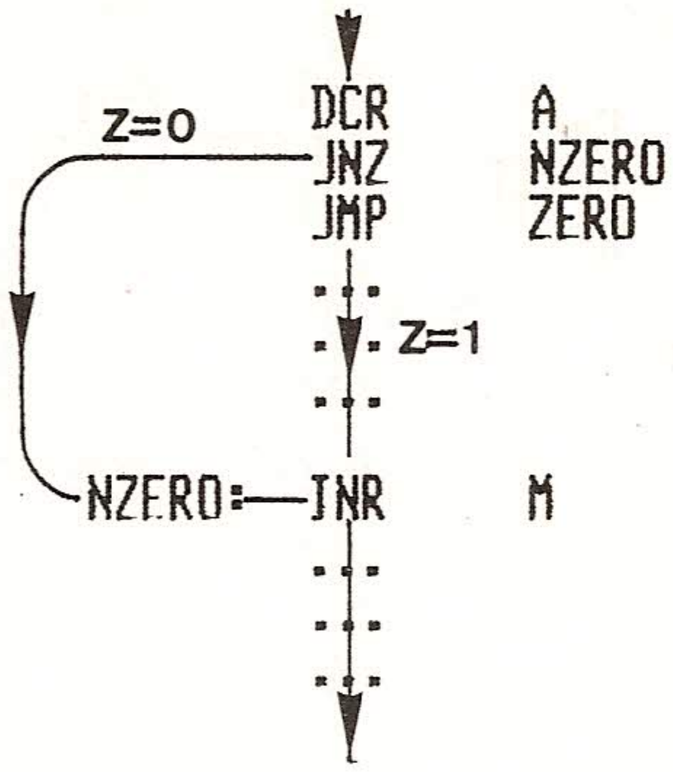
Função:

Esta instrução testará se a flag zero foi setada em "1", ou não. Se a mesma for "0" (o que significa que o resultado da operação anterior foi diferente de zero) o programa saltará para o endereço especificado na instrução. Caso contrário, se for "1", a instrução após "JNZ", será executada e o programa prosseguirá normalmente.

Parâmetros:

Mnemônico	Hexa	Flags Afetadas	Ciclos	Estados
JNZ	C2	-	3(no 8080)	10(no 8080)
			2 ou 3(no 8085)	7 ou 10(no 8085)

Exemplo: JNZ NZERO



35.) JP — JUMP IF POSITIVE

Descrição:

Salta, se positivo.

Formato:

Código de Operação	Operando
JP	endereço

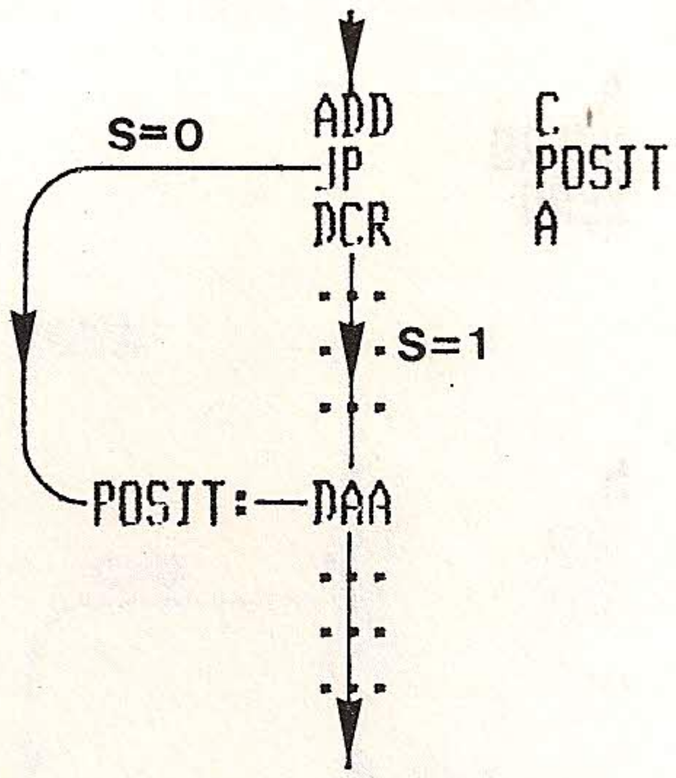
Função:

Esta instrução testará se a flag sinal foi setada em "1" ou não. Se a mesma for "0" (o que significa que o resultado da operação anterior foi positivo) o programa saltará para o endereço especificado na instrução. Caso contrário, se for "1", a instrução após "JP" será executada e o programa prosseguirá normalmente.

Parâmetros:

Mnemônico	Hexa	Flags Afetadas	Ciclos	Estados
JP	F2	-	3(no 8080)	10(no 8080)
			2 ou 3(no 8085)	7 ou 10(no 8085)

Exemplo: JP POSIT



36.) JPE — JUMP IF PARITY EVEN

Descrição:

Salta, se a paridade for par.

Formato:

Código de Operação	Operando
JPE	endereço

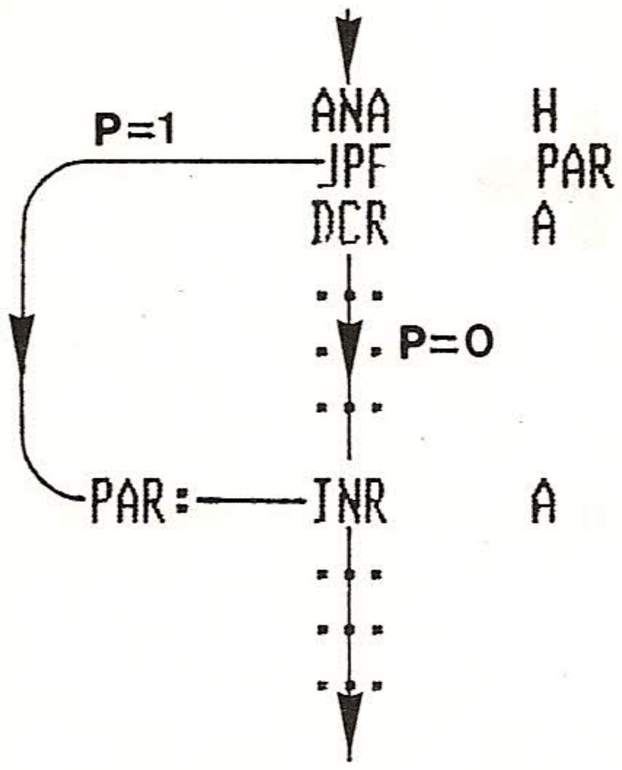
Função:

Esta instrução testará se a flag paridade foi setada em "1" ou não. Se a mesma for "1" (o que significa que a paridade do resultado da operação anterior foi par) o programa saltará para o endereço especificado na instrução. Caso contrário, se for "0", a instrução após "JPE" será executada e o programa prosseguirá normalmente.

Parâmetros:

Mnemônico	Hexa	Flags Afetadas	Ciclos	Estados
JPE	EA	-	3(no 8080)	10(no 8080)
			2 ou 3(no 8085)	7 ou 10(no 8085)

Exemplo: JPE PAR



37.) JPO — JUMP IF PARITY ODD

Descrição:

Salta, se paridade for impar.

Formato:

Código de Operação	Operando
JPO	endereço

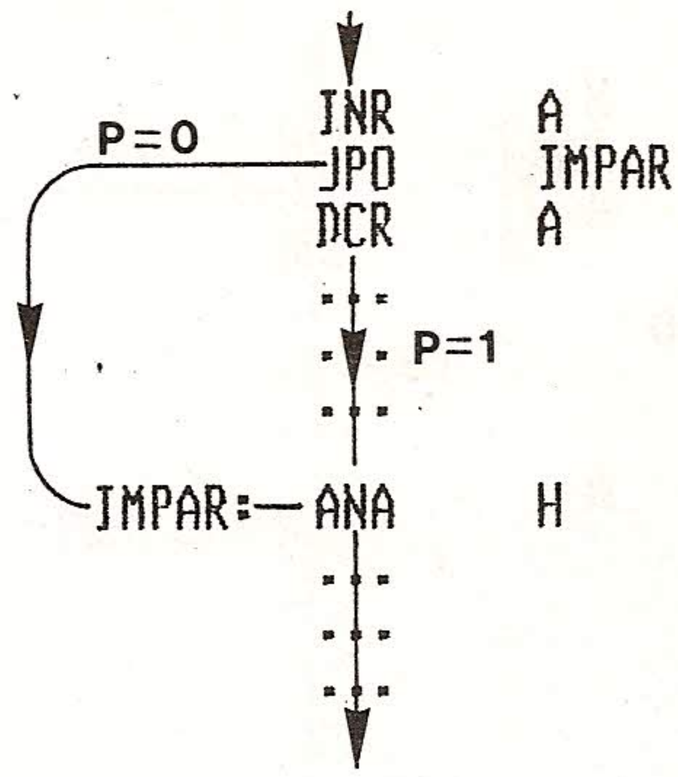
Função:

Esta instrução testará se a flag paridade foi setada em "1" ou não. Se a mesma for "0" (o que significa que a paridade do resultado da operação anterior foi ímpar), o programa saltará para o endereço especificado na instrução. Caso contrário, se for "1", a instrução após "JPO" será executada e o programa prosseguirá normalmente.

Parâmetros:

Mnemônico	Hexa	Flags Afetadas	Ciclos	Estados
JPO	E2	-	3(no 8080)	10(no 8080)
			2 ou 3(no 8085)	7 ou 10(no 8085)

Exemplo: JPO IMPAR



38.) JZ — JUMP IF ZERO

Descrição:

Salta, se zero.

Formato:

Código de Operação	Operando
JZ	endereço

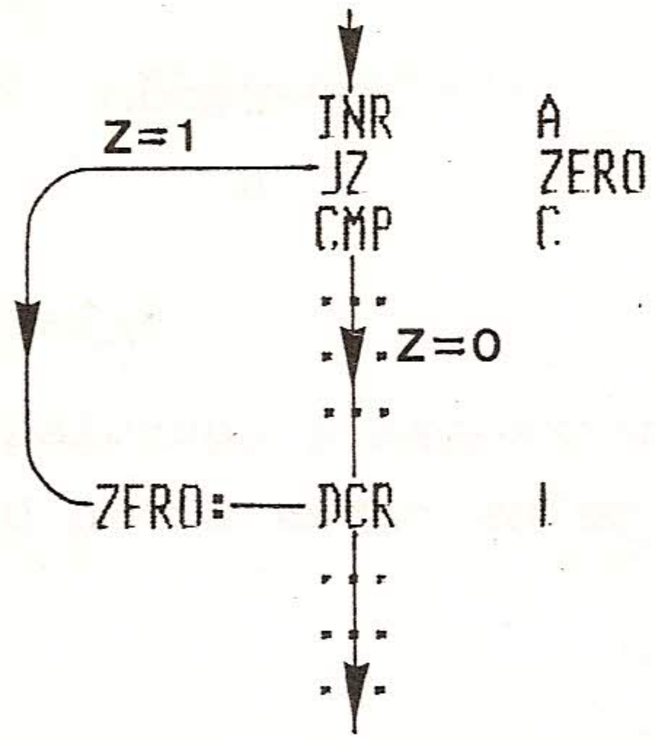
Função:

Esta instrução testará se a flag zero foi setada em "1" ou não. Se a mesma for "1" (o que significa que o resultado da operação anterior foi zero) o programa saltará para o endereço especificado na instrução. Caso contrário, se for "0", a instrução após "JZ" será executada e o programa prosseguirá normalmente.

Parâmetros:

Mnemônico	Hexa	Flags Afetadas	Ciclos	Estados
JZ	CA	-	3(no 8080)	10(no 8080)
			2 ou 3(no 8085)	7 ou 10(no 8085)

Exemplo: JZ ZERO



39.) LDA — LOAD ACCUMULATOR DIRECT

Descrição:

Carrega acumulador diretamente.

Formato:

Código de Operação	Operando
LDA	endereço

Função:

A instrução "LDA" carregará o acumulador com o conteú do da posição de memória de endereço especificado na própria ins trução.

Parâmetros:

Mnemônico	Hexa	Flags Afetadas	Ciclos	Estados
LDA	3A	-	4	13

Exemplo: LDA 002AH

A instrução LDA irá carregar o acumulador com o conteúdo da posição de memória de endereço $002A_{16}$.

40.) LDAX — LOAD ACCUMULATOR INDIRECT

Descrição:

Carrega Acumulador indiretamente.

Formato:

Código de operação	Operando
LDAX	R

Função:

A instrução "LDAX" carregará o acumulador com o conteú do da posição de memória dada pelos pares BC ou DE.

Parâmetros:

Mnemônico	Hexa	Flags Afetadas	Ciclos	Estados
LDAX	xA	-	2	7

Obs.:

"x" poderá assumir os valores 0 ou 1, correspondendo res pectivamente a (LDA B) ou (LDA D).

Exemplo: LDA B

Admitamos que o registrador B contenha 27_{16} e o regis trador C, $4A_{16}$. Assim, a instrução LDAX B carregará o acumulador com o conteúdo da posição de memória de endereço $274A_{16}$.

41.) LHLD — LOAD H AND L DIRECT

Descrição:

Carrega HL diretamente.

Formato:

Código de operação	Operando
LHLD	endereço

Função:

A instrução "LHLD" carregará o registrador L com o con teúdo da posição de memória de endereço especificado na própria instrução e carregará o registrador H, com o conteúdo da posição de memória de endereço seguinte ao endereçc especificado na ins trução.

Parâmetros:

Mnemônico	Hexa	Flags Afetadas	Ciclos	Estados
LHLD	2A	-	5	16

Exemplo: LHLD 24AAH

Admitamos que a memória contenha:

endereço	conteúdo		
$24AA_{16}$	70_{16}	→	L
$24AB_{16}$	$3F_{16}$	→	H

A instrução LHLD 24AAH carregará o registrador L com 70_{16} e o registrador H com $3F_{16}$.

42.) LXI — LOAD REGISTER PAIR IMMEDIATE

Descrição:

Carrega par de registradores com dado imediato.

Formato:

Código de operação	Operando
LXI	R, dado

Função:

A instrução "LXI" carregará o par de registradores especificado no operando com o dado contido no operando. Os pares de registradores poderão ser os pares: BC, DE, HL ou SP.

Parâmetros:

Mnemônico	Hexa	Flags Afetadas	Ciclos	Estados
LXI	x1	-	3	10

Obs.:

"x" poderá assumir os valores 0, 1, 2 ou 3, correspondendo respectivamente a (LXI B,dado),(LXI D, dado),(LXI H, dado) ou (LXI SP,dado).

Exemplo: LXI SP, 3000H

A instrução "LXI SP, 3000H" carregará o par de registradores que formam o stack pointer com 3000_{16}.

43.) MOV — MOVE

Descrição:

Move.

Formato:

Código de operação	Operando
MOV	R ou M, R ou M

Função:

Esta instrução possui três formatos:

a.) MOV R_1, R_2 → Registrador 1, Registrador 2
Neste caso, a instrução MOV irá mover o conteúdo do registrador 2 para o registrador 1.

b.) MOV R, M → Registrador, Memória
Neste caso, a instrução MOV irá mover o conteúdo da posição de memória endereçada pelo par de registradores HL para um dado registrador, A, B, C, D, E, H ou L.

c.) MOV M,R → Memória, Registrador
Neste caso, a instrução MOV irá mover o conteúdo de um dado registrador para a posição de memória endereçada pelo par de registradores HL.

Parâmetros

Mnemônico	Hexa	Flags Afetadas	Ciclos	Estados
MOV (R_1,R_2)	*	-	1	5(no 8080) 4(no 8085)
MOV (R, M)	*	-	2	7
MOV (M, R)	*	-	2	7

* Como o número de instruções de MOV é relativamente grande, sugerimos acompanhar as variações desta instrução no anexo E.

Exemplo:

MOV A,C = Mover o conteúdo do registrador "C" para o acumulador "A".

MOV A,M = Mover para o acumulador o conteúdo da posição de memória endereçada pelo par de registradores HL.

MOV B,E = Mover para o registrador "B" o conteúdo do registrador "E".

Obs.:

A operação "MOV M,M" não é admitida.

44.) MVI — MOVE IMMEDIATE

Descrição:

Move imediato.

Formato:

Código de operação	Operando
MVI	R ou M, dado

Função:

Esta instrução possui dois formatos:

a.) MVI R, dado → Registrador, dado - Neste caso, um dado que poderá apresentar-se como uma constante em ASCII, um Label ou uma expressão, será transferido a um registrador específico, (A,B,C, D,E,H ou L).

b.) MVI M, dado → Memória, dado - Neste caso, um dado que poderá apresentar-se como uma constante em ASCII, um Label ou uma expressão, será transferido a uma posição de memória endereçada pelo par de registradores HL.

Parâmetros:

Mnemônico	Hexa	Flags Afetadas	Ciclos	Estados
MVI (R,dado)	*	-	2	7
MVI (M,dado)	36	-	3	10

* O anexo E ilustrará todas as variações desta instrução.

Exemplo:

```
MVI  M, 47H
MVI  A, 42H
```

45.) NOP — NO OPERATION

Descrição:

Não Operação.

Formato:

Código de operação	Operando
NOP	-

Função:

Nada realizará em um programa, mas será útil na substituição de instruções evitando que se alterem os endereços, será útil no preenchimento de áreas com o objetivo de se gastar tempo e etc.

Parâmetros:

Mnemônico	Hexa	Flags Afetadas	Ciclos	Estados
NOP	00	-	1	4

Exemplo: NOP

Será utilizado em substituição a instruções para evitar que endereços sejam alterados, no preenchimento de posições de memória onde não existem dados, em rotinas de tempo pois as flags não serão afetadas e em inúmeras outras condições.

46.) ORA — INCLUSIVE OR WITH ACCUMULATOR

Descrição:

Inclusive OR com acumulador.

Formato:

Código de operação	Operando
ORA	R ou M

Função:

Esta instrução possui dois formatos:

a.) ORA R → Registrador - Neste caso, a operação lógica "OR" será realizada entre o conteúdo acumulador e um dado registrador, sendo o resultado mantido no acumulador. As flags de carry e auxiliary carry serão resetadas a zero.

b.) ORA M → Memória - Neste caso, a operação lógica "OR" será realizada entre o conteúdo acumulador e o dado referente à posição de memória endereçada pelo par de registradores HL, sendo o resultado mantido no acumulador. As flags de carry e auxiliary carry serão ressetadas para zero.

Parâmetros:

Mnemônico	Hexa	Flags Afetadas	Ciclos	Estados
ORA (R)	Bx	Z,S,P,C,AC	1	4
ORA (M)	B6	Z,S,P,C,AC	2	7

Obs.:

"x" poderá assumir os valores 0, 1, 2, 3, 4, 5 e 7 correspondendo respectivamente aos registradores B, C, D, E, H, L e A.

Exemplo:

```
Acumulador   =  1010 1010
Registrador B =  0010 0001
                 1010 1011
```

Após a instrução ORA B, teremos no acumulador "1010 1011".
Z = 0, S = 1, P = 0, C = 0, AC = 0

47.) ORI — INCLUSIVE OR IMMEDIATE

Descrição:

Inclusive OR imediato.

Formato:

Código de operação	Operando
ORI	dado

Função:

Esta instrução corresponderá a uma operação lógica "OR" entre o acumulador e o segundo byte da instrução (dado), sendo o resultado mantido no acumulador. As flags de carry e auxiliary carry serão ressetadas para zero.

Parâmetros:

Mnemônico	Hexa	Flags Afetadas	Ciclos	Estados
ORI	F6	Z,S,P,C,AC	2	7

Exemplo: ORI 22H

```
Acumulador   = 1000 1000
2º Byte In.  = 0010 0010
               1010 1010
```

Após a instrução de ORI 22H, teremos no acumulador "1010 1010"
Z = 0, S = 1, P = 1, C = 0, AC = 0

48.) OUT — OUTPUT TO PORT

Descrição:

Saída por porta.

Formato:

Código de operação	Operando
OUT	endereço

Função:

Esta instrução carregará a porta de saída especificada com o conteúdo existente no acumulador.

Parâmetros:

Mnemônico	Hexa	Flags Afetadas	Ciclos	Estados
OUT	D3	-	3	10

Exemplo: OUT 80H

Supondo-se que o conteúdo do acumulador seja FF_{16} e que 80_{16} corresponde à porta de uma interface de comunicação paralela 8255, após a instrução OUT 80H, todos os pinos da porta terão nível lógico "1".

49.) PCHL — MOVE HL TO PROGRAM COUNTER

Descrição:

Mover HL para o contador de programa.

Formato:

Código de operação	Operando
PCHL	-

Função:

Mover o conteúdo do par de registradores HL para o par de registradores que forma o contador de programa. Como a próxima instrução a ser executada corresponderá ao endereço constante no contador de programa, a instrução PCHL comportar-se-á como uma instrução de "JUMP".

Parâmetros:

Mnemônico	Hexa	Flags Afetadas	Ciclos	Estados
PCHL	E9	-	1	5(no 8080) 6(no 8085)

50.) POP — POP

Descrição:

Retira do topo do stack.

Formato:

Código de Operação	Operando
POP	R ou PSW

Função:

Esta instrução possui dois formatos:

a.) POP R

- Moverá o conteúdo da posição de memória endereçada pelo registrador stack pointer para o registrador de mais baixa ordem (C, E ou L) e incrementará o conteúdo do stack pointer.

- Moverá o conteúdo da posição de memória endereçada pelo novo conteúdo do stack pointer para o registrador de mais alta ordem (B, D ou H) e incrementará novamente o conteúdo do stack pointer.

b.) POP PSW

- Moverá o conteúdo da posição de memória endereçada pelo registrador stack pointer para o registrador das flags e incrementará o conteúdo do stack pointer.

- Moverá o conteúdo da posição de memória endereçada pelo novo conteúdo do stack pointer para o acumulador e incrementará novamente o conteúdo do stack pointer.

Parâmetros:

Mnemônico	Hexa	Flags Afetadas	Ciclos	Estados
POP (R)	xl	-	3	10
POP (PSW)	Fl	Z,S,P,C,AC	3	10

Obs.:

"x" assumirá os valores C,D, ou E o que corresponderá respectivamente a POP B , POP D, ou POP H .

Exemplo: POP H

Supondo-se que:

- valor inicial do registrador stack pointer 3010.
- valor da memória.

endereço	conteúdo
3010	2B
3011	C4

Como resultado da instrução POP H teremos:

- valor final do registrador stack pointer → 3012
- valor final do registrador H → C4
- valor final do registrador L → 2B

51.) PUSH — PUSH

Descrição:

Coloca no topo do stack.

Formato:

Código de Operação	Operando
PUSH	R ou PSW

Função:

Esta instrução possui dois formatos:

a.) PUSH R

- decrementará o conteúdo do registrador stack pointer e moverá o conteúdo do registrador de mais alta ordem (B, D ou H) para a posição de memória de endereço correspondente ao novo valor do stack pointer.

- decrementará novamente o conteúdo do registrador stack pointer e moverá o conteúdo do registrador de mais baixa ordem (C, E ou L) para a posição de memória de endereço correspondente ao novo valor do stack pointer.

b.) PUSH PSW →

- decrementará o conteúdo do registrador stack pointer e moverá o conteúdo do acumulador para a posição de memória de endereço correspondente ao novo valor do stack pointer.

- decrementará novamente o conteúdo do registrador stack pointer e moverá o conteúdo do registrador das flags para a posição de memória correspondente ao novo valor do stack pointer.

Parâmetros:

Mnemônico	Hexa	Flags Afetadas	Ciclos	Estados
PUSH (R)	x5	-	3	11(no 8080) 13(no 8085)
PUSH (PSW)	F5	-	3	11(no 8080) 12(no 8085)

Obs.:

"x" assumirá os valores C, D ou E o que corresponderá respectivamente à (PUSH B), (PUSH D) ou (PUSH H).

Exemplo: PUSH PSW

Supondo-se que:

- valor inicial do registrador stack pointer 2237.
- valor do registrador das flags.

S	Z		AC		P		C
1	0	0	0	0	1	1	0

$= 86_{16}$

- valor do acumulador

A	A

Como resultado da instrução PUSH PSW, teremos:

- valor final do registrador stack pointer 2235.
- valor final da memória.

endereço	conteúdo
2235	86
2236	AA

52.) RAL — ROTATE LEFT THROUGH CARRY

Descrição:

Girar para a esquerda através do carry.

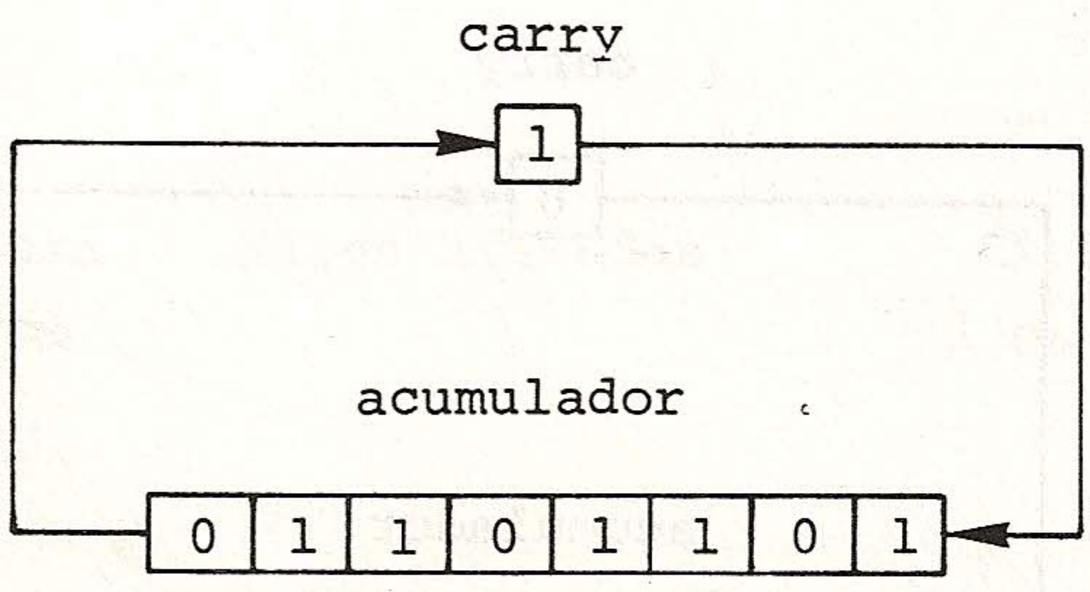
Formato:

Código de operação	Operando
RAL	-

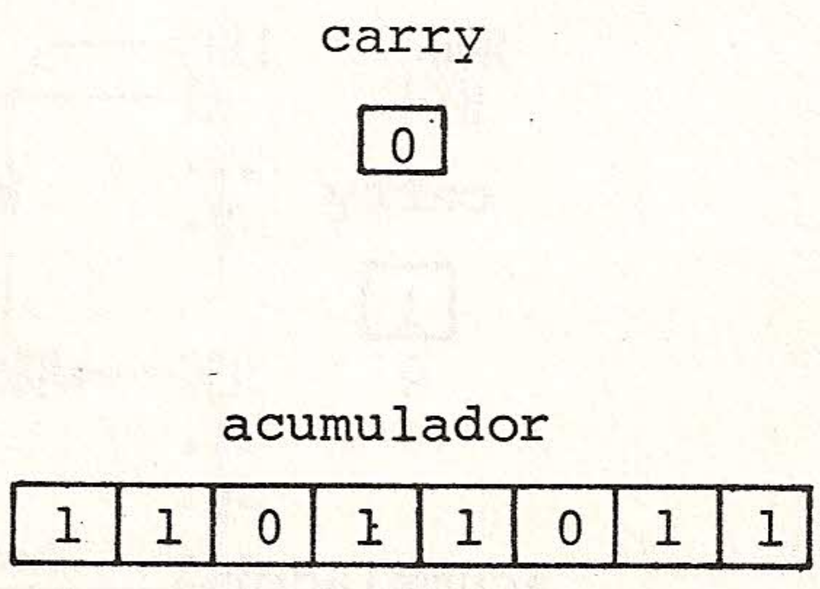
Função:

A instrução "RAL" girará o conteúdo do acumulador de um bit para a esquerda através da flag carry, como mostrada nas duas ilustrações a seguir:

a.) Antes de RAL



b.) Depois de RAL



Parâmetros:

Mnemônico	Hexa	Flags Afetadas	Ciclos	Estados
RAL	17	C	1	4

53.) RAR — ROTATE RIGHT THROUGH CARRY

Descrição:

Girar para a direita através do carry.

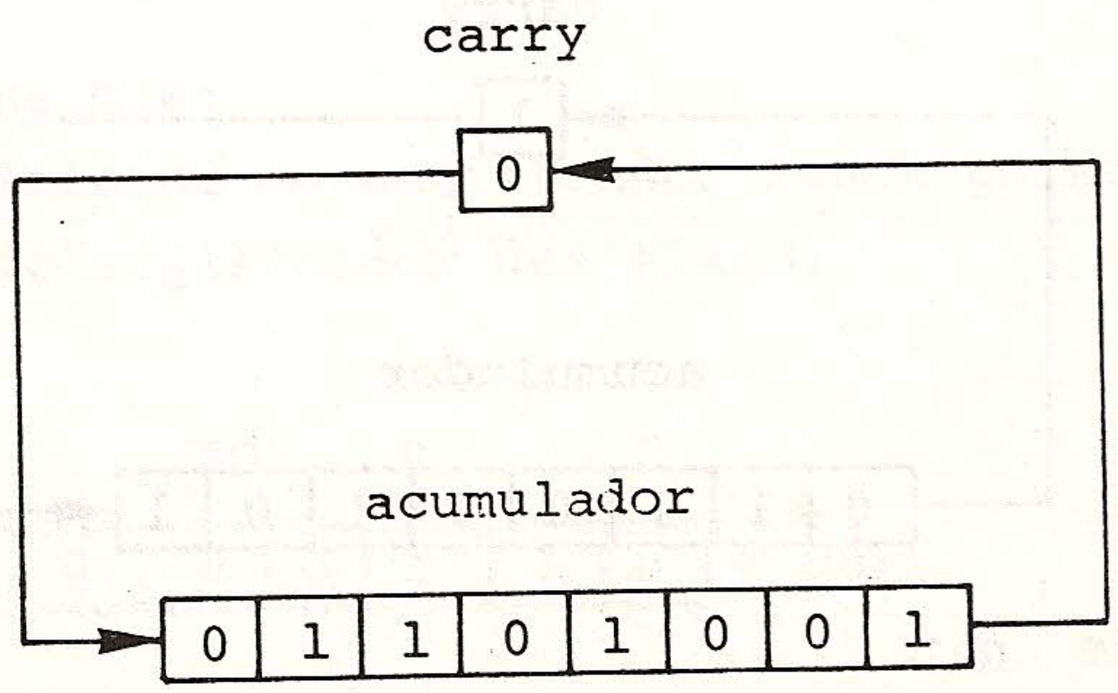
Formato:

Código de operação	Operando
RAR	-

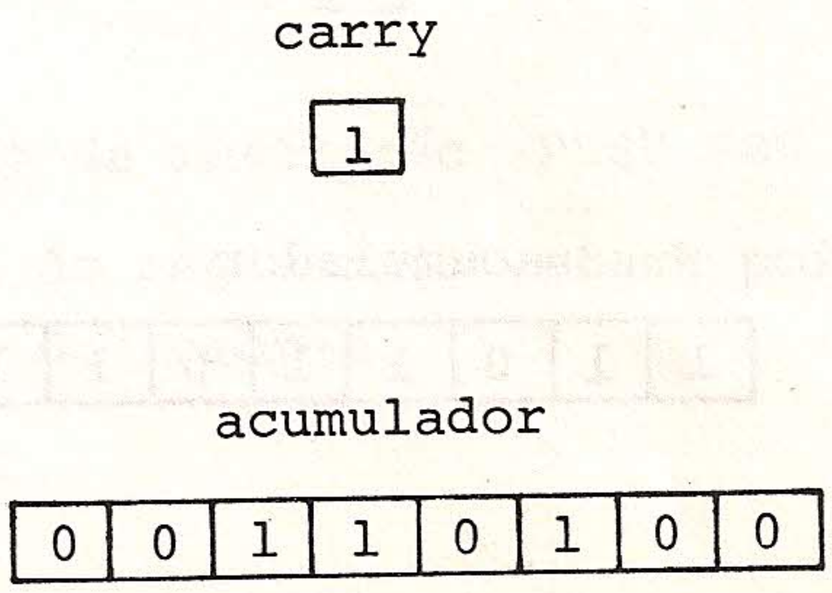
Função:

A instrução "RAR" girará o conteúdo do acumulador de um bit para a direita através da flag carry, como mostrado nas duas ilustrações a seguir:

a.) Antes de RAR



b.) Depois de RAR



Parâmetros:

Mnemônico	Hexa	Flags Afetadas	Ciclos	Estados
RAR	1F	C	1	4

54.) RC — RETURN IF CARRY

Descrição:

Retorno, se houver carry.

Formato:

Código de Operação	Operando
RC	-

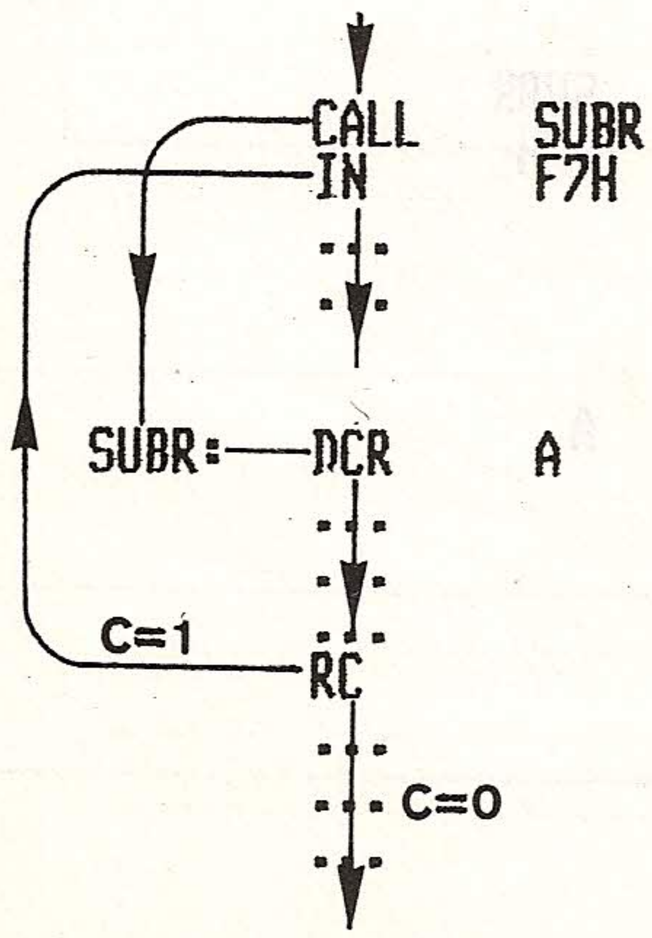
Função:

Esta instrução testará se a flag carry foi setada em "1" ou não. Se a mesma for "1" ela funcionará como uma instrução "RET", o fluxo do programa retornará à instrução seguinte a instrução que chamou a sub-rotina. Caso contrário, se for "0" a instrução após "RC" será executada e o programa prosseguirá normalmente.

Parâmetros:

Mnemônico	Hexa	Flags Afetadas	Ciclos	Estados
RC	D8	-	1 ou 3	5 ou 11(no 8080) 6 ou 12(no 8085)

Exemplo: RC



55.) RET — RETURN FROM SUBROUTINE

Descrição:

Retorno de sub-rotina.

Formato:

Código de operação	Operando
RET	-

Função:

A instrução "RET" será utilizada normalmente como última instrução de uma sub-rotina e fará com que, após o término da execução da sub-rotina, o fluxo de processamento retorne à instrução seguinte a instrução que chamou a sub-rotina.

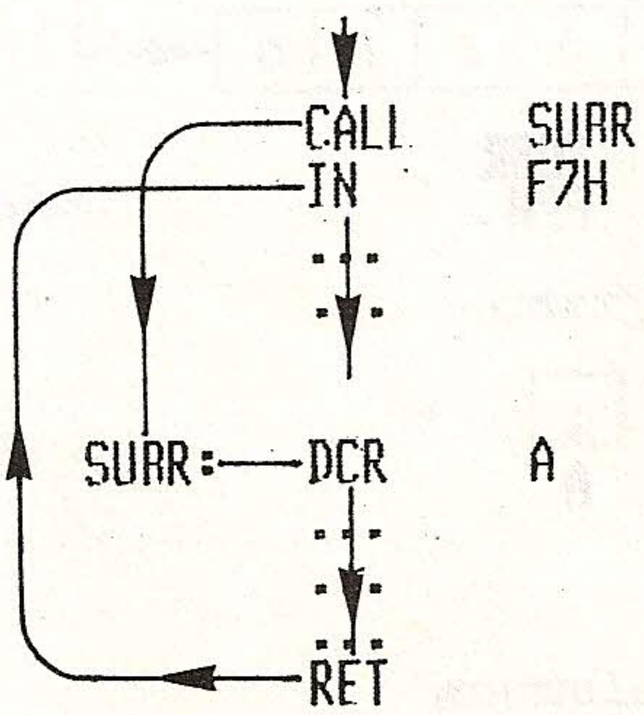
Esta instrução provocará as seguintes ações:

- colocará no registrador, "contador de programa", os dois bytes consecutivos armazenados na memória de dados, cujo endereço será indicado pelo registrador "stack pointer".
- o conteúdo do registrador "stack pointer" será incrementado duas vezes para que este venha conter o endereço da próxima posição do "stack" que será utilizado pelo programa.

Parâmetros:

Mnemônico	Hexa	Flags Afetadas	Ciclos	Estados
RET	C9	-	3	10

Exemplo: RET



56.) RIM — READ INTERRUPT MASK

Obs.:

Instrução existente apenas no microprocessador 8085.

Descrição:

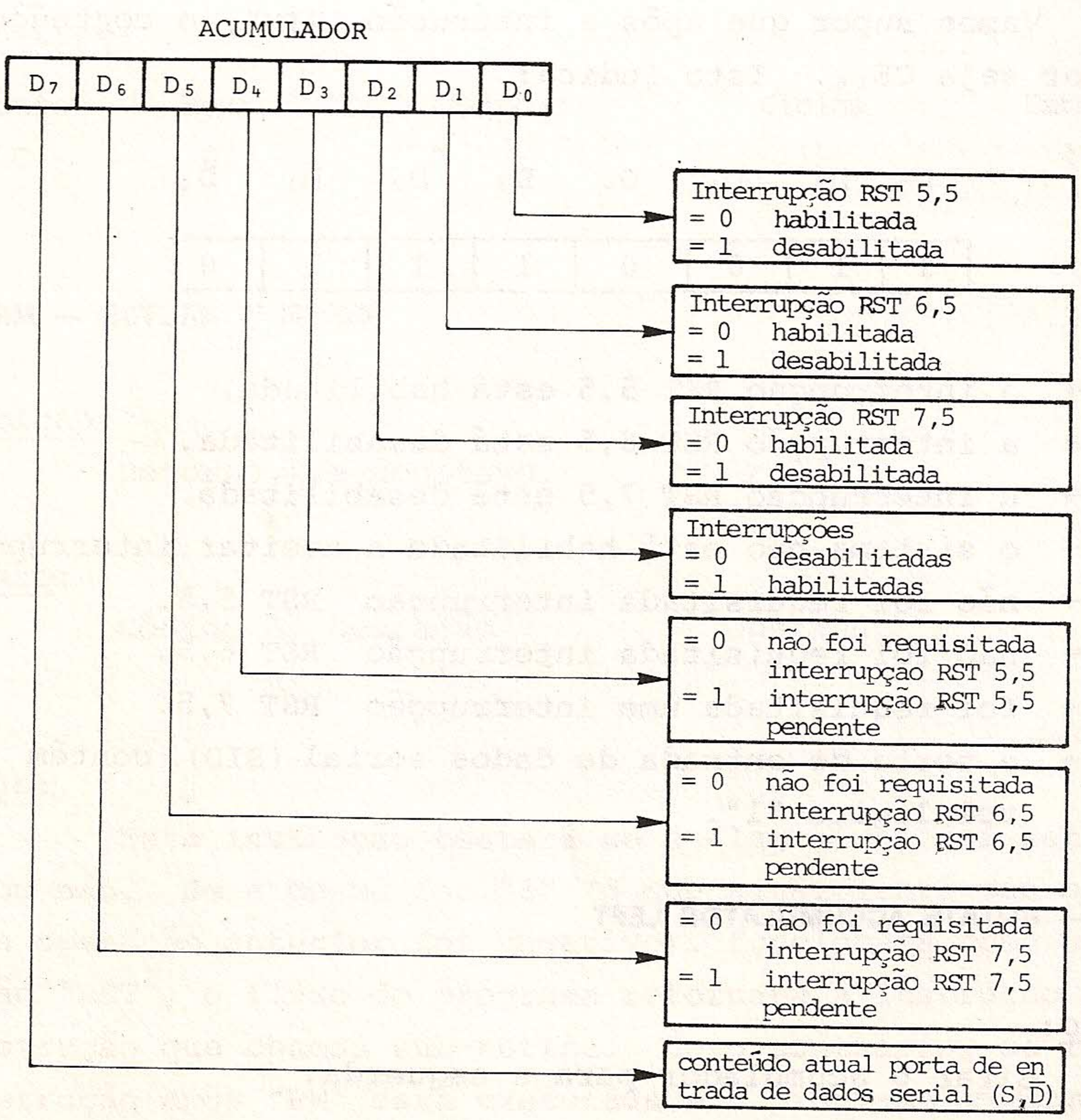
Ler a máscara de interrupção.

Formato:

Código de Operação	Operando
RIM	-

Função:

A instrução "RIM" fará com que o acumulador seja carregado com um dado de oito bit. O formato deste dado será:



Obs.:

As interrupções RST 5,5; RST 6,5; RST 7,5 funcionarão de maneira similar às instruções RST 0 à RST 7, porém o endereço para onde o programa irá saltar, dependendo da interrupção será:

interrupção	endereço
RST 5,5	$002C_{16}$
RST 6,5	0034_{16}
RST 7,5	$003C_{16}$

Parâmetros:

Mnemônico	Hexa	Flags Afetadas	Ciclos	Estados
RIM	20	-	1	4

Exemplo: RIM

Vamos supor que após a instrução "RIM", o conteúdo do acumulador seja CE_{16}. Isto indica:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	1	0	0	1	1	1	0

$D_0 = 0$ → a interrupção RST 5,5 está habilitada.
$D_1 = 1$ → a interrupção RST 6,5 está desabilitada.
$D_2 = 1$ → a interrupção RST 7,5 está desabilitada.
$D_3 = 1$ → o sistema não está habilitado a aceitar interrupções.
$D_4 = 0$ → não foi requisitada interrupção RST 5,5.
$D_5 = 0$ → não foi requisitada interrupção RST 6,5.
$D_6 = 1$ → foi requisitada uma interrupção RST 7,5.
$D_7 = 1$ → a porta de entrada de dados serial (SID) contém o nível lógico "1".

57.) RLC — ROTATE ACCUMULATOR LEFT

Descrição:

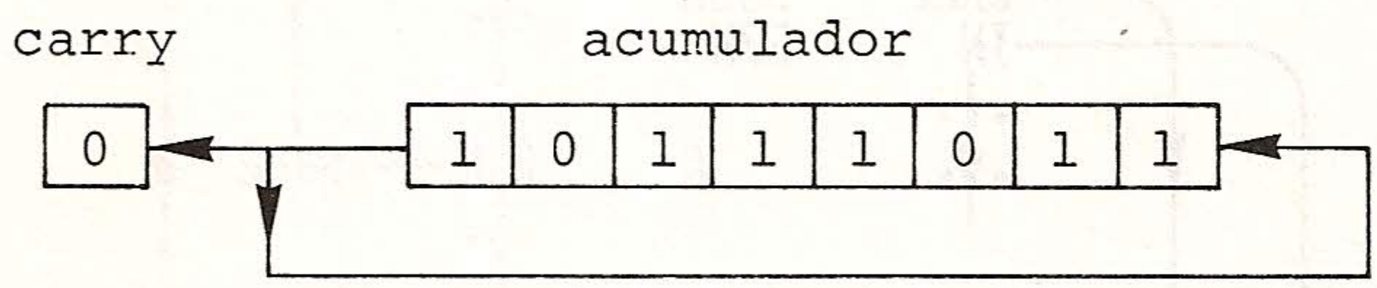
Girar o acumulador para a esquerda.

Formato:

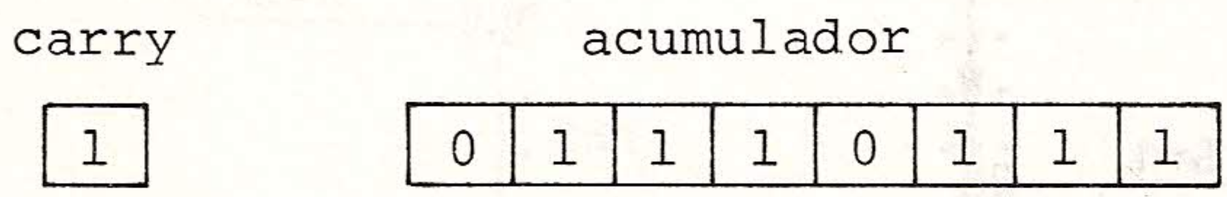
Código de operação	Operando
RLC	-

Função: A instrução RLC girará do conteúdo do acumulador de um bit para a esquerda e transferirá para a flag carry o bit mais significativo do acumulador como mostrado nas duas ilustrações a seguir:

a.) antes de RCL:



b.) depois de RCL:



Parâmetros:

Mnemônico	Hexa	Flags Afetadas	Ciclos	Estados
RLC	07	C	1	4

58.) RM — RETURN IF MINUS

Descrição:

Retorno, se negativo.

Formato:

Código de Operação	Operando
RM	-

Função:

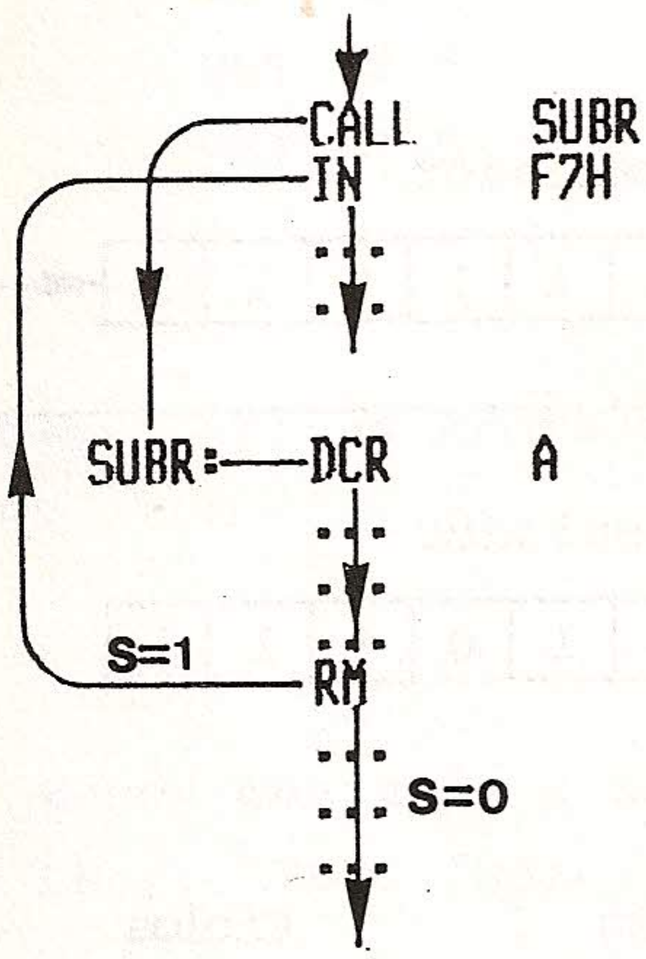
Esta instrução testará se a flag sinal foi setada em "1" ou não. Se a mesma for "1" (o que significará que o resulta do da operação anterior foi negativo), funcionará como uma ins trução "RET", o fluxo do programa retornará à instrução seguinte a instrução que chamou sub-rotina. Caso contrário, se for "0", a instrução após "RM" será executada e o programa prosseguirá nor

malmente.

Parâmetros:

Mnemônico	Hexa	Flags Afetadas	Ciclos	Estados
RM	F8	-	1 ou 3	5 ou 11(no 8080) 6 ou 12(no 8085)

Exemplo: RM



59.) RNC — RETURN IF NO CARRY

Descrição:

Retorno, se não houver carry.

Formato:

Código de Operação	Operando
RNC	-

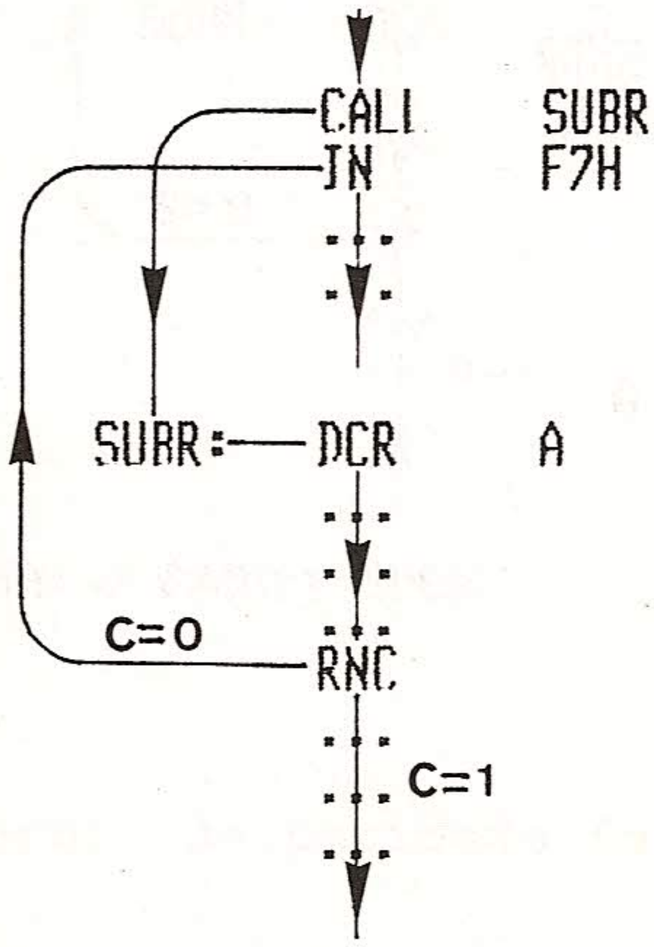
Função:

Esta instrução testará se a flag carry foi setada em "1" ou não. Se a mesma for "0", ela funcionará como uma instrução "RET". O fluxo do programa retornará à instrução seguinte a instrução que chamou a sub-rotina. Caso contrário, se for "1", a instrução após "RNC" será executada e o programa prosseguirá normalmente.

Parâmetros:

Mnemônico	Hexa	Flags Afetadas	Ciclos	Estados
RNC	D0	-	1 ou 3	5 ou 11(no 8080) 6 ou 12(no 8085)

Exemplo: RNC



60.) RNZ — RETURN IF NOT ZERO

Descrição:

Retorno, se não zero.

Formato:

Código de Operação	Operando
RNZ	-

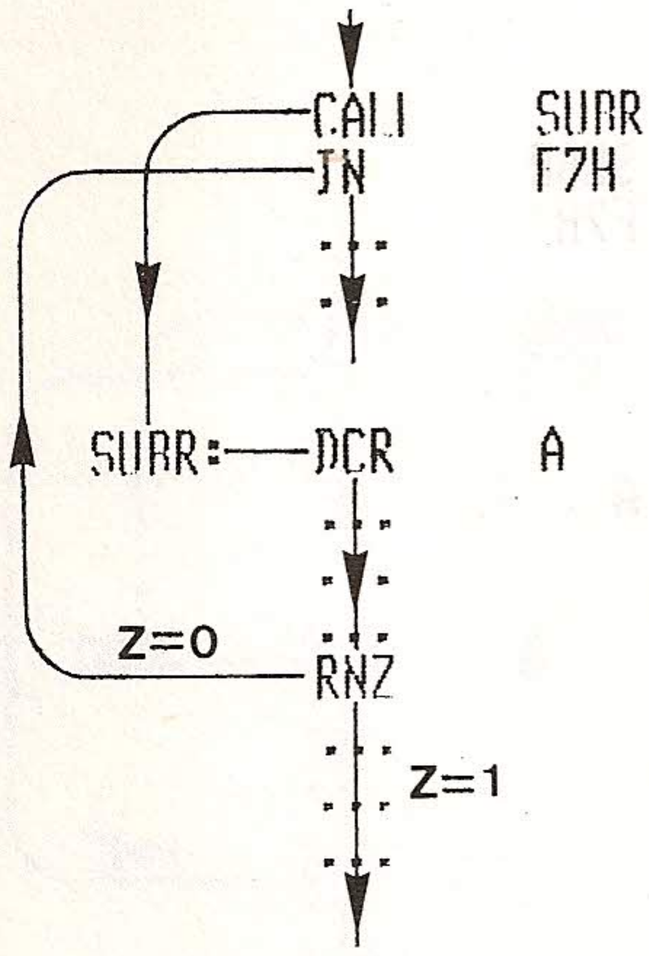
Função:

Esta instrução testará se a flag zero foi setada em "1" ou não. Se a mesma for "0" (o que significará que o resultado da operação anterior foi diferente de zero), funcionará como uma instrução "RET". O fluxo do programa retornará à instrução seguinte a instrução que chamou a sub-rotina. Caso contrário, se for "1", a instrução após "RNZ" será executada e o programa prosseguirá normalmente.

Parâmetros:

Mnemônico	Hexa	Flags Afetadas	Ciclos	Estados
RNZ	C0	-	1 ou 3	5 ou 11(no 8080) 6 ou 12(no 8085)

Exemplo: RNZ



61.) RP — RETURN IF POSITIVE

Descrição:

Retorno, se positivo.

Formato:

Código de Operação	Operando
RP	-

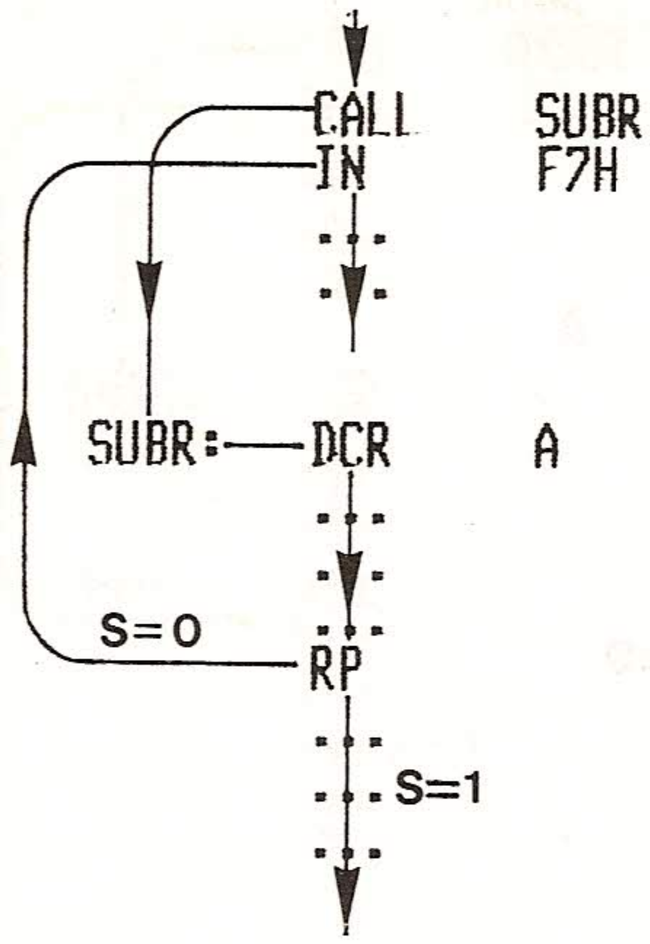
Função:

Esta instrução testará se a flag sinal foi setada em "1" ou não. Se a mesma for "0" (o que significará que o resultado da operação anterior foi positivo), funcionará como uma instrução "RET", o fluxo do programa retornará para a instrução seguinte a instrução que chamou sub-rotina. Caso contrário, se for "1", a próxima instrução após "RP" será executada e o programa prosseguirá normalmente.

Parâmetros:

Mnemônico	Hexa	Flags Afetadas	Ciclos	Estados
RP	F0	-	1 ou 3	5 ou 11(no 8080) 6 ou 12(no 8085)

Exemplo: RP



62.) RPE — RETURN IF PARITY EVEN

Descrição:

Retorno, se paridade for par.

Formato:

Código de Operação	Operando
RPE	-

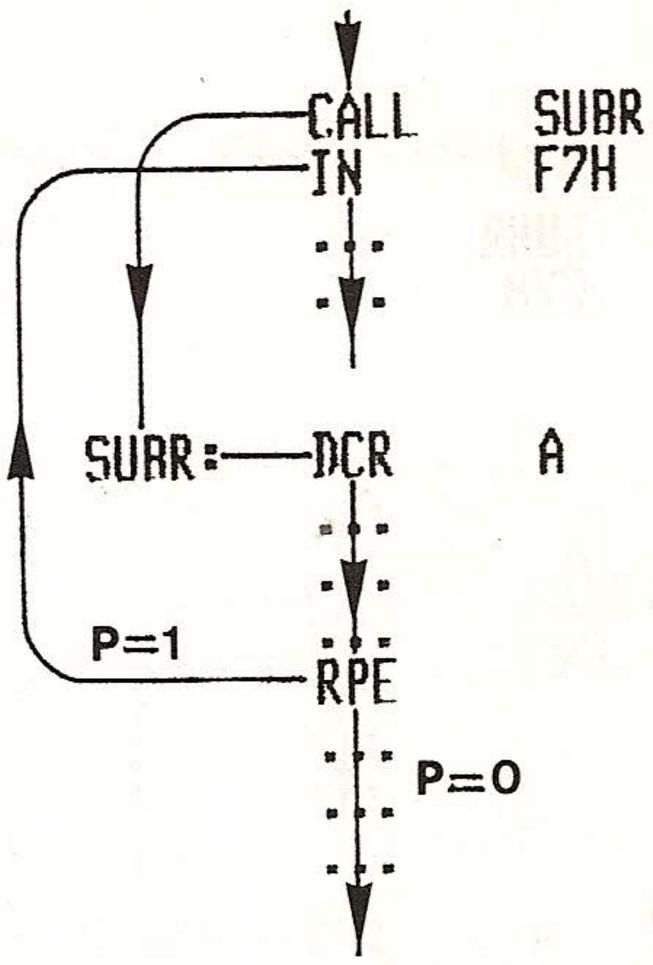
Função:

Esta instrução testará se a flag paridade foi setada em "1" ou não. Se a mesma for "1" (o que significará que a paridade do resultado da operação foi par), funcionará como uma instrução "RET", e o fluxo do programa retornará à instrução seguinte a instrução que chamou sub-rotina. Caso contrário, se for "0", a instrução após "RPE" será executada e o programa prosseguirá normalmente.

Parâmetros:

Mnemônico	Hexa	Flags Afetadas	Ciclos	Estados
RPE	E8	-	1 ou 3	5 ou 11(no 8080) 6 ou 12(no 8085)

Exemplo: RPE



63.) RPO — RETURN IF PARITY ODD

Descrição:

Retorno se a paridade for impar.

Formato:

Código de Operação	Operando
RPO	-

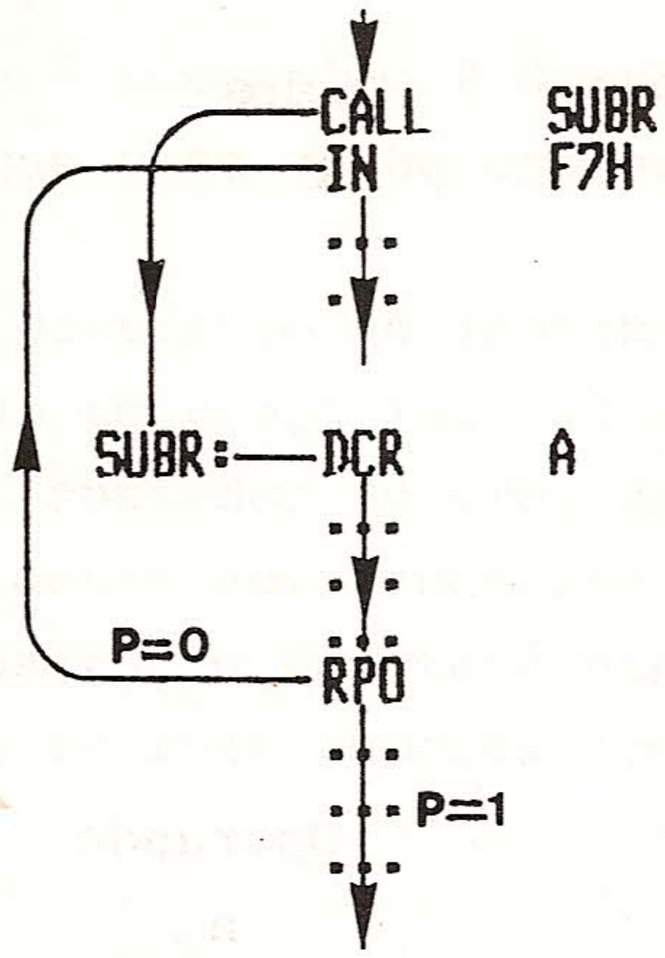
Função:

Esta instrução testará se a flag paridade foi setada em "1" ou não. Se a mesma for "0" (o que significará que o resultado da operação anterior foi impar), funcionará como uma instrução "RET" e o fluxo do programa retornará para a instrução seguinte à que chamou a sub-rotina. Caso contrário, se for "1", a instrução após "RPO" será executada e o programa prosseguirá normalmente.

Párâmetros:

Mnemônico	Hexa	Flags Afetadas	Ciclos	Estados
RPO	E0	-	1 ou 3	5 ou 11(no 8080) 6 ou 12(no 8085)

Exemplo: RPO



64.) RRC — ROTATE ACCUMULATOR RIGHT

Descrição:

Girar o acumulador para a direita.

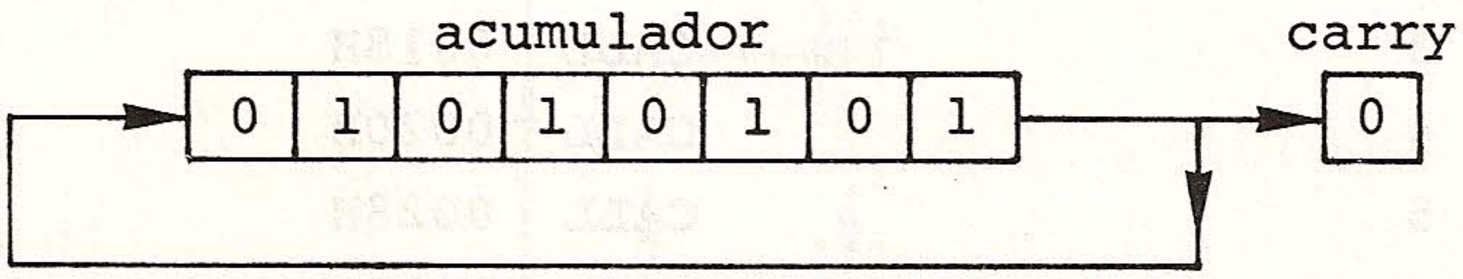
Formato:

Código de Operação	Operando
RRC	-

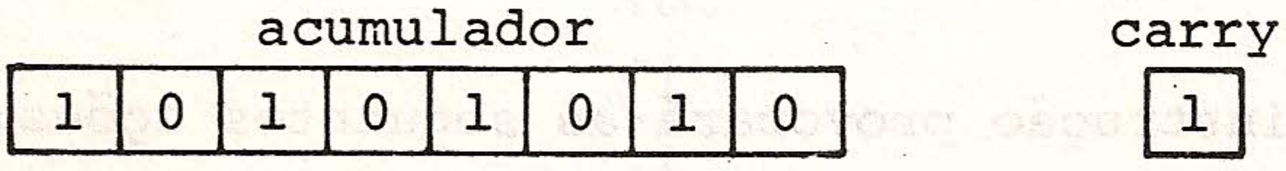
Função:

A instrução RRC girará o conteúdo do acumulador de um bit para a direita e transferirá para a flag carry o bit menos significativo do acumulador, como mostrado nas duas ilustrações a seguir:

a.) Antes de RRC



b.) Depois de RRC



Parâmetros:

Mnemônico	Hexa	Flags Afetadas	Ciclos	Estados
RRC	0F	C	1	4

65.) RST — RESTART

Descrição:

Recomeça.

Formato:

Código de Operação	Operando
RST	n

Obs.:

n pode assumir os valores de 0 à 7.

Função:

A instrução "RST" funcionará como uma instrução "CALL", provocará uma quebra na seqüência normal do programa para a execução de uma sub-rotina. A diferença é que, na instrução "CALL" o endereço inicial da posição de memória onde está armazenado a sub-rotina será o dado imediato do operando e, na instrução "RST", o endereço será a constante do operando multiplicada por "8". Isto é:

instrução	corresponde
RST 0	CALL 0000H
RST 1	CALL 0008H
RST 2	CALL 0010H
RST 3	CALL 0018H
RST 4	CALL 0020H
RST 5	CALL 0028H
RST 6	CALL 0030H
RST 7	CALL 0038H

Esta instrução provocará as seguintes ações:

- o endereço da instrução seguinte será armazenada na memória de dados cujo endereço será indicado pelo registrador

"stack pointer".

- será subtraído 2 do conteúdo do registrador "stack pointer" para que este venha conter o novo endereço disponível do "stack".

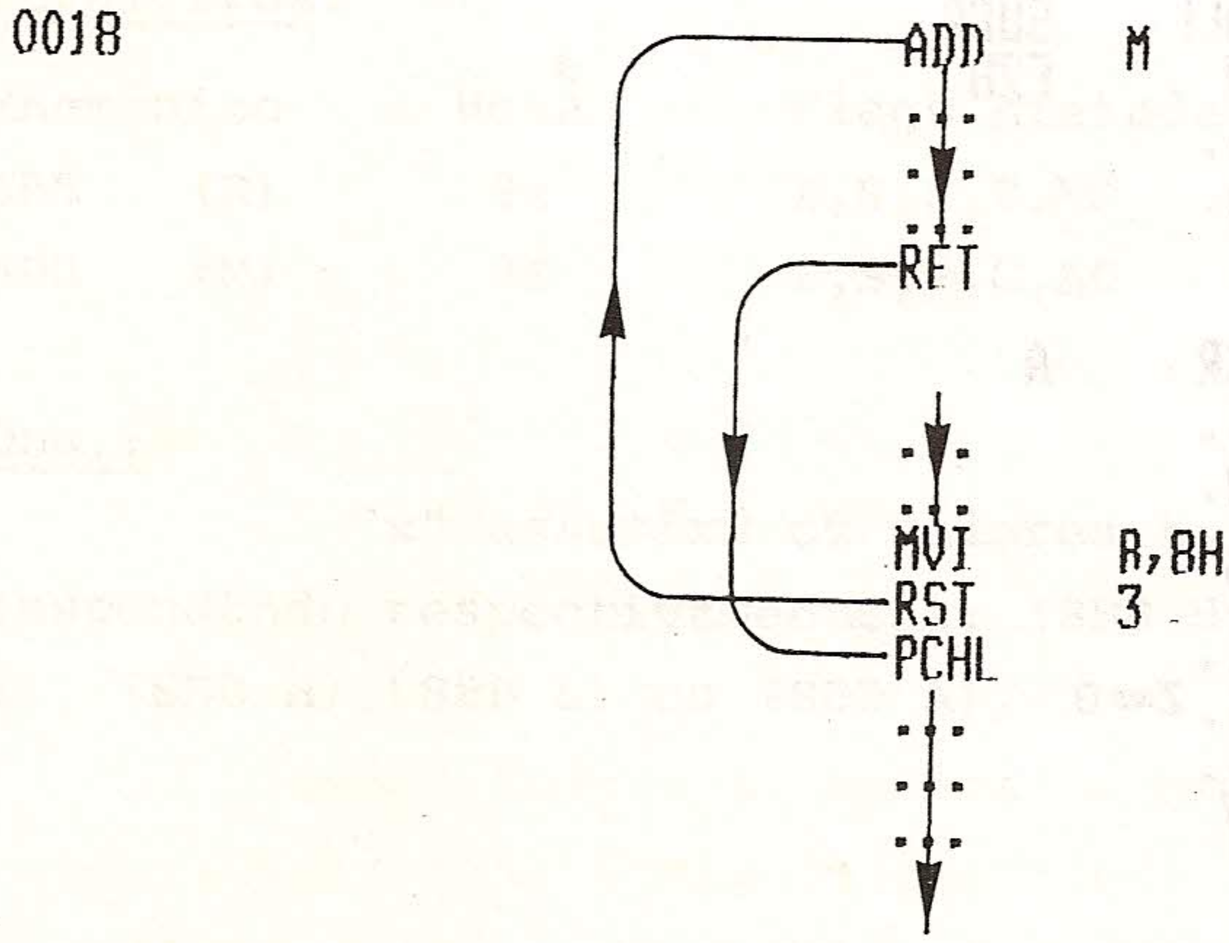
- a constante do operando, multiplicada por "8", o que corresponderá ao endereço inicial da sub-rotina, será armazenada no registrador "contador de programa" para que a execução do programa continue neste novo endereço.

A instrução RST terá como função básica a utilização em conjunto com as interrupções (ver técnicas de programação, capítulo VI).

Parâmetros: código

Mnemônico	Hexa	Flags Afetadas	Ciclos	Estados
RST (0)	C7	-	3	11(no 8080)
RST (1)	CF			12(no 8085)
RST (2)	D7			
RST (3)	DF			
RST (4)	E7			
RST (5)	EF			
RST (6)	F7			
RST (7)	FF			

Exemplo: RST 3



66.) RZ — RETURN IF ZERO

Descrição:

Retorno, se zero.

Formato:

Código de Operação	Operando
RZ	-

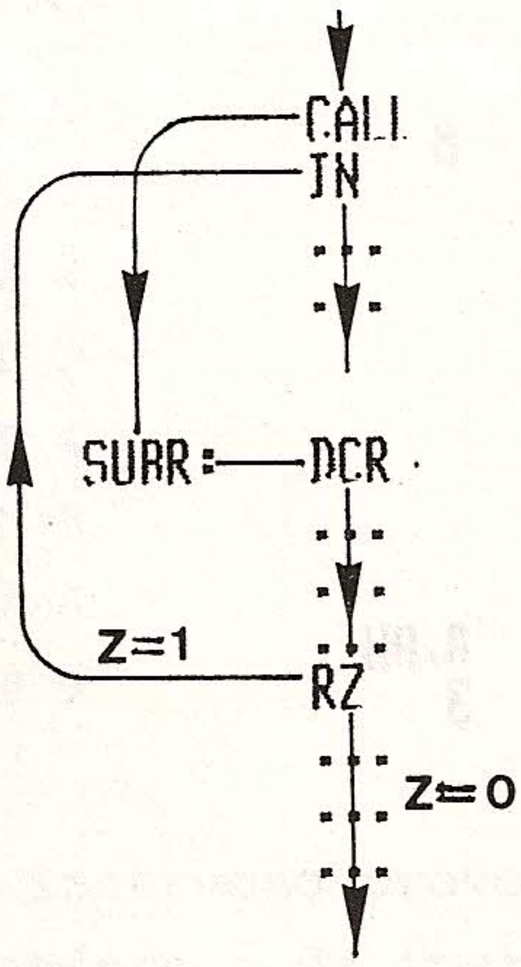
Função:

Esta instrução testará se a flag zero foi setada em "1" ou não. Se a mesma for "1" (o que significará que o resultado da operação anterior foi zero), funcionará como uma instrução "RET", e o fluxo do programa retornará à instrução seguinte a instrução que chamou a sub-rotina. Caso contrário, se for "0", a instrução após "RZ" será executada e o programa prosseguirá normalmente.

Parâmetros:

Mnemônico	Hexa	Flags Afetadas	Ciclos	Estados
RZ	C8	-	1 ou 3	5 ou 11(no 8080) 6 ou 12(no 8085)

Exemplo: RZ



67.) SBB — SUBTRACT WITH BORROW

Descrição:

Subtração com carry.

Formato:

Código de Operação	Operando
SBB	R ou M

Função:

Esta instrução possui dois formatos:

a.) SBB R → Subtrai do conteúdo do acumulador o conteúdo de um dado registrador mais o carry bit, e armazena o resultado no acumulador.

b.) SBB M → Subtrai do conteúdo do acumulador o conteúdo da posição de memória endereçada pelo par de registradores "HL" e mais o carry bit, e armazena o resultado no acumulador.

Obs.:

Para realização da operação de subtração será utilizada a técnica de adição do complemento dois, seguida por uma complementação da flag carry.

Parâmetros:

Mnemônico	Hexa	Flags Afetadas	Ciclos	Estados
SBB (R)	9x	Z,S,P,C,AC	1	4
SBB (M)	9E	Z,S,P,C,AC	2	7

Obs.:

"x" assumirá os valores 8, 9, A, B, C, D ou F, correspondendo respectivamente a: (SBB B), (SBB C), (SBB D), (SBB E), (SBB H),(SBB L) ou (SBB A).

Exemplo: SBB C

Registrador C = 0000 0101 = 05_{16}
Carry bit = 1 = 01_{16}
Reg. C + carry bit = 0000 0110 = 06_{16}

O complemento 2 de 06_{16} é obtido como segue:

0000 0110 = 06_{16}
1111 1001 = $F9_{16}$ → complemento 1 de 06_{16}.
1
1111 1010 = FA_{16} → complemento 2 de 06_{16}.

Acumulador = 0000 0111 = 07_{16}
Compl. 2 de 06_{16} = 1111 1010 = FA_{16}
Soma = 0000 0001 = 01_{16}

C = 1 complementando C = 0, portanto:
Z = 0, S = 0, P = 0, C = 0, AC = 1

68.) SBI — SUBTRACT IMMEDIATE WITH BORROW

Descrição:

Subtração imediata com carry.

Formato:

Código de Operação	Operando
SBI	dado

Função:

A instrução "SBI" subtrairá do conteúdo do acumulador o conteúdo do segundo byte da instrução mais a carry flag, e armazenará o resultado da operação no acumulador.

Para a realização da operação de subtração será utilizada a técnica de adição do complemento dois, seguida de uma complementação da flag carry.

Parâmetros:

Mnemônico	Hexa	Flags Afetadas	Ciclos	Estados
SBI	DE	Z,S,P,C,AC	2	7

Exemplo: SBI 07H

Dado Imediato = 0000 0111 = 07_{16}
Carry bit = 0 + = 00_{16}
Dado ime. + carry bit = 0000 0111 = 07_{16}

O complemento 2 de 07_{16} será obtido como segue:

0000 0111 = 07_{16}
1111 1000 = $F8_{16}$ → complemento 1 de 07_{16}.
1 + = 01_{16}
1111 1001 = $F9_{16}$ → complemento 2 de 07_{16}.

Acumulador = 0000 0111 = 07_{16}
Compl. 2 de 07_{16} = 1111 1001 + = $F9_{16}$
0000 0000 = 00_{16}

C = 1 complemento C = 0, portanto
Z = 1, S = 0, P = 1, C = 0, AC = 1

69.) SHLD — STORE HL DIRECT

Descrição:

Armazenar par de registradores HL diretamente.

Formato:

Código de Operação	Operando
SHLD	endereço

Função:

A instrução "SHLD" transferirá o conteúdo do par de registradores HL para a posição de memória, cujo endereço é especificado no operando da instrução.

Parâmetros:

Mnemônico	Hexa	Flags Afetadas	Ciclos	Estados
SHLD	22	-	5	16

Exemplo: SHLD 0102H

Admitamos que o conteúdo do registrador H seja Bl_{16} e que o conteúdo de L seja 09_{16}.

102	103	→ Posição de Memória.
00	00	→ Conteúdo antes da instrução.
09	Bl	→ Conteúdo após a instrução ser executada.

Como podemos observar, o conteúdo do registrador "H" foi armazenado na posição de memória "103" (mais significativa) e o conteúdo de "L" na posição de memória "102" (menos significativa com relação a "H"), o que podemos tomar como regra para esta instrução.

70.) SIM — SET INTERRUPT MASK

Obs.:

Instrução existente apenas no microprocessador 8085.

Descrição:

Posiciona a máscara de interrupção.

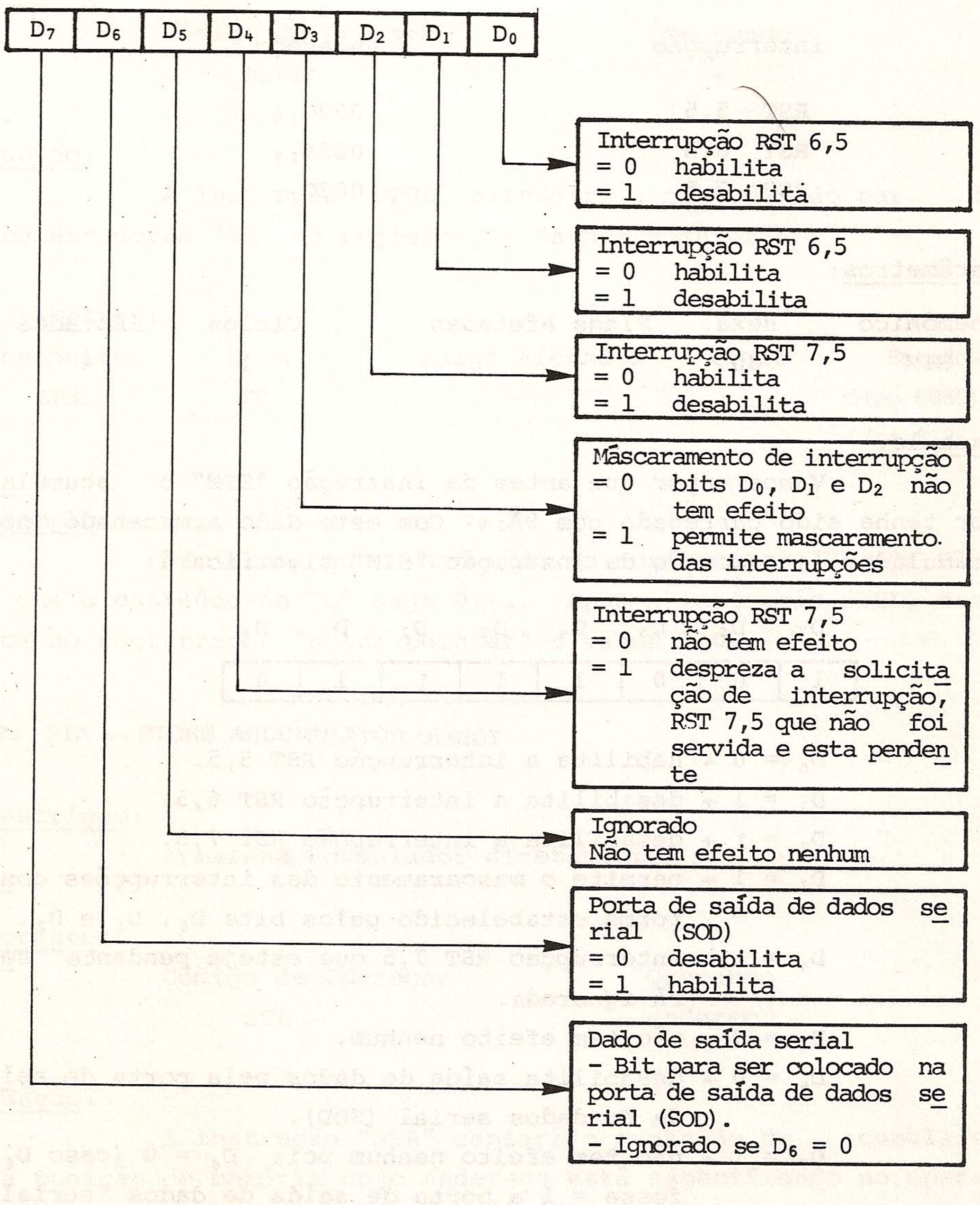
Formato:

Código de Operação	Operando
SIM	-

Função:

A instrução "SIM" fará com que o dado de 8 bits armazenado no acumulador, no instante em que a instrução seja executada, posicione a máscara de interrupções e envie um bit para a porta de saída de dados serial (SOD).

O formato do dado armazenado no acumulador e que será utilizado pela instrução será:



Obs.:

As interrupções RST 5,5; RST 6,5; RST 7,5 funcionarão de maneira similar às instruções RST 0 à RST 7, porém, o endereço para onde o programa irá saltar, dependendo da interrupção será:

interrupção	endereço
RST 5,5	$002C_{16}$
RST 6,5	0034_{16}
RST 7,5	$003C_{16}$

Parâmetros:

Mnemônico	Hexa	Flags Afetadas	Ciclos	Estados
SIM	30	-	1	4

Exemplo:

Vamos supor que antes da instrução "SIM" o acumulador tenha sido carregado com $9A_{16}$. Com este dado armazenado no acumulador, a execução da instrução "SIM" significará:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	1	1	1	1	0

$D_0 = 0 \rightarrow$ habilita a interrupção RST 5,5.

$D_1 = 1 \rightarrow$ desabilita a interrupção RST 6,5.

$D_2 = i \rightarrow$ desabilita a interrupção RST 7,5.

$D_3 = 1 \rightarrow$ permite o mascaramento das interrupções conforme estabelecido pelos bits D_0, D_1 e D_2.

$D_4 = 1 \rightarrow$ interrupção RST 7,5 que esteja pendente será ignorada.

$D_5 = 0 \rightarrow$ não tem efeito nenhum.

$D_6 = 0 \rightarrow$ desabilita saída de dados pela porta de saída de dados serial (SOD).

$D_7 = 1 \rightarrow$ não tem efeito nenhum pois $D_6 = 0$ (caso D_6 fosse = 1 a porta de saída de dados serial (SOD) iria para nível lógico "1").

71.) SPHL — MOVE HL TO SP

Descrição:

Mover par de registradores HL para o Stack Pointer.

Formato:

Código de Operação	Operando
SPHL	-

Função:

A instrução "SPHL" carregará o conteúdo do par de registradores "HL" no registrador "stack pointer".

Parâmetros:

Mnemônico	Hexa	Flags Afetadas	Ciclos	Estados
SPHL	F9	-	1	5(no 8080) 6(no 8085)

Exemplo:

Admitamos que o conteúdo do registrador "H" seja $B1_{16}$ e que o conteúdo do "L" seja 07_{16}. Após a instrução SPHL, teremos no registrador "stack pointer" o valor $B107_{16}$.

72.) STA — STORE ACCUMULATOR DIRECT

Descrição:

Armazena acumulador diretamente.

Formato:

Código de Operação	Operando
STA	endereço

Função:

A instrução "STA" copiará o conteúdo do acumulador na posição de memória cujo endereço está especificado no operando da instrução.

Parâmetros:

Mnemônico	Hexa	Flags Afetadas	Ciclos	Estados
STA	32	-	4	13

Exemplo:

Admitamos que o conteúdo do acumulador seja 27_{16}. Após a instrução STA 402H, teremos armazenado uma cópia do conteúdo do acumulador na posição de memória de endereço 402_{16}.

Memória

endereço	conteúdo
0402_{16}	27_{16}

73.) STAX — STORE ACCUMULATOR INDIRECT

Descrição:

Armazena acumulador indiretamente.

Formato:

Código de Operação	Operando
STAX	R

Função:

A instrução "STAX", armazenará o conteúdo do acumulador na posição de memória endereçada pelo par de registradores "BC" ou "DE".

Parâmetros:

Mnemônico	Hexa	Flags Afetadas	Ciclos	Estados
STAX	x2	-	2	7

Obs.:

"x" poderá assumir os valores 0 ou 1, correspondendo respectivamente a: (STAX B) ou (STAX D).

Exemplo:

Admitamos que o registrador "B" contenha o valor $0E_{16}$, o registrador "C" contenha o valor 21_{16} e o acumulador contenha AA_{16}. Assim sendo, após a instrução STAX B, teremos armazenado uma cópia do conteúdo do acumulador na posição de memória de endereço $0E21_{16}$.

Memória

endereço	conteúdo
$0E21_{16}$	AA_{16}

74.) STC — SET CARRY

Descrição:

Setar a flag carry.

Formato:

Código de Operação	Operando
STC	-

Função:

A instrução "STC" colocará flag carry em um.

Parâmetros:

Mnemônico	Hexa	Flags Afetadas	Ciclos	Estados
STC	37	C	1	4

75.) SUB — SUBTRACT

Descrição:

Subtrai.

Formato:

Código de Operação	Operando
SUB	R ou M

Função:

Esta instrução possui dois formatos:

a.) SUB R → Subtrai do conteúdo do acumulador o conteúdo de um dado registrador, e armazena o resultado no acumulador.

b.) SUB M → Subtrai do conteúdo do acumulador o conteúdo da posição de memória endereçada pelo par de registradores "HL", e armazena o resultado no acumulador.

Obs.:

Para a realização da operação de subtração será utilizada a técnica de adição do complemento dois, seguida por uma complementação da flag carry.

Parâmetros:

Mnemônico	Hexa	Flags Afetadas	Ciclos	Estados
SUB (R)	9x	Z,S,P,C,AC	1	4
SUB (M)	96	Z,S,P,C,AC	2	7

Obs.:

"x" poderá assumir os valores 0, 1, 2, 3, 4, 5 ou 7 correspondendo respectivamente a: (SUB B), (SUB C), (SUB D), (SUB E), (SUB H), (SUB L) ou (SUB A).

Exemplo: SUB B

Acumulador = 0100 0111 = 47_{16}

Registrador B = 0010 0111 = 27_{16}

O complemento 2 de 27_{16} é obtido como segue:

0010 0111 = 27_{16}

1101 1000 = $D8_{16}$ → complemento 1 de 27_{16}

1 +

1101 1001 = $D9_{16}$ → complemento 2 de 27_{16}

Acumulador = 0100 0111 = 47_{16}
Compl. 2 de 27_{16} = 1101 1001 = $D9_{16}$
Soma = 0010 0000 = 20_{16}

C = 1 complemento C = 0, portanto:
Z = 0, S = 0, P = 0, C = 0, AC = 1

76.) SUI — SUBTRACT IMMEDIATE

Descrição:

Subtração Imediata.

Formato:

Código de Operação	Operando
SUI	dado

Função:

A instrução "SUI" subtrairá do acumulador o dado imediato do segundo byte da instrução, e armazenará o resultado no acumulador.

Obs.:

Para a realização da operação de subtração será utilizada a técnica de adição do complemento dois, seguida por uma complementação da flag "carry".

Parâmetros:

Mnemônico	Hexa	Flags Afetadas	Ciclos	Estados
SUI	D6	Z,S,P,C,AC	2	7

Exemplo: SUI 32H

Acumulador = 20_{16} = 0010 0000
Dado imediato = 32_{16} = 0011 0010

O complemento 2 de 32_{16} é obtido como segue:

0011 0010 = 32_{16}
1100 1101 = CD_{16} → complemento 1 de 32_{16}
+ 1
1100 1110 = CE_{16} → complemento 2 de 32_{16}

Acumulador = 0010 0000 = 20_{16}
Compl. 2 de 32_{16} = + 1100 1110 = CE_{16}
Soma = 1110 1110 = EE_{16}

C = 0 → complemento C = 1, portanto:
Z = 0, S = 1, P = 1, C = 1, AC = 0

77.) XCHG — EXCHANGE HL WITH DE

Descrição:

Troca HL com DE.

Formato:

Código de Operação	Operando
XCHG	-

Função:

A instrução"XCHG" trocará o conteúdo dos registradores H e L com o conteúdo dos registradores D e E respectivamente.

Parâmetros:

Mnemônico	Hexa	Flags Afetadas	Ciclos	Estados
XCHG	EB	-	1	4

Exemplo: XCHG

Admitamos que o par de registradores "HL" tenha 2114_{16} e que o par de registradores "DE" tenha 3154_{16}. Após a execução da instrução XCHG, teremos no par de registradores "HL" 3154_{16} e no par de registradores "DE" 2114_{16}.

78.) XRA — EXCLUSIVE OR WITH ACCUMULATOR

Descrição:

OU EXCLUSIVO com acumulador.

Formato:

Código de Operação	Operando
XRA	R ou M

Função:

Esta instrução possui dois formatos:

a.) XRA R → Registrador - Neste caso, a operação lógica "OR EXCLUSIVE" será realizada entre o conteúdo do acumulador e de um dado registrador, e o resultado armazenado no acumulador.

b.) XRA M → Memória - Neste caso, a operação lógica "OR EXCLUSIVE" será realizada entre o conteúdo do acumulador e o conteúdo da posição de memória dado pelo par de registradores HL e o resultado armazenado no acumulador.

Parâmetros:

Mnemônico	Hexa	Flags Afetadas	Ciclos	Estados
XRA (R)	Ax	Z,S,P,C,AC	1	4
XRA (M)	AE	Z,S,P,C,AC	2	7

Obs.:

"x" poderá assumir os valores 8, 9, A, B, C, D ou F, correspondendo respectivamente a: (XRA B), (XRA C), (XRA D), (XRA E), (XRA H), (XRA L), ou (XRA A).

Exemplo: XRA A

Como exemplo, vamos citar um caso muito comum, que é a instrução XRA A, cuja descrição virá a seguir:

A	B	Z
0	0	0
0	1	1
1	0	1
1	1	0

$Z = A \oplus B$

Como podemos notar na tabela apresentada, quando A = B o resultado (Z) será zero, assim XRA A terá como resultado no acumulador somente zeros. Por isso quando desejarmos zerar o acumulador, lançaremos mão da instrução XRA A.

79.) XRI — EXCLUSIVE OR IMMEDIATE WITH ACCUMULATOR

Descrição:

OU EXCLUSIVO imediato com acumulador.

Formato:

Código de Operação	Operando
XRI	dado

Função:

A instrução "XRI" realizará a operação lógica EXCLUSIVE OR entre o conteúdo do acumulador e o conteúdo do segundo byte da instrução, armazenando o resultado no acumulador.

Parâmetros:

Mnemônico	Hexa	Flags Afetadas	Ciclos	Estados
XRI	EE	Z,S,P,C,AC	2	7

Exemplo: XRI 24H

Acumulador = 0011 0110 = 36_{16}
Dado imediato = 0010 0100 = 24_{16}
EXCLUSIVE OR = 0001 0010 = 12_{16}

80.) XTHL — EXCHANGE HL WITH TOP OF STACK

Descrição:

Troca HL com início do "STACK".

Formato:

Código de Operação	Operando
XTHL	-

Função:

A instrução "XTHL" trocará o conteúdo dos registradores H e L com o conteúdo da posição de memória endereçada pelo registrador "stack pointer".

Parâmetros:

Mnemônico	Hexa	Flags Afetadas	Ciclos	Estados
XTHL	E3	-	5	18(no 8080) 16(no 8085)

Exemplo:

Admitamos que o registrador "stack pointer" contenha 1000_{16} o registrador "H" $0E_{16}$ e o registrador "L" $0A_{16}$, e que as posições de memória 1000_{16} e 1001_{16} contenham respectivamente os valores FF_{16} e $F0_{16}$. Após a execução da instrução "XTHL", teremos:

Registradores		Posição de Memória		
H	L	1000	1001	conteúdo antes da instrução ser executada.
0E	0A	FF	F0	
F0	FF	0A	0E	conteúdo após a instrução ser executada.

Cap. V LINGUAGEM DE PROGRAMAÇÃO

V.1 LINGUAGEM DE MÁQUINA

Para que possa haver comunicação existem maneiras de uma pessoa expressar as idéias que tem em mente, de forma que possam ser recebidas e entendidas por outra. A troca de informações deve ser feita através de uma linguagem comum ao emissor e receptor.

Da mesma maneira, o microprocessador deve receber informações, interpretá-las e executá-las, porém o microprocessador e os demais circuitos que formam um sistema usam para troca de informações sinais elétricos e, desta forma, a maneira mais prática de codificar as informações é na forma binária.

Um programa a ser executado por um microprocessador deve estar armazenado na memória de forma que em cada posição deva existir uma informação codificada em oito dígitos binários. Os bus transmitem as informações codificadas na forma binária e as operações lógicas e aritméticas na unidade central de processamento são executadas de acordo com a lógica binária.

Assim sendo, as instruções, os dados e os endereços devem estar codificados na forma binária para que possam ser processados pelo microprocessador.

Esta maneira de se codificar as informações é bastante simples e rápida para uma máquina; difícil e trabalhosa, porém, para nós. Para ilustrar melhor, a figura V.1 apresenta o programa da soma de dois números decimais visto no exemplo 2 do capítulo II, codificado na forma binária o que corresponde exatamente ao conteúdo da memória, caso este programa tenha sido colocado a partir do seu endereço inicial.

O programa como o exemplo mostrado na figura V.1 está escrito numa linguagem que é compreendida pelo microprocessador. Este tipo de linguagem é chamada de linguagem de máquina.

A linguagem de máquina é uma linguagem própria de cada microprocessador e é definida pelo próprio fabricante, portanto é diferente para cada microprocessador.

ENDERECO	CONTEUDO
0000000000000000	00111110
0000000000000001	10000000
0000000000000010	11010011
0000000000000011	00011111
0000000000000100	00100001
0000000000000101	00000000
0000000000000110	00010000
0000000000000111	01111110
0000000000001000	00100011
0000000000001001	10000110
0000000000001010	00100111
0000000000001011	11010011
0000000000001100	00010111
0000000000001101	00111111
0000000000001110	00000001
0000000000001111	11011010
0000000000010000	00010011
0000000000010001	00000000
0000000000010010	10101111
0000000000010011	11010011
0000000000010100	00001111
0000000000010101	01110110

Fig. V.1 - Programa em linguagem de máquina.

O programa em linguagem de máquina é longo e confuso para o ser humano, porém, eventualmente, poder-se-ia representá-lo de uma forma um pouco mais simples, utilizando-se, ao invés da notação binária, a notação hexadecimal. Esta possibilidade deve-se, como visto no Volume I, ao fato de que cada 4 dígitos binários possuirem um correspondente direto hexadecimal o que reduz o tamanho da representação do programa. A figura V.2 mostra o mesmo programa da figura V.1, representado no código hexadecimal.

Outros tipos de sistemas de numeração, como o decimal ou octal, poderiam ser usados para a representação do programa em linguagem de máquina, sem que isto acarretasse qualquer alteração no programa. A representação no sistema hexadecimal é a mais usada e a mais fácil para ser entendida.

O programa em linguagem de máquina pode ser diretamente processado pelo microprocessador, não requerendo nenhuma decodificação ou reorganização. Ele será executado de uma maneira seqüencial e direta. A este tipo de programação dá-se o

nome de programa objeto. Assim sendo, todos programas escritos em linguagem de máquina são programas objeto, pois basta carregá-los na memória do sistema que estarão prontos para serem executados.

ENDERECO	CONTEUDO
0000	3E
0001	80
0002	D3
0003	1F
0004	21
0005	00
0006	10
0007	7E
0008	23
0009	86
000A	27
000B	D3
000C	17
000D	3F
000E	01
000F	DA
0010	13
0011	00
0012	AF
0013	D3
0014	0F
0015	76

Fig. V.2 - Programa em linguagem de máquina.

Como o próprio nome diz, esta é uma linguagem muito mais voltada para a máquina que para o ser humano. Evidentemente, os primeiros computadores eram programados em linguagem de máquina, entretanto esta linguagem possui uma série de inconvenientes, tais como: os programas são muito longos, cansativos de serem escritos e carregados na memória. Os programas escritos, desta maneira, são difíceis de serem entendidos e não ilustram as operações que o microprocessador irá executar. Também são bastante susceptíveis de erros, difíceis de serem encontrados e corrigidos.

Com a evolução dos computadores rapidamente apareceram linguagens mais apropriadas para o trabalho do ser humano, facilitando o trabalho da programação.

V.2 LINGUAGEM ASSEMBLY

A linguagem assembly ou ainda linguagem simbólica foi um primeiro passo na evolução da linguagem de programação, pois um programa em linguagem assembly possui a mesma seqüência de instruções do programa em linguagem de máquina; certos números, porém, são substituídos por símbolos que são mais ilustrativos para o programador. Além disso, cada linha do programa em linguagem assembly possui uma instrução completa, sendo então necessárias, para cada linha do programa, uma, duas ou três posições de memória, dependendo do tamanho da instrução.

Uma primeira simbologia utilizada na linguagem assembly é a substituição do código da instrução pelo mnemônico correspondente ao código de operação e os registradores do operando pelas letras identificadoras do registrador como mostrado no capítulo III.

Os mnemônicos de todas as ilustrações dos microprocessadores 8080 e 8085 foram apresentados durante a descrição dos parâmetros de cada instrução no capítulo IV.

No programa em linguagem de máquina da figura V.2, o conteúdo da primeira posição de memória é "3E" o que (ver anexo D) corresponde ao código da instrução MVI A, dado. Esta é uma instrução de dois bytes, portanto a segunda posição de memória contém o dado de oito bits (80) que é utilizado pela instrução. O conteúdo da posição seguinte da memória é o código da próxima instruçaõ, e, assim, sucessivamente. Substituindo cada código pelo mnemônico da instrução resulta no programa em linguagem assembly da figura V.3.

```
ENDERECO        INSTRUCAO

0000            MVI     A,80H
0002            OUT     1FH
0004            LXI     H,1000H
0007            MOV     A,M
0008            INX     H
0009            ADD     M
000A            DAA
000B            OUT     17H
000D            MVI     A,1H
000F            JC      0013H
0012            XRA     A
0013            OUT     0FH
0015            HLT
```

Fig. V.3 - Programa em linguagem assembly.

Com a substituição dos números por símbolos, os quais com pouco tempo de uso da linguagem assembly tornam-se familiares, já existe uma significativa melhoria para a criação e entendimento do programa. Porém, além desta, existem outras particularidades desta linguagem que facilita a tarefa da programação. Estas outras particularidades são o uso de labels e comentários no programa.

Os labels são nomes simbólicos atribuídos a constantes, posições de memória ou endereços de portas de entradas e saídas, para evitar que se use o valor numérico no programa.

As regras para criação do label são aquelas que foram apresentadas no capítulo III.

Por exemplo:

A instrução da posição de memória 000F do programa na figura V.3 é um salto condicional para a instrução contida no endereço 0013. Caso o programa fosse modificado com a inserção ou retirada de algumas instruções, este endereço seria alterado. Se ao invés de utilizar como operando da instrução de salto condicional, um numérico, fosse utilizado um label que rotulasse a instrução para onde o salto deveria ser executado, alterações do programa não exigiriam alterações no operando da instrução de salto condicional, evitando, assim, trabalho e preocupação do programador. Com a introdução de label no programa da figura V.4, resulta no programa da fig. V.4, aonde "SAIDA" é um label correspondente ao endereço de memória "0013".

```
ENDERECO                INSTRUCAO

0000                    MVI     A,80H
0002                    OUT     1FH
0004                    LXI     H,1000H
0007                    MOV     A,M
0008                    INX     H
0009                    ADD     M
000A                    DAA
000B                    OUT     17H
000D                    MVI     A,1H
000F                    JC      SAIDA
0012                    XRA     A
0013            SAIDA:  OUT     0FH
0015                    HLT
```

Fig. V.4 - Programa em linguagem assembly.

Os comentários dentro de um programa em linguagem assembly são tudo que existe numa linha do programa após um sinal de ponto e vírgula e servem apenas para ilustrar o programa, não sendo utilizado em nada pelo microprocessador. Eventualmente uma linha inteira do programa pode ser utilizada para comentário desde que seja precedida por um ponto e vírgula. Por exemplo, após a instrução da posição de memória 0012, do programa da fig. V.4 poderia ser colocado o seguinte comentário "; zera acumulador", pois a instrução OR-EXCLUSIVO de acumulador com ele mesmo foi utilizado para zerar o conteúdo do acumulador. Os comentários num programa não são obrigatórios, porém, essenciais para compreensão, correção e modificação do mesmo.

Desta forma, basicamente cada linha de um programa em linguagem assembly é constituída por quatro campos distintos: *(LABEL)*, *(CÓDIGO DE OPERAÇÃO)*, *(OPERANDO)* e *(COMENTÁRIO)*.

Os microprocessadores possuem características próprias a cada um deles, com diferente estrutura interna, diferente conjunto de registradores e também conjunto de instrução diferente de um para outro. Desta forma, a linguagem assembly é uma linguagem característica para cada microprocessador, não sendo possível um programa que foi originalmente escrito em linguagem assembly para um microprocessador 8085, ser processado por um microprocessador 6800. Desta forma, a linguagem assembly é uma linguagem específica e diferente para cada microprocessador. Para um mesmo microprocessador podem também existir variações das características da linguagem assembly, podendo serem estabelecidos pontos diferentes. Assim não há uma linguagem universal, nem para um mesmo microprocessador, entretanto a estrutura básica permanecerá a mesma.

Para os microprocessadores 8080 e 8085 o mais usado e também o que é estabelecido pela própria Intel, que é a idealizadora destes microprocessadores, é:

Label:- conforme o definido no capítulo III, isto é, constituído de um a seis caracteres alfanuméricos, sendo que o primeiro é sempre um caracter numérico e quando precede uma instrução deve ser seguido por dois pontos.

Código de Operação:- são válidos para o código de operação das instruções todos os mnemônicos apresentados no capítu

lo IV e mais os mnemônicos das pseudo-instruções que serão analizadas mais adiante neste capítulo.

Operando:- conforme o definido no capítulo III, isto é, pode ser especificado das seguintes maneiras:

- letra identificadora do registrador.
- dado no sistema de numeração hexadecimal, decimal, octal ou binário.
- constantes no código ASCII.
- label.
- contador de localização.
- expressão.

Comentário:- de formato livre podendo ser constituído por qualquer caracter do código ASCII, devendo, porém, sempre ser precedido por ponto e vírgula.

Os seguintes delimitadores são usados para a programação em linguagem assembly:

branco	- um ou mais espaços em branco	- separador dos campos
:	- dois pontos	- final de campo de label
;	- ponto e vírgula	- inicio do campo de comentário
,	- vírgula	- separador de operandos no campo de operando.

Além destas características, aqui expostas, a linguagem assembly possui ainda umas instruções especiais ou pseudo-instruções que são usadas apenas para a estruturação do programa. Estas instruções especiais não são traduzidas para o código de máquina, por não pertencerem ao conjunto de instruções do microprocessador, elas têm apenas funções especiais no programa como definir símbolos, estabelecer o endereço inicial do programa, reservar área de memória, etc, não sendo, portanto, processadas.

As pseudo-instruções mais usadas são:

1.) ORG — ORIGIN

Formato:

Código de Operação	Operando
ORG	endereço

Função:

O endereço especificado no operando é o endereço inicial da memória onde devem ser armazenadas as instruções que estão após a instrução "ORG".

Exemplo: ORG 1000H

Se no início de um programa for colocada a pseudo-instrução "ORG 1000H", o programa objeto será carregado na memória, a partir do endereço 1000H.

2.) EQU — EQUATE

Formato:

Label	Código de Operação	Operando
nome	EQU	endereço ou dado

Função:

O nome especificado no campo de label é definido como igual ao valor do dado ou endereço especificado no operando. Este nome não poderá mais ser redefinido no mesmo programa.

O nome do label não pode ser seguido por dois pontos, como é normalmente requerido para as demais instruções.

Exemplo: CONTR EQU 1FH

```
CONTR   EQU     1FH
        MVI     A,80H
        OUT     CONTR
```

tem o mesmo efeito que:

```
        MVI     A,80H
        OUT     1FH
```

3.) SET — SET

Formato:

Label	Código de Operação	Operando
nome	SET	endereço ou dado

Função:

A mesma que a pseuda-instrução "EQU", porém o nome pode ser redefinido no mesmo programa.

Exemplo:

```
MODO    SET     80H
        MVI     A,MODO
```

tem o mesmo efeito que:

```
        MVI     A,80H
```

4.) DB — DEFINE BYTE

Formato:

Código de Operação	Operando
DB	dado

Função:

Os oito bits do operando são inseridos diretamente no programa objeto.

Exemplo:

```
        MOV     A,M
        DB      23H
        ADD     M
```

tem o mesmo efeito que:

```
        MOV     A,M
        INX     H
        ADD     M
```

5.) DS — DEFINE A BLOCK OF STORAGE

Formato:

Código de Operação	Operando
DS	dado

Função:

Reserva uma área de memória de comprimento igual ao dado especificado no operando.

Exemplo: DS 6H

```
ORG     1000H
EJ
DS      6H
IN      3BH
```

o programa objeto terá o seguinte formato:

```
1000    FB
1001    FF
1002    FF
1003    FF
1004    FF
1005    FF
1006    FF
1007    DB
1008    3B
```

6.) END — END

Formato:

Código de Operação	Operando
END	-

Função:

Indica o final do programa.

7.) SPC — SPACE

Formato:

Código de Operação	Operando
SPC	-

Função:

Reserva uma linha em branco no programa em linguagem assembler.

O programa codificado em linguagem assembler não pode ser diretamente processado pelo microprocessador, deve ser traduzido para a linguagem de máquina. Isto pode ser feito manualmente, com uso de tabelas ou utilizando o próprio computador, como será explicado mais adiante neste capítulo.

O programa em linguagem assembler ou em qualquer outra linguagem diferente da linguagem de máquina é chamado de programa fonte. Portanto programa fonte é qualquer programa escrito em linguagem diferente da linguagem de máquina.

O programa fonte é um programa de sintaxe inteligível para o ser humano, porém, não pode ser diretamente processado pelo microprocessador, devendo, primeiramente, ser traduzido para ser convertido em um programa objeto. A fig. V.5 apresenta o mesmo programa de soma de dois números até agora analisado neste capítulo, melhor estruturado em linguagem assembly. Foram acrescentadas algumas linhas de comentários que tornam o programa mais claro e mostram o seu histórico. A numeração seqüencial que foi introduzida é uma numeração das linhas e não corresponde ao endereço de memória. A numeração de valores saltados de dez em dez é para quando houver necessidade de introdução de linhas adicionais no programa, poder-se utilizar uma numeração intermediária, mantendo-se a seqüência.

```
010     ;*************************************************************
020     ;*****  PROGRAMA    : SOMA DE DOIS NUMEROS                  *****
030     ;*****  PROGRAMADOR : SANDRINI                              *****
040     ;*****  DATA        : 7/AGO/81                              *****
050     ;*****  REVISAO     : A                                     *****
060     ;*****  SISTEMA     : 8085 PROCESSOR                        *****
070     ;*****  LINGUAGEM   : ASSEMBLY 8085                         *****
080     ;*************************************************************
090             SPC
100     ;ENTRADAS : OS FATORES A SEREM SOMADOS DEVEM ESTAR NAS
110     ;           POSICOES DE MEMORIA  DE ENDERECO   1000H E
120     ;           1001H.
130     ;SAIDAS   : O RESULTADO DA SOMA SERA COLOCADO:
140     ;               CENTENA - PORTA B  DA 8255
150     ;               DEZENA  - PORTA CH DA 8255
160     ;               UNIDADE - PORTA CL DA 8255
170             SPC
180     ;*****  ENDERECO DAS PORTAS                                 *****
190             SPC
200     CTR55   EQU     1FH
210     PRTA    EQU     7H
220     PRTB    EQU     0FH
230     PRTC    EQU     17H
240             SPC
250     ;*****  AREAS DE TRABALHO                                   *****
260             SPC
270     FAT1    EQU     1000H
280     FAT2    EQU     1001H
290             SPC
300             ORG     0000H   ;ENDERECO INICIAL DO PROGRAMA
310     INIC:   MVI     A,80    ;MODO ZERO, PORTA A,B,C SAIDA
320             OUT     CTR55
330             LXI     H,FAT1  ;ENDERECO DO FATOR 1 EM HL
340             MOV     A,M     ;COLOCA FATOR 1 NO ACUMULADOR
350             INX     H       ;ENDERECO DO FATOR 2 EM HL
360             ADD     M       ;SOMA FATOR 1 A FATOR 2
370             DAA             ;AJUSTE DECIMAL
380             OUT     PRTC
390             MVI     A,1H    ;ACUMULADOR IGUAL A 1
400             JC      SAIDA   ;SALTA SE HOUVE CARRY NO  AJUSTE
410                             ;DECIMAL
420             XRA     A       ;ZERA ACUMULADOR
430     SAIDA:  OUT     PRTB
440             HLT             ;PARE
450             END
```

Fig. V.5 - Programa em linguagem assembly.

V.3 LINGUAGEM DE ALTO NÍVEL

A linguagem assembly já facilitou muito o trabalho de programação, porém, com o passar dos tempos, surgiu a necessidade de linguagens melhores. Essas novas linguagens são mais voltadas para o problema a ser resolvido, desprendendo dos aspectos inerentes ao equipamento que será utilizado. Apresentam uma estrutura mais técnica, formada por instruções que são decodificadas em diversas instruções de máquinas; são, ainda, linguagens universais, não dependendo do repertório de instruções estabelecido pelos fabricantes.

Com linguagem deste tipo existiriam as seguintes vantagens: os programas seriam escritos de maneira mais concisa e mais rápida; para programar não existiria a necessidade de conhecerem-se as características do sistema nem a linguagem de máquina; os programas nestas linguagens poderiam ser transferidos de um computador para outro; além disso, a linguagem estaria mais próxima da utilizada para a formulação do problema.

Desta maneira, surgiram as linguagens de alto nível ou linguagens automáticas que foram as responsáveis pela crescente difusão dos computadores provocando a sua utilização em massa, pois o usuário não tinha mais necessidade de conhecer a fundo a máquina e nem sua linguagem assembly. Bastariam conhecimentos matemáticos ou específicos de sua área e o conhecimento dos detalhes da linguagem de alto nível. Outra razão é que o usuário poderia usar programas prontos, desenvolvidos por outros, e que independiam do equipamento que possuisse, crescendo rapidamente o número de pacotes de programas disponíveis.

O número de linguagens de alto nível que surgiram em um espaço de vinte anos foi muito grande, chegando a uma centena de linguagens diferentes e, hoje em dia, já existem alguns milhares, cada uma delas voltadas a atender uma área de atividade. Cada fabricante e cada universidade procuram desenvolver uma linguagem própria. Existem comissões internacionais tentando padronizar linguagens de alto nível para cada área específica, para que sejam utilizadas pelos computadores de todo o mundo.

Algumas linguagens tendem a ser padronizadas ou mes

mo não sendo, tornam-se as mais usadas e internacionalmente conhecidas.

As principais linguagens de alto nível são: Fortran (Formula Translation Language), que foi uma das primeiras linguagens de alto nível a surgir e é utilizada para finalidades científicas em problemas que possam ser formulados matematicamente.

Basic (Beginner's All-Purpose Symbolic Instruction Code) para principiantes e uso conversacional com o computador.

Cobol (Common Business Oriented Language) para uso comercial.

Algol (Algorithimic Language) finalidades científicas em problemas que envolvam algoritmos matemáticos.

Pascal para aplicação de controle.

Não iremos fazer um estudo detalhado de nenhuma linguagem de alto nível, pois este estudo é extenso e demorado. Estas linguagens destinam-se ao atendimento de problemas técnico-científicos e são utilizadas em equipamentos com grande quantidade de hardware e software de suporte. Nosso objetivo é analizar e aprofundarmo-nos em aspectos internos de um sistema.

V.4 PROGRAMAS TRADUTORES

Um programa fonte em linguagem assembler é possível de ser convertido manualmente, em um programa objeto, o que não ocorre com um programa em linguagem de alto nível. Para este trabalho árduo e sistemático utiliza-se o próprio computador com programas especialmente desenvolvidos para a execução desta tarefa.

Os programas que traduzem um programa para a linguagem de máquina são chamados de programas tradutores.

Quando um programa é escrito em linguagem diferente da linguagem de máquina, é carregado na memória do computador por meio de um dispositivo de entrada e saída qualquer, por exemplo, um teclado, um terminal de vídeo, uma leitora de cartões, etc. Cada caracter deste programa é convertido para o código ASCII e armazenado na memória do computador.

Após ter sido carregado o programa fonte, é feito o processamento do programa tradutor. Este irá ler cada linha

do programa fonte na memória e transformá-lo para a linguagem de máquina; podendo, então, executar estas instruções, ou gerar uma listagem completa do programa objeto resultante ou, ainda, armazená-lo na memória, o que dependerá do tipo de programa tradutor utilizado.

Os programas tradutores mais comuns são:

- programa assembler.
- programa compilador.
- programa interpretador.

O programa assembler é um programa voltado a converter um programa fonte escrito em linguagem assembly para um programa objeto em linguagem de máquina, conforme o esquema da figura V.6.

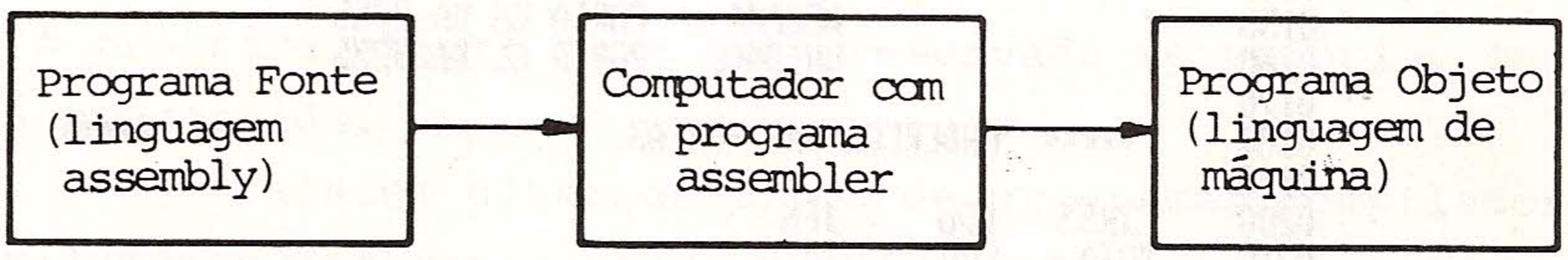


Fig. V.6 - Programa assembler.

Existem diversos programas assemblers, pois como foi visto cada microprocessador possui uma linguagem assembly diferente, e também para um mesmo microprocessador podem existir programas assemblers diferentes, dependendo de quem desenvolveu este programa. Por esta razão deve seguir-se a estrutura estebelecida pelo programa que será utilizado para que ele possa fazer a tradução corretamente. A sintaxe ou estrutura que foi apresentada da linguagem assembly com o formato dos campos, tipo de informação em cada campo, delimitadores permitidos e conjunto de pseudo-instruções, normalmente é o que aceita a maioria dos programas assemblers dos microprocessadores 8080 e 8085. Entretanto alguns equipamentos podem exigir pequenas alterações.

Normalmente, o programa assembler traduz o programa fonte; armazena o programa objeto correspondente na memória; fornece uma listagem do programa fonte resultante, do programa objeto equivalente, uma tabela de símbolos e uma listagem dos erros

de sintaxe.

A fig. V.7 mostra um exemplo da listagem gerada pela tradução do programa mostrado na fig. V.5.

```
 PROGRAMA    !                          PROGRAMA                              !
  OBJETO     !                           FONTE                                !
ENDERE.!CONTEU.! LINHA ! LABEL ! C. OP.!OPERAN.! COMENTARIOS                   !

                0010   ;*************************************************************
                0020   ;*****  PROGRAMA    : SOMA DE DOIS NUMEROS            *****
                0030   ;*****  PROGRAMADOR : SANDRINI                         *****
                0040   ;*****  DATA        : 7/AGO/81                        *****
                0050   ;*****  REVISAO     : A                               *****
                0060   ;*****  SISTEMA     : 8085 PROCESSOR                  *****
                0070   ;*****  LINGUAGEM   : ASSEMBLY 8085                    *****
                0080   ;*************************************************************
                0090
                0100   ;ENTRADAS : OS FATORES A SEREM SOMADOS DEVEM  ESTAR  NAS
                0110   ;           POSICOES  DE MEMORIA  DE  ENDERECO  1000H  E
                0120   ;           1001H.
                0130   ;SAIDAS   : O RESULTADO DA SOMA SERA COLOCADO :
                0140   ;                CENTENA - PORTA B  DA 8255
                0150   ;                DEZENA  - PORTA CH DA 8255
                0160   ;                UNIDADE - PORTA CL DA 8255
                0170
                0180   ;*****  ENDERECO DAS PORTAS                            *****
                0190
                0200   CTR55   EQU     1FH
                0210   PRTA    EQU     7H
                0220   PRTB    EQU     0FH
                0230   PRTC    EQU     17H
                0240
                0250   ;*****  AREAS DE TRABALHO                             *****
                0260
                0270   FAT1    EQU     1000H
                0280   FAT2    EQU     1001H
                0290
                0300           ORG     0000H   ;ENDERECO INICIAL DE PROGRAMA
0000    3E80    0310   INIC:   MVI     A,80H   ;MODO ZERO, PORTA A,B,C SAIDA
0002    D31F    0320           OUT     CTR55
0004    210010  0330           LXI     H,FAT1  ;ENDERECO DO FATOR 1 EM HL
0007    7E      0340           MOV     A,M     ;COLOCA FATOR 1 NO ACUMULADOR
0008    23      0350           INX     H       ;ENDERECO DO FATOR 2 EM HL
0009    86      0360           ADD     M       ;SOMA FATOR 1 A FATOR 2
000A    27      0370           DAA             ;AJUSTE DECIMAL
000B    D317    0380           OUT     PRTC
000D    3E01    0390           MVI     A,1H    ;ACUMULADOR IGUAL A 1
000F    DA1300  0400           JC      SAIDA   ;SALTA SE HOUVE CARRY NO AJUSTE
                0410                           ;DECIMAL
0012    AF      0420           XRA     A       ;ZERA ACUMULADOR
0013    D30F    0430   SAIDA:  OUT     PRTB
0015    76      0440           HLT             ;PARE
                0450           END

SYMBOL TABLE

CTR55   001F    FAT1    1000    FAT2    1001    INIC    0000    PRTA    0007
PRTB    000F    PRTC    0017    SAIDA   0013

ERRORS = 0
```

Fig. V.7 - Listagem gerada pelo programa assembler.

Programa compilador é a designação dada ao programa que traduz programa fonte, escrito em linguagem de alto nível, para programa objeto, em linguagem de máquina, semelhante a função do programa assembler conforme esquema da fig. V.8.

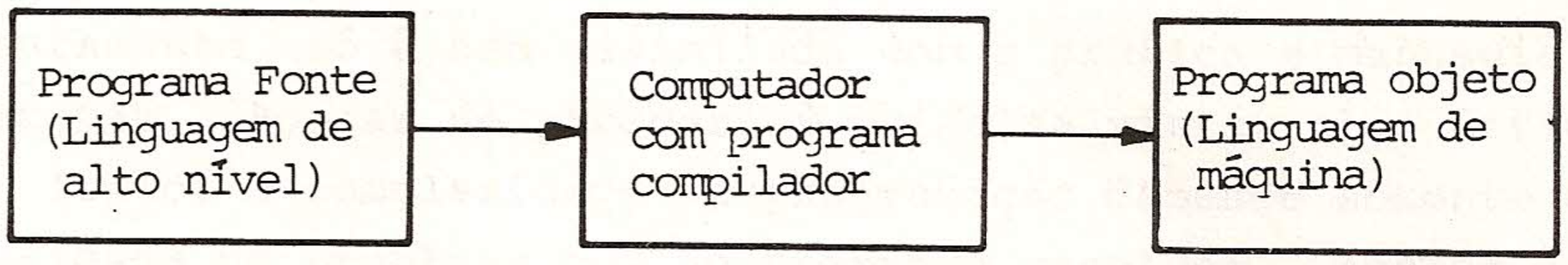


Fig. V.8 - Programa compilador.

O que normalmente o programa compilador faz é armazenar o programa objeto numa área reservada de memória, onde ele será processado.

Existem diversos tipos de programas compiladores; os compiladores Fortran, os compiladores Basic etc, enfim, um compilador para cada linguagem. Cada linguagem possui uma estrutura e uma sintaxe universal, cabendo ao programa compilador de cada computador gerar o programa em linguagem de máquina, característico deste computador.

O programa interpretador também é voltado para linguagem de alto nível. Este não produz um programa objeto e, sim, toma uma instrução do programa fonte, traduz e a executa imediatamente. Após isto, retira uma nova instrução do programa fonte, traduz e executa, e assim por diante, conforme o esquema da fig.V.9.

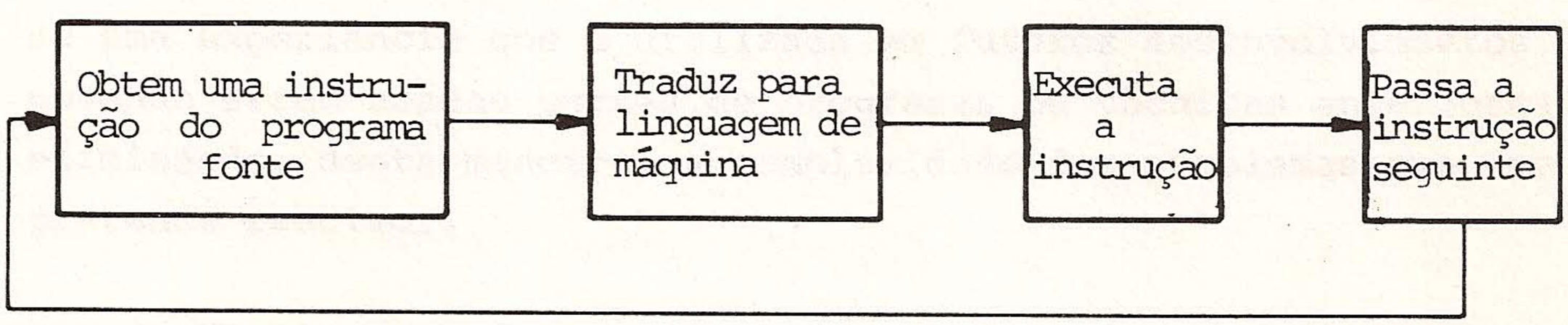


Fig. V.9 - Programa interpretador.

Desta maneira, o programa fonte é executado imediatamente, sem que seja necessário esperar-se pelo programa objeto e depois executá-lo. Porém, o programa fonte é executado instrução por instrução e cada instrução antes de ser executada tem que ser traduzida, aumentando o tempo de processamento.

Cap. VI PRÁTICA DE PROGRAMAÇÃO

VI.1 - APRESENTAÇÃO

A programação do microprocessador é uma atividade que, seguramente, só é bem assimilada com a prática e manuseio de sistemas. Apesar de parecer, à primeira vista, algo difícil de ser feito, a complexidade da programação depende somente da complexidade do problema que se pretende resolver. Apenas os primeiros passos da programação são suficientes para compreender o mecanismo de desenvolvimento de programas, percebendo-se que não é necessário que seja decorada uma série de coisas, bastando somente usar um raciocínio lógico e uma forma de expressar mais técnica.

Para facilitar as atividades mais trabalhosas no desenvolvimento de um programa, existem equipamentos especiais, que são os sistemas de desenvolvimento. Estes sistemas são basicamente constituídos de um teclado para entrada de dados, um visor para acompanhamento das informações e dispositivos para armazenamento de dados de grande capacidade, tais como fita ou disco magnético. Eles possuem também programas especiais para auxiliar o desenvolvimento dos programas, como os programas tradutores, programas para testes, possibilitando a execução de pequenos blocos do programa ou mesmo instrução por instrução e, até mesmo, programas para simular o sistema que estamos projetando. Podem, eventualmente, incluir um programador de EPROM para a gravação direta do programa final em linguagem de máquina na memória que será utilizada em nosso sistema.

Com o desenvolvimento de alguns programas, adquire-se uma experiência que é utilizada em futuros desenvolvimentos, podendo serem usadas partes de programas ou técnicas anteriores eliminando, desta maneira, a complexidade dos problemas que se pretende resolver.

V.2 LOOP

O loop é uma das técnicas mais usadas em programação, e quase todo programa, por mais simples que seja, utiliza loops

em sua extrutura. O loop é um grupo de instruções que são executadas mais de uma vez, sem que seja necessário reescrevê-las. Basta fazer com que o fluxo do programa volte alguns passos, executando novamente instruções que já foram processadas.

Um primeiro exemplo de loop pode ser visto no seguinte exemplo clássico:

- Zerar o bloco de memória RAM de endereço inicial 4200_{16} e comprimento 50_{16}.

Solução:

Este programa não requer nenhum algoritmo especial, é necessário somente carregar com zero, as 80_{10} posições de memória (50 em hexadecimal) a partir de endereço especificado.

A fig. VI.1 apresenta o fluxograma para este programa.

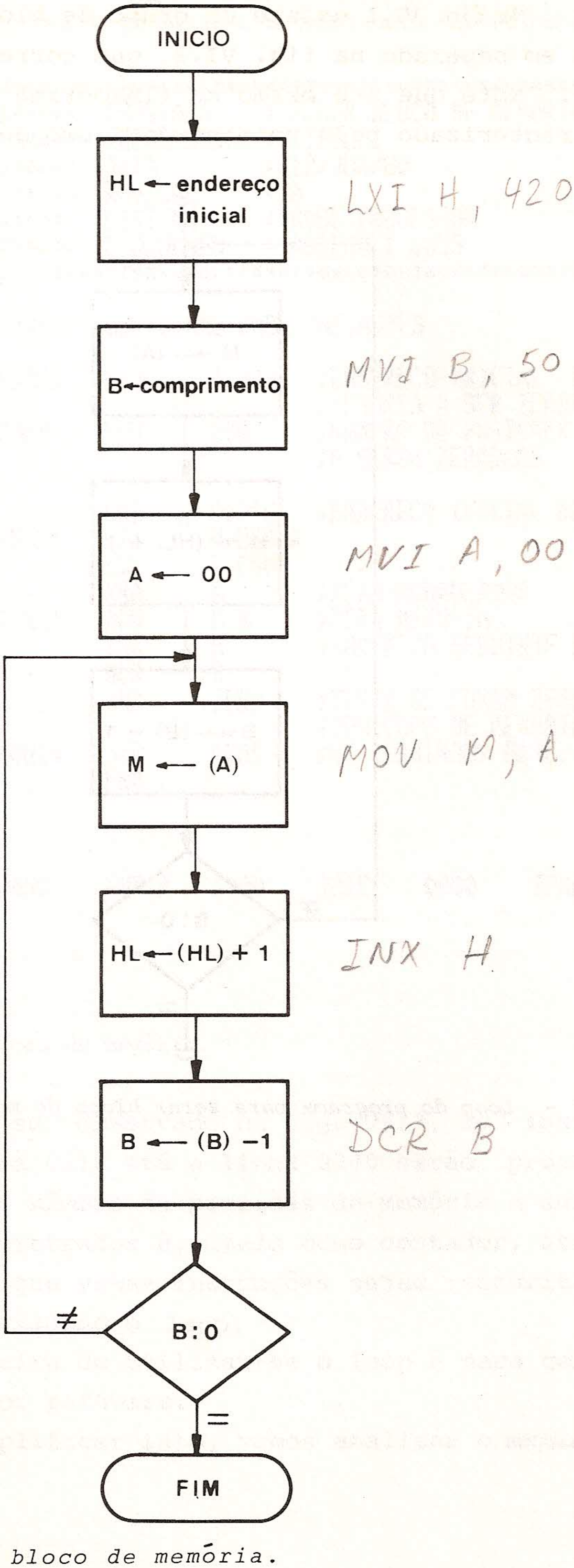


Fig. VI.1 - Zerar bloco de memória.

Na fig. VI.1 existe um grupo de blocos que estão representados em separado na fig. VI.2, que corresponde ao loop do programa. Note que até mesmo no fluxograma de programa o loop é bem caracterizado pelo retorno de fluxo de processamento.

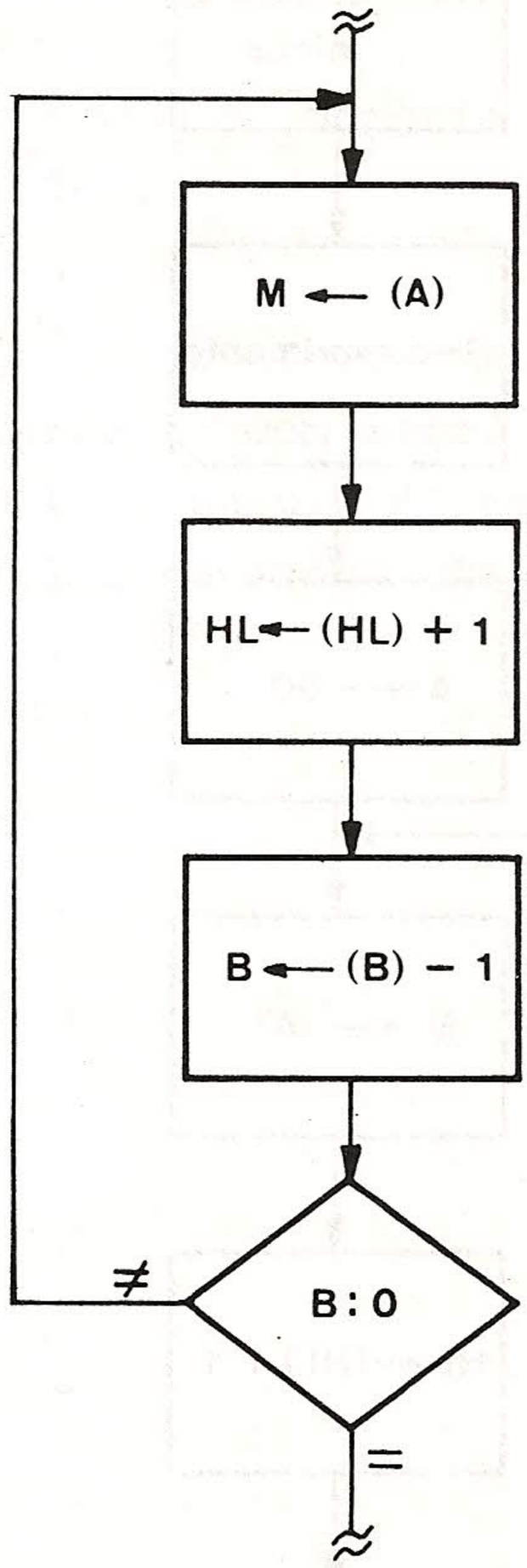


Fig.VI.2 - Loop do programa para zerar bloco de memória.

A listagem deste programa é mostrada na figura VI.3.

```
0010    ;*************************************************************
0020    ;*****  PROGRAMA    : ZERAR BLOCO DE MEMORIA          *****
0030    ;*****  PROGRAMADOR : CIPELLI                         *****
0040    ;*****  DATA        : 12/JUN/80                       *****
0050    ;*****  REVISAO     : A                               *****
0060    ;*****  SISTEMA     : 8085 PROCESSOR                  *****
0070    ;*****  LINGUAGEM   : ASSEMBLY 8085                   *****
0080    ;*************************************************************
0090
0100    ;*****  DEFINICAO DAS CONSTANTES                      *****
0110
0120    ENDER   EQU     4200H   ;ENDERECO INICIAL DO BLOCO DE
0130                            ;MEMORIA A SER ZERADO
0140    COMP    EQU     80D     ;NUMERO DE POSICOES DE MEMORIA
0150                            ;A SEREM ZERADAAS
0160
0170            ORG     0000H   ;ENDERECO INICIAL DO PROGRAMA
0000    210042  0180    INIC:   LXI     H,ENDER
0003    0650    0190            MVI     B,COMP
0005    AF      0200            XRA     A       ;ZERA ACUMULADOR
0006    77      0210    ZERA:   MOV     M,A     ;ZERA MEMORIA
0007    23      0220            INX     H       ;ENDERECO SEGUINTE DE MEMORIA
0008    05      0230            DCR     B
0009    C20600  0240            JNZ     ZERA    ;TESTA SE FORAM ZERADAS TODAS AS
                0250                            ;POSICOES DE MEMORIA
000C    C30C00  0260    AQUI:   JMP     AQUI    ;SE TERMINOU, PARE.
                0270            END

SYMBOL TABLE

AQUI    000C    COMP    0050    ENDER   4200    INIC    0000    ZERA    0006

ERRORS = 0
```

Fig. VI.3 - Zerar bloco de memória.

Como pode ser observado na fig. VI.3, as instruções contidas desde a linha 0210 até a linha 0240 serão processadas tantas vezes quanto o número de posições de memória a serem zeradas, (até que o registrador B, usado como contador, atinja o valor zero) evitando que estas instruções sejam reescritas numerosas vezes, caracterizando o loop.

Outra maneira de utilizar-se o loop é para gerar um intervalo de tempo por software.

Para exemplificar isto, vamos analisar o seguinte exemplo:

-. gerar no bit zero da porta A do 8255 uma onda quadrada de freqüência igual a 50Hz.

Solução:

O sinal é como o mostrado na fig. VI.4.

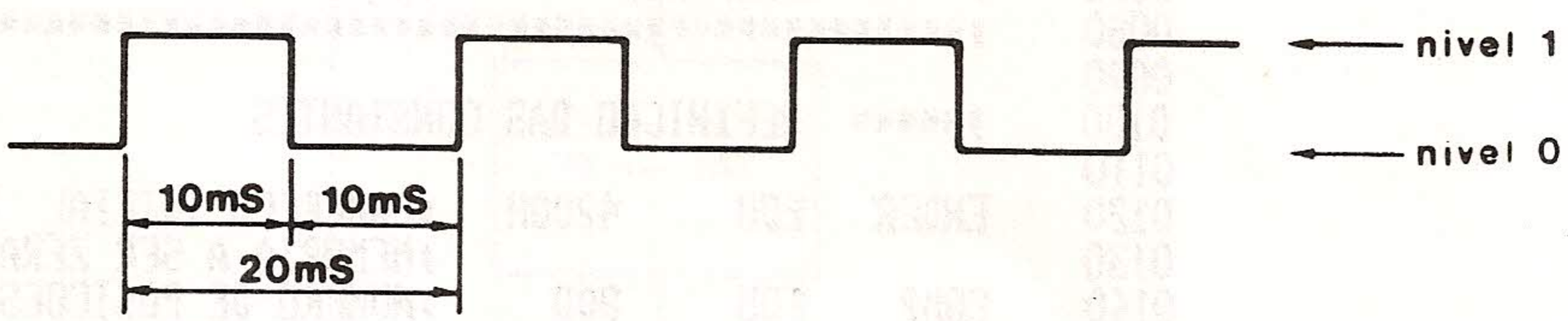


Fig. VI.4 - Onda quadrada de 50Hz.

A freqüência desejada é de 50Hz, o período correspondente é de 20ms. Portanto a solução do problema é enviar o nível lógico "1" à porta, esperar um tempo de 10ms, enviar o nível lógico "0" à porta, esperar o mesmo tempo e repetir esta seqüência indefinidamente.

Para obter o tempo de 10ms, vamos fazer uma rotina que carrega um registrador com um valor inicial e executa um loop decrementando este registrador até que seu conteúdo seja zero. Quando isto acontece o programa sai fora do loop.

Desta maneira, o número de estados das instruções do loop, vezes o número de vezes em que o loop é executado e vezes o tempo de cada estado, deverá ser igual a 10ms.

No caso do microprocessador 8085 com um cristal de 6,144MHz, o sinal de clock é de 3,072MHz e cada estado demora $325,52 \times 10^{-9}$ ou 325,52ns.

Numa primeira tentativa resulta o fluxograma da figura VI.5.

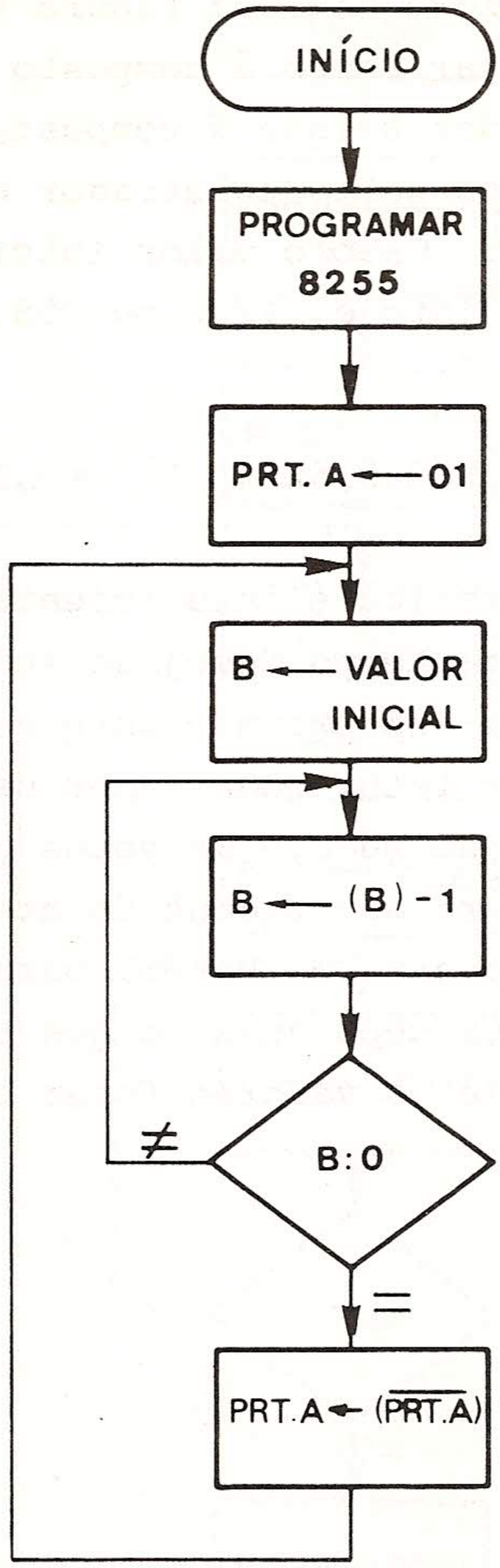


Fig. VI.5 - Gerador de sinal.

Analisando o fluxograma da figura VI.5 podemos concluir que o loop para gastar tempo é composto de uma instrução de decrementó do registrador B (que é composto de 4 estados) e uma instrução de salto caso este registrador não seja zero (que é composta de 10 estados). Caso o valor inicial do registrador B seja o máximo possível, isto é, FF_{16} ou 255_{10}, o tempo gasto para executar esta rotina é:

$$255 \times (4 + 10) \times 325{,}52 \times 10^{-9} = 1{,}17 \times 10^{-3}s = 1{,}16ms$$

Portanto esta rotina é insuficiente para gerar o tempo desejado. Para que este tempo desejado seja atingido, esta rotina deveria ser repetida aproximadamente nove vezes.

Para solucionar isto, poderiamos usar um par de registradores para contagem do número de vezes que o loop é executado. Neste caso ele poderá ser executado até $FFFF_{16}$ ou 65.535_{10} vezes, o que já seria suficiente. Porém, para exercitar, vamos utilizar dois registradores separados, o que resultará no fluxograma da fig. VI.6, aonde os valores foram obtidos após algumas tentativas.

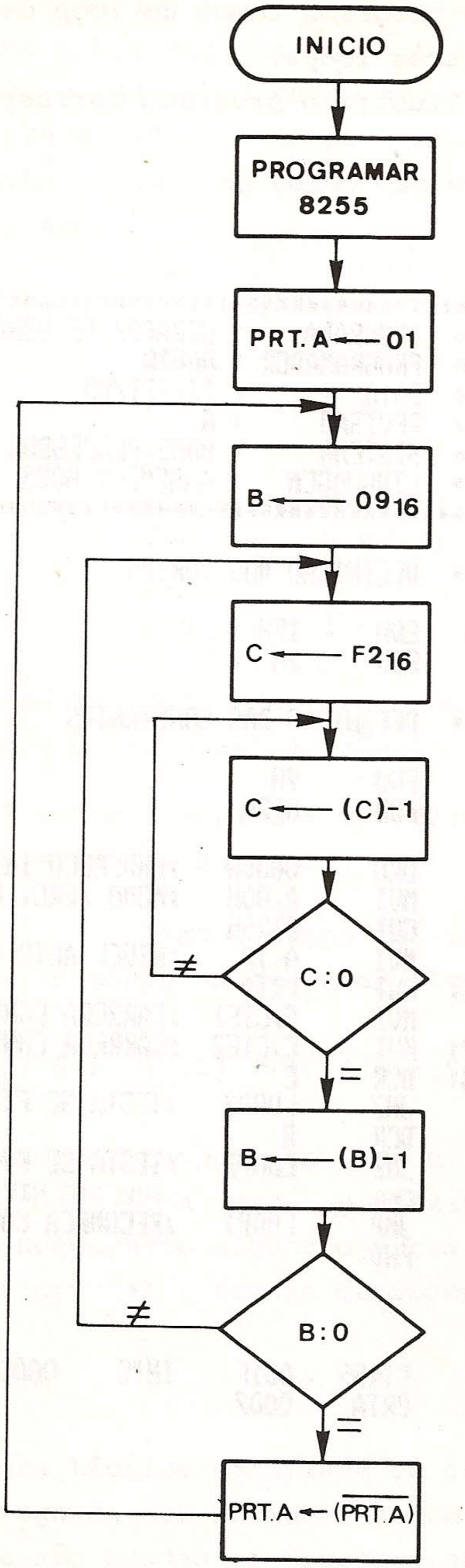


Fig. VI. 6 - Gerador de sinal.

Note que neste programa temos um loop dentro de um loop maior e um total de três loops.

A figura VI.7 ilustra o programa correspondente ao fluxograma da figura VI.6.

```
0010    ;*************************************************************
0020    ;*****  PROGRAMA    : GERADOR DE SINAL                    *****
0030    ;*****  PROGRAMADOR : DAVID                               *****
0040    ;*****  DATA        : 12/SET/80                           *****
0050    ;*****  REVISAO     : A                                   *****
0060    ;*****  SISTEMA     : 8085 PROCESSOR                      *****
0070    ;*****  LINGUAGEM   : ASSEMBLY 8085                       *****
0080    ;*************************************************************
0090
0100    ;*****  DEFINICAO DAS PORTAS                              *****
0110
0120    CTR55   EQU     1FH
0130    PRTA    EQU     7H
0140
0150    ;*****  DEFINICAO DAS CONSTANTES                          *****
0160
0170    CTE1    EQU     9H
0180    CTE2    EQU     0F2H
0190
0200            ORG     0000H   ;ENDERECO INICIAL DO PROGRAMA
0000    3E80    0210    INIC:   MVI     A,80H   ;MODO ZERO, PORTA A,B,C SAIDA
0002    D31F    0220            OUT     CTR55
0004    3E01    0230            MVI     A,1H    ;NIVEL ALTO PARA A SAIDA
0006    D307    0240    LOOP1:  OUT     PRTA
0008    0609    0250            MVI     B,CTE1  ;CARREGA CONST. 1 EM B
000A    0EF2    0260    LOOP2:  MVI     C,CTE2  ;CARREGA CONST. 2 EM C
000C    0D      0270    LOOP3:  DCR     C
000D    C20C00  0280            JNZ     LOOP3   ;TESTA SE FIM DA CONTAGEM EM C
0010    05      0290            DCR     B
0011    C20A00  0300            JNZ     LOOP2   ;TESTA SE FIM DA CONTAGEM EM B
0014    2F      0310            CMA             ;INVERTE NIVEL DE SAIDA
0015    C30600  0320            JMP     LOOP1   ;RECOMECA CONTAGEM DE TEMPO
                0330            END

SYMBOL TABLE

CTE1    0009    CTE2    00F2    CTR55   001F    INIC    0000    LOOP1   0006
LOOP2   000A    LOOP3   000C    PRTA    0007

ERRORS = 0
```

Fig. VI.7 - Gerador de sinal.

O processamento deste programa resultará que por determinado tempo o bit zero da porta A da 8255 terá nível lógico "1" (linhas 230 e 240) e no tempo seguinte, o nível lógico "0" (linhas 310, 320 e 240). O tempo em que cada nível permanecerá estável na porta é igual ao tempo que o processador gasta para executar as linhas.

240 - 1 vez
250 - 1 vez
260 - 9 vezes
270 - 9 x 242 = 2178 vezes
280 - 9 x 242 = 2178 vezes
290 - 9 vezes
300 - 9 vezes
310 - 1 vez
320 - 1 vez

multiplicando-se o número de vezes que cada instrução é executada pelo número de estados de cada instrução, resulta em:

1 x 10 + 1 x 7 + 9 x 7 + 2178 x 4 + 2178 x 10 + 9 x 4 + 9 x 10 + 1 x 7 + 1 x 10 = 30.715 estados.

Como o tempo de cada estado é igual a $325,25 \times 10^{-9}$s, o tempo em que cada nível permanecerá estável é:

$$30.715 \times 325,25 \times 10^{-9} = 9,99 \times 10^{-3} \simeq 10ms$$

É sempre conveniente para um programa como este fazer-se uma checagem final com, por exemplo, um osciloscópio, pois pode ser necessário alguns ajustes das constantes devido às tolerâncias do cristal e demais componentes.

VI.3 SUB-ROTINA

Outra técnica utilizada em programação, também muito útil, é o emprego de sub-rotina. A sub-rotina é um grupo de instruções que são usadas em diversas partes do programa e para que não sejam reescritas diversas vezes no programa são armazenadas em uma área de memória, separada do programa. Cada vez

que o programa necessita executar estas instruções, salta para o endereço em que elas estão armazenadas. Após o processamento destas instruções o fluxo retorna para a instrução seguinte a aquela que provocou o salto, prosseguindo normal.

De uma maneira simples o processamento de um programa com uma sub-rotina é ilustrado no gráfico da figura VI.8.

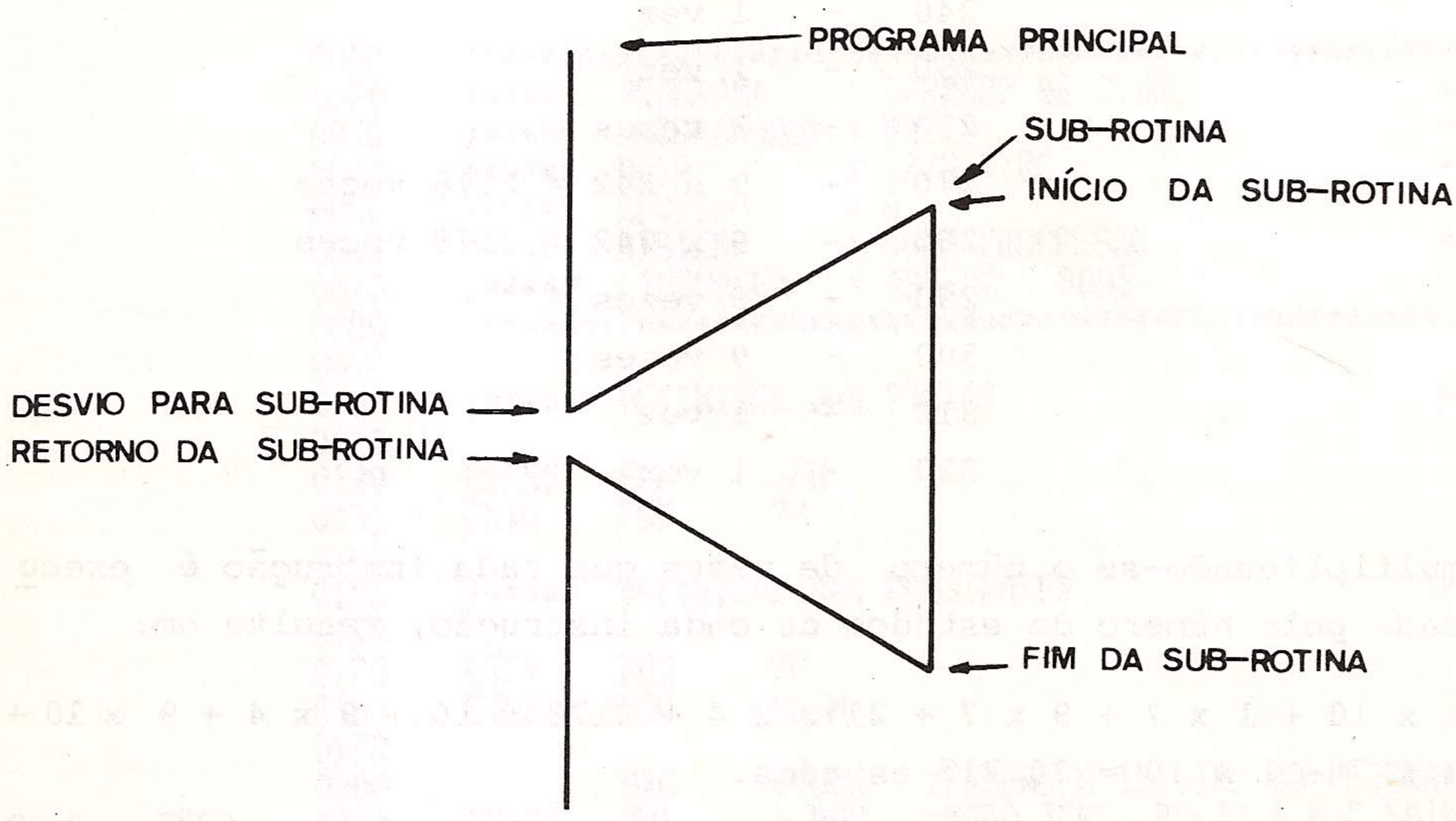


Fig. VI.8 - Execução de uma sub-rotina.

O uso de sub-rotina é empregado para economizar espaço de memória quando um programa possui em diversos pontos, a mesma seqüência de instruções. Também é útil para estruturação mais organizada do programa, pois este fica dividido em grupos de instruções de funções bem definidas, formando uma espécie de macro instrução.

Os microprocessadores 8080 e 8085 possuem um grupo de instruções especiais que fazem com que o programa salte para uma sub-rotina. Este grupo de instruções é formado pela instrução CALL, as instruções CALL condicionais e as instruções RST. Os mnemônicos destas instruções são:

CALL

CC

CNC

CZ

CNZ

CM

CP

CPE

CPO

RST

Existe um grupo de instruções para fazer com que o fluxo do programa retorne após o término de uma sub-rotina. Este grupo é formado pela instrução RET e as instruções RET condicionais, que são, normalmente,as instruções que aparecem na última posição de uma sub-rotina. Os mnemônicos destas instruções são:

RET

RC

RNC

RZ

RNZ

RM

RP

RPE

RPO

Porém, para entendermos bem o mecanismo das sub-rotinas, devemos antes entender o stack (pilha) e o stack pointer (apontador da pilha).

O stack é simplesmente uma área de memória RAM utilizada como registrador de endereços ou dados, que são mantidos pelo microprocessador e endereçados através do registrador stack pointer.

Portanto, quando o programa for utilizar o stack, antes de sua utilização, o registrador stack pointer deve ser carregado com o endereço da posição de memória RAM escolhida como posição inicial do stack. Caso não seja carregado, o endereço no registrador stack pointer, este terá um valor aleatório, indicando um endereço de memória qualquer, causando sérios danos

ao programa.

No stack são armazenados dados para futura utilização pelo programa. Antes de um dado ser armazenado no stack, o registrador stack pointer é decrementado, após o que é então transferido o dado para ele. Isto ocorre para que um dado seja armazenado em seguida a outro sem que o novo destrua o anteriormente armazenado. Note-se que desta maneira, os dados são armazenados em uma seqüência decrescente de endereço de memória.

Uma retirada de um dado do stack sempre retira o último dado armazenado, que é aquele da posição de memória, cujo endereço corresponde ao conteúdo do registrador stack pointer. Após a retirada de um dado, o conteúdo do registrador stack pointer é incrementado.

Para ilustrar o funcionamento do stack, vamos supor que o registrador stack pointer tenha o endereço $43FF_{16}$ e o conteúdo da posição de memória de endereço 43FF seja $7F_{16}$. Após um dado, como por exemplo 01_{16} ser armazenado no stack, a nova situação é mostrada na fig.

antes

STACK POINTER

43 FF

MEMÓRIA RAM

endereço	conteúdo
43 FC	xx
43 FD	xx
43 FE	xx
43 FF	7F

depois

STACK POINTER

43 FE

MEMÓRIA RAM

endereço	conteúdo
43 FC	xx
43 FD	xx
43 FE	01
43 FF	7F

Fig. VI.9 - Armazenamento do dado 01_{16} no stack.

Se agora um dado como 02_{16} for armazenado, a situação é a mostrada na fig. VI.10.

STACK POINTER

43 FD

MEMÓRIA RAM

endereço	conteúdo
43 FC	xx
43 FD	02
43 FE	01
43 FF	7F

Fig. V.10 - Armazenamento do dado 02_{16} no stack.

Nesta condição quando o stack é lido, o dado a ser retirado é o 02_{16} que está na posição de memória de endereço $43FD_{16}$ e o registrador stack pointer será incrementado, indicando o dado 01_{16}, da posição de memória de endereço $43FE_{16}$.

O item mais recente que foi armazenado no stack é designado como o topo do stack. O número de dados que podem ser armazenados no stack é limitado somente pelo tamanho da memória RAM. Deve-se, porém, ter o cuidado para que o tamanho do stack que será gerado pelo programa, não invada área de memória reservada para outros dados o que poderia causar problemas no programa. isto porque, devido ao fato de que quando dois dados forem armazenados na mesma posição de memória, o último destruirá o primeiro que será perdido, o que poderá causar conseqüências imprevisíveis:

Normalmente para endereço inicial do stack usa-se a última posição de memória RAM disponível.

As instruções que afetam o stack são as instruções de mnemônico: PUSH, POP, XTHL, RST e as instruções dos grupos CALL e RET.

A instrução PUSH transfere o conteúdo de um par de registradores para o stack. A instrução POP transfere o conteúdo do stack para um par de registradores. A instrução RST será analisada na próxima seção, pois o seu funcionamento é como o de uma interrupção.

Vamos analisar as instruções CALL e RET que são as que mais nos interessam agora.

Quando uma instrução CALL é armazenada no registrador de instrução do microprocessador, o conteúdo do registrador contador de programa é incrementado três vezes e carregado no stack, salvando o endereço da próxima instrução que deveria ser executada. Os dois bytes do operando da instrução CALL são carregados no registrador contador de programa. Após isto, o ciclo de busca da próxima instrução tem início. Como o endereço da próxima instrução é o conteúdo do registrador contador de programa, ocorre uma quebra no fluxo normal do programa, seguindo este a partir do endereço que era o operando da instrução CALL.

O que a instrução RET faz é transferir o conteúdo do stack para o contador de programa. Desta forma, quando uma instrução RET é encontrada, provoca com que o fluxo do programa continue na instrução seguinte a instrução CALL, a qual provocou o salto para a sub-rotina.

O stack também pode ser usado para salvar os status do programa principal quando uma sub-rotina é processada. A sub-rotina pode ter necessidade de destruir o conteúdo de um registrador que contém um dado usado pelo programa principal. Neste caso, basta no começo da sub-rotina transferir o conteúdo deste registrador para o stack e, antes da instrução de RET, restaurar o conteúdo de registrador.

Para exemplificar o uso da sub-rotina, vamos inicialmente, desenvolver o seguinte programa: transmitir o conteúdo do registrador "C" para uma impressora que está conectada a interface de comunicação serial 8251. As características desta impressora são:

- modo de operação: assincrono.
- velocidade de comunicação: 600 bauds.
- tipo de informação: caracteres codificados no código ASCII.
- comprimento do caracter: 8 bits.
- número de bits de término: 2 bits.
- paridade: não checa.

Solução:

A primeira preocupação é a programação da interface de comunicação. Como a velocidade de transmissão é de 600 bauds,

pode ser selecionado o fator de 64X. Nesta condição é necessário um sinal de 38400Hz para clock de transmissão. Para o nosso sistema, deve ser ligado o ponto D da 7493 ao ponto 9 da 8251 (ver fig. XIII.11 do volume 1).

O modo de operação da interface é:

- assincrono.
- paridade desabitada.
- 8 bits por caracter.
- 2 bits de término.
- sinal de clock igual a 64 vezes a velocidade de transmissão.

Desta maneira a instrução de modo é igual a EF_{16}. (ver figura XI.6 Volume I).

Os comandos necessários para a interface são:

- transmissão habilitada.
- operação de transmissão não bloqueada.
- não dar reset interno.
- não pesquisar caracter SYNC.
- pino RST em nível "0".
- pino DTR em nível "0".

Desta maneira, a instrução de comando é igual à 23_{16} (ver figura XI.7 do Volume I).

Outra coisa importante que deve ser verificada é que antes de enviar um caracter para ser transmitido pela interface, deve ser verificado se ela está disponível para prestar este serviço, isto é, terminou qualquer outro serviço que estava executando. Para isto, basta verificar se o bit zero da palavra de status é igual a "1".

O fluxograma para este programa é o da figura VI.11.

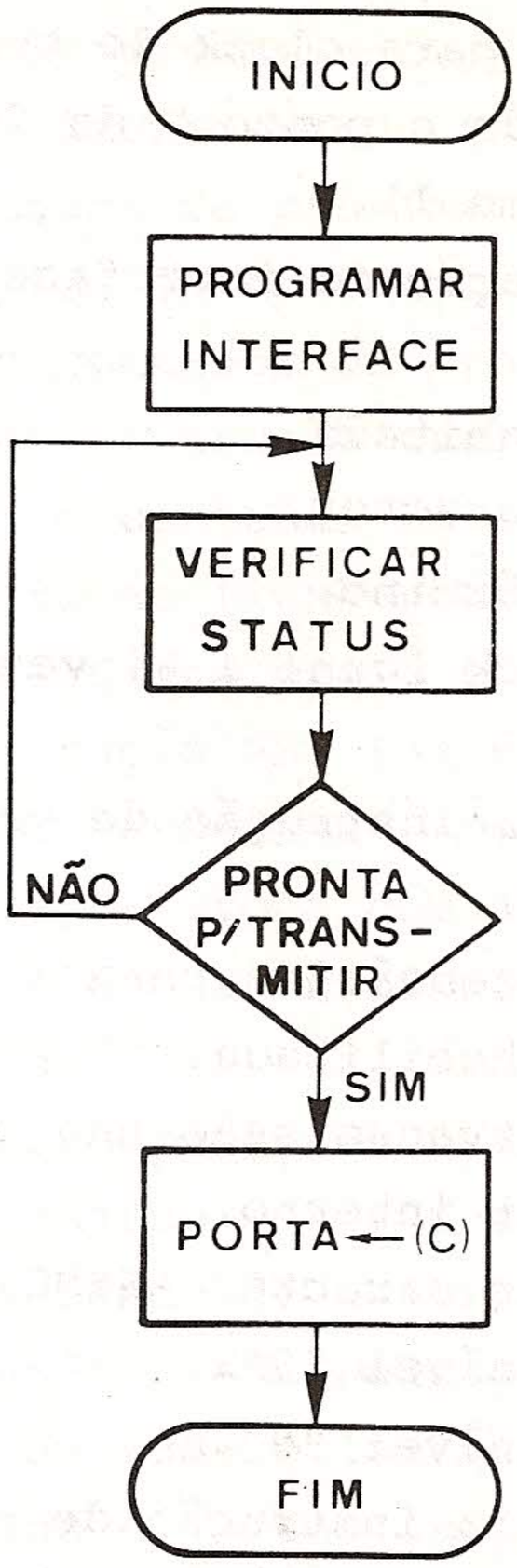


Fig. VI.11 - Transmite dados.

Na figura VI.12 está o programa correspondente ao fluxograma da fig. VI.11.

```
0010    ;*************************************************************
0020    ;*****  PROGRAMA    : TRANSMITE DADOS                     *****
0030    ;*****  PROGRAMADOR : BEZERRA                             *****
0040    ;*****  DATA        : 8/JUN/81                            *****
0050    ;*****  REVISAO     : A                                   *****
0060    ;*****  SISTEMA     : 8085 PROCESSOR                      *****
0070    ;*****  LINGUAGEM   : ASSEMBLY 8085                       *****
0080    ;*************************************************************
0090
0100    ;*****  DEFINICAO DAS PORTAS                              *****
0110
0120    CTR51   EQU     3FH
0130    PRTS    EQU     3BH
0140
0150    ;*****  DEFINICAO DAS CONSTANTES                          *****
0160
0170    MODO    EQU     0EFH    ;ASSINCRONO,SEM PARIDADE
0180                            ;8 BITS DADOS, 2 BITS TERMINO
0190                            ;VELOCIDADE 600 BAUDS, FATOR 64X
0200    CMD     EQU     23H     ;HABILITA PARA TRANSMISSSAO
0210
0220            ORG     0180H   ;ENDERECO INICIAL DO PROGRAMA
0180  3EEF  0230  INIC:  MVI     A,MODO
0182  D33F  0240        OUT     CTR51   ;INSTRUCAO DE MODO PARA 8251
0184  3E23  0250        MVI     A,CMD
0186  D33F  0260        OUT     CTR51   ;INSTRUCAO DE COMANDO PARA 8251
0188  DB3F  0270  TRAN:  IN      CTR51   ;LE STATUS DA INTERFACE
018A  E601  0280        ANI     1H      ;VERIFICA SE ESTA PRONTA
            0290                        ;PARA TRANSMITIR
018C  CA8801 0300       JZ      TRAN    ;SE NAO, VOLTA
018F  79    0310        MOV     A,C     ;TRANSFERE DADO PARA ACUMULADOR
0190  D33B  0320        OUT     PRTS    ;ENVIA DADO PARA INTERFACE
0192  76    0330        HLT             ;PARE
            0340        END

SYMBOL TABLE

CMD     0023    CTR51   003F    INIC    0180    MODO    00EF    PRTS    003B
TRAN    0188

ERRORS = 0
```

Fig.VI.12 - Transmite dados.

O programa de transmissão de dados, visto anteriormente, pode ser usado em qualquer parte de um programa quando for necessário enviar informações do sistema para um periférico. Nestas condições ele pode ser transformado em uma sub-rotina, sendo acrescentado ao final deste programa a instrução RET.

Para uso do programa que transmite dados em uma sub-rotina, vamos desenvolver um programa para a seguinte tarefa:

- imprimir o conteúdo do acumulador, utilizando a impressora com as características do programa anterior.

Solução:

Como a impressora recebe informações codificadas no código ASCII, o conteúdo do acumulador deverá ser convertido para este código, antes de ser transmitido. Como o acumulador possui oito bits, o seu conteúdo é composto por dois dígitos hexadecimais. O primeiro nos quatro bits mais significativos correspondendo aos bits de quatro a sete, e o segundo nos quatro bits menos significativos, os bits de zero a três. Para representação de dígitos hexadecimais são usados os seguintes caracteres:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

e o correspondente a eles no código ASCII é (ver anexo C):

30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 41, 42, 43, 44, 45, 46

A codificação no código ASCII é feita da seguinte maneira:

- aos dígitos de 0 a 9 é somado 30_{16}.
- aos dígitos de A a F é somado 37_{16}.

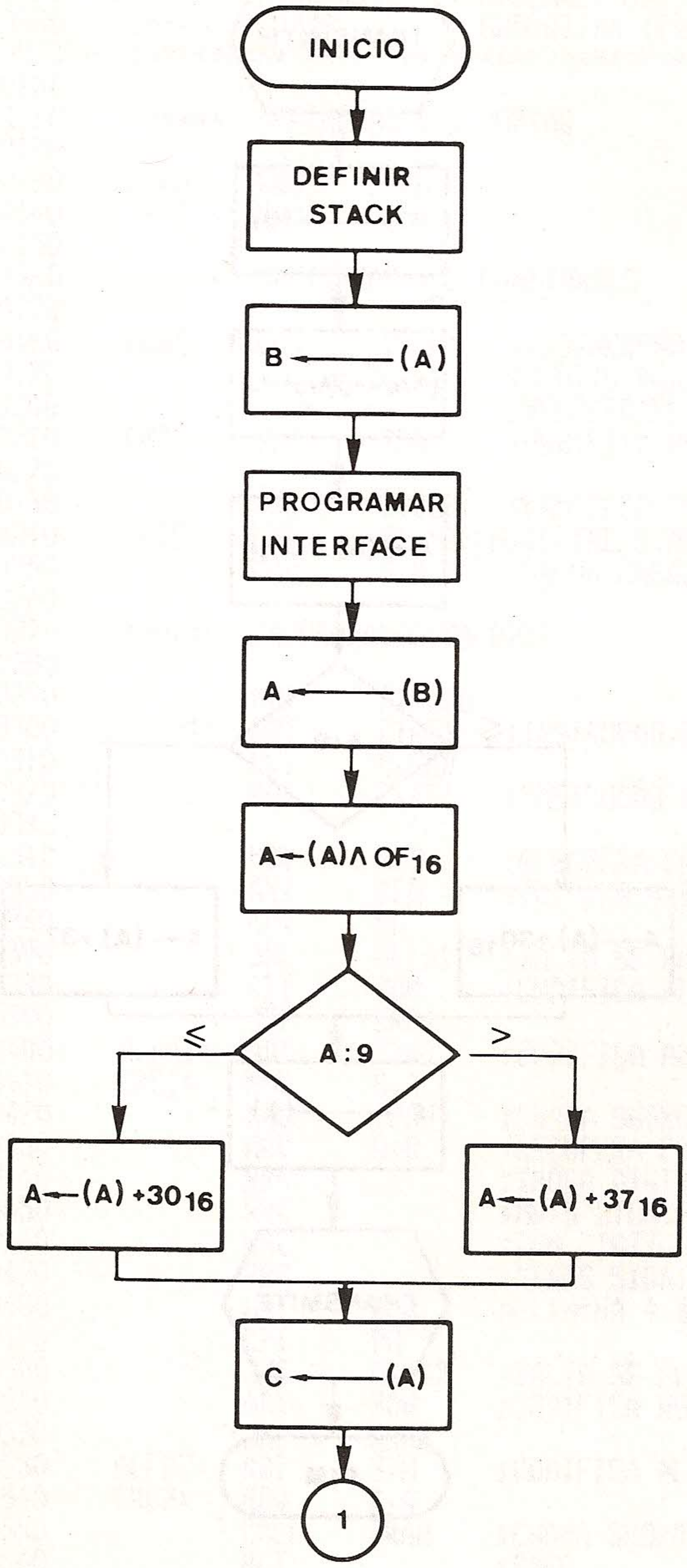
Após a codificação para o código ASCII, cada caracter será enviado a impressora por meio da interface de comunicação série 8251. Para esta transmissão de dados, vamos utilizar o programa visto anteriormente como uma sub-rotina.

Já que será usada uma sub-rotina, o registrador stack pointer deve ser previamente definido. Vamos usar para conteúdo inicial do stack pointer a posição de memória de endereço $43FF_{16}$.

Deve-se lembrar que o dado, a ser transmitido pela sub-rotina (transmite dados) deve estar armazenado no registra

dor C.

Com estas considerações o fluxograma para este programa é mostrado na fig. VI.13 e a listagem do programa correspondente na figura VI.14.



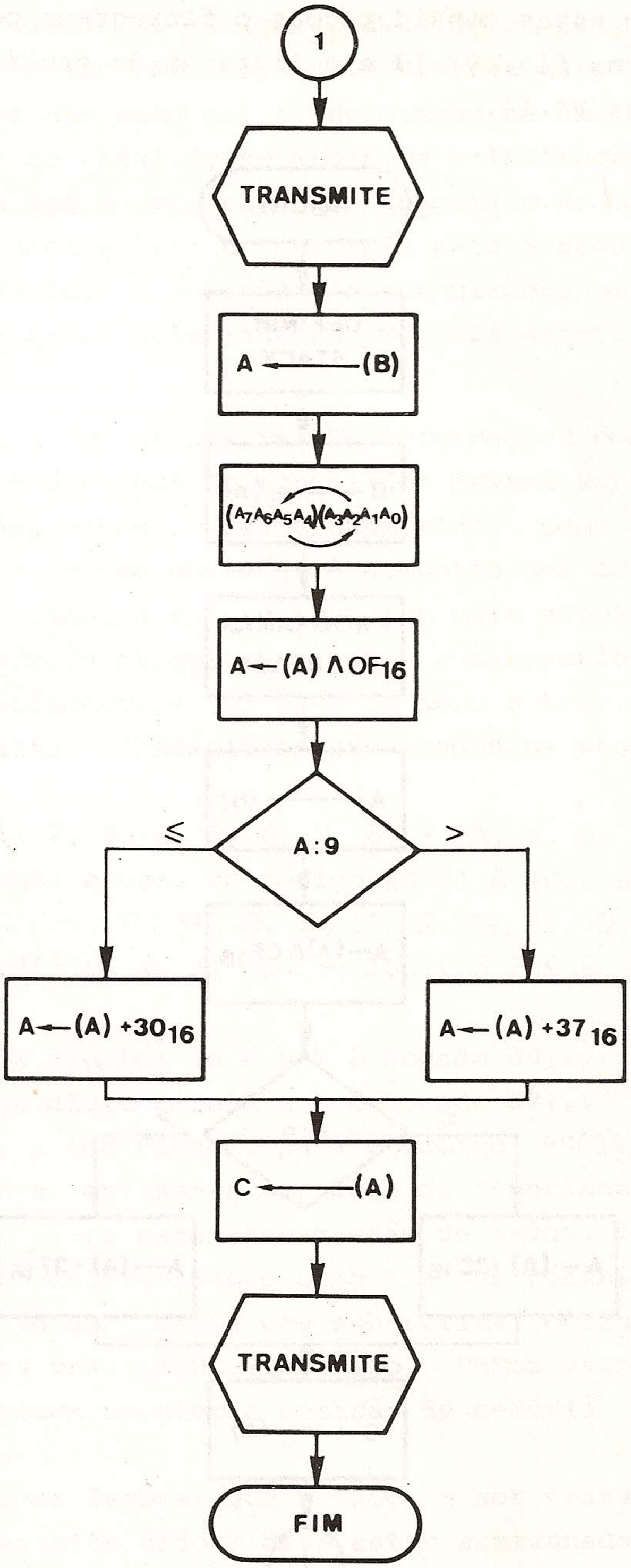


Fig. VI.13 - Imprime conteúdo do acumulador.

```
0010    ;*****************************************************************
0020    ;*****  PROGRAMA    : IMPRIME CONTEUDO ACUMULADOR  *****
0030    ;*****  PROGRAMADOR : RAFAEL                       *****
0040    ;*****  DATA        : 10/JUN/81                    *****
0050    ;*****  REVISAO     : A                            *****
0060    ;*****  SISTEMA     : 8085 PROCESSOR               *****
0070    ;*****  LINGUAGEM   : ASSEMBLY 8085                *****
0080    ;*****  CHAMA       : SUBROTINA TRANSMITE          *****
0090    ;*****************************************************************
0100
0110    ;*****  DEFINICAO DAS PORTAS                       *****
0120
0130    CTR51   EQU     3FH
0140    PRTS    EQU     3BH
0150
0160    ;*****  DEFINICAO DAS CONSTANTES                   *****
0170
0180    MODO    EQU     0EFH    ;ASSINCRONO, SEM PARIDADE
0190                            ;8 BITS DADOS, 2 BITS TERMINO
0200                            ;VELOCIDADE 600 BAUDS, FATOR 64X
0210    CMD     EQU     23H     ;HABILITA PARA TRANSMISSAO
0220
0230            ORG     0100H   ;ENDERECO INICIAL DO PROGRAMA
0100  31FF43  0240    INIC:   LXI     SP,43FFH;DEFINE STACK POINTER
0103  47      0250            MOV     B,A     ;SALVA CARACTER EM B
0260
0270    ;*****  PROGRAMACAO DA 8251                        *****
0280
0104  3EEF    0290            MVI     A,MODO
0106  D33F    0300            OUT     CTR51   ;INSTRUCAO DE MODO
0108  3F23    0310            MVI     A,CMD
010A  D33F    0320            OUT     CTR51   ;INSTRUCAO DE COMANDO
              0330
010C  78      0340            MOV     A,B     ;RECUPERA CARACTER
010D  E60F    0350            ANI     0FH     ;ELIMINA 4 BITS MAIS SIGNIFICAT.
010F  FE09    0360            CPI     9H
0111  D21901  0370            JNC     ALFA1   ;SALTA SE FOR DIGITO ALFABETICO
0114  C630    0380            ADI     30H     ;CODIFICA NUMERICO EM ASCII
0116  C31B01  0390            JMP     CRC1
0119  C637    0400    ALFA1:  ADI     37H     ;CODIFICA ALFABETICO EM ASCII
011B  4F      0410    CRC1:   MOV     C,A
011C  CD8001  0420            CALL    TRAN    ;CHAMA SUBROTINA TRANSMITE DADOS
011F  78      0430            MOV     A,B     ;RECUPERA CARACTER
0120  0F      0440            RRC             ;TROCA DIGITO /
0121  0F      0450            RRC             ;MAIS SIGNIFICATIVO /
0122  0F      0460            RRC             ;COM DIGITO /
0123  0F      0470            RRC             ;MENOS SIGNIFICATIVO
0124  F60F    0480            ANI     0FH     ;ELIMINA 4 BITS MAIS SIGNIFICAT.
0126  FE09    0490            CPI     9H
0128  D23001  0500            JNC     ALFA2   ;SALTA SE FOR DIGITO ALFABETICO
012B  C630    0510            ADI     30H     ;CODIFICA NUMERICO EM ASCII
012D  C33201  0520            JMP     CRC2
0130  C637    0530    ALFA2:  ADI     37H     ;CODIFICA ALFABETICO EM ASCII
0132  4F      0540    CRC2:   MOV     C,A
0133  CD8001  0550            CALL    TRAN    ;CHAMA SUBROTINA TRANSMITE DADOS
0136  76      0560            HLT             ;PARE
              0570
```

Cont.

```
                    0580    ;*****  SUBROTINA TRANSMITE DADOS                    *****
                    0590    ;ENTRADA: DADO A SER TRANSMITIDO NO REGISTRADOR C
                    0600    ;SAIDA  : CARREGA CONTEUDO DE C NA 8251
                    0610    ;DESTROI: A,FLAGS
                    0620
                    0630            ORG     0181H   ;ENDERECO INICIAL DA SUBROTINA
0180    DB3F        0640    TRAN:   IN      CTR51   ;LE STATUS DA INTERFACE
0182    E601        0650            ANI     1H      ;VERIFICA SE ESTA PRONTA
0184    CA8001      0660            JZ      TRAN    ;SE NAO, VOLTA
0187    79          0670            MOV     A,C     ;TRANSFERE DADO PARA ACUMULADOR
0188    D33B        0680            OUT     PRTS    ;ENVIA DADO PARA INTERFACE
018A    C9          0690            RET             ;VOLTA
                    0700            END

SYMBOL TABLE

ALFA1   0119    ALFA2   0130    CMD     0023    CRC1    011B    CRC2    0132
CTR51   003F    INIC    0100    MODO    00EF    PRTS    003B    TRAN    0180

ERRORS = 0
```

Fig. VI.14 - Imprime conteúdo do acumulador.

Outro programa de exemplo:

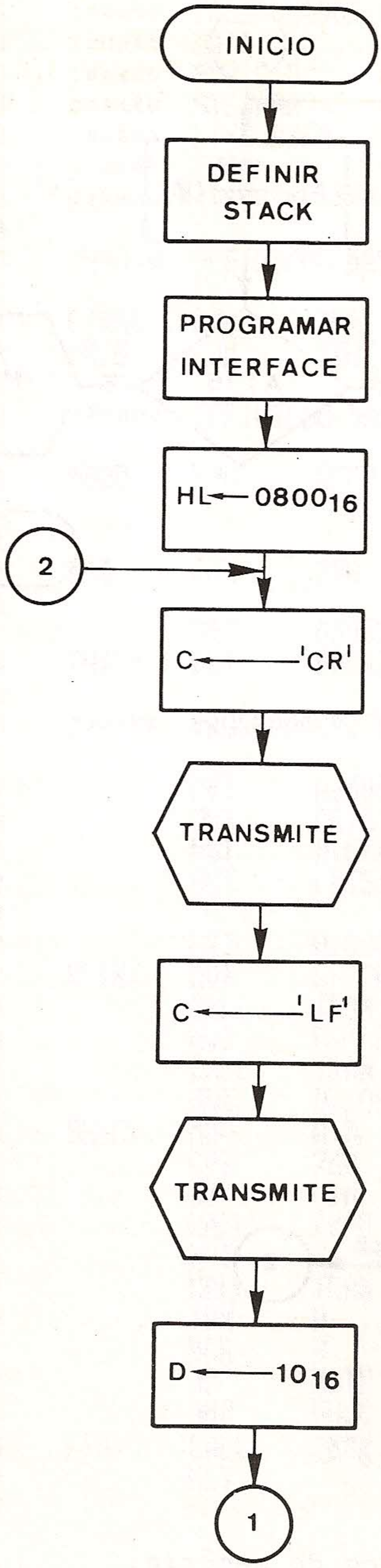
- imprimir o conteúdo de memória a partir do endereço 0800_{16} até a posição de memória cujo conteúdo é 76_{16}.

Solução:

Utilizando os programas anteriores que codificam em ASCII e imprimem o conteúdo do acumulador, vamos imprimir o conteúdo da memória com um espaço branco entre cada dois dígitos e dezesseis grupos de dois dígitos por linha de impressão.

Convém ressaltar que o caracter ASCII, CR (CARRIAGE RETURN) correspondente a $0D_{16}$ é um comando para retorno do carro da impressora para a posição inicial de impressão, o caracter ASCII, LF (LINE FEED) correspondente a $0A_{16}$ é um comando para avanço de papel. O caractere ASCII, SP (SPACE) correspondente a 20_{16} é um comando para dar um espaço na impressão.

O fluxograma para este programa é mostrado na figura VI.15 e o programa na figura VI.16.


INICIO
DEFINIR STACK
PROGRAMAR INTERFACE
HL ← 0800 16
2
C ← 'CR'
TRANSMITE
C ← 'LF'
TRANSMITE
D ← 10 16
1

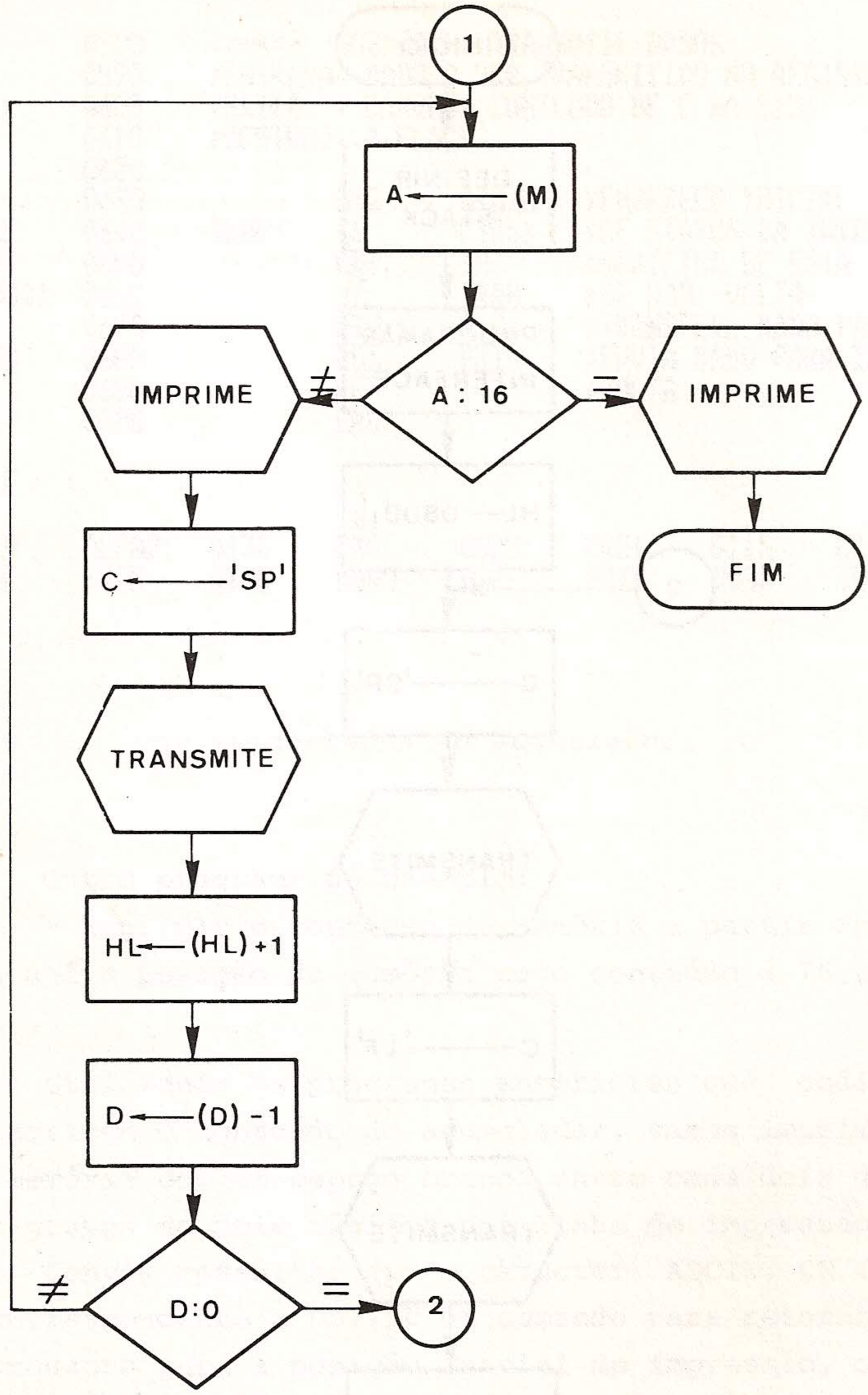


Fig. VI.15 - Imprimir bloco de memória.

```
0010    ;*****************************************************
0020    ;*****  PROGRAMA    : IMPRIME BLOCO DE MEMORIA     *****
0030    ;*****  PROGRAMADOR : CAPUANO                      *****
0040    ;*****  DATA        : 5/JUL/81                     *****
0050    ;*****  REVISAO     : A                            *****
0060    ;*****  SISTEMA     : 8085 PROCESSOR               *****
0070    ;*****  LINGUAGEM   : ASSEMBLY 8085                *****
0080    ;*****  CHAMA       : SUBROTINAS IMPRIME,TRANSMITE *****
0090    ;*****************************************************
0100
0110    ;*****  DEFINICAO DAS PORTAS                       *****
0120
0130    CTRS1   EQU     3FH
0140    PRTS    EQU     3BH
0150
0160    ;*****  DEFINICAO DAS CONSTANTES                   *****
0170
0180    MODO    EQU     0EFH    ;ASSINCRONO,SEM PARIDADE
0190                            ;8 BITS DADOS, 2 BITS TERMINO
0200                            ;VELOCIDADE 600 BAUDS, FATOR 64X
0210    CMD     EQU     23H     ;HABILITA PARA TRANSMISSAO
0220
0230            ORG     0000H   ;ENDERECO INICIAL DO PROGRAMA
0000  31FF43  0240    INIC:   LXI     SP,43FFH;DEFINE STACK POINTER
0250
0260    ;*****  PROGRAMACAO DA 8251                        *****
0270
0003  3EEF    0280            MVI     A,MODO
0005  D33F    0290            OUT     CRT51   ;INSTRUCAO DE MODO
0007  3E23    0300            MVI     A,CMD
0009  D33F    0310            OUT     CRT51   ;INSTRUCAO DE COMANDO
              0320
000B  210008  0330            LXI     H,0800H ;ENDERECO INICIAL DA MEMORIA
000E  0E0D    0340    NLIN:   MVI     C,'CR'
0010  CD8001  0350            CALL    TRAN    ;RETORNO DO CARRO
0013  0E0A    0360            MVI     C,'LF'
0015  CD8001  0370            CALL    TRAN    ;AVANCO DE PAPEL
0018  1610    0380            MVI     D,10H   ;CONTADOR
001A  7E      0390    BUSC:   MOV     A,M     ;RETIRA CONTEUDO A SER IMPRESSO
001B  FE76    0400            CPI     76H     ;COMPARA SE FIM
001D  CA3000  0410            JZ      FIM     ;SE FOR, VA PARA FIM
0020  CD0001  0420            CALL    IMPR    ;SE NAO, IMPRIME
0023  0E20    0430            MVI     C,'SP'
0025  CD8001  0440            CALL    TRAN    ;ESPACO
0028  23      0450            INX     H       ;NOVO ENDERECO DE MEMORIA
0029  15      0460            DCR     D       ;VERIFICA SE TERMINOU LINHA
002A  CA0E00  0470            JZ      NLIN    ;SE SIM; NOVA LINHA
002D  C31A00  0480            JMP     BUSC    ;SE NAO, NOVO CONTEUDO
0030  CD0001  0490    FIM:    CALL    IMPR    ;IMPRIME 76
0033  76      0500            HLT             ;PARE
              0510
```

Cont.

```
                    0520    ;*****  SUBROTINA IMPRIME CONTEUDO DO ACUMULADOR   *****
                    0530    ;ENTRADA:DADO A SER IMPRESSO NO ACUMULADOR
                    0540    ;SAIDA  :CARREGA DADO CODIFICADO EM ASCII NA 8251
                    0550    ;DESTROI:A,B,C,FLAGS
                    0560    ;CHAMA  :SURROTINA TRANSMITE
                    0570
                    0580            ORG     0100H   ;ENDERECO INICIAL DA SUBROTINA
0100    47          0590    IMPR:   MOV     B,A     ;SALVA CARACTER EM B
0101    E60F        0600            ANI     0FH     ;ELIMINA 4 BITS MAIS SIGNIFICAT.
0103    FE09        0610            CPI     9H
0105    D20D01      0620            JNC     ALFA1   ;SALTA SE FOR DIGITO ALFABETICO
0108    C630        0630            ADI     30H     ;CODIFICA NUMERICO EM ASCII
010A    C30F01      0640            JMP     CRC1
010D    C637        0650    ALFA1:  ADI     37H     ;CODIFICA ALFABETICO EM ASCII
010F    4F          0660    CRC1:   MOV     C,A
0110    CD8001      0670            CALL    TRAN    ;CHAMA SUBROTINA TRANSMITE DADOS
0113    78          0680            MOV     A,B     ;RECUPERA CARACTER
0114    0F          0690            RRC             ;TROCA DIGITO /
0115    0F          0700            RRC             ;MAIS SIGNIFICATIVO /
0116    0F          0710            RRC             ;COM DIGITO /
0117    0F          0720            RRC             ;MENOS SIGNIFICATIVO
0018    E60F        0730            ANI     0FH     ;ELIMINA 4 BITS MAIS SIGNIFICAT.
011A    FE09        0740            CPI     9H
011C    D22401      0750            JNC     ALFA2   ;SALTA SE FOR DIGITO ALFABETICO
011F    C630        0760            ADI     30H     ;CODIFICA NUMERICO EM ASCII
0121    C32601      0770            JMP     CRC2
0124    C637        0780    ALFA2:  ADI     37H     ;CODIFICA ALFABETICO EM ASCII
0126    4F          0790    CRC2:   MOV     C,A
0127    CD8001      0800            CALL    TRAN    ;CHAMA SUBROTINA TRANSMITE DADOS
012A    C9          0810            RET             ;VOLTA
                    0820
                    0830    ;*****  SUBROTINA TRANSMITE DADOS                  *****
                    0840    ;ENTRADA:DADO A SER TRANSMITIDO NO REGISTRADOR C
                    0850    ;SAIDA  :CARREGA CONTEUDO DE C NA 8251
                    0860    ;DESTROI:A,FLAGS
                    0870
                    0880            ORG     0180H   ;ENDERECO INICIAL DA SUBROTINA
0180    DB3F        0890    TRAN:   IN      CTR51   ;LE STATUS DA INTERFACE
0182    E601        0900            ANI     1H      ;VERIFICA SE ESTA PRONTA
0184    CA8001      0910            JZ      TRAN    ;SE NAO, VOLTA
0187    79          0920            MOV     A,C     ;TRANSFERE DADO PARA ACUMULADOR
0188    D33B        0930            OUT     PRTS    ;ENVIA DADO PARA INTERFACE
018A    C9          0940            RET             ;VOLTA
                    0950            END

SYMBOL TABLE

ALFA1   010D    ALFA2   0124    BUSC    001A    CMD     0023    CRC1    010F
CRC2    0126    CTR51   003F    FIM     0030    IMPR    0100    INIC    0000
MODO    00EF    NLIN    000E    PRTS    003B    TRAN    0180

ERRORS = 0
```

Fig. VI.15 - Imprimir bloco de memória.

Caso o conteúdo da memória a partir do endereço 0800_{16} seja o mostrado na fig. VI.17, a execução do programa da figura VI.16, gera a listagem mostrada na fig. VI.18.

ender.	cont.	ender.	cont.	ender.	cont.	ender.	cont.
0800	2A	080A	64	0814	1A	081E	66
0801	35	080B	57	0815	10	081F	75
0802	5C	080C	41	0816	6F	0820	79
0803	35	080D	30	0817	00	0821	76
0804	10	080E	23	0818	7B	0822	3A
0805	00	080F	69	0819	1A	0823	14
0806	3B	0810	2A	081A	50	0824	13
0807	0F	0811	19	081B	7B	0825	57
0808	5F	0812	6E	081C	47	0826	19
0809	27	0813	77	081D	3A	0827	3D

Fig. VI.17 - Conteúdo aleatório de memória.

```
2A 35 5C 35 10 00 3B 0F 5F 27 64 57 41 30 23 69
2A 19 6E 77 1A 10 6F 00 7B 1A 50 7B 47 3A 66 75
79 76
```

Fig. VI.18 - Listagem gerada pelo programa de imprimir bloco de memória.

VI.4 INTERRUPÇÃO

O pino de interrupção é uma porta especial de entrada de informação, que o microprocessador examina a todo final de ~~ciclo~~ de instrução para verificar se um sinal de interrupção está presente neste pino. Caso um sinal seja recebido nesse pino, o processo de interrupção ocorre.

O processo de interrupção é semelhante ao da sub-rotina. Quando uma interrupção é solicitada, o microprocessador termina o ciclo de instrução atual, e como na sub-rotina salva o valor do registrador contador de programa na memória stack e carrega um novo endereço de memória neste registrador. Desta for

ma, o programa salta para uma rotina separada. Quando ao final desta rotina existir uma instrução RET, o valor antigo que está no stack é colocado de volta no registrador contador de programa para que o programa retorne para o ponto em que a interrupção ocorreu.

A diferença entre a interrupção e a sub-rotina é que esta ocorre por software através de instruções e a interrupção é gerada por hardware, através de um sinal externo. Outra diferença é que a sub-rotina pode ser colocada em qualquer lugar da memória, já a rotina de interrupção inicia-se em um endereço fixo e pré-estabelecido de memória, como será mostrado mais adiante.

As interrupções são usadas para informar ao microprocessador a ocorrência de um evento externo a ele, podendo, então, o microprocessador executar as ações previamente estipuladas para quando haja a ocorrência deste evento.

As interrupções são muito usadas em equipamentos de controle, sendo produzidas por alarme ou medidores quando estes atingem um valor crítico. Nestas condições, o microprocessador ligará ou desligará chaves, atuará no equipamento de maneira que a situação possa retornar ao normal. A interrupção também pode ser usada em outros equipamentos, como por exemplo, para informar que uma tecla foi pressionada em um teclado, um certo intervalo de tempo foi concluído por um contador, uma interface tem um dado pronto para ser lido. Enfim, informa qualquer evento externo ao microprocessador que exija atenção imediata deste, não havendo necessidade de que ele fique constantemente checando a ocorrência ou não do evento.

A requisição de interrupção no microprocessador 8080 é gerada no pino INT. Quando neste pino estiver presente um nível lógico "1", o seu processamento atual será interrompido, gerando um sinal através do pino $\overline{\text{INTA}}$ do controlador de sistema 8228 (ver capítulo VIII do Volume I) e transferindo o conteúdo atual do bus de dado para o registrador de instrução. O sinal do pino $\overline{\text{INTA}}$ pode ser usado para habilitar alguns circuitos integrados especiais, tais como, o 8214 ou 8259, cuja função será colocar no bus de dados o código de uma instrução RST. As instruções RST's, como visto no capítulo IV, funcionam como um sal

to para uma rotina armazenada a partir de um endereço fixo de memória, dependendo de qual instrução RST está sendo processada.

Outra possibilidade existente é, quando não houver um dos circuitos integrados especiais disponível, ligar o pino $\overline{\text{INTA}}$ por um resistor de 1kΩ à fonte de 12 volts. Neste caso, o controlador de sistema 8228 colocará os pinos que formam o bus de dados em alta impedância (three-state), o que é interpretado pelo microprocessador 8080 com nível lógico "1". Sendo todos os pinos do bus de dados equivalentes ao nível lógico "1" (o que em hexadecimal é FF), a instrução que será armazenada no registrador de instrução é RST 7, fazendo com que o programa salte para a rotina armazenada a partir do endereço de memória 0038_{16}.

O microprocessador 8085 possui um pino de entrada de interrupção INTR que tem a mesma função do pino INT do microprocessador 8080 e pino $\overline{\text{INTA}}$ de mesma função de pino $\overline{\text{INTA}}$ do controlador de sistema 8228. Adicionalmente o microprocessador 8085 possui mais quatro pinos de entradas de interrupção que são chamados de RST 7,5; RST 6,5; RST 5,5; TRAP.

Um sinal nestes pinos produz o mesmo efeito que as instruções RST's, porém os endereços para o salto do programa correspondentes a estes pinos é mostrado na fig. VI.19.

Pino	Endereço para o salto
TRAP	0024_{16}
RST 7,5	$003C_{16}$
RST 6,5	0034_{16}
RST 5,5	$002C_{16}$

Fig. VI.19 - Endereços para interrupção do microprocessador 8085.

Quando ocorre mais de um sinal de interrupção simultâneo para o microprocessador 8085 será atendida, primeiramente, a interrupção que possui mais prioridade, conforme a tabela da fig. VI.20. Por exemplo, se ocorre simultaneamente um sinal de interrupção nos pinos INTR e RST 5,5, primeiramente será atendida a interrupção RST 5,5 e, posteriormente, a interrupção INTR.

Pino	Prioridade de interrupção
TRAP	1ª
RST 7,5	2ª
RST 6,5	3ª
RST 5,5	4ª
INTR	5ª

Fig. VI.20 - Prioridade de interrupção no microprocessador 8085.

Existem, também, diferenças no sinal que deve ser enviada para cada pino para que uma interrupção possa ocorrer. Nos pinos INTR, RST 5,5, RST 6,5, assim como o pino INT do microprocessador 8080, basta que estejam em nível "1" para que uma interrupção seja reconhecida. No pino RST 7,5, uma interrupção somente é reconhecida na subida do sinal, isto é, quando o sinal enviado a ele varia do nível "0" para o nível "1". Já o pino TRAP necessita de uma variação do nível lógico "1" para o nível lógico "0" para que seja habilitado a receber um sinal de interrupção, e só provocará uma interrupção quando o sinal estiver em nível lógico "1".

Os microprocessadores 8080 e 8085 possuem instruções para habilitar (EI) e desabilitar (DI) interrupções. Quando um sinal de RESET é recebido ou uma interrupção é aceita, automaticamente, interrupções são desabilitadas, sendo que nova interrupção pode ocorrer somente após o processamento da instrução EI. Apenas a interrupção TRAP do microprocessador 8085 nunca é desabilitada.

O microprocessador 8085 possui uma instrução especial para habilitar ou desabilitar diferentes interrupções, é a instrução SIM, e outra instrução para verificar o status das interrupções, a instrução RIM (ver capítulo IV).

Para exemplificar o uso da interrupção, vamos desenvolver um programa para realizar a seguinte tarefa:

- Projetar um relógio com saída de dados através de display que inclue decodificador de binário para sete segmentos, conectados conforme esquema da fig. VI.21.

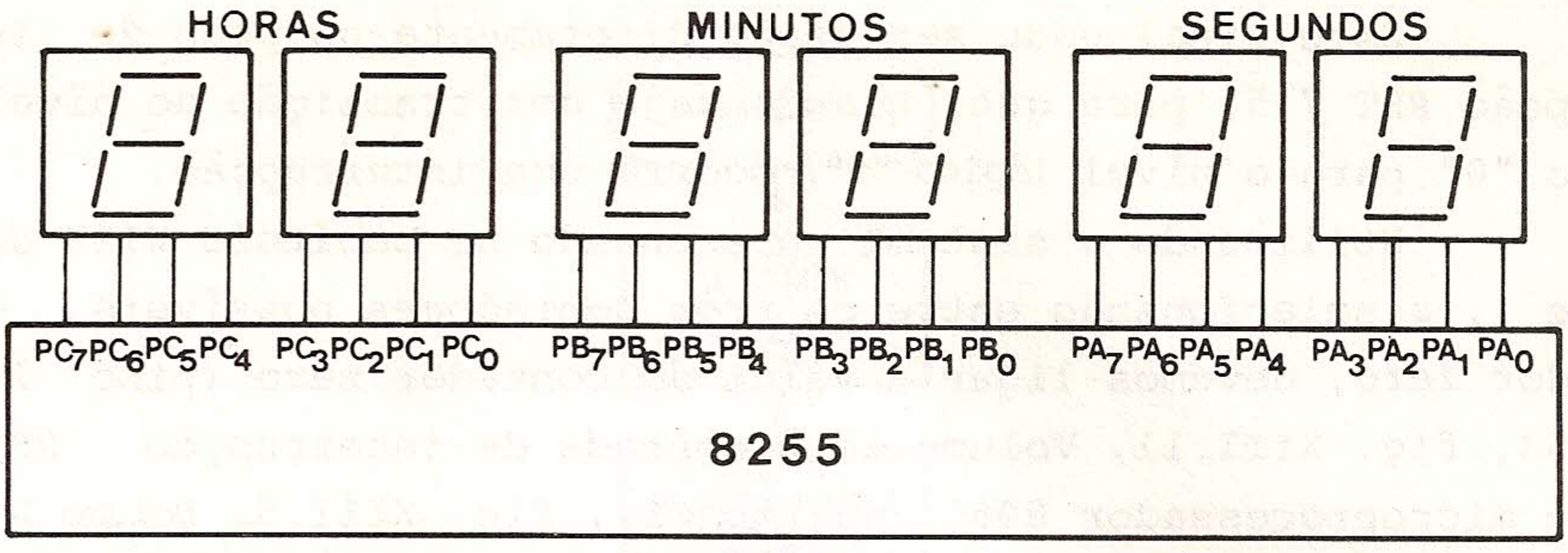


Fig. VI.21 - Esquema de ligação do display.

Solução:

Para a base de tempo, vamos utilizar o contador programavél 8253. A saída do contador será conectada a um pino de interrupção para avisar ao microprocessador o final da contagem, cujo valor iremos determinar. Enquanto isto, o microprocessador ficará sempre atualizando as saídas, mesmo que estas sejam iguais às saídas anteriores.

Este tipo de programa é chamado relógio de tempo real, pois o padrão de tempo não depende do software; enquanto o microprocessador está executando outras tarefas, o padrão de tempo está sendo gerado e, ao final de cada período, é gerado um sinal de hardware para atualização do relógio, não havendo, portanto, variação de um período para outro.

Precisamos de um sinal periódico e de período igual a um segundo para atualizar as informações de saída para o display.

Entre os modos possíveis de funcionamento do contador programavél, vamos selecionar o modo dois, que faz com que o sinal gerado seja um sinal contínuo e igual ao apresentado na fig. VI.22 (ver capítulo XII, Volume 1).

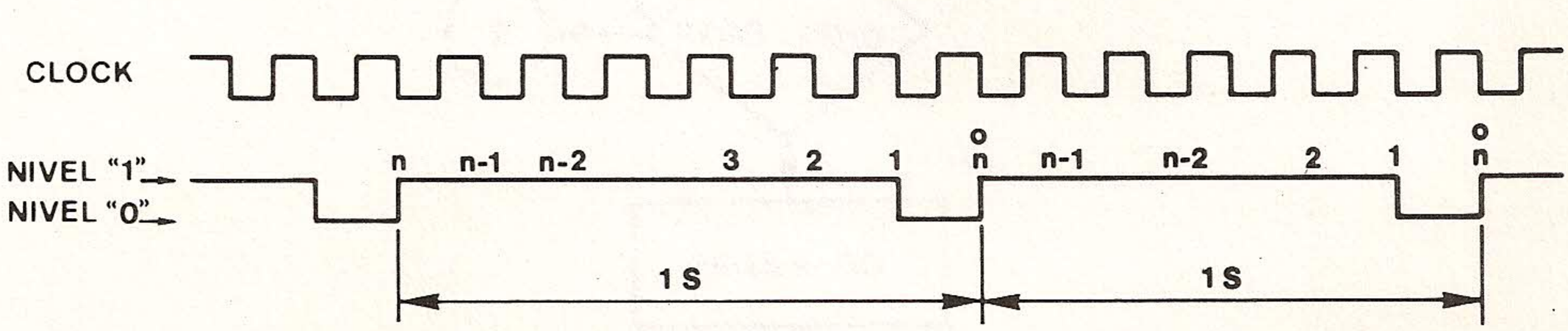


Fig. VI.22 - Sinal do contador no modo 2.

Este sinal pode ser usado diretamente no pino de interrupção RST 7,5, para que, quando haja uma transição do nível lógico "0" para o nível lógico "1", ocorra uma interrupção.

Utilizando o sistema apresentado no capítulo XIII do Volume 1, e selecionando entre os três contadores possíveis, o contador zero, devemos ligar a saída do contador zero (pino 10 da 8253, fig. XIII.11, Volume 1) à entrada de interrupção RST 7,5 do microprocessador 8085 (entrada I_4, fig. XIII.5, Volume 1).

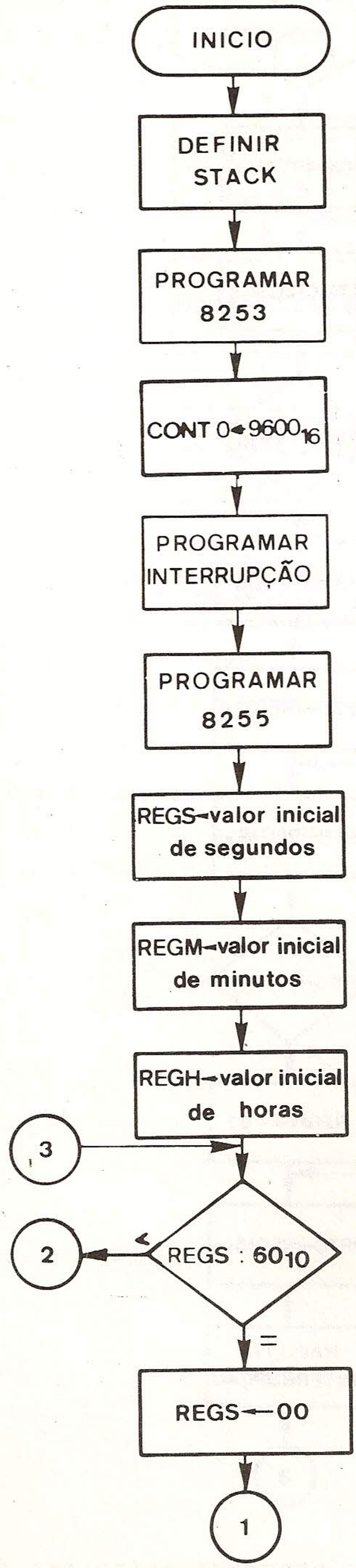
O gate do contador um deve ter nível lógico "1" e já esta conectado a V_{CC}.

Para o sinal de clock do contador, vamos selecionar entre os sinais disponíveis na saída dos divisores, o sinal de freqüência igual a 38.400Hz. O valor inicial do contador deve ser igual a 38.400_{10} (9600_{16}) para que ao final da contagem tenha transcorrido um tempo igual a um segundo. Nestas condições, devemos ligar a saída D dos divisores à entrada de clock do contador zero (pino 9 da 8253, fig. XIII.11, Volume 1).

Somente para exemplo, ao invés de utilizarmos alguns registradores disponíveis da rede de registradores, vamos utilizar três posições de memórias, como registradores para armazenamento das informações de horas, minutos e segundos. A rotina de interrupção deve incrementar o registrador de segundo a cada vez que ocorre uma interrupção.

Com estas considerações podemos passar ao fluxograma do programa principal, representado na fig. VI.23, e o fluxograma da rotina de interrupção, representado na fig. VI.24.

O programa final está mostrado na fig. VI.25.


INICIO
DEFINIR STACK
PROGRAMAR 8253
CONT 0 ← 9600₁₆
PROGRAMAR INTERRUPÇÃO
PROGRAMAR 8255
REGS ← valor inicial de segundos
REGM ← valor inicial de minutos
REGH ← valor inicial de horas
3
REGS : 60₁₀
2
<
=
REGS ← 00
1

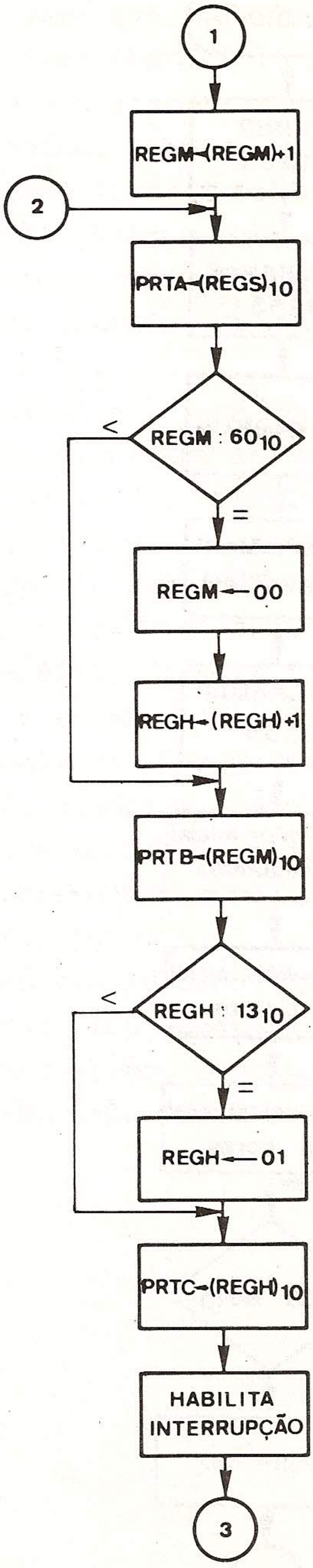


Fig. VI.23 - Fluxograma do programa principal do relógio.

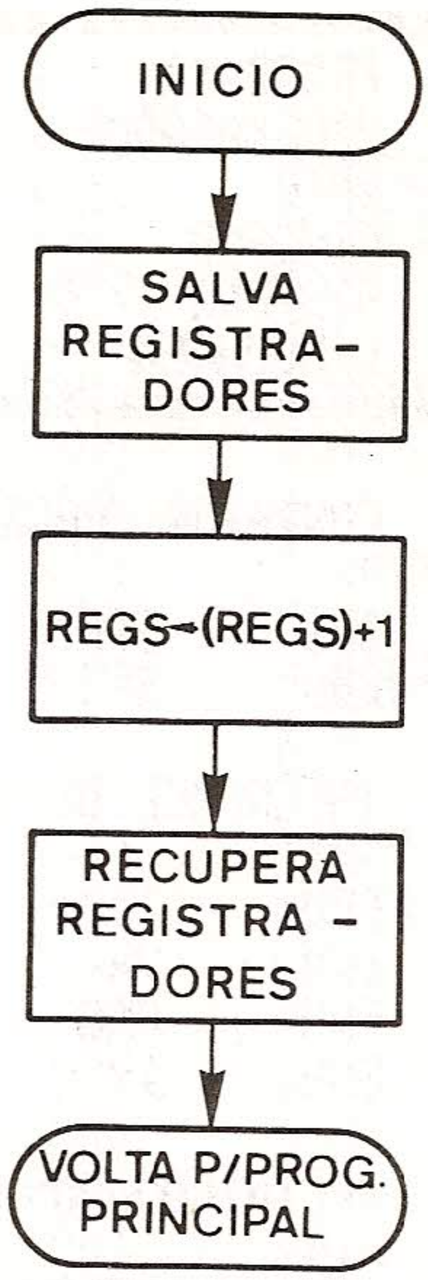


Fig. I.24 - Rotina de interrupção do programa relógio.

```
0010    ;*************************************************************
0020    ;*****  PROGRAMA     : RELOGIO                              *****
0030    ;*****  PROGRAMADOR  : IDOETA                               *****
0040    ;*****  DATA         : 9/OUT/80                             *****
0050    ;*****  REVISAO      : A                                    *****
0060    ;*****  SISTEMA      : 8085 PROCESSOR                       *****
0070    ;*****  LINGUAGEM    : ASSEMBLY 8085                        *****
0080    ;*************************************************************
0090
0100    ;*****  CONTADOR PROGRAMAVEL 8253                          *****
0110
0120    CTR53   EQU     5FH     ;PALAVRA DE CONTROLE
0130    CTDO    EQU     5CH     ;CONTADOR ZERO
0140
0150    ;*****  INTERFACE DE COMUNICACAO PARALELA 8255      *****
0160
0170    CTR55   EQU     1FH     ;PALAVRA DE CONTROLE
0180    PRTA    EQU     7H      ;PORTA A
0190    PRTB    EQU     0FH     ;PORTA B
0200    PRTC    EQU     17H     ;PORTA C
0210
0220    ;*****  DEFINICAO DAS CONSTANTES                           *****
0230
0240    STACK   EQU     43FFH   ;INICIO DO STACK
0250    REGS    EQU     4000H   ;REGISTRADOR DE SEGUNDOS
0260    REGM    EQU     4001H   ;REGISTRADOR DE MINUTOS
0270    REGH    EQU     4002H   ;REGISTRADOR DE HORAS
0280    VALS    EQU     00H     ;VALOR INICIAL DE SEGUNDOS
0290    VALM    EQU     00H     ;VALOR INICIAL DE MINUTOS
0300    VALH    EQU     12D     ;VALOR INICIAL DE HORAS
0310
0320    ;*****  ROTINA DE INTERRUPCAO                              *****
0330
0340            ORG     003CH   ;ENDERECO INICIAL RST 7.5
003C    F5      0350            PUSH    PSW     ;SALVA ACUMULADOR E FLAGS
003D    E5      0360            PUSH    H       ;SALVA HL
003E    210040  0370            LXI     H,REGS  ;SELECIONA REGIST. DE SEGUNDOS
0041    34      0380            INR     M       ;INCREMENTA REGIST. DE SEGUNDOS
0042    E1      0390            POP     H       ;REGENERA HL
0043    F1      0400            POP     PSW     ;REGENERA ACUMULADOR E FLAGS
0044    C9      0410            RET             ;VOLTA PARA PROGRAMA PRINCIPAL
0420
0430    ;*****  PROGRAMA PRINCIPAL                                 *****
0440
0450            ORG     0100H   ;ENDERECO INICIAL DO PROGRAMA
0100    21FF43  0460    INIC:   LXI     SP,STACK;DEFINE STACK POINTER
0103    3E3C    0470            MVI     A,3CH
0105    D35F    0480            OUT     CTR53   ;MODO 2 :CONTADOR ZERO
                0490            ;               DOIS BYTES REQUERIDOS
                0500            ;               CONTAGEM BINARIA
0107    3E00    0510            MVI     A,00H   ;CARREGA BYTE MENOS SIGNIFIC. /
0109    D35C    0520            OUT     CTDO    ;NO CONTADOR
010B    3E96    0530            MVI     A,96H   ;CARREGA BYTE MAIS SIGNIFIC. /
010D    D35C    0540            OUT     CTDO    ;NO CONTADOR
010F    3E1B    0550            MVI     A,1BH
0111    30      0560            SIM             ;HABILITA INTERRUPCAO RST 7.5
0112    3E80    0570            MVI     A,80H
```

```
Cont.
0114    D31F    0580            OUT     CTR55   ;MODO ZERO :A  SAIDA
                0590                            ;           B  SAIDA
                0600                            ;           CL SAIDA
                0610                            ;           CH SAIDA
0116    210040  0620            LXI     H,REGS  ;SELECIONA REGIST. DE SEGUNDOS
0119    3E00    0630            MVI     A,VALS
011B    77      0640            MOV     M,A     ;CARREGA VALOR INICIAL DE SEG.
011C    23      0650            INX     H       ;SELECIONA REGIST. DE MINUTOS
011D    3E00    0660            MVI     A,VALM
011F    77      0670            MOV     M,A     ;CARREGA VALOR INICIAL DE MIN.
0120    23      0680            INX     H       ;SELECIONA REGIST. DE HORAS
0121    3E0C    0690            MVI     A,VALH
0123    77      0700            MOV     M,A     ;CARREGA VALOR INICIAL DE HORAS
0124    210040  0710    CICLO:  LXI     H,REGS  ;SELECIONA REGIST. DE SEGUNDOS
0127    7E      0720            MOV     A,M
0128    FE3C    0730            CPI     60D     ;SEGUNDOS = 60?
012A    C23301  0740            JNZ     DISPS   ;SE NAO, OUT SEGUNDOS
012D    3600    0750            MVI     M,00H   ;SE IGUAL, ZERA SEGUNDOS
012F    23      0760            INX     H       ;SELECIONA REGIST. DE MINUTOS
0130    34      0770            INR     M       ;INCREMENTA REGIST. DE MINUTOS
0131    2B      0780            DCX     H       ;SELECIONA REGIST. DE SEGUNDOS
0132    7E      0790            MOV     A,M     ;RECUPERA VALOR DE SEGUNDOS
0133    27      0800    DISPS:  DAA             ;AJUSTA PARA DECINAL
0134    D307    0810            OUT     PRTA    ;SAIDA DE SEGUNDOS
0136    23      0820            INX     H       ;SELECIONA REGIST. DE MINUTOS
0137    7E      0830            MOV     A,M
0138    FE3C    0840            CPI     60D     ;MINUTOS = 60?
013A    C24301  0850            JNZ     DISPM   ;SE NAO, OUT MINUTOS
013D    3600    0860            MVI     M,00H   ;SE IGUAL ZERA MINUTOS
013F    23      0870            INX     H       ;SELECIONA REGIST. DE HORAS
0140    34      0880            INR     M       ;INCREMENTA REGIST. DE HORAS
0141    2B      0890            DCX     H       ;SELECIONA REGIST. DE MINUTOS
0142    7E      0900            MOV     A,M     ;RECUPERA VALOR DE MINUTOS
0143    27      0910    DISPM:  DAA             ;AJUSTA PARA DECIMAL
0144    D30F    0920            OUT     PRTB    ;SAIDA DE MINUTOS
0146    23      0930            INX     H       ;SELECIONA REGIST. DE HORAS
0147    7E      0940            MOV     A,M
0148    FE0D    0950            CPI     13D     ;HORAS = 13?
014A    C25001  0960            JNZ     DISPH   ;SE NAO, OUT HORAS
014D    3E01    0970            MVI     A,1H
014F    77      0980            MOV     M,A     ;HORAS IGUAL A UM
0150    27      0990    DISPH:  DAA             ;AJUSTA PARA DECIMAL
0151    D31F    1000            OUT     PRTC    ;SAIDA DE HORAS
0153    FB      1010            EI              ;HABILITA INTERRUPCOES
0154    C32401  1020            JMP     CICLO   ;VOLTA AO COMECO
                1030            END

SYMBOL TABLE

CICLO   0124    CTDO    005C    CTR53   005F    CTR55   001F    DISPH   0150
DISPM   0143    DISPS   0133    INIC    0100    PRTA    0007    PRTB    000F
PRTC    0017    REGH    4002    REGM    4001    REGS    4000    STACK   43FF
VALH    000C    VALM    0000    VALS    0000

ERRORS = 0
```

Fig. VI.25 - Relógio.

Este programa completa a apresentação das principais técnicas de programação. Aconselha-se o estudo minucioso de cada passo de todos os programas. Independente disto, vale a pena salientar que a maneira mais eficaz para o aprendizado de programação é através da prática.

Neste ponto, a base necessária para utilização de microprocessadores já foi apresentada, basta, portanto, colocá-la em prática.

ANEXO A

A

TABELA DE CONVERSÃO DOS SISTEMAS DE NUMERAÇÃO.

DECIMAL	BINÁRIO	OCTAL	HEXADECIMAL
000	0000 0000	000	00
001	0000 0001	001	01
002	0000 0010	002	02
003	0000 0011	003	03
004	0000 0100	004	04
005	0000 0101	005	05
006	0000 0110	006	06
007	0000 0111	007	07
008	0000 1000	010	08
009	0000 1001	011	09
010	0000 1010	012	0A
011	0000 1011	013	0B
012	0000 1100	014	0C
013	0000 1101	015	0D
014	0000 1110	016	0E
015	0000 1111	017	0F
016	0001 0000	020	10
017	0001 0001	021	11
018	0001 0010	022	12
019	0001 0011	023	13
020	0001 0100	024	14
021	0001 0101	025	15
022	0001 0110	026	16
023	0001 0111	027	17
024	0001 1000	030	18
025	0001 1001	031	19
026	0001 1010	032	1A
027	0001 1011	033	1B
028	0001 1100	034	1C
029	0001 1101	035	1D
030	0001 1110	036	1E
031	0001 1111	037	1F

DECIMAL	BINÁRIO	OCTAL	HEXADECIMAL
032	0010 0000	040	20
033	0010 0001	041	21
034	0010 0010	042	22
035	0010 0011	043	23
036	0010 0100	044	24
037	0010 0101	045	25
038	0010 0110	046	26
039	0010 0111	047	27
040	0010 1000	050	28
041	0010 1001	051	29
042	0010 1010	052	2A
043	0010 1011	053	2B
044	0010 1100	054	2C
045	0010 1101	055	2D
046	0010 1110	056	2E
047	0010 1111	057	2F
048	0011 0000	060	30
049	0011 0001	061	31
050	0011 0010	062	32
051	0011 0011	063	33
052	0011 0100	064	34
053	0011 0101	065	35
054	0011 0110	066	36
055	0011 0111	067	37
056	0011 1000	070	38
057	0011 1001	071	39
058	0011 1010	072	3A
059	0011 1011	073	3B
060	0011 1100	074	3C
061	0011 1101	075	3D
062	0011 1110	076	3E
063	0011 1111	077	3F
064	0100 0000	100	40
065	0100 0001	101	41
066	0100 0010	102	42

DECIMAL	BINÁRIO	OCTAL	HEXADECIMAL
067	0100 0011	103	43
068	0100 0100	104	44
069	0100 0101	105	45
070	0100 0110	106	46
071	0100 0111	107	47
072	0100 1000	110	48
073	0100 1001	111	49
074	0100 1010	112	4A
075	0100 1011	113	4B
076	0100 1100	114	4C
077	0100 1101	115	4D
078	0100 1110	116	4E
079	0100 1111	117	4F
080	0101 0000	120	50
081	0101 0001	121	51
082	0101 0010	122	52
083	0101 0011	123	53
084	0101 0100	124	54
085	0101 0101	125	55
086	0101 0110	126	56
087	0101 0111	127	57
088	0101 1000	130	58
089	0101 1001	131	59
090	0101 1010	132	5A
091	0101 1011	133	5B
092	0101 1100	134	5C
093	0101 1101	135	5D
094	0101 1110	136	5E
095	0101 1111	137	5F
096	0110 0000	140	60
097	0110 0001	141	61
098	0110 0010	142	62
099	0110 0011	143	63
100	0110 0100	144	64
101	0110 0101	145	65
102	0110 0110	146	66

DECIMAL	BINÁRIO	OCTAL	HEXADECIMAL
103	0110 0111	147	67
104	0110 1000	150	68
105	0110 1001	151	69
106	0110 1010	152	6A
107	0110 1011	153	6B
108	0110 1100	154	6C
109	0110 1101	155	6D
110	0110 1110	156	6E
111	0110 1111	157	6F
112	0111 0000	160	70
113	0111 0001	161	71
114	0111 0010	162	72
115	0111 0011	163	73
116	0111 0100	164	74
117	0111 0101	165	75
118	0111 0110	166	76
119	0111 0111	167	77
120	0111 1000	170	78
121	0111 1001	171	79
122	0111 1010	172	7A
123	0111 1011	173	7B
124	0111 1100	174	7C
125	0111 1101	175	7D
126	0111 1110	176	7E
127	0111 1111	177	7F
128	1000 0000	200	80
129	1000 0001	201	81
130	1000 0010	202	82
131	1000 0011	203	83
132	1000 0100	204	84
133	1000 0101	205	85
134	1000 0110	206	86
135	1000 0111	207	87
136	1000 1000	210	88
137	1000 1001	211	89

DECIMAL	BINÁRIO	OCTAL	HEXADECIMAL
138	1000 1010	212	8A
139	1000 1011	213	8B
140	1000 1100	214	8C
141	1000 1101	215	8D
142	1000 1110	216	8E
143	1000 1111	217	8F
144	1001 0000	220	90
145	1001 0001	221	91
146	1001 0010	222	92
147	1001 0011	223	93
148	1001 0100	224	94
149	1001 0101	225	95
150	1001 0110	226	96
151	1001 0111	227	97
152	1001 1000	230	98
153	1001 1001	231	99
154	1001 1010	232	9A
155	1001 1011	233	9B
156	1001 1100	234	9C
157	1001 1101	235	9D
158	1001 1110	236	9E
159	1001 1111	237	9F
160	1010 0000	240	A0
161	1010 0001	241	A1
162	1010 0010	242	A2
163	1010 0011	243	A3
164	1010 0100	244	A4
165	1010 0101	245	A5
166	1010 0110	246	A6
167	1010 0111	247	A7
168	1010 1000	250	A8
169	1010 1001	251	A9
170	1010 1010	252	AA
171	1010 1011	253	AB
172	1010 1100	254	AC

DECIMAL	BINÁRIO	OCTAL	HEXADECIMAL
173	1010 1101	255	AD
174	1010 1110	256	AE
175	1010 1111	257	AF
176	1011 0000	260	B0
177	1011 0001	261	B1
178	1011 0010	262	B2
179	1011 0011	263	B3
180	1011 0100	264	B4
181	1011 0101	265	B5
182	1011 0110	266	B6
183	1011 0111	267	B7
184	1011 1000	270	B8
185	1011 1001	271	B9
186	1011 1010	272	BA
187	1011 1011	273	BB
188	1011 1100	274	BC
189	1011 1101	275	BD
190	1011 1110	276	BE
191	1011 1111	277	BF
192	1100 0000	300	C0
193	1100 0001	301	C1
194	1100 0010	302	C2
195	1100 0011	303	C3
196	1100 0100	304	C4
197	1100 0101	305	C5
198	1100 0110	306	C6
199	1100 0111	307	C7
200	1100 1000	310	C8
201	1100 1001	311	C9
202	1100 1010	312	CA
203	1100 1011	313	CB
204	1100 1100	314	CC
205	1100 1101	315	CD
206	1100 1110	316	CE
207	1100 1111	317	CF

DECIMAL	BINÁRIO	OCTAL	HEXADECIMAL
208	1101 0000	320	D0
209	1101 0001	321	D1
210	1101 0010	322	D2
211	1101 0011	323	D3
212	1101 0100	324	D4
213	1101 0101	325	D5
214	1101 0110	326	D6
215	1101 0111	327	D7
216	1101 1000	330	D8
217	1101 1001	331	D9
218	1101 1010	332	DA
219	1101 1011	333	DB
220	1101 1100	334	DC
221	1101 1101	335	DD
222	1101 1110	336	DE
223	1101 1111	337	DF
224	1110 0000	340	E0
225	1110 0001	341	E1
226	1110 0010	342	E2
227	1110 0011	343	E3
228	1110 0100	344	E4
229	1110 0101	345	E5
230	1110 0110	346	E6
231	1110 0111	347	E7
232	1110 1000	350	E8
233	1110 1001	351	E9
234	1110 1010	352	EA
235	1110 1011	353	EB
236	1110 1100	354	EC
237	1110 1101	355	ED
238	1110 1110	356	EE
239	1110 1111	357	EF
240	1111 0000	360	F0
241	1111 0001	361	F1
242	1111 0010	362	F2

DECIMAL	BINÁRIO	OCTAL	HEXADECIMAL
243	1111 0011	363	F3
244	1111 0100	364	F4
245	1111 0101	365	F5
246	1111 0110	366	F6
247	1111 0111	367	F7
248	1111 1000	370	F8
249	1111 1001	371	F9
250	1111 1010	372	FA
251	1111 1011	373	FB
252	1111 1100	374	FC
253	1111 1101	375	FD
254	1111 1110	376	FE
255	1111 1111	377	FF

EXPONENCIAIS

2^n	n
1	0
2	1
4	2
8	3
16	4
32	5
64	6
128	7
256	8
512	9
1 024	10
2 048	11
4 096	12
8 192	13
16 384	14
32 768	15
65 536	16
131 072	17
262 144	18
524 288	19
1 048 576	20
2 097 152	21
4 194 304	22
8 388 608	23
16 777 216	24
33 554 432	25
67 108 864	26
134 217 728	27
268 435 456	28
536 870 912	29
1 073 741 824	30

16^n	n
1	0
16	1
256	2
4 096	3
65 536	4
1 048 576	5
16 777 216	6
268 435 456	7
4 294 967 296	8
68 719 476 736	9
1 099 511 627 776	10

ANEXO C *C*

TABELA DE EQUIVALÊNCIA HEXADECIMAL E ASCII (AMERICAN STANDARD CODE FOR INFORMATION INTER CHANGE).

HEXA	ASCII	HEXA	ASCII	HEXA	ASCII	HEXA	ASCII
00	NUL	20	SP	40	@	60	\`
01	SOH	21	!	41	A	61	a
02	STX	22	"	42	B	62	b
03	ETX	23	#	43	C	63	c
04	EOT	24	$	44	D	64	d
05	ENQ	25	%	45	E	65	e
06	ACK	26	&	46	F	66	f
07	BEL	27	'	47	G	67	g
08	BS	28	(	48	H	68	h
09	HT	29	)	49	I	69	i
0A	LF	2A	*	4A	J	6A	j
0B	VT	2B	+	4B	K	6B	k
0C	FF	2C	,	4C	L	6C	l
0D	CR	2D	-	4D	M	6D	m
0E	S0	2E	.	4E	N	6E	n
0F	SI	2F	/	4F	O	6F	o
10	DLE	30	0	50	P	70	p
11	DC1	31	1	51	Q	71	q
12	DC2	32	2	52	R	72	r
13	DC3	33	3	53	S	73	s
14	DC4	34	4	54	T	74	t
15	NAK	35	5	55	U	75	u
16	SYN	36	6	56	V	76	v
17	ETB	37	7	57	W	77	w
18	CAN	38	8	58	X	78	x
19	EM	39	9	59	Y	79	y
1A	SUB	3A	:	5A	Z	7A	z
1B	ESC	3B	;	5B	[	7B	{
1C	FS	3C	<	5C	\	7C	|
1D	GS	3D	=	5D	]	7D	}
1E	RS	3E	>	5E	^	7E	~
1F	US	3F	?	5F	-	7F	DEL

CONJUNTO DE INSTRUÇÕES DOS MICROPROCESSADORES 8080 E 8085 EM ORDEM NUMÉRICA.

HEXA	C.OP.	OPER.	NUMERO DE BYTES	NUMERO DE ESTADOS	FLAGS AFETADAS
00	NOP		1	4	-
01	LXI	B,dado	3	10	-
02	STAX	B	1	7	-
03	INX	B	1	(5)6	-
04	INR	B	1	(5)4	Z,S,P, AC
05	DCR	B	1	(5)4	Z,S,P, AC
06	MVI	B,dado	2	7	-
07	RLC		1	4	C
09	DAD	B	1	10	C
0A	LDAX	B	1	7	-
0B	DCX	B	1	(5)6	-
0C	INR	C	1	(5)4	Z,S,P, AC
0D	DCR	C	1	(5)4	Z,S,P, AC
0E	MVI	C,dado	2	7	-
0F	RRC		1	4	C
11	LXT	D,dado	3	10	-
12	STAX	D	1	7	-
13	INX	D	1	(5)6	-
14	INR	D	1	(5)4	Z,S,P, AC
15	DCR	D	1	(5)4	Z,S,P, AC
16	MVI	D,dado	2	7	-
17	RAL		1	4	C
19	DAD	D	1	10	C
1A	LDAX	D	1	7	-
1B	DCX	D	1	(5)6	-
1C	INR	E	1	(5)4	Z,S,P, AC
1D	DCR	E	1	(5)4	Z,S,P, AC

HEXA	C. OP.	OPER.	NÚMERO DE BYTES	NÚMERO DE ESTADOS	FLAGS AFETADAS
1E	MVI	E,dado	2	7	-
1F	RAR		1	4	C
20	RIM		1	4	-
21	LXI	H,dado	3	10	-
22	SHLD	endereço	3	16	-
23	INX	H	1	(5)6	-
24	INR	H	1	(5)4	Z,S,P, AC
25	DCR	H	1	(5)4	Z,S,P, AC
26	MVI	H,dado	2	7	-
27	DAA		1	4	Z,S,P,C,AC
29	DAD	H	1	10	C
2A	LHLD	endereço	3	16	-
2B	DCX	H	1	(5)6	-
2C	INR	L	1	(5)4	Z,S,P, AC
2D	DCR	L	1	(5)4	Z,S,P AC
2E	MVI	L,dado	2	7	-
2F	CMA		1	4	-
30	SIM		1	4	-
31	LXI	SP,dado	3	10	-
32	STA	endereço	3	13	-
33	INX	SP	1	(5)6	-
34	INR	M	1	10	Z,S,P, AC
35	DCR	M	1	10	Z,S,P, AC
36	MVI	M,dado	2	10	-
37	STC		1	4	C
39	DAD	SP	1	10	C
3A	LDA	endereço	3	13	-
3B	DCX	SP	1	(5)6	-
3C	INR	A	1	(5)4	Z,S,P AC
3D	DCR	A	1	(5)4	Z,S,P AC
3E	MVI	A,dado	2	7	-
3F	CMC		1	4	C
40	MOV	B,B	1	(5)4	-
41	MOV	B,C	1	(5)4	-

HEXA	C.OP.	OPER.	NÚMERO DE BYTES	NÚMERO DE ESTADOS	FLAGS AFETADAS
42	MOV	B,D	1	(5)4	-
43	MOV	B,E	1	(5)4	-
44	MOV	B,H	1	(5)4	-
45	MOV	B,L	1	(5)4	-
46	MOV	B,M	1	7	-
47	MOV	B,A	1	(5)4	-
48	MOV	C,B	1	(5)4	-
49	MOV	C,C	1	(5)4	-
4A	MOV	C,D	1	(5)4	-
4B	MOV	C,E	1	(5)4	-
4C	MOV	C,H	1	(5)4	-
4D	MOV	C,L	1	(5)4	-
4E	MOV	C,M	1	7	-
4F	MOV	C,A	1	(5)4	-
50	MOV	D,B	1	(5)4	-
51	MOV	D,C	1	(5)4	-
52	MOV	D,D	1	(5)4	-
53	MOV	D,E	1	(5)4	-
54	MOV	D,H	1	(5)4	-
55	MOV	D,L	1	(5)4	-
56	MOV	D,M	1	7	-
57	MOV	D,A	1	(5)4	-
58	MOV	E,B	1	(5)4	-
59	MOV	E,C	1	(5)4	-
5A	MOV	E,D	1	(5)4	-
5B	MOV	E,E	1	(5)4	-
5C	MOV	E,H	1	(5)4	-
5D	MOV	E,L	1	(5)4	-
5E	MOV	E,M	1	7	-
5F	MOV	E,A	1	(5)4	-
60	MOV	H,B	1	(5)4	-
61	MOV	H,C	1	(5)4	-
62	MOV	H,D	1	(5)4	-
63	MOV	H,E	1	(5)4	-
64	MOV	H,H	1	(5)4	-

HEXA	C.OP.	OPER.	NÚMERO DE BYTES	NÚMERO DE ESTADOS	FLAGS AFETADAS
65	MOV	H,L	1	(5)4	-
66	MOV	H,M	1	7	-
67	MOV	H,A	1	(5)4	-
68	MOV	L,B	1	(5)4	-
69	MOV	L,C	1	(5)4	-
6A	MOV	L,D	1	(5)4	-
6B	MOV	L,E	1	(5)4	-
6C	MOV	L,H	1	(5)4	-
6D	MOV	L,L	1	(5)4	-
6E	MOV	L,M	1	7	-
6F	MOV	L,A	1	(5)4	-
70	MOV	M,B	1	7	-
71	MOV	M,C	1	7	-
72	MOV	M,D	1	7	-
73	MOV	M,E	1	7	-
74	MOV	M,H	1	7	-
75	MOV	M,L	1	7	-
76	HLT		1	(7)5	-
77	MOV	M,A	1	7	-
78	MOV	A,B	1	(5)4	-
79	MOV	A,C	1	(5)4	-
7A	MOV	A,D	1	(5)4	-
7B	MOV	A,E	1	(5)4	-
7C	MOV	A,H	1	(5)4	-
7D	MOV	A,L	1	(5)4	-
7E	MOV	A,M	1	7	-
7F	MOV	A,A	1	(5)4	-
80	ADD	B	1	4	Z,S,P,C,AC
81	ADD	C	1	4	Z,S,P,C,AC
82	ADD	D	1	4	Z,S,P,C,AC
83	ADD	E	1	4	Z,S,P,C,AC
84	ADD	H	1	4	Z,S,P,C,AC
85	ADD	L	1	4	Z,S,P,C,AC
86	ADD	M	1	7	Z,S,P,C,AC

D

HEXA	C.OP.	OPER.	NÚMERO DE BYTES	NÚMERO DE ESTADOS	FLAGS AFETADAS
87	ADD	A	1	4	Z,S,P,C,AC
88	ADC	B	1	4	Z,S,P,C,AC
89	ADC	C	1	4	Z,S,P,C,AC
8A	ADC	D	1	4	Z,S,P,C,AC
8B	ADC	E	1	4	Z,S,P,C,AC
8C	ADC	H	1	4	Z,S,P,C,AC
8D	ADC	L	1	4	Z,S,P,C,AC
8E	ADC	M	1	7	Z,S,P,C,AC
8F	ADC	A	1	4	Z,S,P,C,AC
90	SUB	B	1	4	Z,S,P,C,AC
91	SUB	C	1	4	Z,S,P,C,AC
92	SUB	D	1	4	Z,S,P,C,AC
93	SUB	E	1	4	Z,S,P,C,AC
94	SUB	H	1	4	Z,S,P,C,AC
95	SUB	L	1	4	Z,S,P,C,AC
96	SUB	M	1	7	Z,S,P,C,AC
97	SUB	A	1	4	Z,S,P,C,AC
98	SBB	B	1	4	Z,S,P,C,AC
99	SBB	C	1	4	Z,S,P,C,AC
9A	SBB	D	1	4	Z,S,P,C,AC
9B	SBB	E	1	4	Z,S,P,C,AC
9C	SBB	H	1	4	Z,S,P,C,AC
9D	SBB	L	1	4	Z,S,P,C,AC
9E	SBB	M	1	7	Z,S,P,C,AC
9F	SBB	A	1	4	Z,S,P,C,AC
A0	ANA	B	1	4	Z,S,P,C,AC
A1	ANA	C	1	4	Z,S,P,C,AC
A2	ANA	D	1	4	Z,S,P,C,AC
A3	ANA	E	1	4	Z,S,P,C,AC
A4	ANA	H	1	4	Z,S,P,C,AC
A5	ANA	L	1	4	Z,S,P,C,AC
A6	ANA	M	1	7	Z,S,P,C,AC
A7	ANA	A	1	4	Z,S,P,C,AC
A8	XRA	B	1	4	Z,S,P,C,AC
A9	XRA	C	1	4	Z,S,P,C,AC

D

HEXA	C.OP.	OPER.	NÚMERO DE BYTES	NÚMERO DE ESTADOS	FLAGS AFETADAS
AA	XRA	D	1	4	Z,S,P,C,AC
AB	XRA	E	1	4	Z,S,P,C,AC
AC	XRA	H	1	4	Z,S,P,C,AC
AD	XRA	L	1	4	Z,S,P,C,AC
AE	XRA	M	1	7	Z,S,P,C,AC
AF	XRA	A	1	4	Z,S,P,C,AC
B0	ORA	B	1	4	Z,S,P,C,AC
B1	ORA	C	1	4	Z,S,P,C,AC
B2	ORA	D	1	4	Z,S,P,C,AC
B3	ORA	E	1	4	Z,S,P,C,AC
B4	ORA	H	1	4	Z,S,P,C,AC
B5	ORA	L	1	4	Z,S,P,C,AC
B6	ORA	M	1	7	Z,S,P,C,AC
B7	ORA	A	1	4	Z,S,P,C,AC
B8	CMP	B	1	4	Z,S,P,C,AC
B9	CMP	C	1	4	Z,S,P,C,AC
BA	CMP	D	1	4	Z,S,P,C,AC
BB	CMP	E	1	4	Z,S,P,C,AC
BC	CMP	H	1	4	Z,S,P,C,AC
BD	CMP	L	1	4	Z,S,P,C,AC
BE	CMP	M	1	7	Z,S,P,C,AC
BF	CMP	A	1	4	Z,S,P,C,AC
C0	RNZ		1	(5/11)6/12	-
C1	POP	B	1	10	-
C2	JNZ	endereço	3	(10)7/10	-
C3	JMP	endereço	3	10	-
C4	CNZ	endereço	3	(11/17)9/18	-
C5	PUSH	B	1	(11)13	-
C6	ADI	dado	2	7	Z,S,P,C,AC
C7	RST	0	1	(11)12	-
C8	RZ		1	(5/11)6/12	-
C9	RET		1	10	-
CA	JZ	endereço	3	(10)7/10	-

HEXA	C.OP.	OPER.	NÚMERO DE BYTES	NÚMERO DE ESTADOS	FLAGS AFETADAS
CC	CZ	endereço	3	(11/17) 9/18	-
CD	CALL	endereço	3	(17) 18	-
CE	ACI	dado	2	7	Z,S,P,C,AC
CF	RST	1	1	(11) 12	-
D0	RNC		1	(5/11) 6/12	-
D1	POP	D	1	10	-
D2	JNC	endereço	3	(10) 7/10	-
D3	OUT	endereço	2	10	-
D4	CNC	endereço	3	(11/17) 9/18	-
D5	PUSH	D	1	(11) 13	-
D6	SUI	dado	2	7	Z,S,P,C,AC
D7	RST	2	1	(11) 12	-
D8	RC		1	(5/11) 6/12	-
DA	JC	endereço	3	(10) 7/10	-
DB	IN	endereço	2	10	-
DC	CC	endereço	3	(11/17) 9/18	-
DE	SBI	dado	2	7	Z,S,P,C,AC
DF	RST	3	1	(11) 12	-
E0	RPO		1	(5/11) 6/12	-
E1	POP	H	1	10	-
E2	JPO	endereço	3	(10) 7/10	-
E3	XTHL		1	(18) 16	-
E4	CPO	endereço	3	(11/17) 9/18	-
E5	PUSH	H	1	11(13)	-
E6	ANI	dado	2	7	Z,S,P,C,AC
E7	RST	4	1	(11) 12	-
E8	RPE		1	(5/11) 6/12	-
E9	PCHL		1	(5) 6	-
EA	JPE	endereço	3	(10) 7/10	-
EB	XCHG		1	4	-
EC	CPE	endereço	3	(11/17) 9/18	-

HEXA	C.OP.	OPER.	NÚMERO DE BYTES	NÚMERO DE ESTADOS	FLAGS AFETADAS
EE	XRI	dado	2	7	Z,S,P,C,AC
EF	RST	5	1	(11)12	-
F0	RP		1	(5/11)6/12	-
F1	POP	PSW	1	10	Z,S,P,C,AC
F2	JP	endereço	3	(10)7/10	-
F3	DI		1	4	-
F4	CP	endereço	3	(11/17)9/18	-
F5	PUSH	PSW	1	(11)12	-
F6	ORI	dado	2	7	Z,S,P,C,AC
F7	RST	6	1	(11)12	-
F8	RM		1	(5/11)6/12	-
F9	SPHL		1	(5)6	-
FA	JM	endereço	3	(10)7/10	-
FB	EI		1	4	-
FC	CM	endereço	3	(11/17)9/18	-
FE	CPI	dado	2	7	Z,S,P,C,AC
FF	RST	7	1	(11)12	-

CONJUNTO DE INSTRUÇÕES DOS MICROPROCESSADORES 8080 E 8085 SEPARADAS EM GRUPOS.

MOVE

MOV	B,B	40
MOV	B,C	41
MOV	B,D	42
MOV	B,E	43
MOV	B,H	44
MOV	B,L	45
MOV	B,M	46
MOV	B,A	47
MOV	C,B	48
MOV	C,C	49
MOV	C,D	4A
MOV	C,E	4B
MOV	C,H	4C
MOV	C,L	4D
MOV	C,M	4E
MOV	C,A	4F
MOV	D,B	50
MOV	D,C	51
MOV	D,D	52
MOV	D,E	53
MOV	D,H	54
MOV	D,L	55
MOV	D,M	56
MOV	D,A	57
MOV	E,B	58
MOV	E,C	59
MOV	E,D	5A
MOV	E,E	5B

MOV	E,H	5C
MOV	E,L	5D
MOV	E,M	5E
MOV	E,A	5F
MOV	H,B	60
MOV	H,C	61
MOV	H,D	62
MOV	H,E	63
MOV	H,H	64
MOV	H,L	65
MOV	H,M	66
MOV	H,A	67
MOV	L,B	68
MOV	L,C	69
MOV	L,D	6A
MOV	L,E	6B
MOV	L,H	6C
MOV	L,L	6D
MOV	L,M	6E
MOV	L,A	6F
MOV	M,B	70
MOV	M,C	71
MOV	M,D	72
MOV	M,E	73
MOV	M,H	74
MOV	M,L	75
MOV	M,A	77
MOV	A,B	78
MOV	A,C	79
MOV	A,D	7A
MOV	A,E	7B

MOVE

MOV	A,H	7C
MOV	A,L	7D
MOV	A,M	7E
MOV	A,A	7F

MOVE IMEDIATO

MVI	B, dado	06
MVI	C, dado	0E
MVI	D, dado	16
MVI	E, dado	1E
MVI	H, dado	26
MVI	L, dado	2E
MVI	M, dado	36
MVI	A, dado	3E

ACUMULADOR IMEDIATO

ADI	dado	C6
ACI	dado	CE
SUI	dado	D6
SBI	dado	DE
ANI	dado	E6
XRI	dado	EE
ORI	dado	F6
COI	dado	FE

ACUMULADOR

ADD	B	80
ADD	C	81
ADD	D	82
ADD	E	83
ADD	H	84
ADD	L	85

ACUMULADOR

ADD	M	86
ADD	A	87
ADC	B	88
ADC	C	89
ADC	D	8A
ADC	E	8B
ADC	H	8C
ADC	L	8D
ADC	M	8E
ADC	A	8F
SUB	B	90
SUB	C	91
SUB	D	92
SUB	E	93
SUB	H	94
SUB	L	95
SUB	M	96
SUB	A	97
SBB	B	98
SBB	C	99
SBB	D	9A
SBB	E	9B
SBB	H	9C
SBB	L	9D
SBB	M	9E
SBB	A	9F
ANA	B	A0
ANA	C	A1
ANA	D	A2
ANA	E	A3
ANA	H	A4

ACUMULADOR

ANA	L	A5
ANA	M	A6
ANA	A	A7
XRA	B	A8
XRA	C	A9
XRA	D	AA
XRA	E	AB
XRA	H	AC
XRA	L	AD
XRA	M	AE
XRA	A	AF
ORA	B	B0
ORA	C	B1
ORA	D	B2
ORA	E	B3
ORA	H	B4
ORA	L	B5
ORA	M	B6
ORA	A	B7
CMP	B	B8
CMP	C	B9
CMP	D	BA
CMP	E	BB
CMP	H	BC
CMP	L	BD
CMP	M	BE
CMP	A	BF

INCREMENTA

INR	B	04
INR	C	0C

INCREMENTA

INR	D	14
INR	E	1C
INR	H	24
INR	L	2C
INR	M	34
INR	A	3C
INX	B	03
INX	D	13
INX	H	23
INX	SP	33

DECREMENTA

DCR	B	05
DCR	C	0D
DCR	D	15
DCR	E	1D
DCR	H	25
DCR	L	2D
DCR	M	35
DCR	A	3D
DCX	B	0B
DCX	D	1B
DCX	H	2B
DCX	SP	3B

SALTO

JMP	endereço	C3
JNZ	endereço	C2
JZ	endereço	CA
JNC	endereço	D2
JC	endereço	DA

JPO	endereço	E2
JPE	endereço	EA
JP	endereço	F2
JM	endereço	FA
PCHL		E9

RETORNO DE SUB-ROTINA

RET	C9
RNZ	C0
RZ	C8
RNC	D0
RC	D8
RPO	E0
RPE	E8
RP	F0
RM	F8

CARREGA

LDAX	B	0A
LDAX	D	1A
LHLD	endereço	2A
LDA	endereço	3A

ARMAZENA

STAX	B	02
STAX	D	12
SHLD	endereço	22
STA	endereço	32

OPERAÇÃO COM STACK

PUSH B	C5
PUSH D	D5
PUSH H	E5

OPERAÇÃO COM STACK

PUSH PSW	F5
POP B	C1
POP D	D1
POP H	E1
POP PSW	F1
XTHL	E3
SPHL	F9

CHAMA SUB-ROTINA

CALL	endereço	CD
CNZ	endereço	C4
CZ	endereço	CC
CNC	endereço	D4
CC	endereço	DC
CPO	endereço	E4
CPE	endereço	EC
CP	endereço	F4
CM	endereço	FC

RST 0	C7
RST 1	CF
RST 2	D7
RST 3	DF
RST 4	E7
RST 5	EF
RST 6	F7
RST 7	FF

CARREGA IMEDIATO

LXI B, dado	01
LXI D, dado	11

CARREGA IMEDIATO

LXI H, dado	21
LXI SP, dado	31

ESPECIAIS

XCHG	EB
DAA	27
CMA	2F
STC	37
CMC	3F

GIRAR ACUMULADOR

RLC	07
RRC	0F
RAL	17
RAR	lF

SOMA PAR DE REGISTRADORES

DAD B	09
DAD D	19
DAD H	29
DAD SP	39

ENTRADA/SAÍDA

IN	endereço	D3
OUT	endereço	DB

CONTROLE

NOP	00
SIM	20
RIM	30

CONTROLE

HLT	76
DI	F3
EI	FB

PSEUDO INSTRUÇÕES

ORG	endereço
EQU	dado
SET	dado
DB	dado
DS	dado
END	
SPC	

BIBLIOGRAFIA

8080A/8085 - ASSEMBLY LANGUAGE PROGRAMMING
Lance A. Leventhal, Aorborne/McGraw-Hill, Inc.

8080/8085 - ASSEMBLY LANGUAGE PROGRAMMING MANUAL
Intel Corporation.

COMPONENT DATA CATALOG, 1979
Intel Corporation.

DIGITAL COMPUTER FUNDAMENTALS
Yaohan Chu, McGraw-Hill Book Company.

ELETRONICS ENGINEERS HANDBOOK
Donald G. Fink, McGraw-Hill Book Company.

FORTRAN - MONITOR PRINCÍPIOS
Tércio Pacitti, Ao Livro Técnico S.A.

INTRODUÇÃO AOS MICROCOMPUTADORES
Joaquim A. Moura Relvas, Figueirinhas.

MANUAL DE INSTRUÇÕES DE MONTAGEM SIKIT - DK 80
Siemens A.G.

MANUAL DE SISTEMAS, SDD - G80/85
Gepeto Eletrônica Ltda.

MICROPROCESSOR DEVICES - DATA BOOK 1976/1977
Siemens A.G.

MICROCOMPUTER EXPERIMENTATION WITH THE INTEL SDK 85
Lance Leventhal/Colin Walsh, Prentice-Hall, Inc.

PROJETO DE COMPUTADORES DIGITAIS
Glen George Langdon Jr./Edson Fregni, Edgard Blücher Ltda.

Impresso em off-set por

EDITORA SANTUÁRIO

Rua Pe. Claro Monteiro, 342

Aparecida - São Paulo

Fone DDD (0125) 36-2140

com filmes fornecidos pelo editor.

PUBLICAÇÕES EM ELETRÔNICA

Rádio — Propagação

Envolve propagação de ondas longas até microondas, ondas ópticas e seus meios, bem como, atmosfera, guias de onda, fibras óticas e os métodos de Propagação, através de estudos da:

Reflexão, Refração, Zonas de Fresnel, Princípio de Huygens, Critérios de Rayleigh, Antena, Radar, Satélites, etc.

N.º de páginas: 168

Autor: **Jaroslav Smit**

Física — Volume 01

Aborda Funções e Gráficos; Grandezas e Medidas; Movimento Retilíneo Uniforme e Movimento Retilíneo Uniformemente Variado; Queda dos Corpos; Vetores; Velocidade Relativa; Composição de Movimentos; Leis de Newton; Energia.

Em cada capítulo contém uma série de exercícios resolvidos e exercícios propostos. O Apêndice contém dezesseis experiências propostas de fácil execução.

N.º de páginas: 304 1.ª Edição.

Autor: **Rocco Lence**

Eletrônica de potência

O livro aborda o estudo dos Conversores Estáticos, implementados com Tiristores. Seqüencialmente são tratados: classificação dos Conversores, em forma resumida e com uma análise detalhada, fixados com exemplos numéricos e, aplicação de Conversores no acionamento de motores elétricos.

N.º de páginas: 300 — 1.ª Edição

Autor: **José Luiz Antunes de Almeida**

PUBLICAÇÕES EM ELETRÔNICA

Elementos de Eletrônica Digital

Iniciação a Eletrônica Digital, Álgebra de Boole, Minimização de Funções Booleanas, Circuitos Contadores, Decodificadores, Multiplex, Demultiplex, Display, Registradores de Deslocamento, Desenvolvimento de Circuitos Lógicos, Circuitos Somadores/Subtratores e outros. 10.ª Edição, 512 páginas.
Autores: **Capuano / Idoeta.**

Teoria e Processo de Desenvolvimento em Eletrônica

Estudo e Associação de Bipolos Passivos e Ativos, Circuitos Ressonantes, Aparelhos de Medidas, Válvulas, Semicondutores, Leis de Kirchhoff, Teoremas de Thevenin e Norton, Osciloscópio e outros. 2.ª Edição, 260 páginas.
Autor: **Eng.º Sidnei David**

Microprocessadores 8080 e 8085 - Hardware - Vol. 1

Memórias RAM, ROM, PROM e EPROM, o 8224, 8228, 8080, 8085, 8255 e 8253, suas aplicações e montagem de um microprocessador. 5.ª Edição, 144 páginas.
Autor: **Eng.º Antonio Carlos José Franceschini Visconti**

Microprocessadores 8080 e 8085 - Software - Vol. 2

Estudo das instruções dos microprocessadores 8080 e 8085, Fluxogramas, iniciação a programação e desenvolvimento de programas com a utilização dos microprocessadores 8080 e 8085. 6.ª Edição, 208 páginas
Autor: **Eng.º Antonio Carlos José Franceschini Visconti**

Amplificador Operacional

Ideal e Real, em componentes discretos, Realimentação, Compensação, Buffer, Somadores, Detetor de Picos, Integrador, Gerador de Sinais, Amplificadores de Audio, Modulador, Sample-Hold, etc.

Possui cálculos e projetos de circuitos e salienta cuidados especiais.
3.ª Edição, 272 páginas.
Autores: **Eng.º Roberto Antonio Lando/Eng.º Serg Rios Alves**

Basic para Computadores Pessoais

Apresentação da Linguagem, com desenvolvimento detalhado das instruções, com uma série de exercícios resolvidos e propostos (com respostas), jogos e programas aplicativos. Linguagem simples e de aceitação pelos aficcionados no ramo dos microcomputadores.

3.ª Edição, 270 páginas.
Autor: **Arsonval Fleury Pereira**

Microprocessador Z-80 - Hardware - Vol. 1

Estudo dos Algoritmos, Arquitetura, Estrutura e Ciclo de Tempo do Microprocessador Z-80, CTC (contador), PIO (porto), Memórias 4801, 4802, 2732 Circuito de Clock, Reset, Teclado, Display e outros circuitos. 3.ª Edição, 196 páginas.
Autores: **Eng.º Luiz Benedito Cypriano**
Eng.º Paulo Roberto Cardinalli

Microprocessador Z-80 Software Vol. 2

Pesquisa do SET de Instruções do microprocessador Z-80. Tipos de Endereçamento. Tipo de Instrução. Fluxo de Dados. Interrupção. Linguagem de Máquina e Assembly. Pseudo-Instrução. Desenvolvimento de Programas.

Este livro também se destina à aplicação de micros pessoais que operam em linguagem de máquina. 3.ª Edição, 334 páginas.
Autor: **Eng.º Luiz Benedito Cypriano.**