PROGRAMANDO O

Z-80

NEY ACYR R. DE OLIVEIRA

ANDRÉ GIL RUBENS

O objetivo deste texto é apresentar de uma forma didática e ordenada as instruções do microprocessador Z-80 com inúmeras ilustrações e centenas de exemplos e exercícios envolvendo a linguagem Assembly do Z-80.

A filosofia que seguimos no desenvolvimento deste trabalho foi a de apresentar, junto com as instruções, exemplos de programas que crescem de complexidade à medida que novas instruções são estudadas.

Esperamos que este livro possa ser entendido, em toda a sua plenitude, tanto por estudantes como por profissionais e autodidatas das áreas de eletrônica e informática.

PROGRAMANDO O Z-80

NEY ACYR R. DE OLIVEIRA

AeN

ANDRÉ GIL RUBENS

PROGRAMANDO O Z-80

AUTORES: **ANDRÉ GIL RUBENS**
NEY ACYR R. DE OLIVEIRA

1º edição 1984

Composição:
STUDIO MORAES
Tel.: 256-0939 - 257-3478

Diagramação e Arte Final:
João R. Reis Jr.

Capa:
SERIARTE
Tel.: 232-5299

Revisão Final:
Selma Regina da Silva

CIP-Brasil. Catalogação-na-fonte
Sindicato Nacional dos Editores de Livros, RJ.

Oliveira, Ney Acyr Rodrigues de.
048p Programando o Z-80 / Ney Acyr Rodrigues de Oliveira e André Gil Rubens. — Rio de Janeiro : AeN Publicações, 1984.

Apêndices
Bibliografia
ISBN 85-85039-01-9

1. Microprocessadores - Programação I. Rubens, André Gil II. Título

84-0628 CDD - 001.642

Editado pela

A e N CONSULTORIA PROJETOS E PUBLICAÇÕES LTDA
Rua da Conceição 105 sala 1906 - Tel: 233-3838
Rio de Janeiro - RJ - CEP 20051
IMPRESSO NO BRASIL

Por mais hábil que seja um homem para fazer abstrações e chegar a noções gerais, não poderá fazer nenhum progresso sem o auxílio das línguas, que é o duplo: um quando fala e o outro quando escreve.

Leonhard Euler (1707-1783)

Aos meus pais
e as minhas sobrinhas
Daniela, Diana e Ellen

André

As minhas filhas,
aos meus pais
e à Luzia

Ney

PREFÁCIO

O objetivo deste texto é apresentar, de uma forma didática e ordenada, as instruções do microprocessador Z-80 com inúmeras ilustrações e centenas de exemplos e exercícios.

Este livro de software completa o estudo do microprocessador Z-80, pois a parte relativa ao hardware foi apresentada no livro "MICROPROCESSADOR Z-80 HARDWARE" também de nossa autoria.

A nossa intenção é que este livro possa ser entendido, em toda a sua plenitude, tanto por estudantes como por profissionais e autodidatas das áreas de eletrônica e informática.

A filosofia que seguimos no desenvolvimento deste trabalho foi a de apresentar, junto com instruções, exemplos de programas que crescem de complexidade à medida que novas instruções são estudadas.

Assim o leitor já começa desde o estudo das instruções iniciais a desenvolver os primeiros programas em Assembly. Isto é fundamental para o aprendizado, pois a eficiência e a otimização de um programa decorrem diretamente dos programas anteriormente criados.

No capítulo 1 apresentamos um resumo dos diversos tipos de linguagens, com ênfase nos conceitos da linguagem Assembly.

Os capítulos 2,3 e 4 abordam, respectivamente, a arquitetura do Z-80, a relação dos bits de condição com as operações aritméticas e lógicas e os modos de endereçamento.

Estes capítulos iniciais e o capítulo 5, que aborda os detalhes da Linguagem Assembly, formam a base mínima para o entendimento e utilização do conjunto de instruções do Z-80.

No capítulo 6 iniciamos o estudo das instruções do Z-80, através das instruções de transferências de 8 bits.

A apresentação detalhada do conjunto de instruções prossegue até o capítulo 16, sendo que, em cada capítulo, temos uma tabela com todas as informações necessárias a completa descrição das instruções.

As instruções foram agrupadas nos capítulos de acordo com as suas características funcionais. Deste modo, o estudo torna-se menos cansativo do que, por exemplo, a apresentação em ordem alfabética destas instruções.

Em adição, cada capítulo possui exemplos de programas relacionados com cada tipo de instrução apresentada. E, ainda, ao final de cada capítulo temos uma lista de exercícios para que o leitor possa treinar os conhecimentos adquiridos.

No capítulo 17 apresentamos algumas subrotinas aplicativas que normalmente são usadas no software básico de microcomputadores.

Finalmente, nos apêndices, apresentamos algumas tabelas envolvendo o código das instruções e o código ASCII.

A necessidade de se escrever em português, sobre um assunto cuja origem é um inglês técnico específico, foi para nós um desafio. É fato sabido que os profissionais evitam traduzir os nomes dos elementos e conceitos introduzidos em inglês. Isto ocorre, principalmente, devido ao receio de não se fazerem entender, caso se aventurem a traduzí-los. Assim, os termos técnicos cujas traduções não são utilizadas no dia a dia dos profissionais foram deixadas na versão original. Por outro lado, aqueles cujas traduções já começam a se popularizar foram traduzidos, sendo que a versão original encontra-se entre aspas e parênteses.

Os Autores
Rio de Janeiro, junho de 1984

ÍNDICE

CAPÍTULO 4 – MODOS DE ENDEREÇAMENTO

CAPÍTULO 5 – A LINGUAGEM ASSEMBLY

CAPÍTULO 6 – INSTRUÇÕES DE TRANSFERÊNCIAS DE 8 BITS

CAPÍTULO 16 – INSTRUÇÕES DE ENTRADA E SAÍDA

CAPÍTULO 17 – APLICAÇÕES

1 INTRODUÇÃO

1.1 – APRESENTAÇÃO

Este livro ensina como programar o microprocessador Z-80 e complementa os conhecimentos apresentados no livro "Microprocessador Z-80 Hardware" também de nossa autoria.

Antes de iniciarmos o estudo do software do microprocessador Z-80 apresentaremos neste capítulo um resumo dos tipos de linguagens existentes e dos diversos conceitos envolvidos.

1.2 – LINGUAGEM DE MÁQUINA

O modo mais rudimentar de programar uma unidade de controle é fornecer os *códigos binários* de cada instrução a ser executada.

A razão de termos que trabalhar com códigos binários decorre da natureza das máquinas eletrônicas que foram projetadas para reconhecer dois níveis de tensão ('0' e '1'). Isto porque, seria inviável uma máquina eletrônica que pudesse reconhecer, com confiabilidade, dez níveis de tensão, que seria a opção natural.

Assim, as unidades de controle são projetadas para executar um certo nº de códigos, que formam o seu *conjunto de instruções.*

Portanto, já podemos concluir que cada unidade de controle possui o seu repertório de instruções que é intrínseco ao seu hardware.

Na fig. 1.1 temos um exemplo de um programa em linguagem de máquina com os códigos representados em binário.

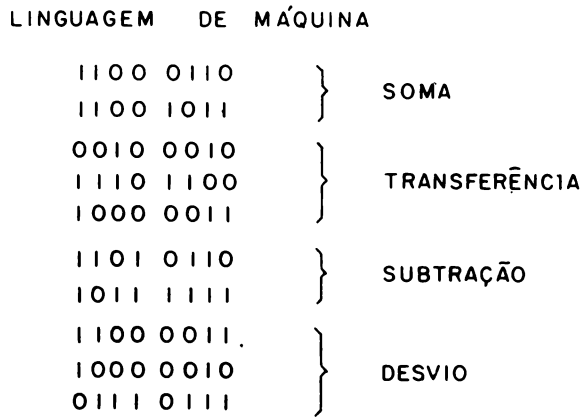


Fig. 1.1 – Programa em Linguagem de Máquina

Normalmente, os programas em linguagem de máquina são escritos em *hexadecimal* ou *octal* de modo a facilitar a utilização dos códigos das instruções.

Na base hexadecimal representamos os dezesseis elementos com os números de 0 a 9 e as letras A, B, C, D, E e F.

Na figura 1.2 temos a representação de um nº binário de quatro bits nas bases hexadecimal, octal e decimal.

BIN	HEX	OCTAL	DEC
0000	0	0	0
0001	1	1	1
0010	2	2	2
0011	3	3	3
0100	4	4	4
0101	5	5	5
0110	6	6	6
0111	7	7	7
1000	8	10	8
1001	9	11	9
1010	A	12	10
1011	B	13	11
1100	C	14	12
1101	D	15	13
1110	E	16	14
1111	F	17	15

Fig. 1.2 – Relação entre as bases

Observe que a representação na base hexadecimal é a mais compacta e a mais fácil, pois a mudança para binário ou vice-versa é a mais utilizada para representação de nºs binários na linguagem de máquina.

Temos na figura 1.3 o programa da figura 1.1 escrito em hexadecimal e octal.

HEX	OCTAL	
C 6	306	SOMA
C B	313	
2 2	042	TRANSFERÊNCIA
E C	354	
8 3	203	
D 6	326	SUBTRAÇÃO
B F	277	
C 3	303	DESVIO
8 2	202	
7 7	167	

Fig. 1.3 – Programa em Linguagem de Máquina

1.3 – LINGUAGEM ASSEMBLY

Na linguagem de máquina a representação das instruções através de códigos numéricos é extremamente cansativa e, além disso, a lógica do programa fica difícil de ser compreendida.

Assim, representamos na linguagem assembly as instruções através de *mnemônicos* de 3 ou 4 letras. Estes mnemônicos facilitam o entendimento e a criação dos programas.

Na figura 1.4 temos o programa da figura 1.1 usando-se os mnemônicos de cada instrução.

MÁQUINA	ASSEMBLY	
C 6	ADD	OCBH
C B		
2 2	LD	(083ECH), HL
E C		
8 3		
D 6	SUB	OBFH
B F		
C 3	JP	07782H
8 2		
7 7		

Fig. 1.4 – Programa na Linguagem Assembly

Para utilizarmos esta linguagem, cujos detalhes estudaremos no capítulo 5, é necessário um programa que faça a tradução dos mnemônicos para os correspondentes código de máquina. Este programa é chamado de MONTADOR ("Assembler").

Na fig. 1.5 temos uma ilustração da função deste programa montador.

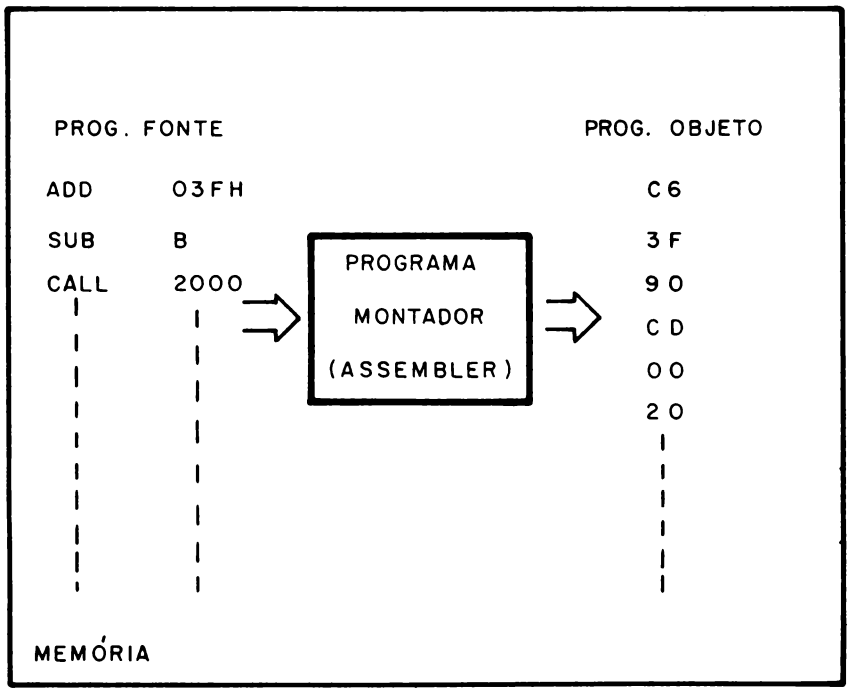


Fig. 1.5 – Programa Montador

O programa montador converte um *programa fonte* escrito em Assembly para o correspondente *programa objeto* em código de máquina.

A denominação *programa fonte* identifica o programa escrito pelo usuário e que ainda não foi montado.

O *programa objeto* é o programa em código de máquina pronto para ser executado.

Na linguagem assembly as instruções são as mesmas existentes na corrrespondente línguagem de máquina, só que representadas por mnemônicos. Isto é, nesta linguagem só podemos utilizar as instruções que são reconhecidas pela unidade de controle.

Assim, cada computador possui o seu conjunto próprio de instruções e a correspondente linguagem assembly. Isto porque, as suas instruções estão implementadas no *hardware* da unidade de controle, que varia de acordo com o porte, o tipo e o fabricante do computador.

Normalmente, este *programa montador* é fornecido junto com o microcomputador, sendo função do microprocessador utilizado. O montador mais popular para microprocessadores 8080 e Z-80 é o M-80 da Microsoft.

Voltaremos a linguagem assembly no capítulo 5 quando estudaremos a sua sintaxe.

1.4 – LINGUAGEM DE ALTO NÍVEL

Uma evolução natural nas linguagems de programação foi a criação das linguagens de alto nível. Elas caracterizam-se pela não dependência direta com o hardware da unidade de controle. Isto é, define-se um padrão de linguagem com seus comandos e sintaxe, de tal modo que as variações sejam mínimas quando ocorre mudança de máquina.

Isto facilita a programação, pois pode-se aprender uma lin-

guagem que seja utilizada em vários computadores. Com isso, os programas que realizam a conversão para o código de máquina (programa objeto) são bem mais complexos. Estes programas são chamados *compiladores.*

Temos compiladores para as diversas linguagens, cada uma com suas características próprias, voltadas para aplicações específicas.

Quando se adquire um computador é necessário especificar junto ao fabricante os compiladores relativos as linguagens desejadas.

Como exemplo de linguagens e alto nível podemos citar: PASCAL, FORTRAN, BASIC, COBOL, etc . . .

A função básica de um compilador é traduzir um programa fonte escrito em linguagem de alto nível para o correspondente código de máquina (programa objeto).

Outrossim, é importante salientar que nas aplicações de controle de processo a linguagem predominante é o Assembly. Isto porque nesta linguagem o programador pode dimensionar e otimizar o tamanho e o tempo de execução dos programas.

Finalizando, ressaltamos que o repertório de instruções do microprocessador Z-80 inclui o conjunto de instruções do microprocessador 8080 e 80 instruções adicionais. O que faz com que os programas desenvolvidos para o 8080 também sejam executados no Z-80. Nos próximos capítulos, sempre que possível, ressaltaremos esta compatibilidade.

1.5 – EXERCÍCIOS DE FIXAÇÃO

1.5.1 – Qual a relação entre as representações em hexadecimal, octal, binário e decimal?

1.5.2 – Porque a base hexadecimal é mais usada na representação de código de máquina?

1.5.3 – Qual a relação entre o hardware da unidade de controle e os códigos da linguagem de máquina?

1.5.4 – O que é linguagem Assembly?

1.5.5. – Qual a função do montador?

1.5.6. – Cite uma vantagem e uma desvantagem da linguagem Assembly.

1.5.7 – O que são compiladores?

2 A ARQUITETURA DO Z-80

2.1 – APRESENTAÇÃO

Iniciando o estudo do software do microprocessador Z-80, estudaremos neste capítulo a sua organização lógica interna e seus registradores.

Apresentaremos os registradores estudando-os conceitualmente e dividindo-os didaticamente em três grupos.

2.2 – ORGANIZAÇÃO LÓGICA INTERNA

A figura 2.1 mostra a organização lógica interna do microprocessador Z-80. A seguir descreveremos cada uma de suas partes:

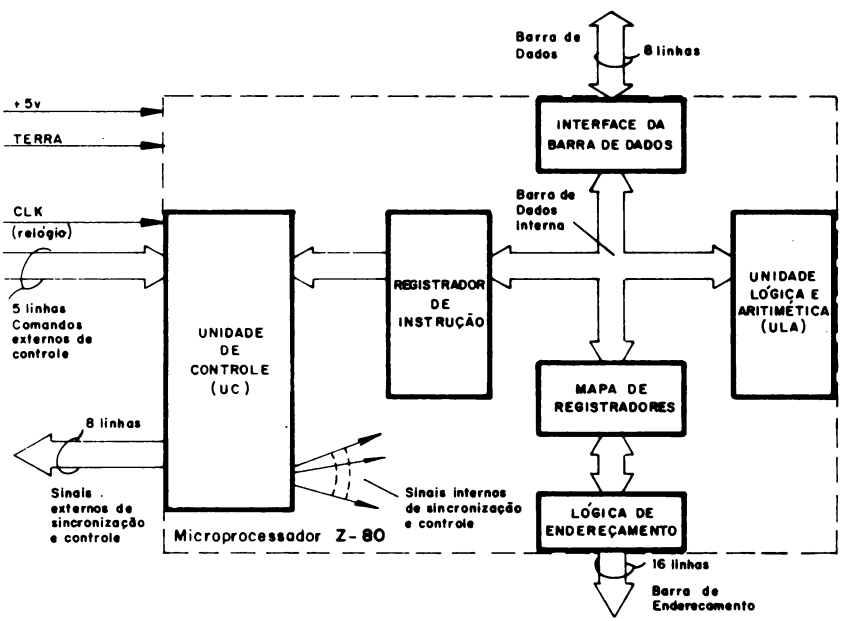


Fig. 2.1 – Organização Lógica do Z-80

A barra de endereçamento ("Address Bus") é composta de 16 bits, possibilitando endereçar até 2 EXP (16) = 65536 = 64K bytes de memória. Além disso, o Z-80 utiliza 8 bits desta barra para endereçar até 2 EXP (8) = 256 dispositivos de entrada ou saída.

Os dados (operandos ou instruções) destinados ao Z-80 ou dele procedentes trafegam através da *barra de dados* ("Data Bus"), que é bidirecional e possui 8 bits. Esta barra se ramifica no interior do Z-80, através da *barra de dados interna* que estabelece a comunicação entre o *mapa de registradores,* o *registrador de instrução* e a *unidade lógica e aritmética* (ULA).

A *interface da barra de dados* acopla os setores internos e externos desta barra e tem duas funções básicas:

1 — Isolamento elétrico
2 — Captura e armazenamento

O *isolamento elétrico* ("buffer") entre circuitos elétricos das barras de dados interna e externa, faz com que as memórias e dispositivos de entrada e saída não sobrecarreguem os circuitos internos do Z-80, permitindo que um número maior destes dispositivos externos seja conectado a esta barra.

A *captura e armazenamento* ("latch") permite que estados lógicos de curta permanência na barra de dados sejam capturados e armazenados temporariamente.

A *lógica de endereçamento* determina, a cada momento, o endereço que deve ser colocado na barra de endereçamento, que também é equipada com buffers e latches.

A ULA concentra em si todas as operações lógicas e aritméticas.

O *mapa de registradores* reúne todos os registradores de processamento que o programador tem acesso.

O *registrador de instrução* encerra o código da instrução que está em processo de execução.

A função básica da *unidade de controle* (UC) é gerar os sinais de *sincronização e controle,* internos e externos ao Z-80, responsáveis pela consecução das atividades da máquina.

O automatismo do processamento é conseguido por meio destes sinais, que determinam quais as funções que devem ser executadas em determinados instantes de tempo coordenando desta forma: as operações da ULA, as transferências internas ao Z-80 e as partes externas que compõem o microcomputador. Tais sinais são precisamente distribuídos no tempo e no hardware, através de linhas individuais ligadas a pontos de controle dos circuitos internos ou externos ao Z-80.

A figura 2.2 ilustra o funcionamento da unidade de controle (UC).

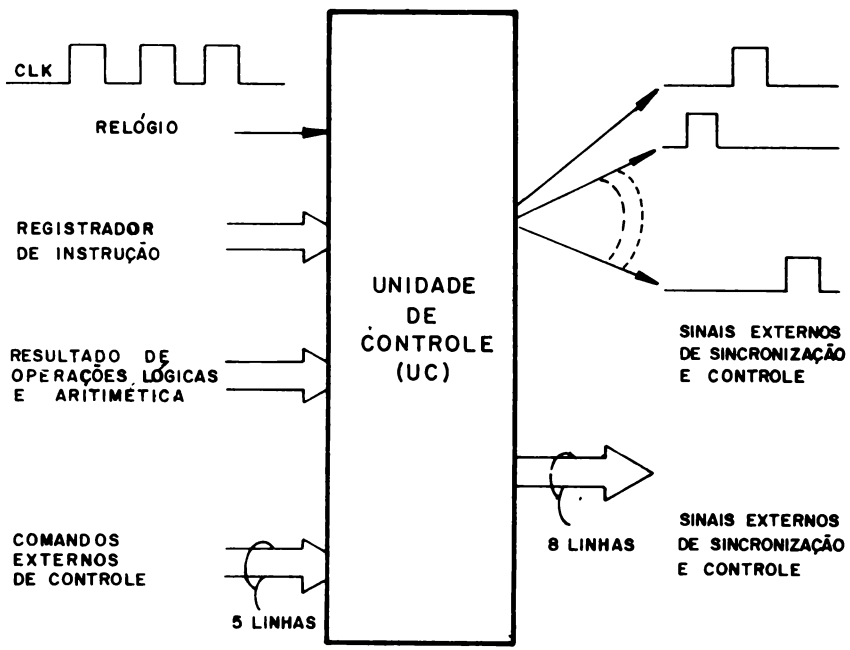


Fig. 2.2 – Funcionamento da UC

Observe que os sinais são gerados com base no conteúdo do registrador de instrução, nos resultados das operações lógicas e artiméticas e nos *comandos externos de controle.* O cadenciamento destes sinais é determinado pela entrada pulsada proveniente do relógio.

O *registrador de instrução* fornece informações relativas a instrução que está em fase de execução.

Através dos resultados das operações lógicas e aritméticas são escolhidos cursos alternativos de ação de programas.

Os comandos externos de controle reúnem um grupo de cinco linhas, que permitem aos módulos externos ao Z-80 interferir na seqüência natural de processamento. Estes comandos externos, quando ativados, forçam a execução de programas específicos (interrupção, reset), ou mesmo, podem paralisar temporariamente o processamento (wait, hold).

Os *sinais externos de sincronização e controle* reúnem um grupo de oito linhas que servem basicamente para controlar os acessos à memória e aos dispositivos de entradas e saída.

Finalmente, observando a figura 2.1, podemos identificar 40 pontos de comunicação do Z-80 com o meio exterior, que reúnem: as barras, os sinais de controle e a alimentação.

2.3 – MAPA DE REGISTRADORES

Para o programador, um dos aspectos mais importantes a considerar no hardware do Z-80 é a arquitetura de seus registradores, ou seja, a organização lógica estrutural e funcional desses elementos.

A figura 2.3 apresenta o mapa de registradores do Z-80.

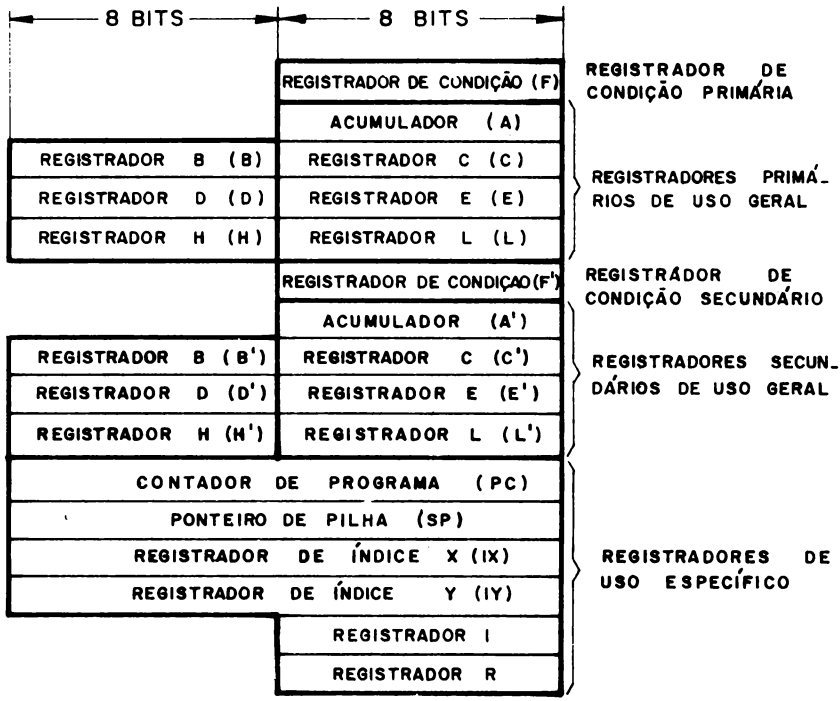


Fig. 2.3 – Mapa de Registradores do Z-80

Note que existem registradores de 8 e de 16 bits. Além disso, temos registradores *primários* e *secundários* que estudaremos a seguir.

Apresentaremos estes registradores dividindo-os em três tipos: *registradores de uso geral, registradores de condição* e *registradores de uso específico.*

2.4 – REGISTRADORES DE USO GERAL

O microprocessador Z-80 possui sete registradores de uso geral, de 8 bits cada um, que se apresentam de forma duplicada. Assim, para fins didáticos, vamos classificá-los em *primários* e *secundários.*

Estes registradores são apresentados na figura 2.4.

REGISTRADORES DE USO GERAL	
PRIMÁRIOS	SECUNDÁRIOS
A	A'
B	B'
C	C'
D	D'
E	E'
H	H'
L	L'

Fig. 2.4 – Registradores de Uso Geral

Os registradores de uso geral e os registradores F e F', que serão estudados mais adiante, são também classificados durante a execução de um programa em *ativos* e *passivos*. Em qualquer instante temos oito registradores ativos e oito passivos. Os registradores ativos são aqueles que respondem pelos operandos das instruções, enquanto que os passivos mantêm seus conteúdos inalterados até o momento de se tornarem ativos e, conseqüentemente, terem seus conteúdos modificados.

Existem duas instruções no Z-80, EX AF, AF' e EXX, que permitem selecionar A e F ou A' e F' e B, C, D, E, H, L ou B', C' D', E', H', L' como ativos, respectivamente. Na figura 2.5 temos as possíveis combinações destes registradores.

A	F
B	C
D	E
H	L

A'	F'
B'	C'
D'	E'
H'	L'

A'	F'
B	C
D	E
H	L

A	F
B'	C'
D'	E'
H'	L'

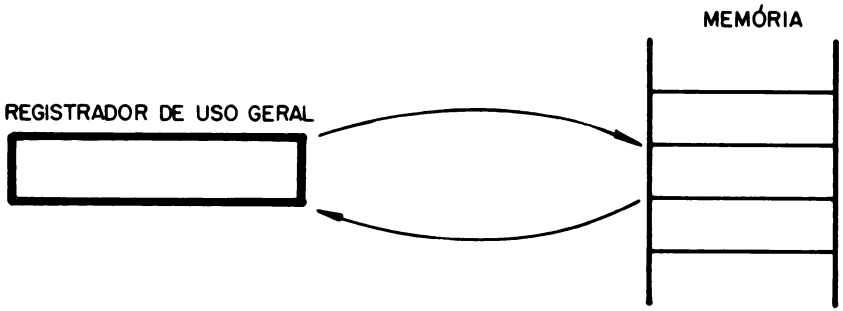
Fig. 2.5 – Combinações de Registradores Ativos

Uma vantagem de termos dois blocos de registradores de uso geral é a maior disponibilidade de registradores no interior do Z-80, o que permite o acesso mais rápido às informações do que se as mesmas estivessem armazenadas na memória. Outra vantagem é que o programador pode trocar rapidamente (uma ou duas instruções) de um bloco de registradores para outro. Isto é usado, principalmente, na troca de contexto durante o atendimento de interrupções.

Convém ressaltar que o microprocesador 8080 possui os mesmos registradores de uso geral que o Z-80 porém sem duplicação.

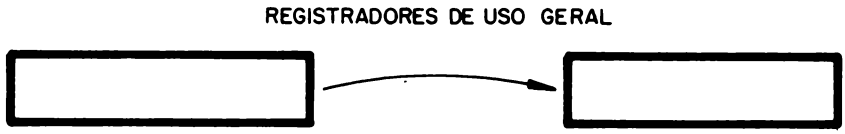
A seguir temos um resumo das principais funções dos registradores de uso geral no Z-80 ou no 8080:

a) Transferência do conteúdo de oito bits de um registrador ativo para uma posição de memória ou vice-versa.



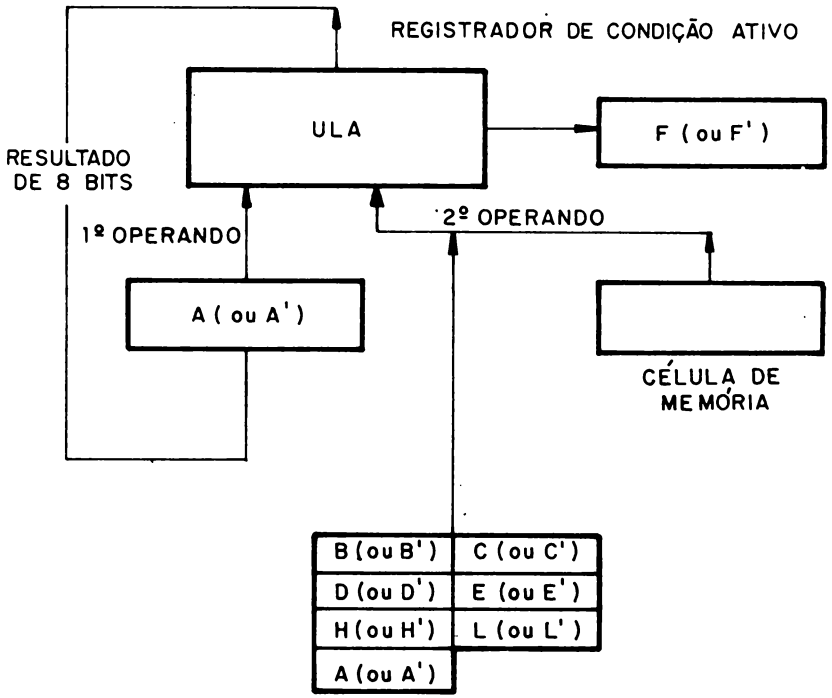
Por exemplo, a instrução LD A, (1000H) transfere o conteúdo da posição de memória 1000H para o acumulador.

b) Transferência do conteúdo de um registrador de uso geral ativo para um outro.



Por exemplo, a instrução LD E,H transfere o conteúdo do registrador H para E.

c) Operações lógicas e aritméticas de 8 bits são realizadas usando o acumulador ativo, outro registrador ativo de uso geral ou o conteúdo de uma posição de memória. O resultado sempre é depositado no acumulador.



Por exemplo, a instrução ADD A,D adiciona o conteúdo do registrador D ao acumulador, e este resultado fica depositado no próprio acumulador.

d) Os registradores de uso geral podem ser usados em *pares*, formando registradores de 16 bits. Os pares possíveis são BC, DE, HL (ou B'C', D'E' ou H'L').

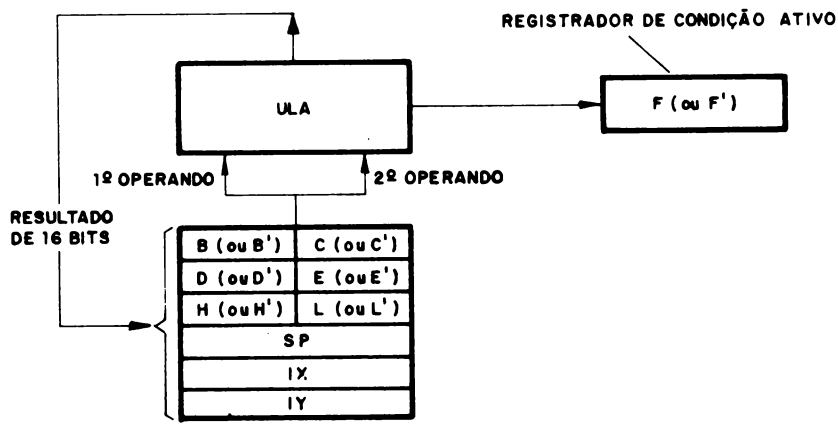
B (ou B')	C (ou C')	par BC (ou B'C')
D (ou D')	E (ou E')	par DE (ou D'E')
H (ou H')	L (ou L')	par HL (ou H'L')

Em várias instruções dos microprocessadores Z-80 e 8080, os dados contidos nos três *pares de registradores* representam endereços de posições de memória. Neste caso, o conteúdo dos registradores B, D e H representam a parte mais significativa do endereço e o conteúdo dos registradores C, E e L representam a parte menos significativa. Quando o conteúdo dos pares de registradores representam endereço, eles são chamados de *ponteiros* ("pointers").

Por exemplo, a instrução LD (HL),C transfere o conteúdo do registrador C para a posição de memória endereçada pelo par HL.

e) Os pares de registradores e os registradores SP, IX e IY podem também ser usados em *operações aritméticas de dupla precisão,* isto é, como operandos de 16 bits. Este tipo de operação é usado principalmente na manipulação de ponteiros.

Sem as instruções de dupla precisão seriam necessárias duas operações aritméticas de 8 bits.



Por exemplo, a instrução ADD HL, SP adiciona o conteúdo de SP ao conteúdo de HL, e o resultado fica depositado no próprio HL.

Vejamos agora alguns comentários complementares sobre o uso dos registradores de uso geral.

O par de registradores HL (ou H' L') é privilegiado em relação aos outros dois, pois o seu conteúdo é usado como ponteiro em diversas modalidades de instruções. O Z-80 e o 8080 possuem instruções que transferem um byte entre qualquer registrador de uso geral e uma posição de memória endereçada pelo conteúdo do par HL (ou H' L'). Além disso, as instruções lógicas e aritméticas permitem usar o par HL (ou H' L') como ponteiro de seus operandos. Assim, o programador deve, preferencialmente, usar o par HL (ou H' L') como ponteiro no acesso à memória. Por outro lado, os registradores B, C, D e E (ou B', C', D' e E') devem ser usados para armazenamento temporário de dados ou servir de segundo operando nas operações lógicas e aritméticas. Além disso, existem algumas instruções que usam os pares BC (ou B' C') e DE (ou D' E') como ponteiros no acesso à memória. No entanto, tal grupo de instruções limita-se a transferir informações entre o acumulador e a memória.

Por exemplo, a instrução LD A,(DE) transfere o conteúdo da posição de memória endereçada pelo par DE para o acumulador.

2.5 – REGISTRADORES DE CONDIÇÃO

O Z-80 possui os registradores de condição F e F' ("Flag Register') sendo que, como já foi visto, um deles é ativo e o outro passivo.

Cada registrador de condição tem oito bits, dos quais seis são usados.

O registrador de condição ativo (F ou F') funciona associado ao acumulador e à ULA, e cada um de seus bits indica uma situação definida, resultante basicamente de operações lógicas e aritméticas.

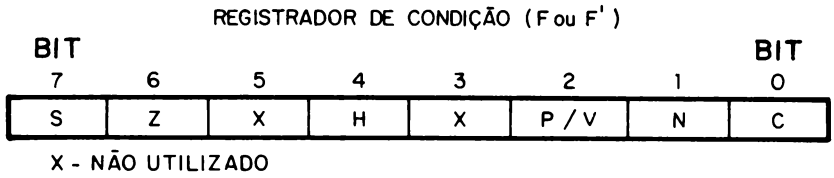


Fig. 2.6 – Registrador F ou F'

São os seguintes os bits de condição do registrador F ou F':

a) BIT C ("Carry bit")
Indica a ocorrência de "vai um" no 8º bit após a realização de uma soma ou de empréstimo ("borrow") quando se efetua uma subtração, assumindo o valor 1 se houver tal ocorrência ou o valor 0 caso contrário.

b) BIT N ("Subtract bit")
De uma maneira geral, assume o valor 0 após as instruções de soma e o valor 1 após as instruções de subtração.

c) BIT P/V ("Parity/Overflow bit")
Este bit pode representar, de um modo geral, *paridade ou overflow,* dependendo da natureza da instrução executada.

A paridade representa a natureza par ou ímpar do número de bits "1" do resultado.

A ocorrência de overflow indica que o resultado da operação aritmética não coube no registrador a ele destinado.

Quando o bit P/V representar a paridade do resultado, P/V = 1 indica paridade par e P/V = 0 indica paridade ímpar.

No caso do bit P/V indicar overflow, se P/V = 1 após uma operação aritmética, significa a ocorrência de overflow.

d) BIT H ("Half-carry bit")
Indica a ocorrência de "vai um" ou empréstimo do quarto para o quinto bit do resultado, na realização da soma ou da subtração, assumindo o valor 1 se houver tal ocorrência.

e) BIT Z ("Zero bit")
Z = 1 indica que o resultado de uma operação lógica ou aritmética é nulo e Z = 0 indica que o resultado é diferente de zero.

f) BIT S ("Signal bit")
Representa o sinal do resultado respeitando-se a convenção de sinal algébrico da *notação complemento a 2,* de modo que se S = 0 indica um resultado positivo e S = 1 um resultado negativo.

Convém ressaltar que os bits de condição Z, S, C e P/V podem ser testados pelas instruções que alteram o curso normal de um programa. As escolhas de cursos alternativos dos programas são realizadas com base nos estados dos referidos bits de condição.

Os bits H e N são usados para facilitar as operações aritméticas com números representados na *notação BCD.*

2.6 – REGISTRADORES DE USO ESPECÍFICO

Cada registrador específico tem uma função bem definida durante o processamento. Temos quatro registradores de 16 bits e dois de 8 bits:

1 – Contador de programa (PC – "Program Counter")
2 – Ponteiro de pilha (SP – "Stack Pointer")
3 – Registradores de índice (IX e IY – "Index Register")
4 – Registrador de interrupção (I – "Interrupt Register")
5 – Registrador de refresh (R – "Memory Refresh")

O microprocessador 8080 somente possui os registradores PC e SP que realizam funções semelhantes às desempenhadas no Z-80.

A seguir apresentaremos conceitualmente estes registradores específicos.

2.6.1 – O Contador de Programa (PC)

O PC é um registrador de 16 bits que é atualizado em cada instrução, de modo a conter o endereço da próxima instrução a ser executada.

As instruções do Z-80 podem ocupar de um a quatro bytes consecutivos da memória.

Do ponto de vista do programador, o conteúdo do PC endereça sempre o início da próxima instrução a ser executada, isto é, o seu primeiro byte. Desta forma, a unidade de controle incrementa automaticamente o conteúdo da instrução que está em fase de execução.

Como conseqüência natural do funcionamento do PC, as instruções de um programa são organizadas na memória em posições consecutivas e em ordem crescente de endereços.

A figura 2.7 ilustra o funcionamento do PC.

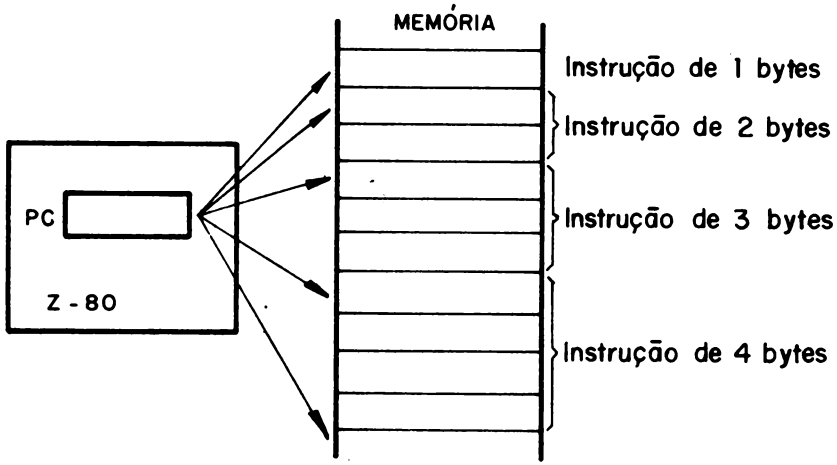


Fig. 2.7 – O Funcionamento do PC

Esta atualização seqüencial do PC é quebrada quando o Z-80 executa instruções que alteram o curso normal do programa. Estas instruções, por exemplo: JP, CALL, RET e RST, carregam o PC com um novo valor.

2.6.2 — O Ponteiro da Pilha (SP)

O SP é um registrador de 16 bits que tem a função de *ponteiro da pilha.*

A *pilha* é uma estrutura de informação implementada em RAM, que é muito utilizada em computadores de um modo geral.

Numa linguagem mais formal, a pilha é uma *lista linear* em que inserções e deleções de elementos são realizadas em um dos extremos desta lista.

Listas lineares são conjuntos de elementos X (1), X (2), . . . X (n), cujas propriedades estruturais envolvem, essencialmente, posições relativas unidimensionais.

A dinâmica de funcionamento da pilha pode ser compreendida na prática com a inserção e retirada de bolas em uma caçapa conforme mostrado na fig. 2.8.

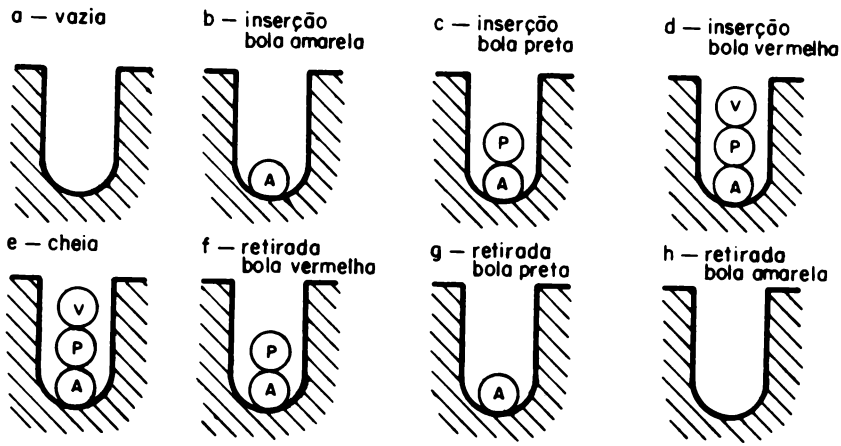


Fig. 2.8 — O Funcionamento da Pilha

Observe que nesta estrutura o *último elemento inserido é o primeiro a ser retirado.* Este é o princípio básico de funcionamento da pilha. Quando implementada na memória, o SP endereça o último dado inserido ou o seguinte ao que foi retirado, isto é, o *topo da pilha.*

Nos microcomputadores, a principal utilização da pilha é o armazenamento temporário do conteúdo de registradores de uma maneira sistemática e organizada.

As instruções do Z-80 que manipulam a pilha realizam transferências de 2 bytes. Assim, podem ser transferidos em uma única operação, o conteúdo de qualquer registrador específico de 16 bits (exceto o SP), o de qualquer par de registradores de uso geral e o par formado pelo acumulador (A ou A') e o registrador ativo de condição (F ou F').

Outra aplicação da pilha é o armazenamento do endereço de retorno em *chamadas de sub-rotinas* e no *tratamento de pedidos de interrupção.*

2.6.3 – Os Registradores de Índice (IX e IY)

IX e IY são dois registradores de 16 bits que permitem ao Z-80 calcular endereços de operandos de instruções que utilizam a técnica de *endereçamento indexado.* Nesta técnica, o endereço efetivo do operando é a soma do campo de endereços da instrução, denominado deslocamento ("displacement"), com o conteúdo de um dos registradores de índice, especificado no campo do código da instrução.

As instruções com endereçamento indexado são usadas principalmente em aplicações com *listas e tabelas.*

A figura 2.9 ilustra o funcionamento do registrador de índice.

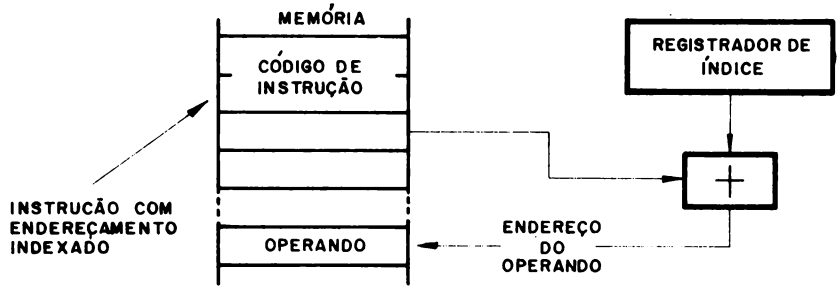


Fig. 2.9 – Endereçamento Indexado

Por exemplo, a instrução LD E,(IX + 15H) transfere para o registrador E o conteúdo da posição de memória cujo endereço é dado pela soma do conteúdo de IX e o deslocamento que é 15H (base hexadecimal). Se IX = 1000H então o endereço efetivo é 1015H.

2.6.4 – O Registrador Vetor de Interrupção (I)

O Registrador I, também chamado *Registrador Vetor de Interrupção,* tem 8 bits e é usado no atendimento de interrupções. O seu conteúdo é previamente carregado pelo programador e representa os 8 bits mais significativos do endereço do vetor de interrupção. A parte menos significativa deste endereço é gerada pelo dispositivo de entrada e saída que originou o pedido de interrupção. Estes 8 bits menos significativos são fornecidos através da barra de dados, em intervalo de tempo sincronizado pelo relógio do Z-80 e durante o processo de atendimento de interrupção.

O vetor de interrupção é o endereço da primeira instrução da *rotina de tratamento de interrupção.* Associado a uma instrução de desvio leva o Z-80 a executar esta rotina. Assim, cada dispositivo de entrada e saída ao solicitar uma interrupção força o Z-80 a executar a sua rotina de tratamento específica.

A figura 2.10 ilustra a formação do endereço de interrupção.

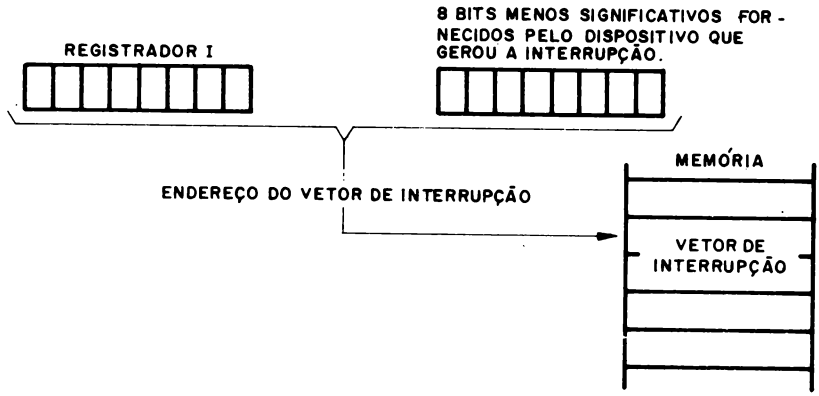


Fig. 2.10 – Formação do Endereço Vetor de Interrupção

2.6.5. – O Registrador de Refresh de Memória (R)

O Registrador R, chamado *registrador de refresh da memória,* possui 8 bits e sua função é controlar o *refresh* automático de memórias semicondutoras dinâmicas.

Normalmente, cada célula de memória dinâmica sofre um *refresh* periódico, num intervalo inferior a 2ms. O mecanismo de *refresh* é, na quase totalidade dos casos, a realização de um ciclo de leitura, que na maioria das pastilhas processa-se de uma só vez, em grupos de até 128 células de memória. O conteúdo do registrador R designa, de uma forma cíclica, o grupo que deve sofrer o *refresh*.

Após a busca de cada instrução, o Z-80 incrementa de uma unidade o conteúdo do registrador R e o coloca nos 7 bits menos significativos da barra de endereços. Isto ocorre em paralelo com o processamento interno da instrução, justamente nos intervalos de tempo em que a execução da instrução não requer o uso das barras.

Isto é uma grande virtude do Z-80, pois além do *refresh* transcorrer de forma transparente ao programa, também dispensa o uso de circuitos externos.

2.7 – EXERCÍCIOS DE FIXAÇÃO

2.7.1 – Por que a capacidade máxima de endereçamento de memória é 65536 bytes e a de dispositivos de entrada e saída é 256?

2.7.2 – Quais as funções da interface da barra de dados?

2.7.3 – Explique o funcionamento da unidade de controle (UC).

2.7.4 – O que são registradores Ativos e Passivos?

2.7.5 – Quais as vantagens de termos os registradores de uso geral de forma duplicada?

2.7.6 – Qual o par de registradores que, preferencialmente, deve ser usado como ponteiro? Por que?

2.7.7 – Explique o significado de cada um dos bits de condição.

2.7.8 – Qual a função do registrador contador de programa (PC)?

2.7.9 – O que é memória pilha?

2.7.10 – O que é refresh de memória?

2.7.11 – Quais os registradores que permitem a utilização do endereçamento indexado?

2.7.12 – Por que as instruções são organizadas em endereços seqüenciais na memória?

2.7.13 – Quais são as principais funções dos registradores de uso geral?

2.7.14 – Explique como é determinado o endereço do operando no endereçamento indexado?

2.7.15 — O que é vetor de interrupção?

2.7.16 — Como é determinado o endereço do vetor de interrupção?

2.7.17 — Quais são as vantagens do registrador R?

3 OS BITS DE CONDIÇÃO E AS OPERAÇÕES ARITMÉTICAS

3.1 – APRESENTAÇÃO

Após termos estudado no capítulo anterior a arquitetura e os registradores do Z-80, iniciaremos o software apresentando a relação entre os bits de condição e as operações aritméticas, incluindo a representação de números em *complemento a dois* e em *BCD.*

3.2 – O REGISTRADOR F ou F'

Estudaremos este registrador com detalhes pois o poder de decisão de um programa, que é a sua "inteligência", é baseado nos testes dos bits de condição.

Na figura 3.1 temos a identificação dos bits do registrador F (ou F'), que são chamados *bits de condição.*

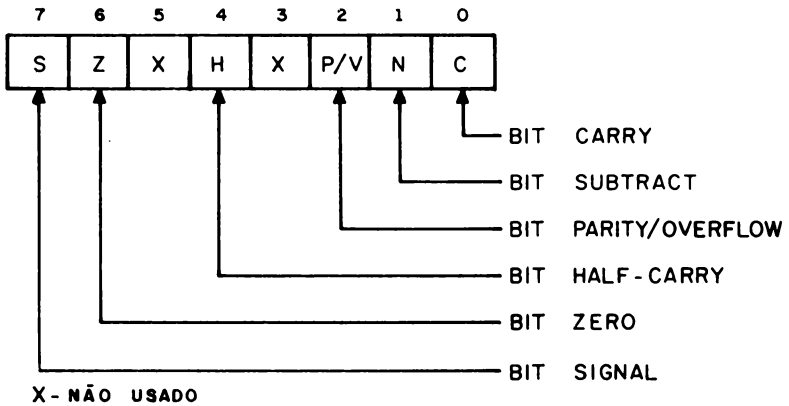


Fig. 3.1 – Os Bits de Condição do Z-80

Estes bits de condição, que possuem significado próprio, estão agrupados logicamente no registrador F ou F' de modo a facilitar o salvamento destes na pilha. Isto ocorre, principalmente, na chamada e retorno de sub-rotinas e no atendimento de interrupções.

Os bits de condição C, P/V, S e Z podem ser testados pelas instruções que ocasionam desvios condicionais – JUMP'S, CALL'S e RETURN'S – que na verdade avaliam os resultados de operações lógicas e aritméticas.

Os bits de condição H e N são usados para facilitar as operações aritméticas com números representados na notação BCD – Binary Coded Decimal.

O microprocessador 8080 não dispõe do bit N e o bit P/V tem a função de indicar, exclusivamente, a ocorrência de paridade, sendo identificado por P.

Visando simplificar as representações das várias bases numéricas no transcurso do texto usaremos a seguinte convenção: números terminados por "B" estão representados na base binária, por "D" na base decimal e por "H" na base hexadecimal.

3.3 – REPRESENTAÇÃO EM COMPLEMENTO A DOIS

3.3.1 – Formato dos Números

O hardware da Unidade Lógica e Aritmética (ULA) do Z-80 foi projetado para operar números algébricos na representação complemento a dois (2's). A figura 3.2 mostra a forma geral desta técnica de representar números algébricos.

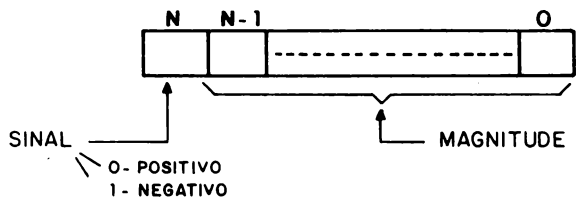


Fig. 3.2 – Representação 2's

O bit mais significativo representa o sinal do número; "0" representa o sinal positivo e "1" o negativo. O restante dos bits representam a magnitude do número.

Quando um byte representa um número na notação 2's, o bit 7 representa o sinal e os bits 0 a 6 representam a magnitude. A magnitude de um número positivo é a sua representação na base binária. Portanto, a faixa de números positivos que pode ser representada com um byte é de 0D a 127D, conforme ilustra a figura 3.3.

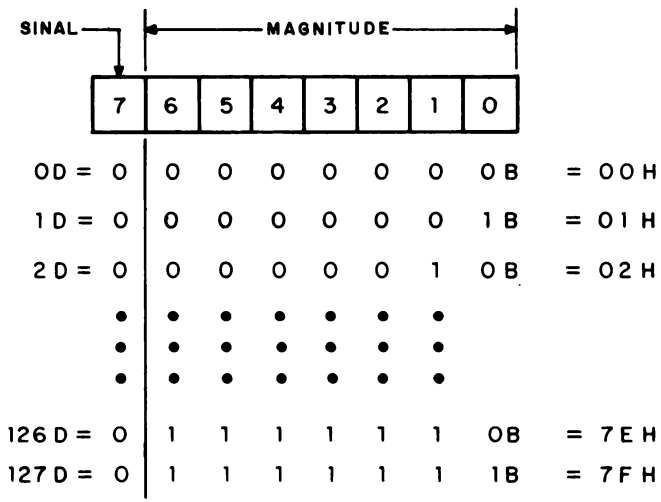


Fig. 3.3 – Faixa de nºs positivos em 2's utilizando-se um byte

A mudança de sinal de um número representado na notação 2's se faz da seguinte forma:

1. Cada bit é individualmente complementado, produzindo o chamado complemento a 1 (1's).

2. Somar "1" ao resultado anterior, desprezando, caso exista, o "vai um" do bit de mais alta ordem.

Por exemplo: -10D representado na notação 2's é 11110110B, conforme ilustra a figura 3.4. Observe que o bit 7, representativo

do sinal, ficou automaticamente setado, indicando um número negativo.

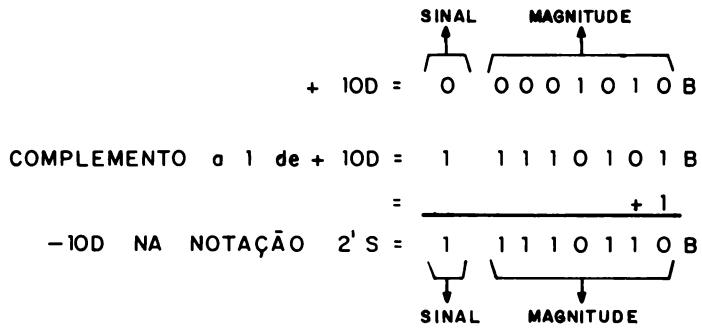


Fig. 3.4 – Cálculo de -10D na notação 2's

Portanto, a faixa de números negativos que pode ser representada com um byte é de -1D a - 128D, conforme ilustra a figura 3.5.

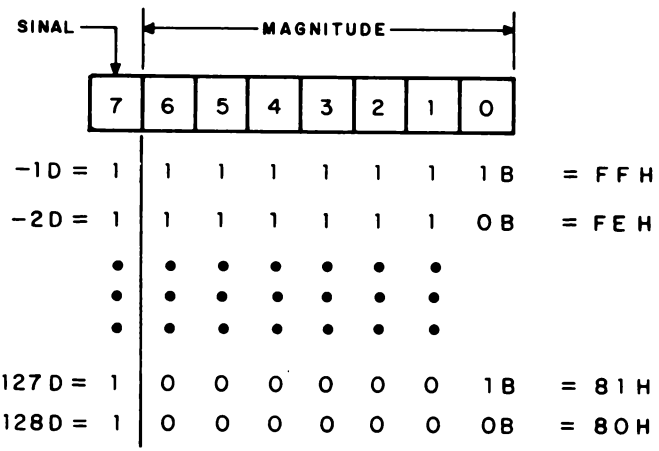


Fig. 3.5 – Faixa de nºs negativos em 2's representados por um byte.

3.3.2 – A Operação de Subtração

As operações de subtração com números representados na notação 2's são realizadas somando-se o minuendo ao subtraendo de sinal invertido (2's), e desprezando-se o possível "vai um".

Por exemplo, a instrução SUB E subtrai o conteúdo do registrador E do conteúdo do acumulador e armazena o resultado no próprio acumulador. Esta operação realiza-se conforme ilustra a figura 3.6. Vamos assumir que em A temos 1AH e em E temos 0CH. Então, mudamos o sinal de 0CH e somamos a 1AH, obtendo como resultado 0EH, ignorando-se o "vai um".

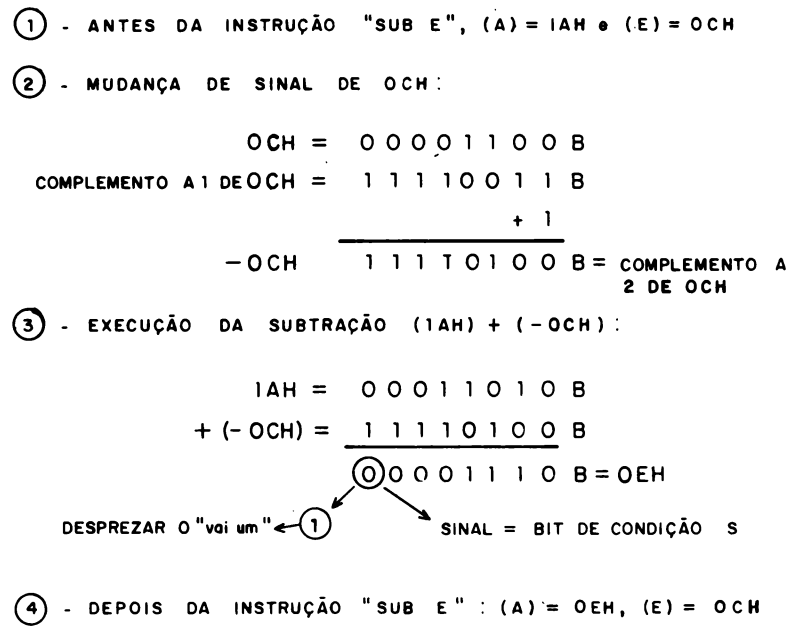


Fig. 3.6 – Execução da Instrução SUB E

Note que para realizar subtrações de números representados na notação 2's basta que o hardware tenha circuitos inversores e o mesmo somador usado nas operações de adição. Isto simplifica bastante a implementação da unidade lógica e aritmética, razão pela qual a notação 2's é usada na maioria dos computadores digitais.

O bit de condição S indica o sinal do resultado e, nada mais é que o bit mais significativo (bit 7).

O bit de condição Z é setado (Z = 1) se o resultado de uma

operação for igual a zero. Caso contrário, assume o valor zero (Z = 0).

3.3.3 – O Overflow

Nas operações de soma ou subtração de números de oito bits, representados na notação 2's, os resultados são armazenados no acumulador, que é um registrador de 8 bits.

Se o resultado de uma soma de dois números positivos ultrapassar 127D, ou se o resultado de uma soma de dois números negativos for inferior a -128D, significa que tais resultados não podem ser representados somente pelos 8 bits do acumulador. Esta condição é chamada de overflow, isto é, o resultado da operação não coube no registrador a ele destinado.

O bit de condição P/V indica o overflow em operações aritméticas. Se ocorrer tal condição após uma operação aritmética o bit P/V é setado (P/V = 1). Assim, ele nos alerta para a validade ou não do resultado.

Na figura 3.7 temos dois exemplos de ocorrência de overflow. No primeiro caso, temos a soma de dois números positivos de 8 bits, cujo resultado ficou negativo. No segundo caso, temos a soma de dois números negativos, cujo resultado ficou positivo.

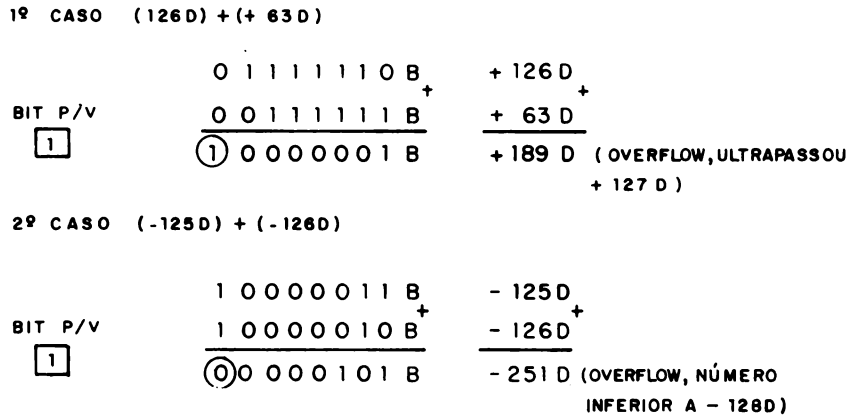


Fig. 3.7 – Exemplos de Overflow

3.3.4 – Operações com nºs 2's sem sinal

Se um byte for interpretado como sendo a magnitude de um número binário positivo representado na notação 2's, poderemos estender a faixa de números positivos representados por um byte para 0D a 255D, conforme ilustra a figura 3.8.

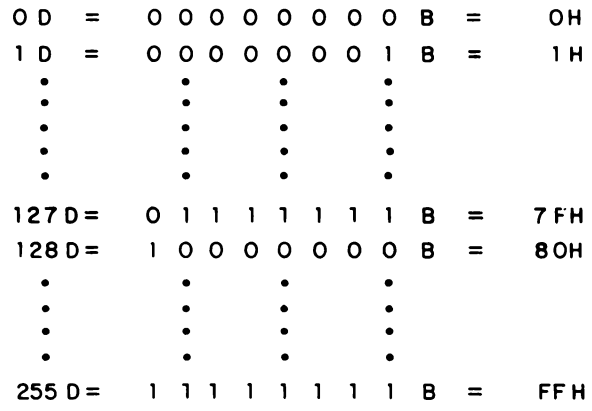


Fig. 3.8 – Faixa de nºs positivos na representação 2's sem sinal

Note que as operações de soma e subtração na aritmética 2's continuam válidas.

Na realização de uma soma, se ocorrer "vai um" significa que o resultado ultrapassou 255D, fazendo com que o bit de condição C fique setado (C = 1).

As instruções que realizam operações de subtração, automaticamente, invertem o "vai um" do resultado, antes de armazená-lo no bit de condição C. Tal inversão permite que o bit C indique o sinal do resultado de operações de subtração com números na representação 2's sem o sinal.

Assim, o bit C resetado (C = 0) significa que o resultado é maior ou igual a zero e, o bit C setado (C = 1) simboliza um resultado negativo.

A figura 3.9 exemplifica uma operação de subtração de números positivos representados na notação 2's sem o sinal, usando-se uma instrução de subtração. Para facilitar o entendimento do sinal foi adicionado um nono bit fictício. Observe que o bit C é o inverso do "vai um", o que significa que o resultado é positivo.

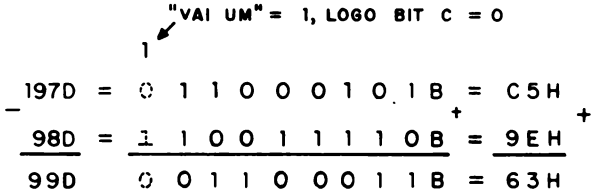


Fig. 3.9 – Subtração de nºs positivos na representação 2's sem sinal

3.4 – REPRESENTAÇÃO BCD

Na representação BCD (Binary Coded Decimal) cada dígito decimal é representado pelo seu correspondente código binário. A tabela da figura 3.10 mostra a representação binária de cada dígito decimal.

DECIMAL	BINÁRIO
0 D	0 0 0 0 B
1 D	0 0 0 1 B
2 D	0 0 1 0 B
3 D	0 0 1 1 B
4 D	0 1 0 0 B
5 D	0 1 0 1 B
6 D	0 1 1 0 B
7 D	0 1 1 1 B
8 D	1 0 0 0 B
9 D	1 0 0 1 B

Fig. 3.10 – Representação em BCD

A figura 3.11 ilustra a representação dos números 89D e 123D, na notação BCD.

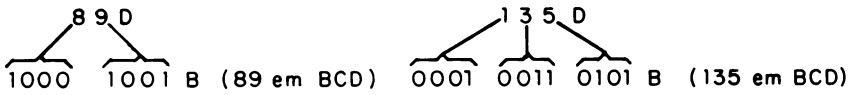


Fig. 3.11 – Exemplos de nºs BCD

3.5 – OS BITS DE CONDIÇÃO H e N

Os bits de condição H e N não podem ser testados pelas instruções de desvio e servem para auxiliar nas operações aritméticas com números na representação BCD.

O bit H setado (H = 1) simboliza a ocorrência de "vai um" do quarto para o quinto bit do resultado.

O bit N setado (N = 1) simboliza que a operação realizada foi de subtração e, caso contrário (N = 0), simboliza a realização de uma operação de adição.

Na realidade, a Unidade Lógica e Aritmética (ULA) do Z-80 foi projetada pera operar números na representação 2's. No entanto, em muitas aplicações fica mais cômodo operar números em BCD, principalmente, quando envolve a manipulação de dados de entrada ou saída.

O Z-80 dispõe da instrução "DAA", que realiza os ajustes dos resultados obtidos quando números BCD são adicionados ou subtraídos pela ULA. Isto porque a ULA assume que está operando números representados na notação 2's. A figura 3.12 ilustra a função da instrução "DAA". Observe que os bits de condição H e N também são parâmetros de entrada desta instrução.

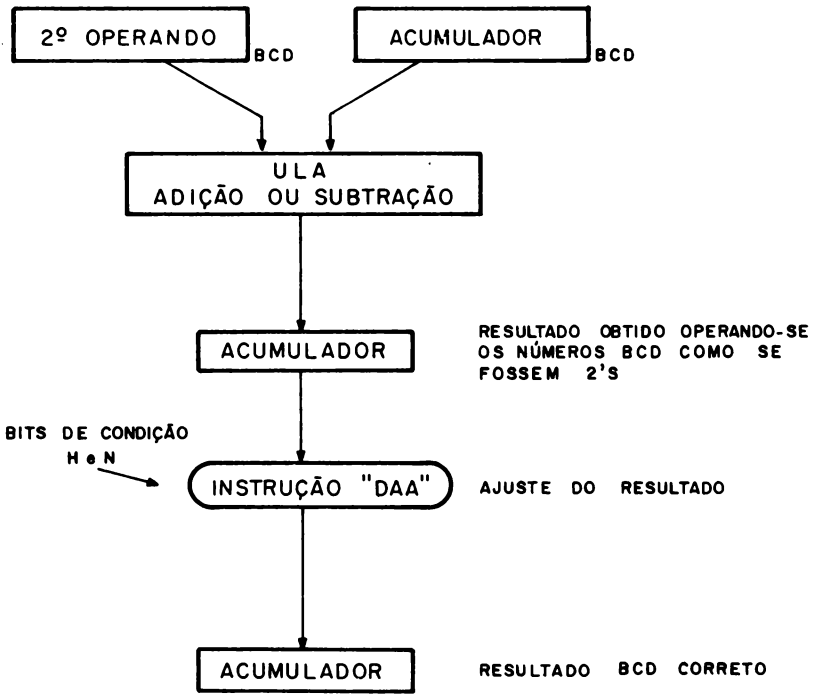


Fig. 3.12 – Instrução DAA

Para o caso de operações de adição, isto é, se N = 0 o resultado é ajustado pela instrução 'DAA" da seguinte maneira:

1. Se H = 1, somar seis ao dígito BCD menos significativo.

2. Se C = 1, somar seis ao dígito BCD mais significativo.

3. Somar seis ao dígito BCD que tiver valor superior a 1001B.

A figura 3.13 mostra alguns exemplos de operações de adição com números BCD.

EX Nº 1 ADICIONAR 12D E 26D EM BCD

```
  0 0 0 1 | 0 0 1 0   (12 BCD)
  0 0 1 0 | 0 1 1 0   (26 BCD)
  -------   -------
  0 0 1 1 | 1 0 0 0   (38 BCD) C = 0, H = 0 (Neste caso a
                                instrução DAA não realiza
                                nenhuma alteração)
```

EX Nº 2 ADICIONAR 29D E 39D EM BCD

```
                0 0 1 0 | 1 0 0 1   (29 BCD)
                0 0 1 1 | 1 0 0 1   (39 BCD)
                -------   -------
Instrução       0 1 1 0 | 0 0 1 0   (62 BCD É ERRADO) C = 0, H = 1
"DAA" ------>   0 0 0 0 | 0 1 1 0   (Ajusta somando + 6 ao dígito menos significativo)
                -------   -------
                0 1 1 0 | 1 0 0 0   (68 BCD)
```

EX Nº 3 ADICIONAR 94D E 93D EM BCD

```
                1 0 0 1 | 0 1 0 0   (94 BCD)
                1 0 0 1 | 0 0 1 1   (93 BCD)
                -------   -------
  c [1] <---    0 0 1 0 | 0 1 1 1   (27 BCD É ERRADO) C = 1, H = 0
Instrução
"DAA"           0 1 1 0 | 0 0 0 0   (Ajusta somando + 6 ao dígito mais significativo)
                -------   -------
  c [1]         1 0 0 0 | 0 1 1 1   (87 BCD COM C = 1 CORRETO)
```

EX Nº 4 ADICIONAR 99D E 98D EM BCD

```
                1 0 0 1 | 1 0 0 1   (99 BCD)
                1 0 0 1 | 1 0 0 0   (98 BCD)
                -------   -------
  c [1] <---    0 0 1 1 | 0 0 0 1   (31 BCD É ERRADO) C = 1, H = 1
Instrução
"DAA"           0 1 1 0 | 0 1 1 0   (Ajusta somando + 6 a ambos os dígitos)
                -------   -------
  c [1]         1 0 0 1 | 0 1 1 1   (97 BCD COM C = 1 CORRETO)
```

Fig. 3.13 – Exemplos de Adição com nºs BCD

Para o caso de operações de subtração, isto é, se N = 1 o resultado é ajustado pela instrução "DAA", subtraindo seis ao dígito BCD nas seguintes condições:

1. Se H = 1, subtrair seis ao dígito BCD menos significativo.

2. Se C = 1, subtrair seis ao dígito BCD mais significativo.

3. Se C = 1 e H = 1, subtrair seis de ambos os dígitos BCD.

A figura 3.14 mostra alguns exemplos de operações de subtração com números BCD. Note que os resultados dos exemplos 3 e 4 são representados na notação complemento a 10 (10's), que é semelhante à forma binária 2's.

EX Nº 1 SUBTRAIR 10 D DE 98 D EM BCD

```
  1001 | 1000  (98 BCD)
 -0001 | 0000  (10 BCD)
 ------+------
  1000 | 1000  (88 BCD) C = 0, H = 0  (Neste caso a
                                        instruçao DAA não realiza
                                        nenhuma alteração)
```

EX Nº2 SUBTRAIR 19 D DE 91 D EM BCD

```
          1001 | 0001  (91 BCD)
         -0001 | 1001  (19 BCD)
         ------+------
Instrução 0111 | 1000  (78 BCD É ERRADO) C = 0, H = 1
"DAA"     0000 | 0110  (Ajusta sub. 6 do dígito BCD menos significativo)
         ------+------
          0111 | 0010  (72 BCD COM C = 0, CORRETO)
```

EX Nº 3 SUBTRAIR 91D DE 19 D EM BCD

```
          0001 | 1001  (19 BCD)
         -1001 | 0001  (91 BCD)
         ------+------
  c[1]    1000 | 1000  (88 BCD É ERRADO) C = 1, H = 0

          0110 | 0000  (Ajusta sub. 6 do dígito BCD mais significativo)
         ------+------
  c[0]    0010 | 1000  (28 BCD COM C = 0 CORRETO)
```

EX Nº 4 SUBTRAIR 99D DE 11D EM BCD

```
          0001 | 0001  (11 BCD)
         -1001 | 1001  (99 BCD)
         ------+------
  c[1]    0111 | 1000  (78 BCD É ERRADO) C = 1, H = 1

          0110 | 0110  (Ajusta sub. 6 de ambos os dígitos)
         ------+------
  c[1]    0001 | 0010  (12 BCD COM C = 1 CORRETO)
```

Fig. 3.14 – Exemplos de Subtração com nºs BCD

O 10's é obtido subtraindo-se cada dígito de nove e somando-se um ao dígito menos significativo do resultado anterior.

As operações de subtração com números BCD representados na notação 10's são realizadas somando-se o minuendo ao subtraendo, complementado a 10, e desprezando-se o possível "vai um".

Por exemplo, o 10's de 72 é 27 + 1 = 28, que corresponde ao resultado do exemplo 3 da figura 3.14.

O exemplo abaixo compara uma operação de subtração normal e utilizando a notação 10's:

subtração normal	subtração com 10's
91D	91D
- 72D	+ 28D
19D	19D
	1 despreza o "vai um"

Da mesma forma que na notação 2's, na notação 10's o bit mais à esquerda representa o sinal, respeitando-se a convenção de 1 para sinal negativo e 0 para o sinal positivo.

Como exemplo, temos uma operação de adição com números de sinais diferentes:

+ 692D	= 0.692D
- 342D	= 1.658D
+ 350D	0.350D
	1 despreza o "vai um"

Observe que neste tipo de representação é muito mais prático utilizar o carry bit como sinal do resultado, de maneira semelhante à representação complemento a 2 sem sinal.

3.6 – EXERCÍCIOS DE FIXAÇÃO

3.6.1 – Quais os bits que podem ser testados pelas instruções de mudança de seqüência?

3.6.2 – Qual a posição de cada bit no registrador F ou F'?

3.6.3 – Por que o hardware do microprocessador Z-80 foi implementado para operar números na representação 2's?

3.6.4 Para os nºs apresentados abaixo, determine o complemento 2'S de cada um deles:

a) 01101110B	d) 10000011B	g) 8DH
b) 10100011B	e) 10000100B	h) 7FH
c) 01111110B	f) 01101110B	i) 34D

3.6.5 – Qual é a faixa de nºs representados na notação 2's com sinal quando utilizamos um byte de tamanho de palavra?

3.6.6 – Qual a função do bit S?

3.6.7 – Determine o resultado das subtrações apresentadas abaixo, utilizando-se a notação complemento 2's com sinal:

a) 34D – 27D	d) 53D – 72D	g) 7DH – 2FH
b) 26D – 34D	e) 12D – 11D	h) F5H – 37H
c) 79D – 63D	f) 15D – 17D	i) 8BH – 5CH

3.6.8 – Para as subtrações do exercício 3.6.7, determine o valor dos bits de condição para cada caso.

3.6.9 – Como um número na representação 2's, com mais de 8 bits pode ser representado na memória do Z-80?

3.6.10 – Observe que para se obter o 2's de um número de 8 bits podemos, também, subtraí-lo de 100000000B. A partir disto, explique porque o "vai um" em operações com números na notação 2's é desprezado.

3.6.11 – Quais são as informações fornecidas pelos bits de condição S e Z?

3.6.12 – O que é overflow?

3.6.13 – Determine um algorítmo que identifica se houve overflow em uma operação aritmética com números 2's sem usar o bit P/V.

3.6.14 – Quais são as informações fornecidas pelo bit de condição P/V, fornecendo resultado de overflow ou de paridade?

3.6.15 – Como podemos saber se o bit de condição P/V nos informa sobre a condição de overflow ou paridade?

3.6.16 – Dê uma solução, para recuperarmos o resultado correto, no caso de obtermos um resultado de uma operação com overflow.

3.6.17 – Como testar se o resultado de uma operação com números representados na notação 2's é nulo?

3.6.18 – Como testar se o resultado de uma operação com números 2's, representados com o sinal, é positivo ou negativo?

3.6.19 – Dê uma vantagem para o uso de números 2's sem o sinal?

3.6.20 – Por que as operações de subtração no microprocessador Z-80 invertem o bit C?

3.6.21 – Diga como deve ser interpretado o bit C em operações com números 2's sem o sinal se:

a) os operandos forem de sinais contrários;
b) os dois operandos tiverem sinais negativos.

3.6.22 – Explique a função dos bits de condição H e N.

3.6.23 – Qual a diferença entre os bits P/V e C após as operações aritméticas?

3.6.24 – O que são números representados na notação BCD?

3.6.25 – Dê duas aplicações em que é interessante o uso da representação BCD.

3.6.26 – Como atua a instrução DAA se N = 0?

3.6.27 – Como atua a instrução DAA se N = 1?

3.6.28 – Determine um algorítmo que converta números na representação 2's para BCD e, outro algorítmo que faça a operação inversa.

3.6.29 – Determine uma forma de testar se o resultado de uma operação com números representados em BCD é nulo.

3.6.30 – Determine uma forma de testar o sinal do resultado de operações com números representados em BCD.

3.6.31 – Os nºs abaixo estão representados na notação BCD. Realize as operações em binário e, em seguida, corrija o resultado explicando a atuação da instrução DAA.

a) 23D + 78D
b) 15D + 17D
c) 29D + 39D
d) 35D – 27D
e) 82D – 48D
f) 19D – 28D

4 MODOS DE ENDEREÇAMENTO

4.1 – APRESENTAÇÃO

Após termos estudado no capítulo anterior a relação entre as operações aritméticas e os bits de condição, apresentaremos os modos de endereçamento que são fundamentais para o perfeito entendimento das instruções deste microprocessador.

Adiantamos, ainda, que no estudo do conjunto de instruções do Z-80, para fins didáticos, classificaremos subgrupos de instruções com base nas suas técnicas de endereçamento.

4.2 – INTRODUÇÃO

O código objeto de uma instrução na memória tem um campo que designa o código de operação e outro que designa o operando. A figura 4.1 ilustra a instrução armazenada na memória.

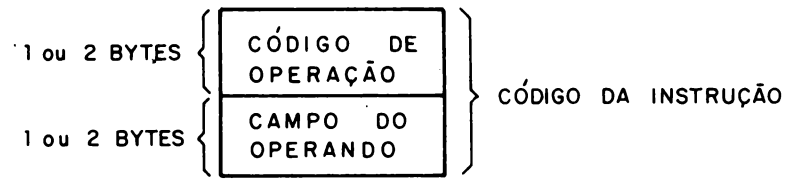


Fig. 4.1 – A Instrução na Memória

Dentro do variado repertório de instruções do Z-80, que começaremos a estudar no próximo capítulo, encontraremos algumas instruções que não necessitam do campo do operando e que podem ser executadas durante os últimos períodos do ciclo M1 (busca de instrução). No entanto, a maioria das instruções requerem o campo do operando para consecução de suas funções. Estes operandos podem ser conteúdos de registradores internos do Z-80 ou de posições de memória.

Os modos de endereçamento do Z-80 são as formas pelas quais as suas instruções podem designar os seus operandos.

Por exemplo, uma instrução de soma (ADD) opera dois números de 8 bits. Um dos operandos é o conteúdo do acumulador, enquanto que o segundo operando pode estar contido em um registrador interno do Z-80 ou em uma posição de memória.

Os modos de endereçamento do Z-80 classificam-se em:

1. Implícito
2. Imediato
3. Imediato estendido
4. Por registrador
5. Por registrador indireto
6. Estendido
7. Página zero
8. Relativo
9. Indexado
10. Por bit
11. Por Ponteiro de Pilha

4.3 – ENDEREÇAMENTO IMPLÍCITO

As instruções com endereçamento implícito não possuem campo de operando. O mesmo fica subentendido no próprio código de operação. Assim, os códigos de operação destas instruções são fixos.

Por exemplo, na instrução CPL que inverte cada bit do acumulador (1's) o operando é fixo, sendo sempre o acumulador. A figura 4.2 mostra o formato e a operação desta instrução.

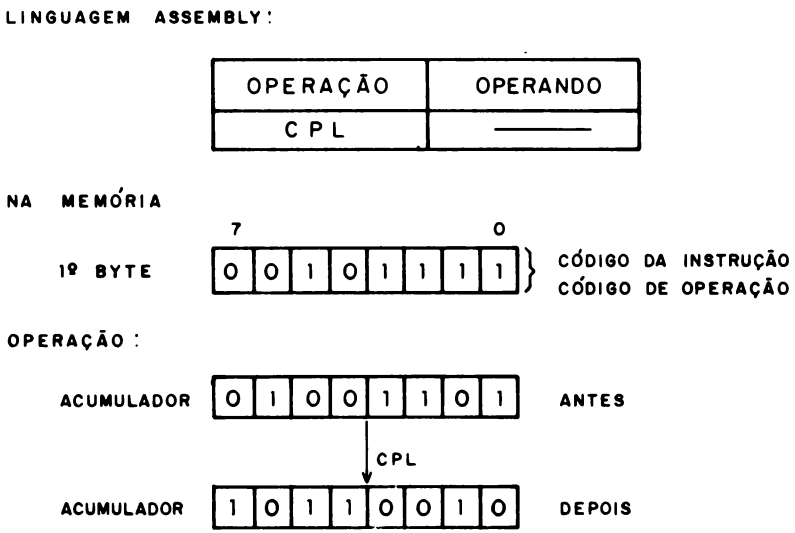


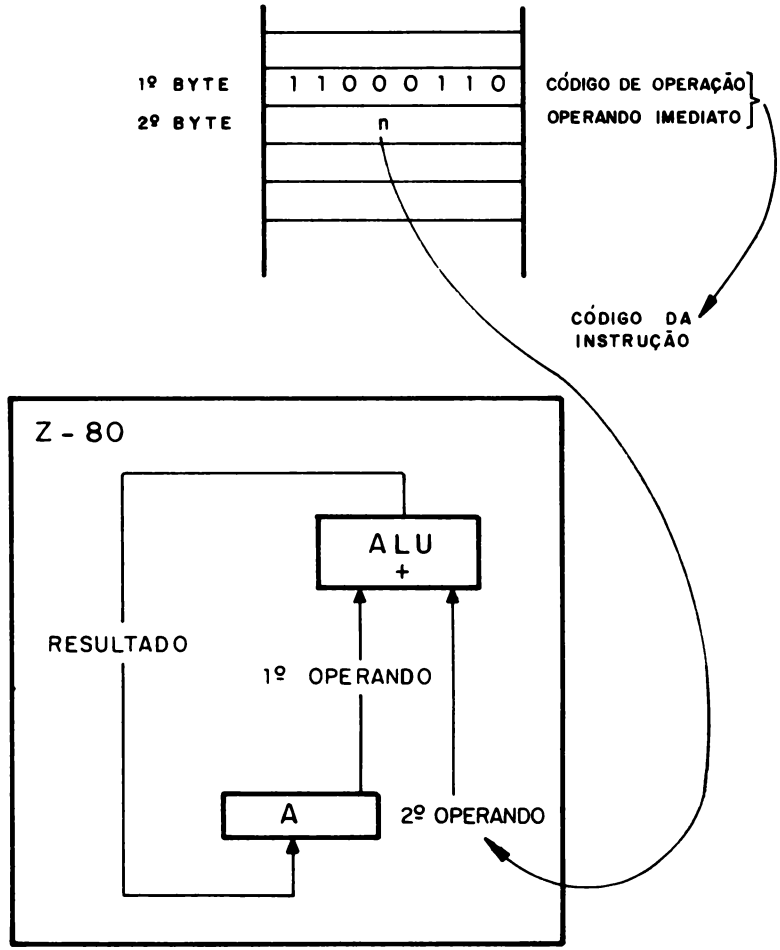
Fig. 4.2 – Endereçamento Implícito

4.4 – ENDEREÇAMENTO IMEDIATO

No modo de endereçamento imediato ou o segundo ou o terceiro ou o quarto byte do código da instrução é o próprio operando.

Tal categoria de endereçamento é muito útil quando desejamos operar constantes de 8 bits.

Por exemplo, a instrução ADD A,n soma o conteúdo do acumulador ao conteúdo do seu segundo byte, e deposita o resultado no acumulador. Note que o código desta instrução varia em função da constante que o programador escolhe para ser adicionada ao acumulador. A figura 4.3 mostra o formato e a operação desta instrução.



LINGUAGEM ASSEMBLY :

OPERAÇÃO	OPERANDO
ADD	A, n

Fig. 4.3 – Endereçamento Imediato

4.5 – ENDEREÇAMENTO IMEDIATO ESTENDIDO

As instruções com endereçamento imediato estendido possuem operandos de 16 bits que fazem parte do código da instrução.

Esta categoria de endereçamento é usada principalmente para carregar constantes em registradores de 16 bits (IX, IY e SP) e em pares de registradores (BC, DE e HL).

Por exemplo, a instrução LD IY,nn carrega o registrador IY com 16 bits imediatos. A figura 4.4 ilustra o formato e a operação desta instrução.

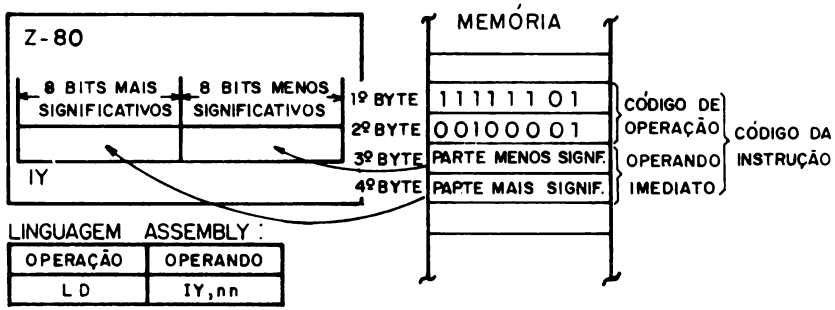


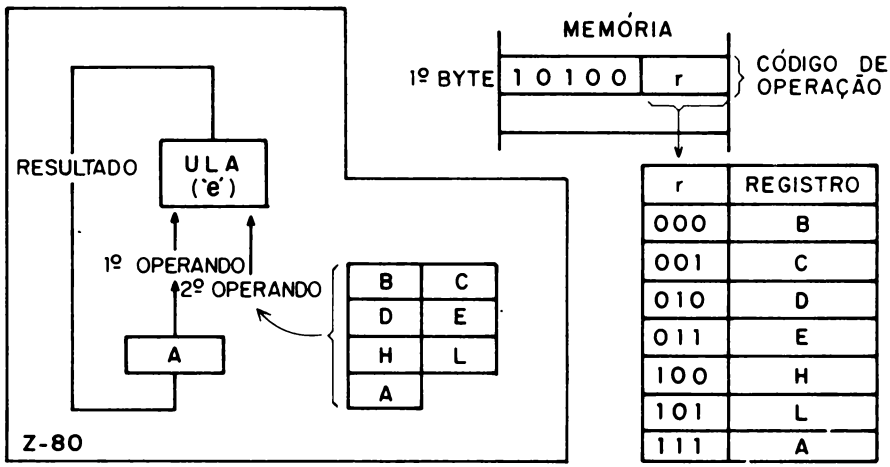
Fig. 4.4 – Endereçamento Imediato Estendido

Observe que o código de operação desta instrução é composto de dois bytes, e os dois bytes seguintes armazenam o operando imediato. A parte relativa aos oito bits menos significativos do operando fica no 3º byte, e os outros oito bits mais significativos ocupam o 4º byte.

4.6 – ENDEREÇAMENTO POR REGISTRADOR

No modo de endereçamento por registrador um ou mais registradores do Z-80 são usados como operandos da instrução. Para tanto, o código de operação destas instruções contém campos que especificam os registradores do Z-80, que devem ser usados na execução da instrução.

Por exemplo, a instrução AND r faz o "e lógico" do acumulador com um registrador r de uso geral, e deposita o resultado no acumulador. A figura 4.5 ilustra o formato e a operação desta instrução.



LINGUAGEM ASSEMBLY:

OPERAÇÃO	OPERANDO
AND	r

Fig. 4.5 – Endereçamento por Registrador

Observe que os 5 bits mais significativos do código de operação são fixos e os três menos significativos selecionam o registrador de uso geral (2º operando), de acordo com a tabela mostrada na figura 4.5.

4.7 – ENDEREÇAMENTO POR REGISTRADOR INDIRETO

No modo de endereçamento por registrador indireto, também chamado por ponteiro, o operando fica em uma posição de memória cujo endereço está contido em um par de registradores BC, DE ou HL.

Esta técnica de endereçamento é útil para acessar posições de memória de endereço aleatório.

Por exemplo, a instrução INC (HL) incrementa de uma unidade o conteúdo da posição de memória endereçada pelo par HL. A figura 4.6 mostra o formato e a operação desta instrução.

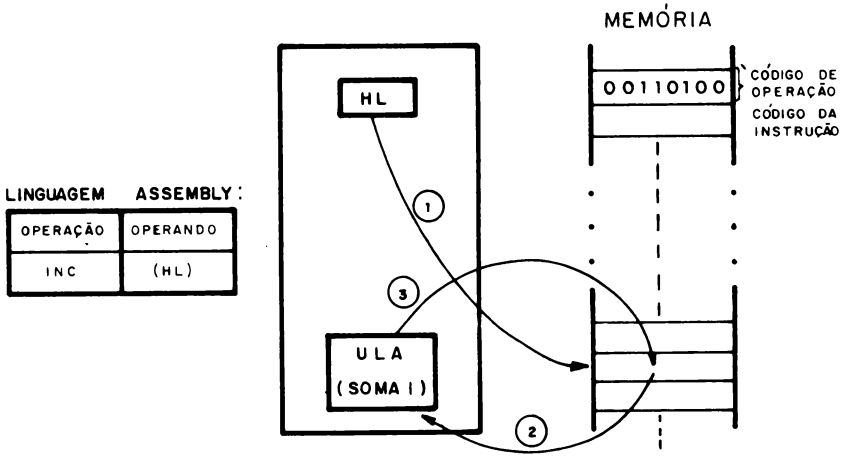


Fig. 4.6 – Endereçamento por Registrador Indireto

Observe que esta operação de incremento possui apenas um operando e o resultado é depositado na própria origem. Na linguagem Assembly do Z-80 o conteúdo de uma posição de memória é representado entre parênteses.

4.8 – ENDEREÇAMENTO ESTENDIDO

No modo de endereçamento estendido, também chamado de endereçamento direto, a instrução contém no seu próprio código o endereço do operando.

Da mesma forma que no modo de endereçamento por registrador indireto, a presente técnica permite o acesso a posições de memória aleatórias. No entanto, as instruções que utilizam a técnica de endereçamento estendido necessitam de um número maior de bytes para armazenar o seu código, pois nele está contido o endereço do operando.

O exemplo clássico de uma instrução com endereçamento estendido é a LD A, (nn), cujo formato e operação encontram-se ilustrados na figura 4.7.

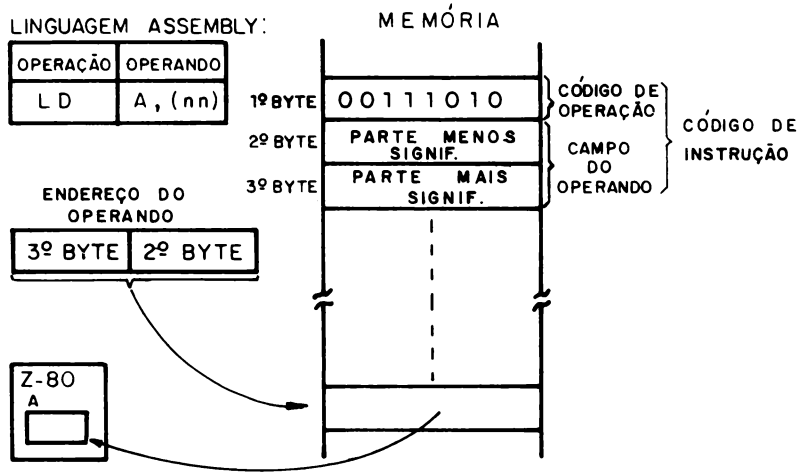


Fig. 4.7 – Endereçamento Estendido

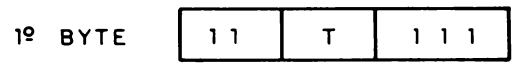
O 2º e o 3º byte da instrução LD A, (nn) especificam uma posição de memória cujo conteúdo é transferido para o acumulador. Observe que a parte menos significativa do endereço fica armazenada no 2º byte, e a mais significativa no 3º byte.

4.9 – ENDEREÇAMENTO PÁGINA ZERO

Este modo de endereçamento é usado somente na instrução RST T (RESTART). A ação desta instrução é armazenar o conteúdo do contador de programa no topo da pilha e efetuar um desvio no programa para uma das oito posições fixas de memória na página zero.

A página zero é a região de memória que pode ser endereçada com os oito bits menos significativos da barra de endereços, mantendo os seus oito bits mais significativos em zero.

O formato da instrução RST T está mostrado na figura 4.8. O campo "T" ocupa três bits e dependendo do estado destes, o desvio ocorre para os endereços 0H, 8H, 10H, 18H, 20H, 28H, 20H ou 38H. As instruções RESTART serão abordadas com mais detalhes quando estudarmos o grupo de instruções chamada de subrotina e retornos do conjunto de instruções do Z-80.



ENDEREÇO DE DESVIO NA PÁGINA ZERO	VALOR DE T	ASSEMBLY
0 0 0 0 H	0 0 0 B	RST 0 0 H
0 0 0 8 H	0 0 1 B	RST 0 8 H
0 0 1 0 H	0 1 0 B	RST 1 0 H
0 0 1 8 H	0 1 1 B	RST 1 8 H
0 0 2 0 H	1 0 0 B	RST 2 0 H
0 0 2 8 H	1 0 1 B	RST 2 8 H
0 0 3 0 H	1 1 0 B	RST 3 0 H
0 0 3 8 H	1 1 1 B	RST 3 8 H

Fig. 4.8 – Endereçamento Página Zero

4.10 – ENDEREÇAMENTO RELATIVO

O modo de endereçamento relativo permite o acesso a 256

posições de memória localizadas em torno do endereço da instrução corrente.

O endereçamento relativo no Z-80 é usado somente no grupo de instruções de mudança de seqüência do programa: JUMP'S condicionais e incondicionais.

Nestas instruções, o segundo byte é chamado de deslocamento ou "displacement" e representa um número algébrico na notação complemento a 2 (2's) na faixa de -128D a + 127D (1000000B a 01111111B). Tal valor é adicionado ao conteúdo do contador de programa e o resultado desta operação é o endereço efetivo da instrução que deve dar continuidade ao programa. A figura 4.9 ilustra a técnica de endereçamento relativo.

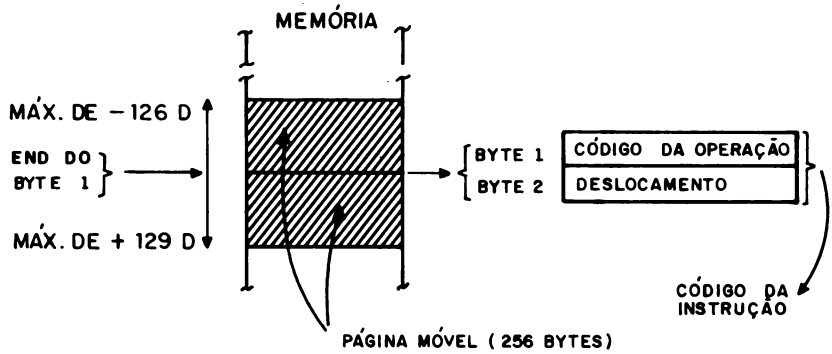


Fig. 4.9 – Endereçamento Relativo

Uma vez que as instruções com endereçamento relativo ocupam dois bytes, o endereço efetivo fica na faixa de -126D a + 129D em relação ao primeiro byte da instrução. O conjunto de 256 bytes que pode ser acessado pelo endereçamento relativo é chamado de *página móvel.*

Como exemplo podemos citar a instrução JR Z, e, que causa um desvio no curso normal do programa se o bit de condição Z for igual a "1". A figura 4.10 ilustra esta instrução com deslocamento igual 11D.

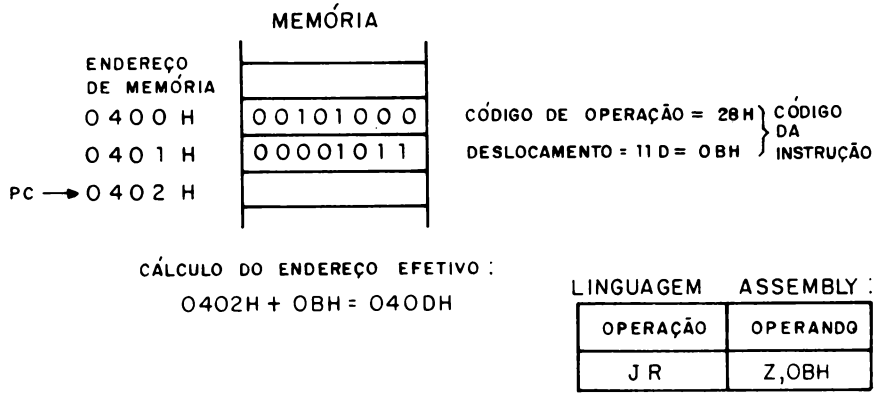


Fig. 4.10 – Instrução JR Z, 0BH

Neste caso da instrução JR Z, e, se o bit de condição Z for igual a "1" o programa prossegue na posição de memória 040DH. Se o bit Z for igual a zero, o programa prossegue na orientação natural do contador de programa, isto é, na posição de memória 0402H.

4.11 – ENDEREÇAMENTO INDEXADO

A figura 4.11 mostra o formato geral das instruções com endereçamento indexado. Observe que o código de operação é composto de dois bytes; o terceiro byte contém o valor do deslocamento "d" e o quarto byte pode existir ou não, dependendo da natureza da instrução.

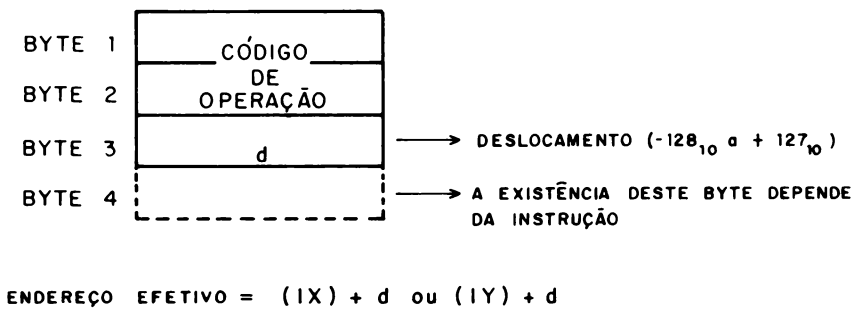


Fig. 4.11 – Endereçamento Indexado

Consideremos, por exemplo, a instrução LD (IY + d),n. Esta instrução transfere o operando imediato n (byte 4) para a posição de memória, cujo endereço é dado pela soma do conteúdo de IY com o byte do deslocamento "d" (byte 3).

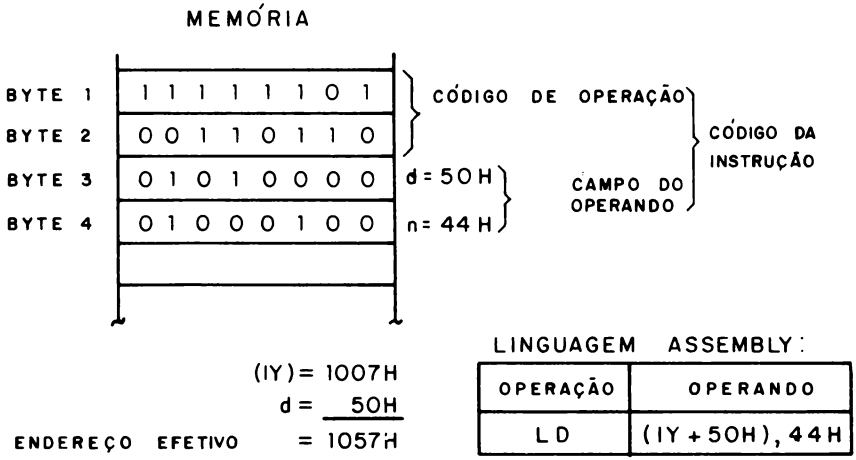


Fig. 4.12 – A Instrução LD (IY + d),n

A figura 4.12 ilustra o formato e a operação desta instrução atribuindo os seguintes valores numéricos: (IY) = 1007H,d = 50H e n = 44H. Após a execução desta instrução, com estes valores, teremos o valor 44H armazenado na posição da memória 1057H.

4.12 – ENDEREÇAMENTO POR BIT

O modo de endereçamento por bit permite o acesso individual a qualquer um dos oito bits de um byte. Esta técnica de endereçamento conjugada com as demais já apresentadas permite setar, resetar e testar qualquer bit de um registrador do Z-80 ou de uma posição de memória.

Por exemplo, a instrução SET b, r seta o bit "b" do registrador de uso geral r, e o seu formato geral está mostrado na figura 4.13

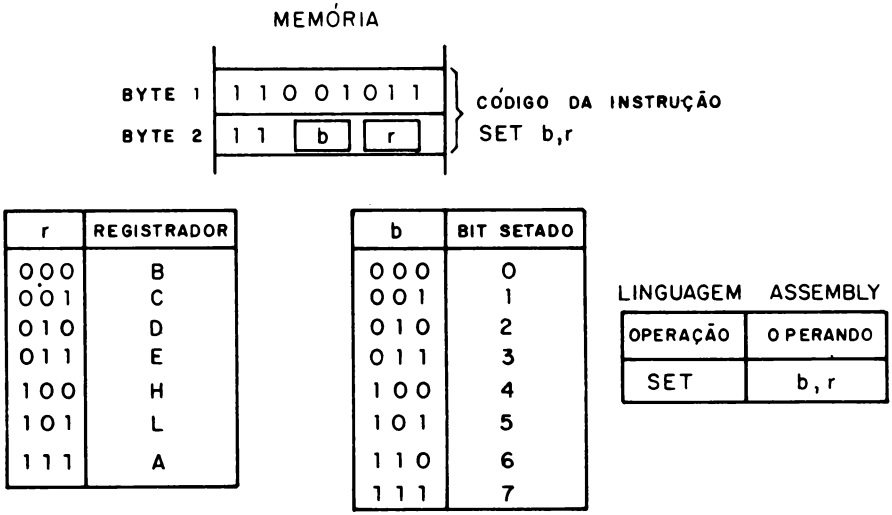


Fig. 4.13 – A Instrução SET b,r

Para b = 2 e r = E temos a instrução SET 2,E, ilustrada na figura 4.14, que seta o bit 2 do registrador E. Suponha que o conteúdo inicial do registrador E seja F0H.

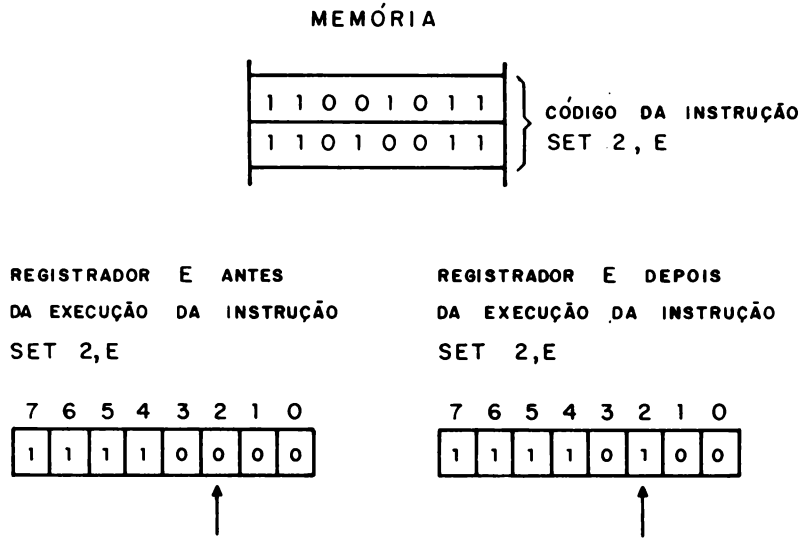


Fig. 4.14 – A Instrução SET 2,E

4.13 – ENDEREÇAMENTO POR PONTEIRO DE PILHA

No modo de endereçamento por ponteiro de pilha o operando fica na posição de memória endereçada pelo ponteiro de pilha (SP). Na realidade, este tipo de endereçamento poderia ficar englobado no modo de endereçamento por registrador indireto. No entanto, para fins didáticos, classificamos à parte as instruções que utilizam o ponteiro de pilha.

Apresentaremos as instruções PUSH e POP, que são casos típicos de endereçamento por ponteiro de pilha.

4.13.1 – A Instrução PUSH

Nas instruções de PUSH os operandos de 16 bits são transferidos para a pilha. Estes operandos podem ser: os pares AF, BC, DE ou HL e os registradores IX e IY. Os endereços de memória acessados durante uma operação de PUSH são determinados da seguinte maneira:

- Os oito bits mais significativos do operando de 16 bits (A, B, D, H, IX_H e IY_H) são armazenados no endereço de memória fornecido pelo conteúdo do SP decrementado de uma unidade.

- Os oito bits menos significativos do operando de 16 bits (F, C, E, L, IX_L e IY_L) são armazenados no endereço de memória fornecido pelo conteúdo do SP decrementado de 2 unidades.

- O SP é automaticamente decrementado de 2 (SP ← SP-2).

Como exemplo, a figura 4.15 mostra como transcorre a instrução PUSH BC, em que o par BC é transferido para a pilha.

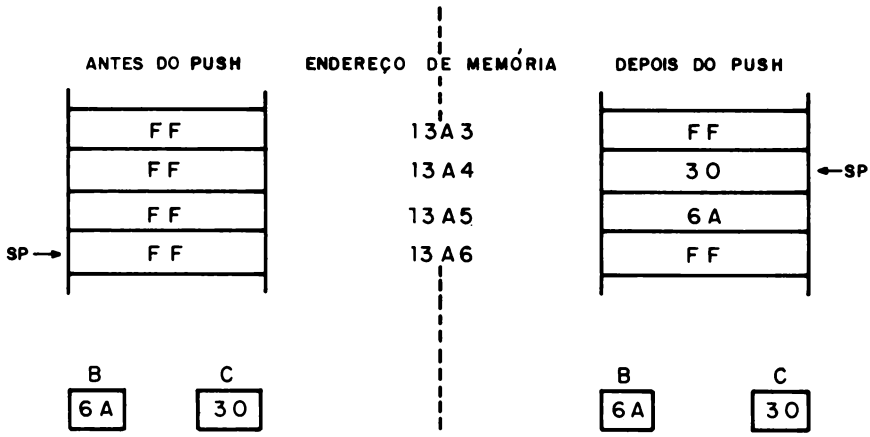


Fig. 4.15 – A Instrução PUSH BC

4.13.2 – A Instrução POP

Nas instruções de POP os operandos de 16 bits são transferidos da pilha para os seguintes registradores no Z-80: AF, BC, DE, HL, IX ou IY. Os endereços de memória acessados durante uma operação de POP são determinados da seguinte maneira:

- O conteúdo da posição de memória endereçada pelo SP é transferido para a parte menos significativa do registrador do Z-80 (F, C, E, L, IX_L, IY_L).

- O conteúdo da posição de memória endereçada pelo SP incrementado de uma unidade é transferido para a parte mais significativa do registrador do Z-80 (A, B, D, H, IX_H e IY_H).

- O SP é automaticamente incrementado de 2 (SP ← SP + 2).

Como exemplo, a figura 4.16 mostra como transcorre a instrução POP HL, em que o par HL é carregado com o conteúdo das posições localizadas no topo da pilha.

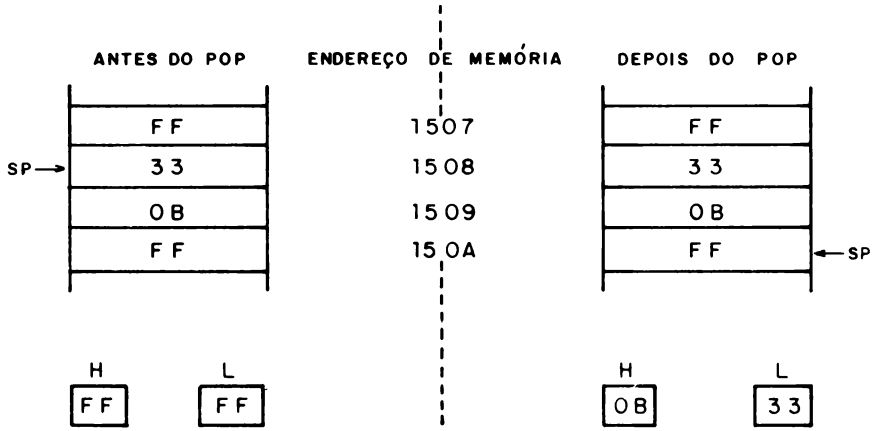


Fig. 4.16 – A Instrução POP HL

4.14 – COMPARANDO OS MODOS DE ENDEREÇAMENTO do Z-80 e do 8080

O microprocessador 8080 possui oito dos onze modos de endereçamento do Z-80 exceto o relativo, o indexado e o por bit.

Assim, por exemplo, para manipularmos bits com o 8080 torna-se necessário o uso de máscaras, o que envolve no mínimo duas instruções.

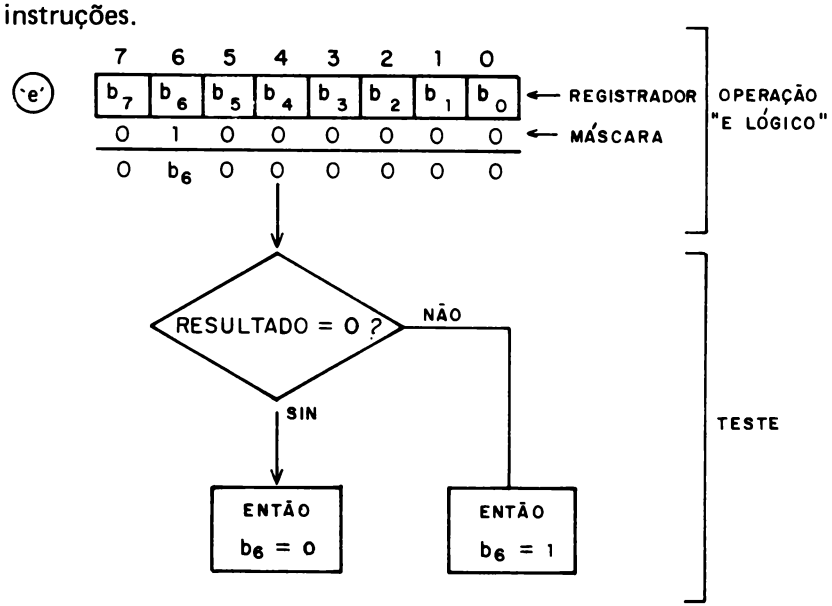


Fig. 4.17 – Uso de Máscara no 8080

A figura 4.17 ilustra o uso da máscara para testar o bit 6 de um registrador. Observe que é necessário pelo menos uma operação "e lógico" e um teste.

A indexação mostra-se muito útil em programas que utilizam tabelas na memória.

O endereçamento relativo além de facilitar, em alguns casos, a relocação de programas, também reduz a quantidade de memória necessária, porque uma instrução de desvio relativo só possui dois bytes.

Podemos concluir que os três modos de endereçamento acrescidos ao Z-80 vieram facilitar a programação, reduzir a quantidade de memória necessária aos programas e, na maioria dos casos, contribuir para aumentar a velocidade de processamento.

4.15 – EXERCÍCIOS DE FIXAÇÃO

4.15.1 – Quais os modos de endereçamento do Z-80?

4.15.2 – Qual a principal característica do formato do código das instruções que possuem endereçamento implícito?

4.15.3 – Qual a diferença entre os modos de endereçamento imediato e imediato estendido?

4.15.4 – Em que aplicações você usaria o modo de endereçamento imediato?

4.15.5 – Qual o nº mínimo de bytes de uma instrução com endereçamento imediato estendido?

4.15.6 – Explique como é feita a identificação dos operandos no endereçamento por registrador.

4.15.7 – Porque o endereçamento por registrador indireto é chamado "por ponteiro"?

4.15.8 – Explique o modo de endereçamento estendido.

4.15.9 – Cite uma vantagem e uma desvantagem do endereçamento estendido quando comparado com o endereçamento por registrador indireto.

4.15.10 – O que é página zero?

4.15.11 – O que é página móvel no endereçamento relativo?

4.15.12 – Cite duas vantagens da utilização do endereçamento relativo?

4.15.13 – Cite uma aplicação em que é útil a utilização do modo de endereçamento indexado.

4.15.14 – No endereçamento por bit quais os operandos necessários nas instruções?

4.15.15 – Admita que os valores iniciais dos registradores DE, SP e BC são, respectivamente, 03FAH, 1000H e F3C9H. Determine a situação final da pilha e dos registradores envolvidos após a execução seqüencial das instruções PUSH DE, PUSH DE e POP BC.

5 A LINGUAGEM ASSEMBLY

5.1 – APRESENTAÇÃO

No capítulo 1 apresentamos os conceitos básicos existentes nesta linguagem. Agora, vamos introduzir a *sintaxe* e as *pseudo-instruções* para podermos, durante o estudo das instruções do Z-80, escrever programas usando os recursos e as facilidades da linguagem assembly.

A referência para este capítulo é o programa montador M-80 da Microsoft que reconhece instruções dos microprocessadores Z-80 e 8080, sendo encontrado disponível no mercado.

5.2 – SINTAXE

Temos quatro campos que são identifidados e agrupados do seguinte modo:

[endereço:] [código de operação] [operando] [; comentários]

A separação entre os campos é feita por um ou qualquer de espaços (maior que um), sendo que é considerado pelo montador somente o primeiro espaço, os outros são descartados.

O *campo de endereço* ("LABEL"), quando existe, inicia-se na primeira coluna e termina com o caractere ":". Este campo e usado quando desejamos referenciar uma instrução, por exemplo, em instruções de desvio.

O campo de *código de operação* é usado para armazenar, basicamente, os mnemônicos das instruções do Z-80 e do 8080. Este campo também é usado para armazenar as pseudo-instruções que serão estudadas no ítem 5.7 deste capítulo.

O terceiro campo é o de *operando* que armazena os operandos das instruções. Estes podem ser o resultado de operações aritméticas e lógicas.

O último campo é o de *comentários* e começa com ";". O conteúdo deste campo não é analisado pelo montador e serve apenas para ajudar a compreensão do programa. Para começarmos uma linha com comentários, basta colocar na primeira coluna o caractere ";".

Na figura 5.1 temos um exemplo de um programa fonte escrito em assembly.

```
; EXEMPLO DE PROGRAMA
; TRANSFERE BLOCO DE MEMÓRIA
ORIGEM    EQU     1000H
DESTINO   EQU     2000H
ELEM      EQU     00FFH
          ORG     100H          ; ORIGEM 100H
INÍCIO:   LD      HL, ORIGEM    ; HL ← END. ORIGEM
          LD      DE, DESTINO   ; DE ← END. DESTINO
          LD      BC, ELEM      ; BC ← Nº DE ELEMENTOS
          LDIR                  ; TRANSFERE ATÉ BC  0
          HALT                  ; PARE
          END                   ; FIM
```

Fig. 5.1 - Exemplo de Programa

Neste exemplo da figura 5.1 os mnemônicos LD, LDIR e HALT são instruções do Z-80 e os mnemônicos EQU, ORG e END são pseudo-instruções que o programa montador necessita para gerar corretamente o código de máquina.

5.3 – SÍMBOLOS VÁLIDOS

A – Z 0-9 . ? $ @

5.4 – CONSTANTES NUMÉRICAS

nnnn B – binário
nnnn D – decimal
nnnn O – octal
nnnn Q – octal
nnnn H – hexadecimal
X' nnnn' – hexadecimal

5.5 – STRINGS

Os strings de caracteres são delimitados por aspas (ex: "strings"). O montador associa a cada caractere o código ASCII correspondente. A tabela do código ASCII encontra-se no Apêndice C.

5.6 – OPERAÇÕES ARITMÉTICAS E LÓGICAS

NULL
LOW, HIGH
* , / , MOD, SHR, SHL
+ , –
EQ, NE, LT, LE, GT, GT
NOT, AND, OR, XOR

5.7 – PSEUDO-INSTRUÇÕES

A parte responsável pela geração do código objeto, em cada linha de um programa fonte escrito na linguagem Assembly, é o código da instrução.

No entanto, os montadores, via de regra, dispõe de algumas instruções, cujos mnemônicos são colocados no campo do código de instrução, que não geram código objeto. O objetivo é auxiliar o montador no processo de montagem. Estas instruções são denominadas *PSEUDO-INSTRUÇÕES* ou *DIRETIVAS,* porque, na realidade, não são códigos operacionais representativos de instruções de máquina.

Na maioria dos montadores existem cinco tipos básicos de pseudo-instruções que realizam as seguintes funções:

1 – Definição da área de montagem do programa;
2 – Definição de símbolos;
3 – Reserva de memória;
4 – Indicação de final do programa fonte;
5 – Montagem condicional.

Existem outras pseudo-instruções peculiares a cada montador que implementam funções tais como: a colocação de cabeçalhos em páginas, mudança de página e outras funções. No entanto, neste texto, daremos atenção especial aos tipos básicos de pseudo-instruções mencionados anteriormente.

Na maioria dos montadores, no campo de endereço podemos ter *nomes* ou *endereços*. Os *nomes* são colocados nas pseudo-instruções que definem símbolos ; EQU e SET e, via de regra, não são seguidos de dois pontos. Nas demais pseudo-instruções podemos ter endereços (opcionais) seguidos de dois pontos, como nas instruções de máquina do Z-80.

No campo do código de operação é colocado o mnemônico. O operando depende da natureza da pseudo-instrução.

5.7.1 – Área de montagem do programa

A pseudo-instrução *ORG* estabelece a área de montagem do programa através do seu operando, que define o endereço da primeira instrução do programa (Origem).

No programa da figura 5.2 a primeira pseudo-instrução ORG informa ao montador que o programa objeto inicia-se no endereço 1000H. A segunda ORG estabelece que a partir da instrução XOR A o programa tem que ser montado em 1050 H. Observe que entre as posições de memória 1006H e 104FH o montador não garante a colocação de nenhum dado específico.

Endereços	Dados		
1000	79	LD	A, C
1001	C6 02	ADD	A, 2
1003	C3 50 10	JP	CONTI
		ORG	1050H
1050	CONTI	XOR	A

Fig. 5.2 – Exemplo da pseudo-instrução ORG

5.7.2 – Definição de símbolos

As pseudo-instruções que definem símbolos em um programa são *EQU* e *SET*.

A pseudo-instrução EQU é usada para atribuir símbolos a valores numéricos. Após a definição de um símbolo pela EQU, o seu valor numérico é usado no programa objeto sempre que o símbolo for encontrado.

No programa a figura 5.3 a instrução LD A, VALOR é equivalente a instrução LD A, 12H.

```
VALOR    EQU     12H
         .
         .
         .
         LD      A, VALOR
```

Fig. 5.3 – Exemplo da pseudo-instrução EQU

Observe que se, futuramente, o programador desejar que o nome VALOR assuma outro valor numérico, basta apenas mudar o operando da pseudo-instrução EQU; sendo desnecessário alterar cada instrução separadamente.

Da mesma forma que na pseudo-instrução EQU, a SET é usada para atribuir símbolos a valores numéricos. Após a definição de um símbolo pela SET, o seu valor numérico é usado no programa objeto sempre que o símbolo for encontrado. No entanto, o valor do símbolo definido pela SET pode ser alterado a partir de uma nova definição por uma outra pseudo-instrução SET.

A pseudo-instrução SET é idêntica a EQU, exceto que os símbolos podem ser definidos mais de uma vez.

No programa da figura 5.4, na primeira instrução ADD A,IMMED o símbolo IMMED tem o valor 05H e na segunda instrução ADD A, IMMED este mesmo símbolo tem o valor 0AH.

```
IMMED     SET     5
          ADD     A, IMMED
IMMED     SET     10H-6
          ADD     A,IMMED
```

Fig. 5.4 – Exemplo da pseudo-instrução SET

5.7.3 – Reserva de memória

As pseudo-instruções *DB* ("define byte") e *DW* ("define word") definem valores de 8 a 16 bits respectivamente.

O operando de DB é um valor ou uma expressão simbólica que é representada por 8 bits. Por outro lado, o operando de DW é representado por 16 bits.

Estas duas pseudo-instruções são muito úteis para gerar tabelas, constantes e variáveis na memória.

O programa fonte da figura 5.5 gera uma tabela de 5 bytes, e cada byte representa um número de 1 a 5 em ASCII.

```
TABELA:   DB      31H
          DB      32H
          DB      33H
          DW      3435H
```

Fig. 5.5 – Exemplo das pseudo-instrução DB e DW

A pseudo-instrução DS especifica a reserva de um número de bytes na memória para armazenamento de dados. A montagem do programa fonte não coloca nenhum dado em especial nestas posições. O programador não deve assumir que exista valor inicial nestas posições.

O programa da figura 5.6 reserva 10 bytes consecutivos a partir do endereço PROX e, em seguida, reserva mais 16 bytes.

```
PROX:       DS 1
```

```
PROX:   DS 10D      ; RESERVA 10 BYTES
        DS 10H      ; RESERVA 16 BYTES
```

Fig. 5.6 – Exemplo da pseudo-instrução DS

5.7.4 – Final de programa

A pseudo-instrução *END* indica ao montador o final físico do programa fonte e, também, que a geração do programa objeto deve ser iniciada.

Um programa fonte só pode ter uma pseudo-instrução END que, por sua vez, tem que ser a última instrução física do programa.

Alguns montadores permitem a colocação de uma expressão no lugar do operando que indica ao programa carregador ("loader") o endereço inicial de carregamento do programa. Isto permite que se inicie a execução do programa imediatamente após o seu carregamento ("load and go").

5.7.5 – Montagem Condicional

O recurso de montagem condicional faz com que trechos de programa somente sejam montados se uma determinada condição for atendida. Isto permite que certos programas sejam instalados em sistemas somente quando houver necessidade, desta forma economiza-se memória.

A figura 5.7 mostra o formato geral desta pseudo-instrução.

```
(opcional):     IF          expressão
                .
                .
                .
                instruções
                .
                .
                .
                .
                .
(opcional):     ENDIF
```

Fig. 5.7 – Formato da pseudo-instrução IF

A expressão colocada no operando da pseudo-instrução IF é desenvolvida pelo montador. Se o seu resultado for igual a zero as instruções colocadas entre as pseudo-instruções IF e ENDIF *não são montadas.* Se o resultado da expressão for *diferente de zero* as referidas instruções *são montadas* como se as pseudo-instruções IF e ENDIF não estivessem presentes no programa. Alguns montadores permitem o *aninhamento* de IF e ENDIF, em que montagens condicionais são colocadas dentro do escopo de outras montagens condicionais. Em geral, isto é permitido até oito níveis.

No programa da figura 5.8, na primeira vez que surge a instrução LD A,C esta é montada, porque o valor de COND é 0FFH. Na segunda vez, a instrução LD A,C não é montada porque o valor de COND foi redefinido para zero pela pseudo-instrução SET.

```
COND    SET     0FFH
        IF      COND
        LD      A,C
        ENDIF
COND    SET     0
        IF      COND
        LD      A,C
        ENDIF
```

Fig. 5.8 – Exemplo de montagem condicional

5.8 – EXERCÍCIOS DE FIXAÇÃO

5.8.1 – Quais os campos do formato da linguagem assembly?

5.8.2 – Como é realizada a separação entre os campos?

5.8.3 – Quais as informações que podem ser armazenadas no campo de código de operação?

5.8.4 – O que são pseudo-instruções?

5.8.5 – Quais as pseudo-instruções que são usadas em qualquer programa?

5.8.6 – Qual a diferença entre a instrução HALT e a pseudo-instrução END?

5.8.7 – Qual a diferença entre as pseudo-instruções EQU e SET?

5.8.8 – Explique as pseudo-instruções DB, DW e DS.

5.8.9 – O que é montagem condicional?

5.8.10 – Explique o funcionamento da pseudo-instrução IF.

6 INSTRUÇÕES DE TRANSFERÊNCIAS DE 8 BITS

6.1 – APRESENTAÇÃO

Neste capítulo iniciaremos o estudo do conjunto de instruções do Z-80, através do grupo de transferências de 8 bits.

Recomendamos ao leitor que antes de iniciar a leitura deste capítulo faça uma revisão nos capítulos anteriores, principalmente no capítulo 2, pois este apresenta os registradores do Z-80.

6.2 – INTRODUÇÃO

O grupo de instruções de transferências de 8 bits será dividido nos seguintes subgrupos de operações, visando facilitar o seu estudo:

SUBGRUPO 1 – Registrador ← Registrador
SUBGRUPO 2 – Registrador ← Memória
SUBGRUPO 3 – Memória ← Registrador
SUBGRUPO 4 – Registrador, Memória ← Constante

As funções de cada subgrupo estão esquematizadas na figura 6.1. Os números no interior dos pequenos círculos correspondem aos subgrupos.

O subgrupo 1 engloba todas as instruções de transferência entre registradores de 8 bits internos ao Z-80.

O subgrupo 2 engloba todas as instruções que transferem dados da memória para registradores internos do Z-80, exceto operandos com endereçamento imediato (constantes).

O subgrupo 3 engloba todas as instruções que transferem dados de registradores internos do Z-80 para a memória.

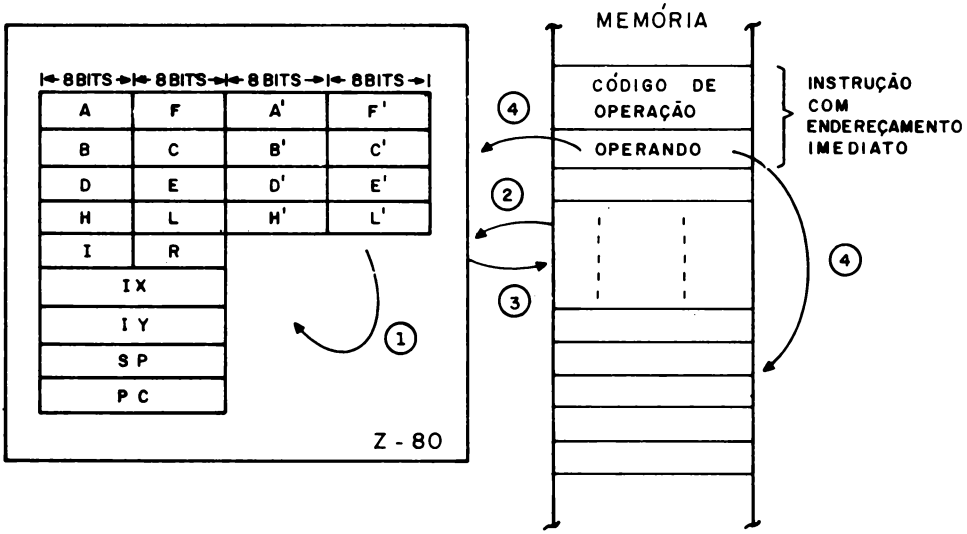


Fig. 6.1 – Funções dos Subgrupos de Transferência de 8 bits

O subgrupo 4 engloba todas as instruções que transferem constantes, isto é, operandos imediatos, para registradores internos do Z-80 e posições de memória.

A tabela 6.1 apresenta as instruções de transferências de 8 bits, com um resumo sucinto das suas principais características. Esta tabela está dividida nos mesmos subgrupos apresentados anteriormente.

A primeira coluna da esquerda está subdividida em duas e mostra os mnemônicos de cada instrução na linguagem Assembly do 8080 e Z-80. O travessão "—" simboliza que não existe instrução compatível no 8080.

O mnemônico que identifica as instruções de transferências de 8 bits no Z-80 é "LD", decorrente de "LOAD". Os operandos são separados por uma vírgula. O operando da esquerda representa sempre o registrador para onde é destinada (destino) a informação e o da direita representa o registrador de onde se origina (origem) a informação. Por exemplo, a instrução LD B,E transfere o conteúdo do registrador E para o registrador B.

Em qualquer caso, o registrador de origem, isto é, aquele do qual a informação é retirada, permanece inalterado após a transferência.

A segunda coluna da tabela 6.1 mostra uma representação simbólica da operação realizada por cada instrução. Os operandos entre parênteses representam o conteúdo de uma posição de memória.

Os operandos com endereçamento indireto por par de registradores estão simbolizados por (BC), (DE) e (HL), que representam posições de memória endereçadas pelos pares BC, DE e HL, respectivamente.

Os operandos com endereçamento indexado estão simbolizados por (IX + d) e (IY + d), que representam posições de memória endereçadas pelos registradores de índice IX e IY, respectivamente.

Os operandos com endereçamento estendido estão simbolizados por (nn) e com endereçamento imediato por n.

A seta (←) apontando para a esquerda mostra o sentido da transferência. Por exemplo, a instrução LD B,E, do exemplo anterior, realiza a seguinte operação simbólica: B ← E.

A terceira coluna da tabela 6.1 mostra como são afetados os bits de condição após a execução de cada instrução.

A quarta coluna da tabela 6.1 mostra o código de máquina de cada instrução em binário.

Cada linha desta coluna corresponde a um byte do código de instrução. Estes códigos de máquina, em hexadecimal, estão relacionados no apêndice B.

As quatro últimas colunas mostram respectivamente o número de bytes, o número de ciclos de máquina — M ciclos —, o número de estados — T ciclos — e o número do subgrupo.

TABELA 6.1 – INSTRUÇÕES DE TRANSFERÊNCIAS DE 8 BITS

Mnemônico 8080/85	Mnemônico Z-80	Operação Simbólica	Bits de Condição C	Z	P/V	S	N	H	Código de Máquina Binário 76	543	210	Nº de Bytes	Nº de M ciclos	Nº de T ciclos	SUB GRUPO
		OPERAÇÕES REGISTRADOR ← REGISTRADOR													
MOV r,r'	LD r,r'	r ← r'	●	●	●	●	●	●	01	r	r'	1	1	4	
–	LD A,I	A ← I	●	↕	IFF	↕	0	0	11	101	101	2	2	9	
									01	010	111				
–	LD A,R	A ← R	●	↕	IFF	↕	0	0	11	101	101	2	2	9	
									01	011	111				1
–	LD I,A	I ← A	●	●	●	●	●	●	11	101	101	2	2	9	
									01	000	111				
–	LD R,A	R ← A	●	●	●	●	●	●	11	101	101	2	2	9	
									01	001	111				
		OPERAÇÕES REGISTRADOR ← MEMÓRIA													
MOV r,M	LD r,(HL)	r ← (HL)	●	●	●	●	●	●	01	r	110	1	2	7	
–	LD r, (IX + d)	r ← (IX + d)	●	●	●	●	●	●	11	011	101	3	5	19	
									01 ←	r	101 →				
										d					
–	LD r, (IY + d)	r ← (IY + d)	●	●	●	●	●	●	11	111	101	3	5	19	
									01 ←	r	110 →				2
										d					
LDAX B	LD A,(BC)	A ← (BC)	●	●	●	●	●	●	00	001	010	1	2	7	
LDAX D	LD A,(DE)	A ← (DE)	●	●	●	●	●	●	00	011	010	1	2	7	
LDA end	LD A,(nn)	A ← (nn)	●	●	●	●	●	●	00	111	010	3	4	13	
									←	n	→				
									←	n	→				

OPERAÇÕES MEMÓRIA ← REGISTRADOR

MOV M,r	LD (HL),r	(HL) ← r	●	●	●	●	●	●	01	110	r	1	2	7	
—	LD (IX + d),r	(IX + d) ← r	●	●	●	●	●	●	11	011	101	3	5	19	
									01	110	r				
									←	d	→				
—	LD (IY + d),r	(IY + d) ← r	●	●	●	●	●	●	11	111	101	3	5	19	
									01	110	r				
									←	d	→				3
STAX B	LD (BC),A	(BC) ← A	●	●	●	●	●	●	00	000	010	1	2	7	
STAX D	LD (DE),A	(DE) ← A	●	●	●	●	●	●	00	010	010	1	2	7	
STA end	LD (nn),A	(nn) ← A	●	●	●	●	●	●	00	110	010	3	4	13	
									←	n	→				
									←	n	→				

OPERAÇÕES REGISTRADOR, MEMÓRIA ← CONSTANTE

MVI r, Const.	LD r,n	r ← n	●	●	●	●	●	●	00	r	110	2	2	7	
									←	n	→				
MVI M, Const.	LD (HL),n	(HL) ← n	●	●	●	●	●	●	00	110	110	2	3	10	
									←	n	→				
—	LD (IX + d), n	(IX + d) ← n	●	●	●	●	●	●	11	011	101	4	5	19	
									00	110	110				
									←	d	→				4
									←	n	→				
—	LD (IY + d),n	(IY + d) ← n	●	●	●	●	●	●	11	111	101	4	5	19	
									00	110	110				
									←	d	→				
									←	n	→				

NOTAS:

1 — "r,r' " simbolizam qualquer um dos registradores A, B, C, D, E, H, L e na representação do código de máquina binário respeitam a seguinte convenção: B = 000, C = 001, D = 010, E = 011, H = 100, L = 101, A = 111

2 — "IFF" indica que o conteúdo do FLIP-FLOP IFF1 é transferido para o flag P/V.

3 — "d" simboliza o deslocamento no endereçamento indexado.

4 — "n" simboliza o operando imediato.

5 — "nn" simboliza o endereço no modo de endereçamento estendido.

6 — Notação dos bits de condição: ● = não afetado, 0 = resetado, 1 = setado, X = indefinido.
↕ = o bit é afetado em função do resultado da operação.

6.3 – SUBGRUPO 1 – REGISTRADOR ← REGISTRADOR

As instruções do tipo LD r,r' transferem o conteúdo do registrador r' para o registrador r. Os operandos r e r' podem ser qualquer um dos registradores de uso geral: A, B, C, D, E, H e L. Como estas instruções não envolvem a ULA, os bits de condição não são afetados.

A figura 6.2 mostra como transcorre a instrução LD C,E. A palavra "Antes" significa a situação dos registradores antes da execução da instrução e, "Depois" a situação após a execução.

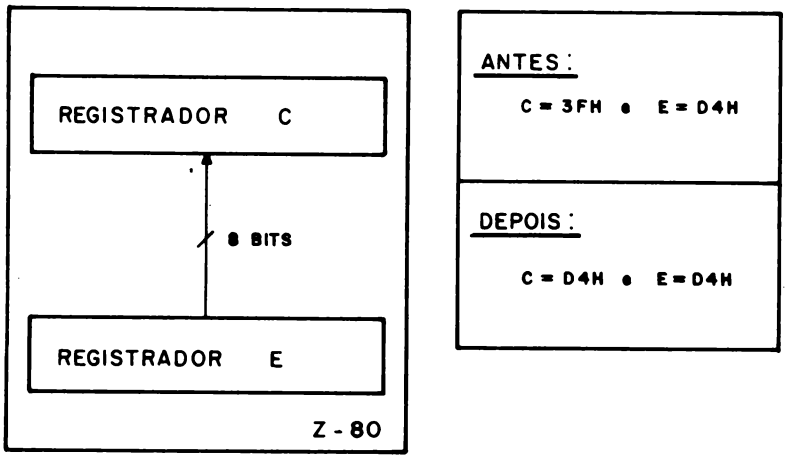


Fig. 6.2 – A Instrução LD C,E

As demais quatro instruções deste grupo transferem os conteúdos dos registradores I (vetor de interrupção) e R (refresh da memória) para o acumulador e vice-versa.

As instruções LD A,I e LD A,R transferem os conteúdos dos registradores I e R, respectivamente, para o acumulador. Apesar destas instruções não envolverem a ULA, os bits de condição Z e S são afetados como se o dado transferido fosse o resultado de uma operação aritmética e, o conteúdo do flip-flop de interrupção IFF1 é transferido para o bit de condição P/V.

A figura 6.3 mostra como transcorre a instrução LD A,I.

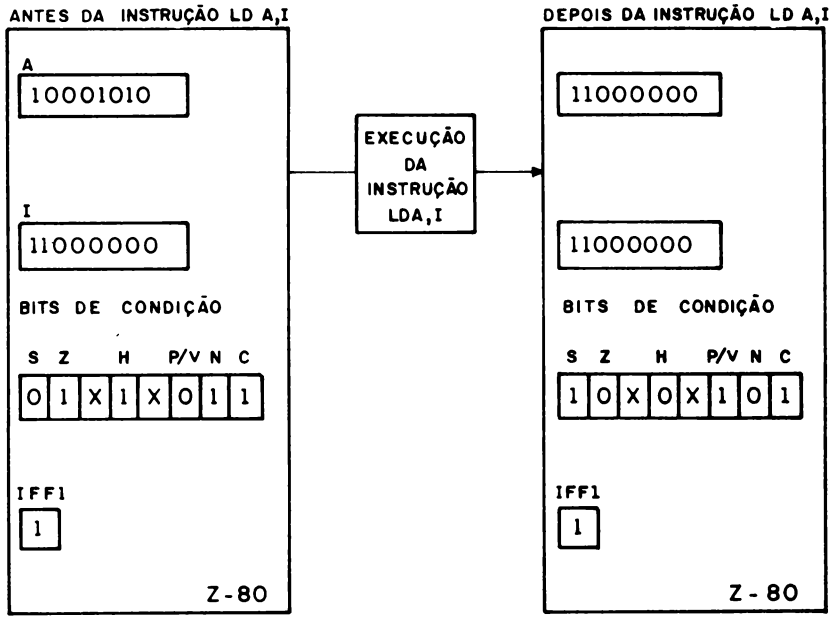


Fig. 6.3 – A Instrução LD A,I

As instruções LD I,A e LD R,A transferem o conteúdo do acumulador para os registradores I e R, respectivamente, e não afetam os bits de condição.

6.4 – SUBGRUPO 2 – REGISTRADOR ← MEMÓRIA

As instruções do subgrupo 2 transferem o conteúdo de uma posição de memória para um registrador do Z-80 e não afetam os bits de condição.

As instruções com endereçamento por par de registradores são: LD r, (HL), LD A, (BC) e LD A, (DE). O par HL é privilegiado em relação aos pares BC e DE, pois quando usado como ponteiro permite a transferência para qualquer registrador de uso geral. A figura 6.4 mostra como transcorre a instrução LD B, (HL).

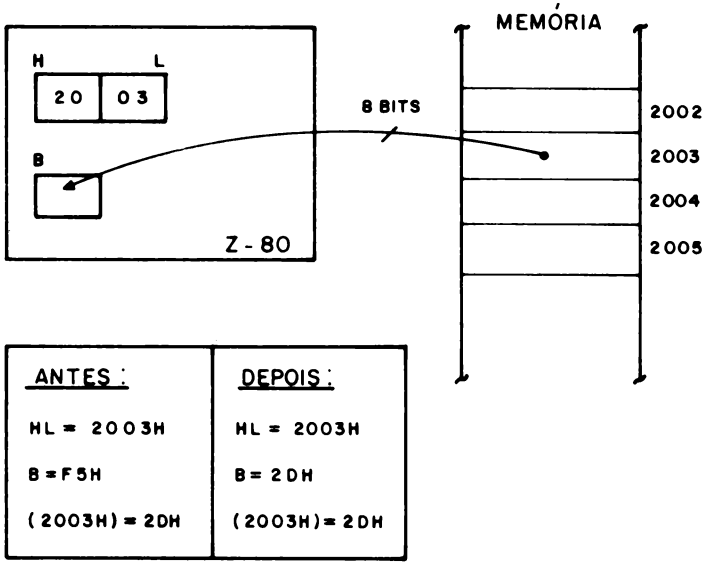
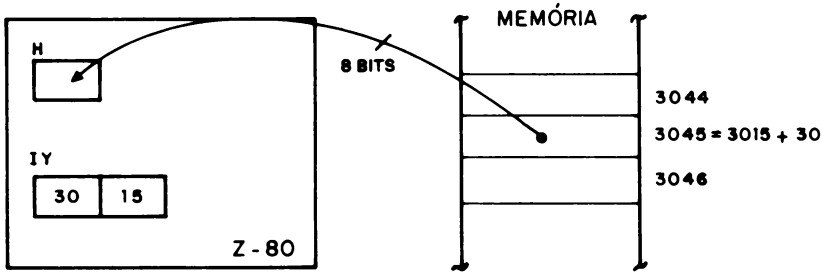


Fig. 6.4 – A Instrução LD B,(HL)

As instruções com endereçamento indexado são LD r, (IX + d) e LD r,(IY + d). A figura 6.5 mostra como transcorre a instrução LD H,(IY + 30H).



ANTES :	DEPOIS :
H = 23H	H = 4BH
IY = 3015H	IY = 3015H
(3045H) = 4BH	(3045H) = 4BH

Fig. 6.5 – A Instrução LD H, (IY + 30H)

Com endereçamento estendido temos a instrução LD A, (nn). A figura 6.6 mostra um exemplo desta instrução, que transfere o conteúdo da posição de memória 7F3CH para o acumulador. Observe como o endereço é montado na memória; no segundo byte fica a parte menos significativa e no terceiro byte e parte mais significativa. Alertamos ao leitor que esta convenção prevalece para as demais instruções.

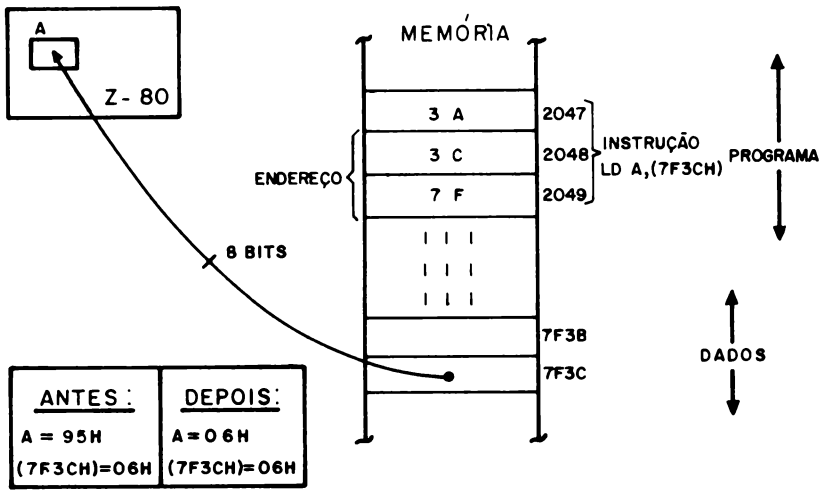


Fig. 6.6 – A Instrução LD A,(7F3CH)

6.5 – SUBGRUPO 3 – MEMÓRIA ← REGISTRADOR

As instruções do subgrupo 3 transferem o conteúdo de um registrador interno do Z-80 para uma posição de memória e não afetam os bits de condição. Na realidade, efetuam as operações inversas do subgrupo 2.

As instruções com endereçamento por par de registradores são: LD (HL),r, LD (BC), A e LD (DE),A. Neste caso, também, o par HL é privilegiado em relação aos pares BC e DE, pois quando usado como ponteiro permite a transferência de qualquer registrador de uso geral.

As instruções com endereçamento indexado são:

LD (IX + d), r e LD (IY + d),r. E, com endereçamento estendido temos a instrução LD (nn),A.

6.6 – SUBGRUPO 4 – REGISTRADOR, MEMÓRIA ← CONSTANTE

As instruções do subgrupo 4 transferem um operando imediato (constante) para um registrador interno do Z-80 ou para uma posição de memória e não afetam os bits de condição.

As instruções do tipo LD r,n transferem o segundo byte do código de instrução para o registrador de uso geral r. A título de exemplo, a figura 6.7 mostra como transcorre a instrução LD C, 9AH.

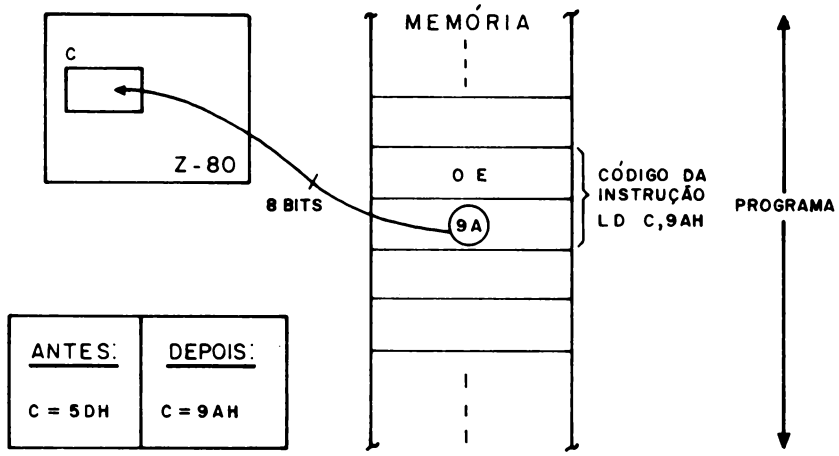


Fig. 6.7 – A Instrução LD C,9AH

A instrução LD (HL),n transfere o segundo byte do código de instrução para a posição de memória endereçada pelo par HL. A figura 6.8 mostra como transcorre a instrução LD (HL), 7FH.

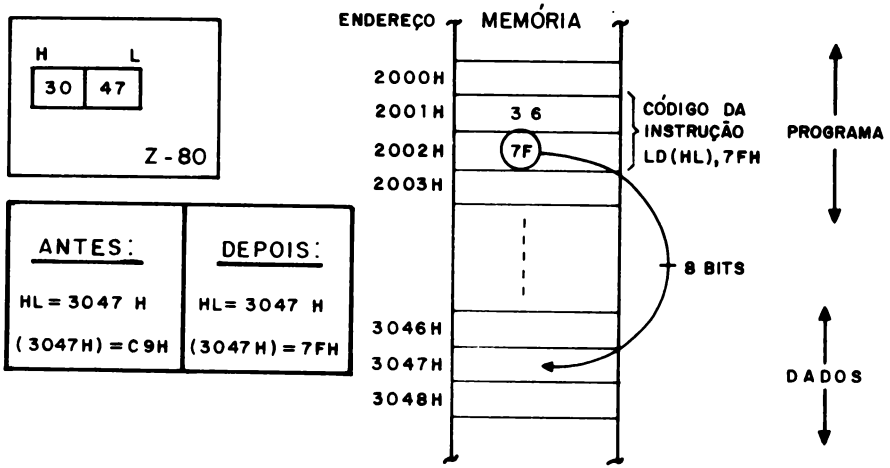


Fig. 6.8 – A Instrução LD (HL), 7FH

As instruções LD (IX + d),n e LD (IY + d),n transferem o quarto byte do código de suas instruções para as posições de memória endereçadas pela soma do deslocamento d com o conteúdo dos registradores IX e IY, respectivamente.

A figura 6.9 mostra como transcorre a instrução LD (IX + 20H), 3DH.

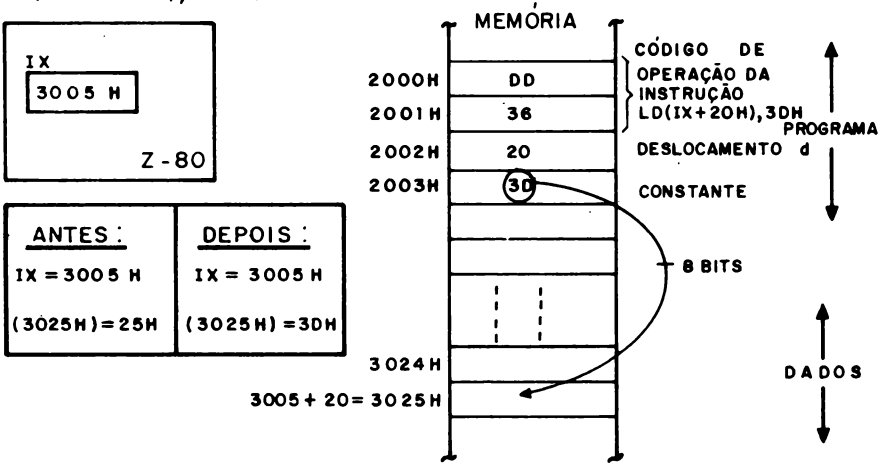


Fig. 6.9 – A Instrução LD (IX + 20H), 3DH

6.7 – EXEMPLO Nº 1

O exemplo nº 1, ilustrado na figura 6.10, apresenta um programa que transfere o conteúdo do registrador L para a posição de memória 010AH e o conteúdo do registrador H para a posição de memória 010BH. Neste programa as transferências para a memória se fazem através do acumulador com a instrução LD (nn),A. Primeiro é transferido o conteúdo do L e depois o de H.

A instrução "HALT", a última deste programa, faz o Z-80 paralisar a busca de instruções da memória até que ocorra uma interrupção externa ou um reset.

```
                    ORG     0000H
0000    7D          LD      A, L          ; A ← L
0001    32 OA 01    LD      (010AH), A    ; (010AH) ← A
00o4    7C          LD      A, H          ; A ← H
0005    32 0B 01    LD      (010BH), A    ; (010BH) ← A
0008    76          HALT                  ; PARADA
                    END
```

Fig. 6.10 – Exemplo nº 1

Observe que o programa foi montado a partir do endereço 0000H. A título de exercício confirme os códigos das instruções utilizando a tabela 6.1.

6.8 – EXEMPLO Nº 2

Na figura 6.11 temos um programa que troca o conteúdo dos registradores B e C.

```
                    ORG     0000H
0000    78          LD      A, B          ; SALVA B EM A
0001    41          LD      B, C          ; TRANSFERE C PARA B
0002    4F          LD      C, A          ; COMPLETA A TROCA
0003    76          HALT                  ; PARADA
                    END
```

Fig. 6.11 – Exemplo nº 2

Observe que foi usado o acumulador para salvar o conteúdo de B. Isto porque,quando é realizada a transferência de C para B, o conteúdo de B é perdido.

Novamente, o programa foi montado com endereço inicial 0000H.

6.9 – EXERCICIOS DE FIXAÇÃO

Todos os programas desta lista devem ser realizados na linguagem Assembly do Z-80 e montados a partir do endereço 0000H.

6.9.1 – Escreva um programa que armazene 00H nos registradores B, C, D, E, H e L.

6.9.2 – Escreva um programa que troque o conteúdo dos pares BC e DE.

6.9.3 – Escreva um programa que troque o conteúdo das posições 1000H e 1001H de memória.

6.9.4 – Escreva um programa que armazene no registrador H o conteúdo da posição 2000H e no registrador L o conteúdo da posição 2001H.

6.9.5 – Faça um programa que escreva 00H na posição 10ADH de memória.

6.9.6 – Faça um programa que transfere o conteúdo da posição de memória endereçada pelo registrador de índice IX para a posição de memória endereçada pelo par BC.

6.9.7 – Faça um programa que transfere quatro bytes armazenados em posições consecutivas na memória, a partir do endereço contido no registrador de índice IX, para outras quatro posições consecutivas a partir do endereço contido no registrador de índice IY.

6.9.8 – Explique como funcionam os bits Z, P/V e S nas instruções LD A, R e LD A, I.

6.9.9 -- Cite uma aplicação para a instrução LD A,R. E para a instrução LD R,A?

6.9.10 – Escreva um programa que carregue 20H em B e 0AH em C.

7 INSTRUÇÕES DE TRANSFERÊNCIAS DE 16 BITS

7.1 – APRESENTAÇÃO

No capítulo anterior iniciamos o estudo do conjunto de instruções apresentando as instruções de transferências de 8 bits.

Neste capítulo estudaremos as instruções de transferências de 16 bits, que formam juntamente com as de transferências de 8 bits, o grupo de instruções mais usado pelos programadores.

7.2 – INTRODUÇÃO

O grupo de instruções de transferências de 16 bits será dividido nos seguintes subgrupos de operações, visando facilitar o seu estudo:

SUBGRUPO 1 – Registrador ← Registrador
SUBGRUPO 2 – Registrador ← Memória
SUBGRUPO 3 – Memória ← Registrador
SUBGRUPO 4 – Registrador ← Constante
SUBGRUPO 5 – Pilha ← Registrador
SUBGRUPO 6 – Registrador ← Pilha

As funções de cada subgrupo estão esquematizadas na figura 7.1. Os números no interior dos pequenos círculos correspondem aos subgrupos.

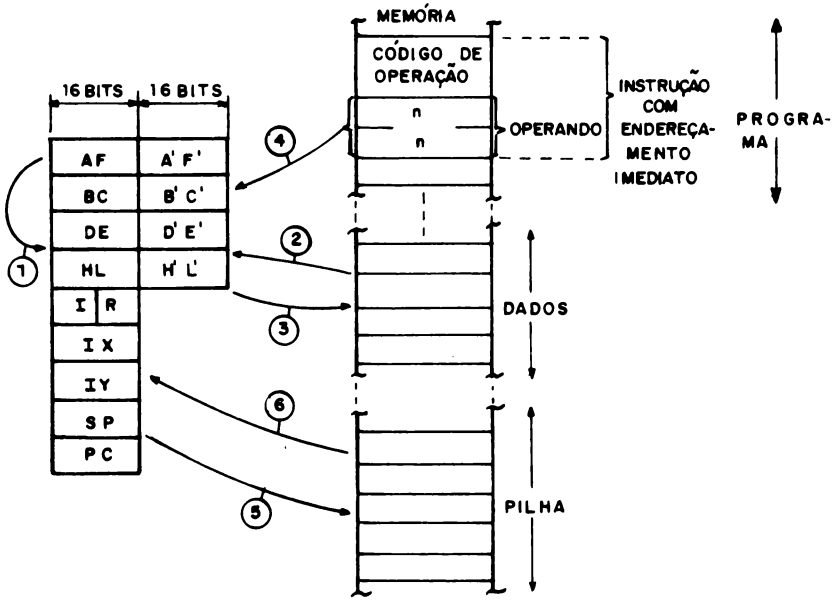


Fig. 7.1 – Funções dos Subgrupos

O subgrupo 1 engloba todas as instruções de transferência entre registradores internos do Z-80.

O subgrupo 2 engloba todas as instruções que transferem dados da memória para registradores internos do Z-80, exceto dados da pilha e constantes decorrentes de endereçamento imediato.

O subgrupo 3 engloba todas as instruções que transferem dados dos registradores internos do Z-80 para a memória, exceto à pilha.

O subgrupo 4 engloba todas as instruções que transferem constantes para registradores internos do Z-80.

O subgrupo 5 engloba as instruções que transferem pares de registradores (AF, BC, DE ou HL) e os registradores IX ou IY para o topo da pilha.

O subgrupo 6 engloba as instruções que transferem o topo da

pilha para um par de registradores (AF, BC, DE ou HL) ou para os registradores IX ou IY.

As instruções de transferência de 16 bits não afetam os bits de condição.

A tabela 7.1 apresenta as instruções de transferências de 16 bits, com um resumo sucinto das suas principais características. Esta tabela está dividida nos mesmos subgrupos apresentados anteriormente.

TABELA 7.1 – INSTRUÇÕES DE TRANSFERÊNCIAS DE 16 BITS

Mnemônico 8080/85	Mnemônico Z-80	Operação Simbólica	Bits de Condição C	Z	P/V	S	N	H	Código de Máquina Binário 76	543	210	Nº de Bytes	Nº de M ciclos	Nº de T ciclos	SUB-GRUPO
		OPERAÇÕES REGISTRADOR ← REGISTRADOR													
SPHL	LD SP,HL	SP ← HL	●	●	●	●	●	●	11	111	001	1	1	6	
–	LD SP,IX	SP ← IX	●	●	●	●	●	●	11 11	011 111	101 001	2	2	10	1
–	LD SP,IY	SP ← IY	●	●	●	●	●	●	11 11	111 111	101 001	2	2	10	
		OPERAÇÕES REGISTRADOR ← MEMÓRIA													
LHLD end	LD HL,(nn)	H ← (nn + 1) L ← (nn)	●	●	●	●	●	●	00 ←	101 n	010 →	3	5	16	
									←	n	→				
–	LD dd,(nn)	dd_H ← (nn + 1) dd_L ← (nn)	●	●	●	●	●	●	11 01 ←	101 dd1 n	101 011 →	4	6	20	
									←	n	→				
–	LD IX,(nn)	IX_H ← (nn + 1) IX_L ← (nn)	●	●	●	●	●	●	11 00 ←	011 101 n	101 010 →	4	6	20	2
									←	n	→				
–	LD IY,(nn)	IY_H ← (nn + 1) IY_L ← (nn)	●	●	●	●	●	●	11 00 ←	111 101 n	101 010 →	4	6	20	
									←	n	→				

OPERAÇÕES MEMÓRIA ← REGISTRADOR

SHLD end	LD (nn),HL	(nn + 1) ← H (nn) ← L	●	●	●	●	●	●	00 ← ← n	100 n n	010 → →	3	5	16
–	LD (nn),dd	(nn + 1) ← dd_H (nn) ← dd_L	●	●	●	●	●	●	11 01 ← ← n	101 dd0 n n	101 011 → →	4	6	20
–	LD (nn),IX	(nn + 1) ← IX_H (nn) ← IX_L	●	●	●	●	●	●	11 00 ← ← n	011 100 n n	101 010 → →	4	6	20
–	LD (nn),IY	(nn + 1) ← IY_H (nn) ← IY_L	●	●	●	●	●	●	11 00 ← ← n	111 100 n n	101 010 → →	4	6	20

3

OPERAÇÕES REGISTRADOR ← CONSTANTE

LXI dd, const.	LD dd,nn	dd ← nn	●	●	●	●	●	●	00 ← ← n	dd0 n n	001 → →	3	3	10
–	LD IX,nn	IX ← nn	●	●	●	●	●	●	11 00 ← ← n	011 100 n n	101 001 → →	4	4	14
–	LD IY,nn	IY ← nn	●	●	●	●	●	●	11 00 ← ← n	111 100 n n	101 001 → →	4	4	14

4

continua na próxima página

OPERAÇÕES PILHA ← REGISTRADOR

PUSH p	PUSH qq	$(SP-2) \leftarrow qq_L$ $(SP-1) \leftarrow qq_H$ $SP \leftarrow SP-2$	●	●	●	●	●	●	11	qq0	101	1	3	11	
–	PUSH IX	$(SP-2) \leftarrow IX_L$ $(SP-1) \leftarrow IX_H$ $SP \leftarrow SP-2$	●	●	●	●	●	●	11 11	011 100	101 101	2	4	15	5
–	PUSH IY	$(SP-2) \leftarrow IY_L$ $(SP-1) \leftarrow IY_H$ $SP \leftarrow SP-2$	●	●	●	●	●	●	11 11	111 100	101 101	2	4	15	

OPERAÇÕES REGISTRADOR ← PILHA

POP p	POP qq	$qq_H \leftarrow (SP+1)$ $qq_L \leftarrow (SP)$ $SP \leftarrow SP+2$	●	●	●	●	●	●	11	qq0	001	1	3	10	
–	POP IX	$IX_H \leftarrow (SP+1)$ $IX_L \leftarrow (SP)$ $SP \leftarrow SP+2$	●	●	●	●	●	●	11 11	011 100	101 001	2	4	14	6
–	POP IY	$IY_H \leftarrow (SP+1)$ $IY_L \leftarrow (SP)$ $SP \leftarrow SP+2$	●	●	●	●	●	●	11 11	111 100	101 001	2	4	14	

NOTAS:

1 – "dd" simboliza qualquer um dos registradores de 16 bits: BC(00), DE(01), HL(10) ou SP(11).
2 – "qq" simboliza qualquer um dos registradores de 16 bits: BC(00), DE(01), HL(10) ou AF(11).
3 – Os 8 bits menos significativos de um registrador de 16 bits são representados por $(PAR)_L$ e os 8 bits mais significativos por $(PAR)_H$. Por exemplo, BC_L = C e AF_H = A.
4 – "p" na linguagem Assembly do 8080 é representado por B, D, H ou PSW.
5 – Notação dos bits de condição: ● = não afetado.

Tabela 7.1 – Instruções de Transferências de 16 Bits

7.3 – SUBGRUPO 1 – REGISTRADOR ← REGISTRADOR

As instruções do subgrupo 1 são: LD SP, HL, LD SP, IX ou LD SP, IY. Estas instruções transferem para o SP, respectivamente, os conteúdos dos registradores HL, IX e IY.

A título de exemplo, a figura 7.2 mostra como transcorre a instrução LD SP, IX.

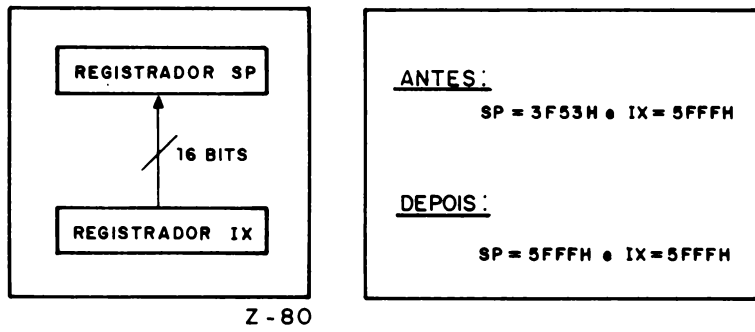


Fig. 7.2 – A Instrução LD SP, IX

7.4 – SUBGRUPO 2 – REGISTRADOR ← MEMÓRIA

As instruções do subgrupo 2 transferem os conteúdos de dois bytes em endereços consecutivos na memória para os pares de registradores BC, DE, HL e para os registradores de 16 bits, SP, IX ou IY. A figura 7.3 mostra o procedimento de transferência para cada registrador, com suas respectivas instruções. Observe que o conteúdo da posição de memória de menor endereço é transferido para a parte menos significativa do registrador de 16 bits. Este endereço é formado concatenando-se o penúltimo e último bytes do código de instrução, que formam, respectivamente, as partes menos e mais significativas do referido endereço.

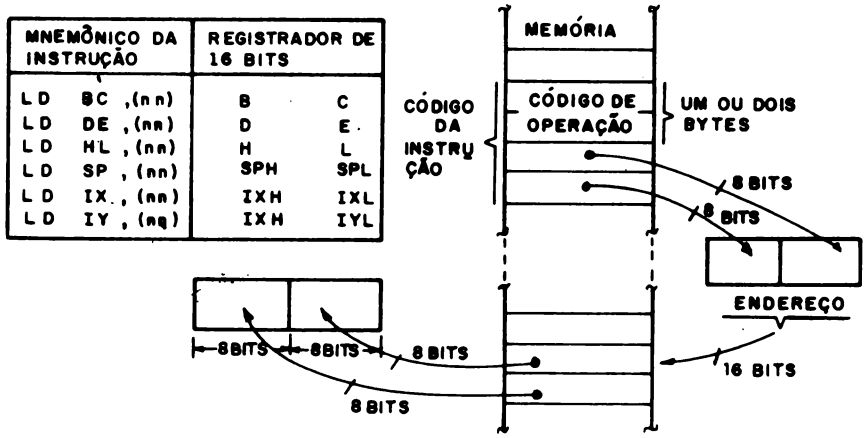


Fig. 7.3 – Operações Registrador ← Memória

Como exemplo, a figura 7.4 mostra como transcorre a instrução LD DE, (127FH).

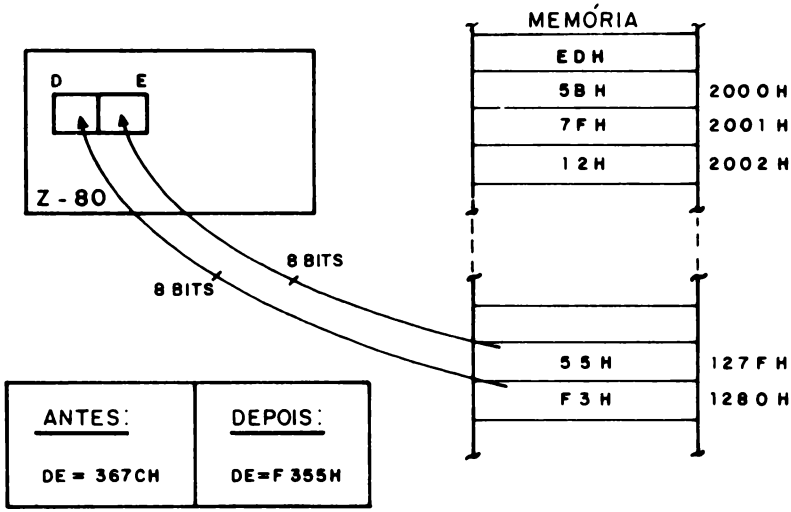


Fig. 7.4 – A Instrução LD DE, (127FH)

7.5 – SUBGRUPO 3 – MEMÓRIA ← REGISTRADOR

As instruções do subgrupo 3 transferem os conteúdos dos pares de registradores BC, DE ou HL e dos registradores de 16 bits, SP, IX ou IY, para dois bytes em endereços consecutivos na memória. Na realidade, estas instruções realizam as operações inversas do subgrupo anterior. A figura 7.5 mostra o procedimento de transferência de cada registrador com suas respectivas instruções. Observe que a parte menos significativa do registrador de 16 bits é transferido para a posição de memória de menor endereço. E, a formação do endereço obedece a mesma convenção usada no subgrupo anterior.

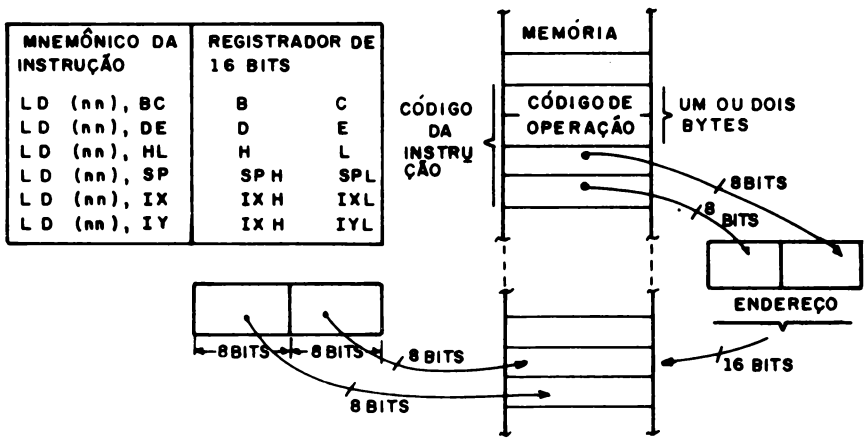


Fig. 7.5 – Operações Memória ← Registrador

A figura 7.6 mostra como transcorre a instrução LD (010AH), HL.

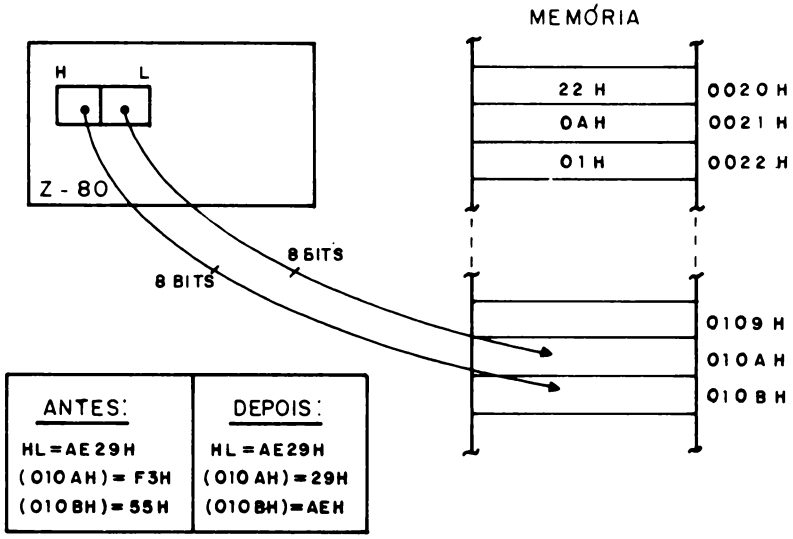


Fig. 7.6 – A Instrução LD (010AH), HL

7.6 – SUBGRUPO 4 – REGISTRADOR ← CONSTANTE

As instruções do subgrupo 4 transferem uma constante de 16 bits para os pares de registradores BC, DE ou HL e para os registradores de 16 bits, SP, IX ou IY. A figura 7.7 mostra o procedimento de transferência para cada registrador, com suas respectivas instruções. Observe que o penúltimo byte do código de instrução é transferido para a parte menos significativa do registrador de 16 bits, e o último byte para a parte mais significativa.

MNEMÔNICO DA INSTRUÇÃO	REGISTRADOR DE 16 BITS	
LD BC, nn	B	C
LD DE, nn	D	E
LD HL, nn	H	L
LD SP, nn	SP H	SP L
LD IX, nn	IX H	IX L
LD IY, nn	IY H	IY L

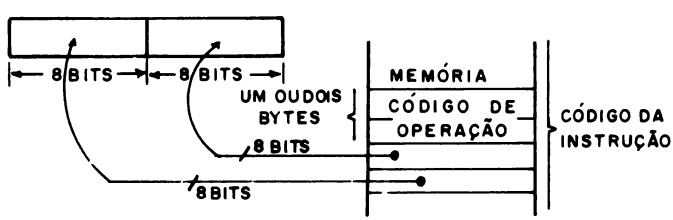
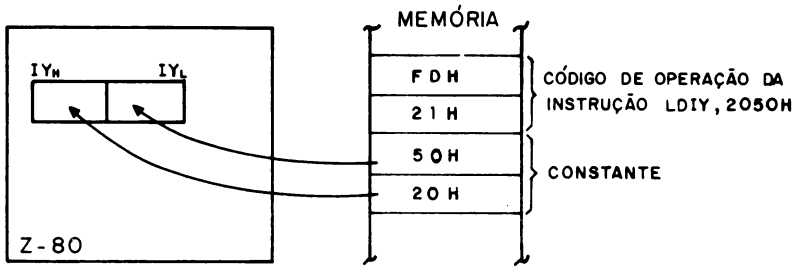


Fig. 7.7 – Operações Registrador ← Constante

A figura 7.8 mostra como transcorre a instrução LD IY, 2050H.



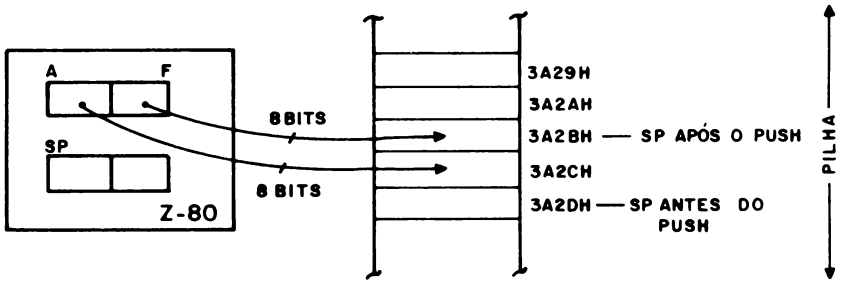
ANTES:	DEPOIS:
IY = FFD5H	IY = 2050H

Fig. 7.8 – A Instrução LD IY, 2050H

7.7 – SUBGRUPO 5 – PILHA ← REGISTRADOR

As instruções do subgrupo 5 transferem o conteúdo dos registradores de 16 bits para o topo da pilha. Este tipo de operação já foi apresentado no capítulo 4, quando estudamos o endereçamento por ponteiro de pilha.

A figura 7.9 mostra como transcorre a instrução PUSH AF.



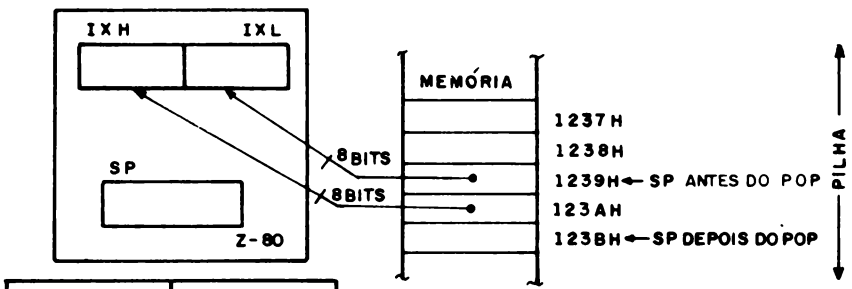
ANTES:	DEPOIS:
A = 1FH	A = 1FH
F = 2AH	F = 2AH
SP = 3A2DH	SP = 3A2BH
(3A2CH) = FFH	(3A2CH) = 1FH
(3A2BH) = FFH	(3A2BH) = 2AH

Fig. 7.9 – A Instrução PUSH AF

7.8 – SUBGRUPO 6 – REGISTRADOR ← PILHA

As instruções do subgrupo 6 transferem o topo da pilha para um registrador de 16 bits do Z-80. Este tipo de operação, também, já foi visto no endereçamento por ponteiro de pilha.

A figura 7.10 mostra como transcorre a instrução POP IX.



ANTES	DEPOIS
IX = 0FF0H	IX = 933DH
SP = 1239H	SP = 123BH
(1239H) = 3DH	(1239H) = 3DH
(123AH) = 93H	(123AH) = 93H

Fig. 7.10 – A Instrução POP IX

7.9 – EXEMPLO Nº 1

No primeiro exemplo do capítulo 6 apresentamos um programa implementado somente com instruções de transferências de 8 bits. Agora, mostraremos como este mesmo programa fica mais simples se utilizarmos instruções de transferências de 16 bits. A figura 7.11 apresenta esta nova solução.

```
                        ORG     0000H
0000    22 0A 01        LD      (010AH), HL
0003    76              HALT
                        END
```

Fig. 7.11 – Exemplo nº 1

Observe que o programa foi reduzido para duas instruções, sendo que o código objeto ocupa quatro bytes na memória e a origem é 0000H.

7.10 – EXEMPLO Nº 2

O exemplo nº 2, mostrado na figura 7.12, apresenta um programa que transfere o conteúdo das posições de memória 12F7H para o registrador C e 12F8H para o registrador H.

```
                VAR     EQU     12F7H
                        ORG     100H
0100  ED 4B F7 12 TRANS: LD      BC, (VAR)   ; B ← (12F8H) e
0104  60                LD      H, B        ; C ← (12F7H)
0105  76                HALT
                        END
```

Fig. 7.12 – Exemplo nº 2

A instrução LD BC. (VAR), transfere o conteúdo da posição de memória 12F8H para B e de 12F7H para C. Em seguida, a instrução LD H,B transfere o conteúdo de B para H.

7.11 – EXEMPLO Nº 3

O exemplo nº 3, mostrado na figura 7.13, apresenta um programa que inicia a pilha em 1000H e, transfere para a mesma os registradores de 16 bits AF, BC, DE, HL, IX e IY, nesta ordem. Observe que a medida que os dados são inseridos na pilha, a mesma vai ocupando posições de memória com endereços decrescentes. Por esta razão é que na maioria dos microcomputadores a pilha é alocada no final da memória RAM.

```
TOPI    EQU     1000H
        ORG     100H
        LD      SP, TOPI    ; INICIA PILHA EM 1000H
        PUSH    AF          ; (0FFFH) ← A, (0FFEH) ← F
        PUSH    BC          ; (0FFD) ← B, (0FFCH) ← C
        PUSH    DE          ; (0FFBH) ← D, (0FFAH) ← E
        PUSH    HL          ; (0FF9H) ← H, (0FF8H) ← L
        PUSH    IX          ; (0FF7H) ← IXH, (0FF6H) ← IXL
        PUSH    IY          ; (0FF5H) ← IYH, (0FF4H) ← IYL
        HALT
        END
```

Fig. 7.13 – Exemplo nº 3

A título de exercício monte o código de máquina do exemplo nº 3, consultando as tabelas do Apêndice B.

7.12 – EXEMPLO Nº 4

O exemplo nº 4, mostrado na figura 7.14, apresenta um programa que troca o conteúdo dos registradores IX e IY, através da pilha. Note que os dados são recuperados da pilha na ordem inversa daquela que foram guardados.

```
                ORG     0000H
0000 31 00 40   LD      SP, 4000H   ; INICIA PILHA
0003 DD E5      PUSH    IX
0005 FD E5      PUSH    IY
0007 DD E1      POP     IX
0009 FD E1      POP     IY
0008 76         HALT
                END
```

Fig. 7.14 – Exemplo nº 4

7.13 – EXERCÍCIOS DE FIXAÇÃO

Todos os programas desta lista devem ser escritos em Assembly e montados a partir do endereço 0100H.

7.13.1 – Faça um programa que inicie a pilha no endereço 3FFFH.

7.13.2 – Faça um programa que troque os conteúdos das posições de memória 1000H e 2000H e, 1001H e 2001H, entre si.

7.13.3 – Faça um programa que troque os conteúdos dos pares BC e DE através da pilha.

7.13.4 – Faça um programa que troque os conteúdos das posições de memória endereçadas pelo par DE e pelo registrador de índice IY.

7.13.5 – Faça um programa que troque o conteúdo do topo da pilha com o conteúdo do par BC.

7.13.6 – Determine o código objeto hexadecimal dos programas dos Exemplos nºs 1, 2, 3 e 4.

7.13.7 – Com relação aos programas do exercício anterior, calcule o número de bytes e o tempo de execução de cada programa, assumindo que o Z-80 opera com um relógio a uma freqüência de 2,5MHz.

7.13.8 – Faça um programa que carregue A0CFH no registrador IX, F5C3H em IY e A000H em DE.

7.13.9 – Faça um programa que transfere para a posição de memória 8000H o conteúdo da posição de memória cujo endereço está armazenado nas posições 5000H (menos significativo) e 5001H (mais significativo).

7.13.10 – Com uma instrução, transfira o conteúdo da posição de memória 5CF4H para o registrador D.

8 INSTRUÇÕES DE PERMUTA

8.1 – APRESENTAÇÃO

Continuando com o estudo das instruções do Z-80, apresentaremos neste capítulo as instruções de permuta.

8.2 – INTRODUÇÃO

O grupo de *instruções de permuta* será dividido nos seguintes subgrupos de operações visando facilitar o seu estudo:

SUBGRUPO 1 – Permuta entre pares de registradores ativos.
SUBGRUPO 2 – Permuta entre registradores ativos e passivos.
SUBGRUPO 3 – Permuta entre o topo da pilha e registrador.

As funções de cada subgrupo estão esquematizadas na figura 8.1.

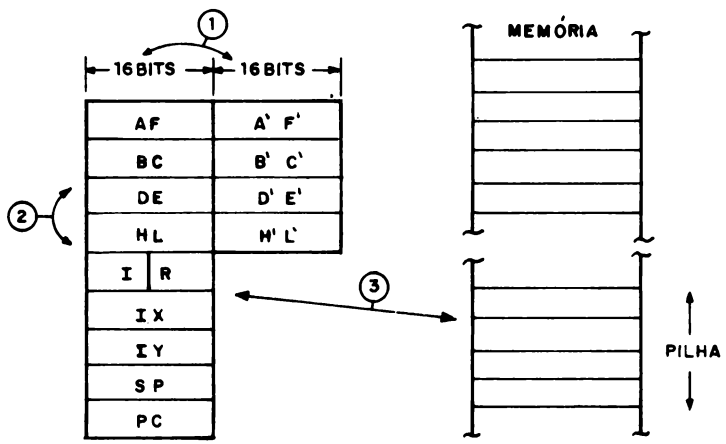


Fig. 8.1 – Funções dos Subgrupos

A tabela 8.1 apresenta as *instruções de permuta,* com um resumo sucinto das suas principais características. Esta tabela está dividida nos mesmos subgrupos apresentados anteriormente.

TABELA 8.1 – GRUPO DE INSTRUÇÕES DE PERMUTA

Mnemônico		Operação Simbólica	Bits de Condição						Código de Operação Binário			Nº de Bytes	Nº de M ciclos	Nº de T ciclos	SUB-GRUPO
8080/85	Z-80		C	Z	P/V	S	N	H	76	543	210				
		PERMUTA ENTRE REGISTRADORES ATIVOS													
XCHG	EX DE,HL	DE ⇔ HL	●	●	●	●	●	●	11	101	011	1	1	4	1
		PERMUTA ENTRE REGISTRADORES ATIVOS E PASSIVOS													
–	EX AF,AF'	AF ⇔ AF'	●	●	●	●	●	●	00	001	000	1	1	4	
–	EXX	BC ⇔ BC' DE ⇔ DE' HL ⇔ HL'	●	●	●	●	●	●	11	011	001	1	1	4	2
		PERMUTA ENTRE O TOPO DA PILHA E REGISTRADOR													
XTHL	EX (SP),HL	H ⇔ (SP + 1) L ⇔ (SP)	●	●	●	●	●	●	11	100	011	1	5	19	
–	EX (SP),IX	IX_H ⇔ (SP + 1) IX_L ⇔ (SP)	●	●	●	●	●	●	11 11	011 100	101 011	2	6	23	3
–	EX (SP),IY	IY_H ⇔ (SP + 1) IY_L ⇔ (SP)	●	●	●	●	●	●	11 11	111 100	101 011	2	6	23	

Notação dos bits de condição: ● = não afetado.

Tabela 8.1 – Grupo de Instruções de Permuta

8.3 – SUBGRUPO 1

No *subgrupo 1* a permuta se faz entre pares de registradores ativos e, na verdade, é composto de apenas uma instrução: EX DE, HL que troca o conteúdo dos pares DE e HL entre si. A figura 8.2 mostra como transcorre esta instrução.

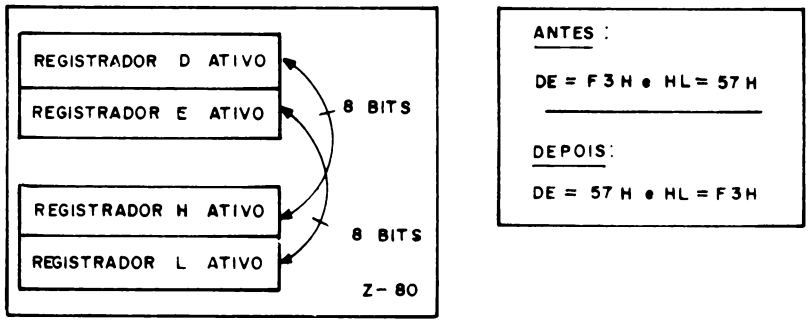


Fig. 8.2 – A Instrução EX DE, HL

8.4 – SUBGRUPO 2

O *subgrupo 2* engloba as operações que permutam registradores ativos e passivos, sendo composto das instruções EX AF, AF' e EXX.

A instrução EX AF, AF' transforma o acumulador e flags ativos em passivos e vice-versa. A figura 8.3 mostra a ação desta instrução.

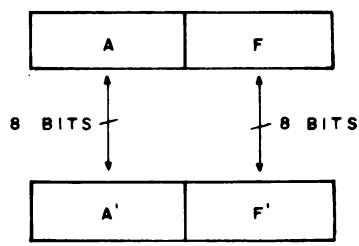


Fig. 8.3 – A Instrução EX AF, AF'

A instrução EXX transforma os registradores BC (B'C') DE (D'E') e HL (H'L') ativos em passivos e vice-versa. A figura 8.4 mostra a ação desta instrução.

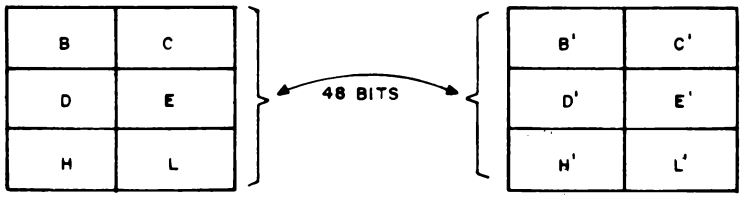


Fig. 8.4 – A Instrução EXX

8.5 – SUBGRUPO 3

O *subgrupo 3* engloba as instruções que permutam o topo da pilha com um registrador ativo de 16 bits. As instruções EX (SP), HL, EX (SP), IX e EX (SP), IY permutam o topo da pilha com o conteúdo dos registradores HL, IX e IY, respectivamente. A figura 8.5 mostra o procedimento geral destas instruções:

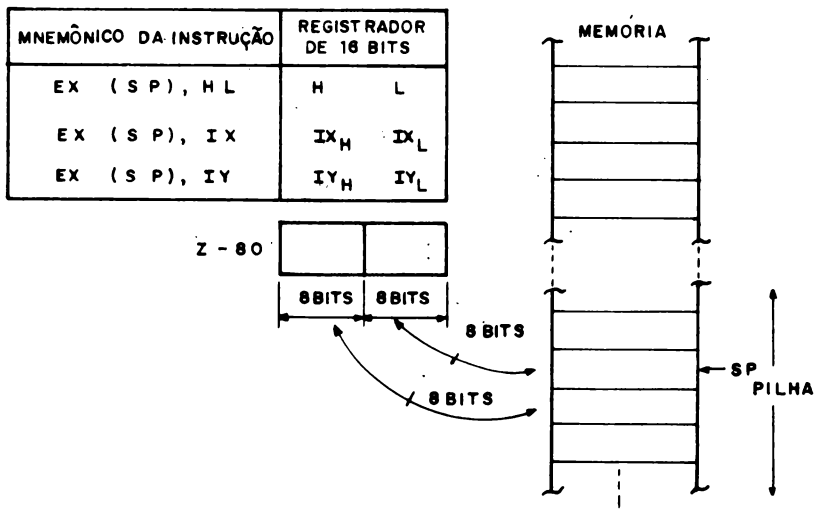


Fig. 8.5 – Permuta entre o topo da pilha e registrador

Como exemplo, a figura 8.6 mostra como transcorre a instrução EX (SP), HL.

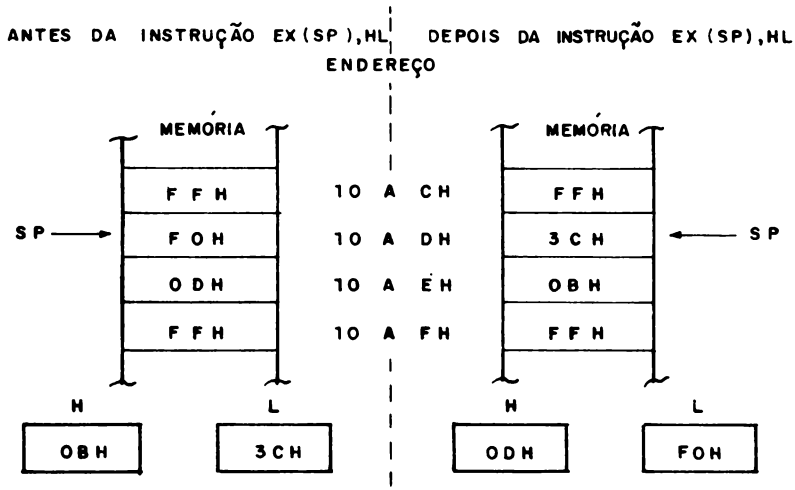


Fig. 8.6 – A Instrução EX (SP), HL

8.6 – EXEMPLO Nº 1

O exemplo nº 1, mostrado na figura 8.7, apresenta um programa que troca o conteúdo do par DE com o topo da pilha.

```
                ORG     200H
0200 EB         EX      DE, HL
0201 E3         EX      (SP), HL
0202 EB         EX      DE, HL
0203 76         HALT
                END
```

Fig. 8.7 – Exemplo nº 1

A primeira instrução EX DE, HL troca DE com HL. A instrução EX (SP), HL troca o conteúdo original do par de registrador DE com o topo da pilha, ficando o último no par HL. E, a segunda instrução EX DE, HL restaura o conteúdo inicial de HL e coloca o topo da pilha em DE.

8.7 – EXEMPLO Nº 2

O exemplo nº 2, mostrado na figura 8.8, apresenta uma forma de preservar temporariamente os registradores A e F no decorrer de um processamento.

```
                    .
                    .
                    .
EX          AF, AF'          ; SALVA A, F


PROCESSAMENTO


EX          AF, AF'          ;RESTAURA A, F
```

Fig. 8.8 – Exemplo nº 2

8.8 – EXEMPLO Nº 3

Da mesma forma que no exemplo anterior, agora preservaremos os pares BC, DE e HL, conforme ilustrado na figura 8.9.

```
EXX                                  ; SALVA BC, DE e HL
LD          BC, DADO1                ; NOVOS DADOS
LD          DE, DADO2
LD          HL, DADO3

PROCESSAMENTO

EXX                                  ; RESTAURA BC, DE e HL
```

Fig. 8.9 – Exemplo nº 3

8.9 – EXERCÍCIOS DE FIXAÇÃO

Todos os programas desta lista devem ser escritos em Assembly e montados a partir do endereço 0100H.

8.9.1 – Faça um programa que troque os conteúdos dos registradores IX e IY através do topo da pilha.

8.9.2 – Faça um programa que troque o topo da pilha com o conteúdo do par BC.

8.9.3 – Faça um programa que troque o topo da pilha com o conteúdo das posições de memória 902AH e 902BH.

8.9.4 – Faça um programa que transfere o topo da pilha para o par HL e coloca no topo da pilha o valor 1000H.

9 INSTRUÇÕES LÓGICAS E ARITMÉTICAS DE 8 BITS

9.1 – APRESENTAÇÃO

Após termos estudado nos capítulos anteriores as instruções de transferência de 8 e 16 bits e as de permuta, apresentaremos agora as instruções lógicas e aritméticas de 8 bits.

9.2 – INTRODUÇÃO

O grupo de *instruções lógicas e aritméticas de 8 bits* realiza operações de adição, subtração, "e" lógico, "ou" lógico, "ou exclusivo", comparação e incrementação e decrementação de uma unidade.

As instruções deste grupo, por sua vez, classificam-se em dois subgrupos. O *primeiro subgrupo* envolve operações com dois operandos. Neste caso, um operando é sempre o acumulador e o outro pode ser especificado por endereçamento imediato, registrador indireto através do par HL, indexado ou registrador.

O *segundo subgrupo* envolve operações com um operando, que pode ser especificado pelas mesmas técnicas de endereçamento do subgrupo 1, exceto o imediato.

A tabela 9.1 apresenta as *instruções lógicas e aritméticas de 8 bits,* com um resumo sucinto das suas principais características.

TABELA 9.1 – INSTRUÇÕES LÓGICAS E ARITMÉTICAS DE 8 BITS

Mnemônico		Operação Simbólica	Bits de Condição						Código de Máquina binário			Nº de Bytes	Nº de M ciclos	Nº de T ciclos	SUB-GRUPO
8080/85	Z-80		C	Z	P/V	S	N	H	76	543	210				
OPERAÇÕES ARITMÉTICAS DE 8 BITS COM DOIS OPERANDOS															
ADD r	ADD A,r	A ← A + r	↕	↕	V	↕	0	↕	10	000	r	1	1	4	
ADI n	ADD A,n	A ← A + n	↕	↕	V	↕	0	↕	11	000	110	2	2	7	
									←	n	→				
ADD M	ADD A,(HL)	A ← A + (HL)	↕	↕	V	↕	0	↕	10	000	110	1	2	7	
–	ADD A,(IX + d)	A ← A + (IX + d)	↕	↕	V	↕	0	↕	11	011	101	3	5	19	
									10	000	110				
									←	d	→				
–	ADD A,(IY + d)	A ← A + (IY + d)	↕	↕	V	↕	0	↕	11	111	101	3	5	19	
									10	000	110				
									←	d	→				1
ADC,ADI	ADC A,s	A ← A + s + CY	↕	↕	V	↕	0	↕		001					
SUB,SUI	SUB s	A ← A – s	↕	↕	V	↕	1	↕		010					
SBB,SBI	SBC A,s	A ← A – s – CY	↕	↕	V	↕	1	↕		011					
OPERAÇÕES LÓGICAS DE 8 BITS COM DOIS OPERANDOS															
ANA,ANI	AND s	A ← A ∧ s	0	↕	P	↕	0	1		100					
ORA,ORI	OR s	A ← A ∨ s	0	↕	P	↕	0	0		110					
XRA,XRI	XOR s	A ← A ⊕ s	0	↕	P	↕	0	0		101					
CMP,CPI	CP s	A – s	↕	↕	V	↕	1	↕		111					

OPERAÇÕES ARITMÉTICAS COM UM OPERANDO

INR r	INC r	r ← r + 1	●	↕	V	↕	0	↕	00	r	100	1	1	4
INR M	INC (HL)	(HL) ← (HL) + 1	●	↕	V	↕	0	↕	00	110	100	1	3	11
—	INC (IX + d)	(IX + d) ← (IX + d) + 1	●	↕	V	↕	0	↕	11 00 ←	011 110 d	101 100 →	3	6	23
—	INC (IY + d)	(IY + d) ← (IY + d) + 1	●	↕	V	↕	0	↕	11 00 ←	111 110 d	101 100 →	3	6	23
DCR	DEC d	d ← d − 1	●	↕	V	↕	1	↕			101			

NOTAS:

1 — "r" simboliza qualquer um dos registradores de 8 bits: B (000), C (001), D (010), E (011), H (100), L (101), A (111).
2 — "s" simboliza r, n, (HL), (IX + d) e (IY + d), conforme mostrado nas instrucões ADD.
3 — Os bits 000 nas instruções ADD são substituídos pelos correspondentes nas instruções ADC, SUB, SBB, AND, OR, XOR e CP.
4 — O "d" na instrução DEC simboliza r, (HL), (IX + d) e (IY + d), conforme mostrado nas instruções INC.
5 — Para se obter os códigos de máquina binários das instruções DEC, basta substituir os bits 100 os bits 100 nas instruções INC por 101.

6 — A letra V indica que o bit P/V fornece o overflow do resultado, e a letra P a paridade.
7 — V = 1 indica a ocorrência de overflow.
P = 1 indica que a paridade do resultado é par
8 — Notações dos bits de condição: ● = não afetado, 0 = resetado, 1 = setado.
X = indefinido.
↕ = o bit é afetado em função do resultado da operação.

Tabela 9.1 — Instruções Lógicas e Aritméticas de 8 bits

9.3 – SUBGRUPO 1 – OPERAÇÕES COM DOIS OPERANDOS

As funções lógicas e aritméticas processadas pelo subgrupo 1 operam o acumulador e o outro operando especificado, e o resultado é sempre depositado no acumulador. Os bits de condição são atualizados em função do resultado da operação realizada.

Os mnemônicos das instruções do subgrupo 1 são compostos sempre do código da operação (ADD, ADC, SUB, SBC, AND, OR, XOR e CP), e do segundo operando, que varia em função da técnica de endereçamento usada: r, n, (HL), (IX + d) e (IY + d).

No Z-80 as operações aritméticas com operandos de 8 bits são as de adição e subtração.

9.3.1 – Operações de Adição

Nas operações de adição o registrador especificado é adicionado ao acumulador através da aritmética complemento a dois. Existem dois tipos de operações de adição: *sem o carry e com o carry.*

No primeiro caso, o conteúdo do acumulador é adicionado ao segundo operando e o resultado é depositado no acumulador.

No segundo caso, o conteúdo do acumulador é adicionado ao segundo operando e ao bit carry. E, este resultado é depositado no acumulador.

As operações de *adição com o carry* permitem a realização de operações de *adição de multiprecisão,* que estudaremos posteriormente.

9.3.1.1 – Exemplo nº 1

Assuma que o registrador D contém 2EH e o acumulador contém 6CH, então, a instrução ADD A,D realiza a operação de adição, conforme mostra a figura 9.1.

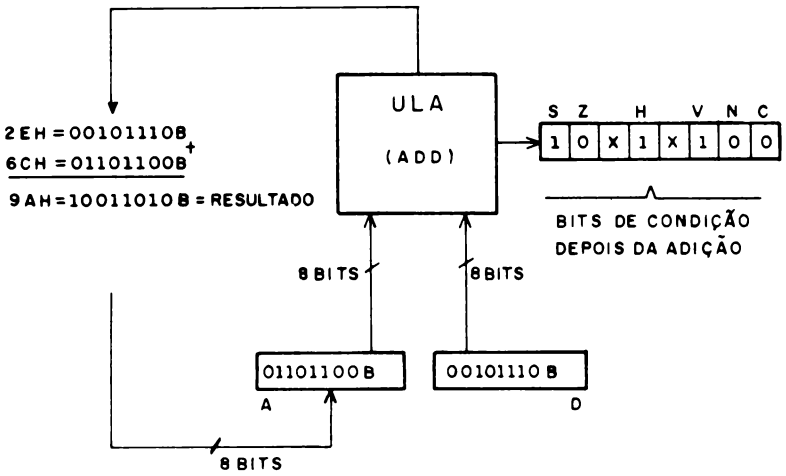


Fig. 9.1 – A Instrução ADD A, D

9.3.1.2 – Exemplo nº 2

Assuma que o registrador C contém 3DH, o acumulador 42H e o carry-bit = 1, então, a instrução ADC A, C realiza a operação de adição conforme mostra a figura 9.2.

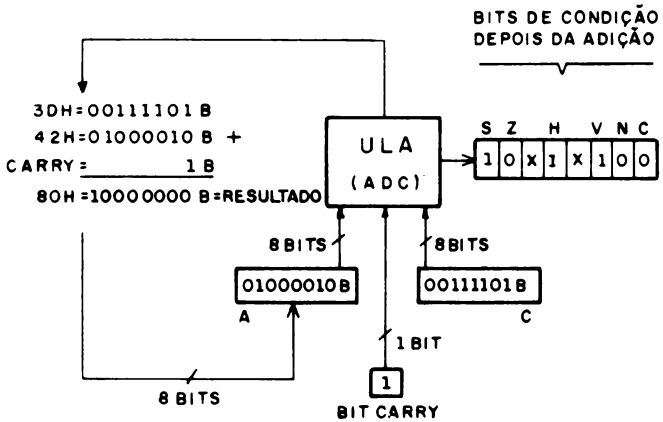


Fig. 9.2 – A Instrução ADC A, C

9.3.2 – Operações de Subtração

Nas operações de subtração o conteúdo do registrador especificado é subtraído do conteúdo do acumulador na aritmética complemento a dois, ficando o resultado no acumulador.

A ação do bit carry nas operações de subtração difere da operação de adição. Na subtração, se não for gerado "vai um" do bit mais significativo então o carry-bit é setado, indicando a ocorrência do "um emprestado" (borrow). Caso contrário, o carry-bit é resetado.

Da mesma forma que nas operações de adição, existem dois tipos de operação de subtração: *sem o borrow* e *com o borrow.*

Na subtração sem o borrow o segundo operando é subtraído do conteúdo do acumulador. Em contrapartida, na subtração com o borrow, do acumulador é subtraído o resultado da soma do segundo operando com o estado atual do bit carry.

As instruções de subtração com borrow são muito úteis na implementação de operações de *subtração de multiprecisão.*

9.3.2.1 – Exemplo nº 3

Assuma que o acumulador contém 3EH. Então, a instrução SUB A subtrai o acumulador dele mesmo, produzindo um resultado nulo. A figura 9.3 ilustra esta operação. Observe que como o "vai um" do bit de mais alta ordem é um e, tratando-se de uma operação de subtração, o bit carry fica resetado.

Note que a instrução SUB A pode ser usada para resetar o carry-bit e zerar o acumulador.

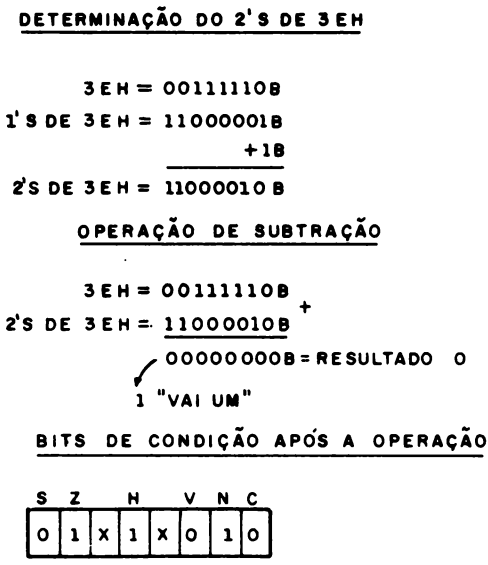


Fig. 9.3 – A Instrução SUB A

9.3.2.2 – Exemplo nº 4

Assuma que o conteúdo da posição de memória 1020H contém 02H, o acumulador contém 04H e o carry bit = 1. Então, a instrução SBC A, (IX + 20H) realiza a operação de subtração conforme mostra a figura 9.4, sabendo-se, ainda, que no momento de sua execução IX = 1000H.

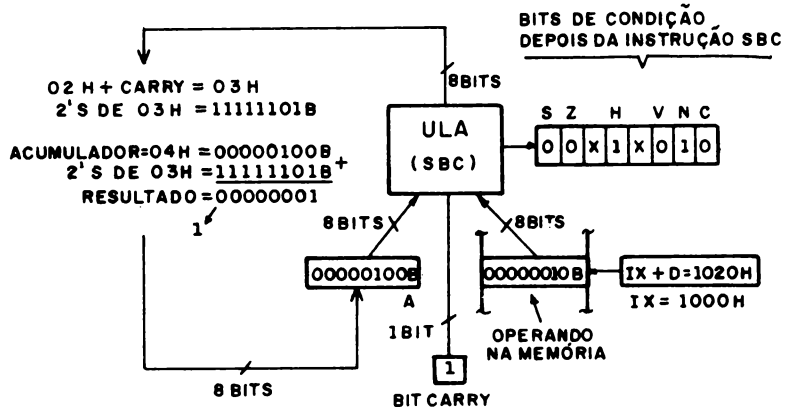


Fig. 9.4 – A Instrução SBC A, (IX + 20H)

9.3.3 – Operações Lógicas

As três operações lógicas possíveis são o "e", "ou" e "ou exclusivo" lógicos, e a comparação. A tabela 9.2 mostra a simbologia destas operações, exceto a comparação, e a tabela 9.3 mostra as suas *tabelas-verdades.* A instrução de comparação será estudada separadamente.

FUNÇÃO LÓGICA	MNEMÔNICO DA INSTRUÇÃO	SÍMBOLO DA OPERAÇÃO
"e"	AND	$\wedge$
"ou"	OR	V
"ou exclusivo"	XOR	$\oplus$

Tabela 9.2 – Simbologia das Operações Lógicas

OPERANDOS		RESULTADOS		
OPERANDO 1	OPERANDO 2	$\wedge$	V	$\oplus$
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Tabela 9.3 – Tabelas-Verdades

As funções lógicas no Z-80 operam os *bits de mesma ordem* em cada byte, realizando a operação *bit a bit.* Como um bit não afeta o seu adjacente, não há ocorrência de "vai um" (carry-bit = 0).

9.3.3.1 – Exemplo nº 5

Assuma que o acumulador contém FCH e o registrador C contém 0FH, então, a instrução AND C realiza as ações mostradas na figura 9.5. Observe que este exemplo garante que os qua-

tro bits de mais alta ordem do acumulador fiquem zerados e, os quatro bits menos significativos fiquem inalterados; na terminologia de processamento de dados dizemos que os quatro bits mais significativos foram *"mascarados"*.

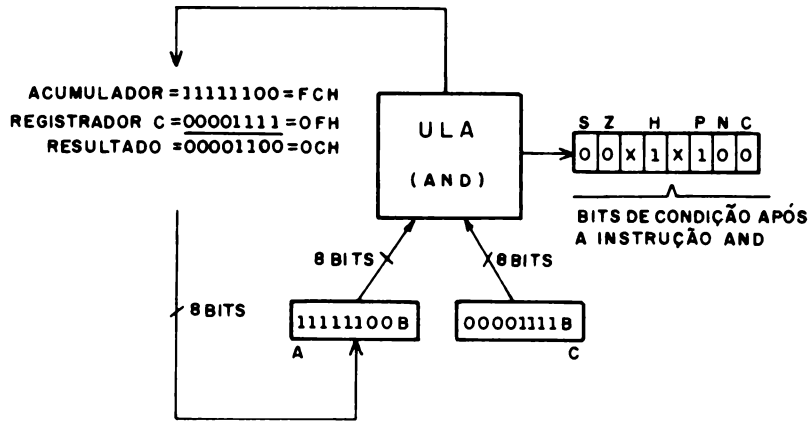


Fig. 9.5 – A Instrução AND C

De um modo geral, como o "e" lógico de um bit com "zero" dá "zero", e o "e" lógico de "um" com qualquer bit mantém este bit inalterado, esta operação lógica é usada para *zerar grupos de bits.*

A operação de resetar ou setar bits mantendo os demais inalterados é denominada de *mascaramento ("mask").*

9.3.3.2 – Exemplo nº 6

Assuma que o registrador D contém B5H, então a instrução OR 0FH procede conforme mostra a figura 9.6. Observe neste exemplo que os quatro bits de mais baixa ordem do acumulador ficam setados e os quatro bits mais significativos ficam inalterados. Podemos dizer que os quatro bits menos significativos foram *"mascarados"*.

De um modo geral, o "ou" lógico de um bit com "um" dá "um" e o "ou" lógico de "zero" com qualquer bit mantém este bit inalterado. O "ou" lógico é usado *para setar grupos de bits.*

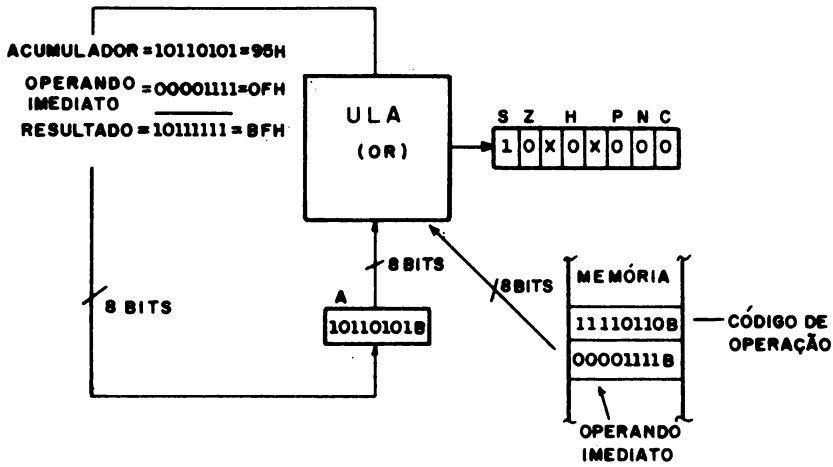


Fig. 9.6 – A Instrução OR 0FH

9.3.3.3 – Exemplo nº 7

Como o "ou exclusivo" de qualquer bit com ele mesmo produz um zero, então a instrução XOR A pode ser usada para zerar o conteúdo do acumulador.

9.3.3.4 – Exemplo nº 8

Como o "ou exclusivo" de "um" lógico com qualquer bit é o seu complemento (0 XOR 1 = 1 e 1 XOR 1 = 0), então, se o conteúdo do acumulador é FFH a instrução XOR B produz o *complemento a um* do registrador B no acumulador.

9.3.4 – Operação de Comparação

A instrução CP compara o segundo operando com o acumula-

dor. Esta comparação se faz subtraindo internamente o operando do acumulador, deixando ambos inalterados, e atualizando os bits de condição em função do resultado.

Se Z = 1 significa que as quantidades são iguais, Z = 0 significa que as quantidades são diferentes.

Como se trata de uma operação de subtração, o bit carry fica setado se não houver "vai um" do bit mais significativo do resultado. Se os operandos fôrem números positivos, este resultado (C = 1) significa que o segundo operando é maior do que o acumulador. A figura 9.7 resume a análise dos bits Z e C, assumindo a comparação de números positivos.

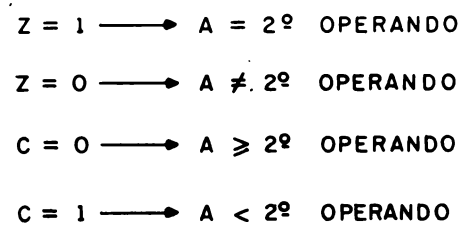


Fig. 9.7 – Análise dos Bits "Z" e "C"

9.3.4.1 – Exemplo nº 9

Assuma que o conteúdo do acumulador é 0BH e o registrador D contém 04H. Então, a instrução CP D realiza a operação de subtração interna mostrada na figura 9.8.

Após a subtração temos A = 0BH, D = 04H e o bit C = 0, indicando que o conteúdo de D é menor ou igual ao conteúdo do acumulador.

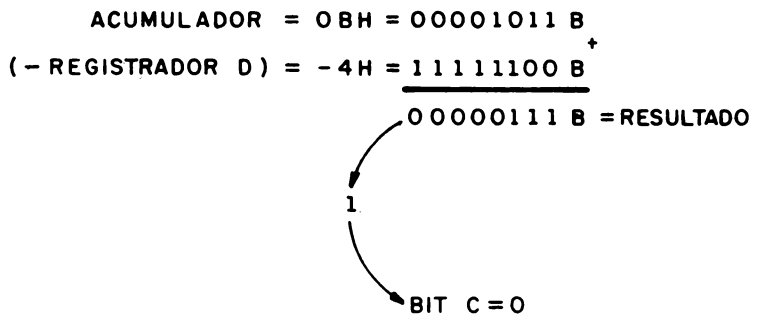


Fig. 9.8 – Subtração Interna

9.4 – SUBGRUPO 2 – OPERAÇÕES COM UM OPERANDO

O subgrupo 2 envolve as instruções que incrementam e decrementam de uma unidade o conteúdo de um registrador interno do Z-80 ou de uma posição de memória. O resultado destas operações é depositado no próprio operando. Os bits de condição S, Z, H e V são atualizados em função do resultado. No entanto, nas operações do subgrupo 2 o *carry bit fica inalterado.*

Os mnemônicos das instruções do subgrupo 2 são compostos sempre do código da operação (INC e DEC) e do operando (r,(HL), (IX + d) e (IY + d)).

9.4.1 – Exemplo nº 10

Se o registrador H contém 79H, então, após a execução da instrução INC H passa a ter 7AH. A figura 9.9 mostra o procedimento da instrução INC H para este exemplo.

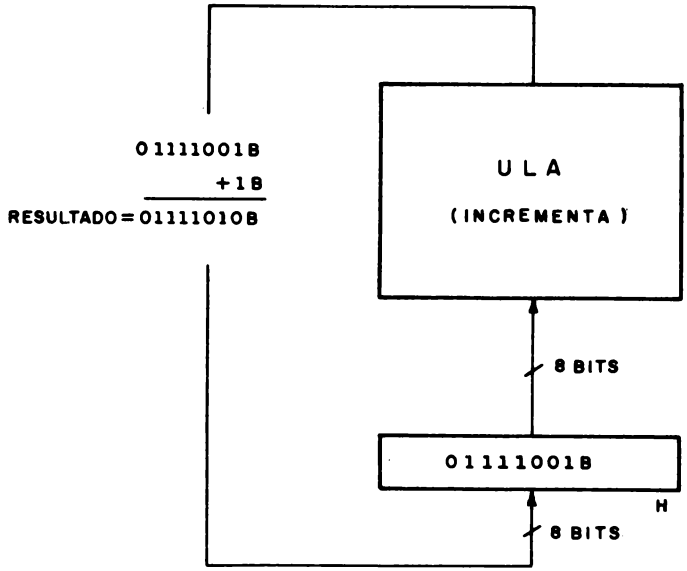


Fig. 9.9 – A Instrução INC H

Para praticar um pouco, determine os valores dos bits de condição após a execução da instrução INC H.

9.4.2 – Exemplo nº 11

Assuma que o registrador IY contém 5000H e que a posição de memória 5010H contém 04H. Então, após a execução da instrução DEC (IY + 10H) a posição de memória 5010H passa a ter 03H.

9.5 – SOMA DE MULTIPRECISÃO

A instrução ADC pode ser usada para somar números que precisam de mais de um byte para sua representação. Considere a

soma dos dois números hexadecimais sem sinal da figura 9.10. Esta soma pode ser feita no Z-80, somando-se primeiramente os dois bytes menos significativos e em seguida o "vai um" resultante à parcela seguinte da operação.

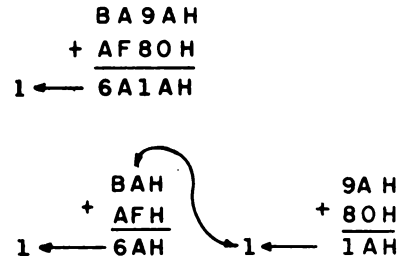


Fig. 9.10 – Soma de Multiprecisão

Observe que desta forma podemos adicionar parcelas com qualquer número de dígitos. A isto denominamos de *adição de multiprecisão.*

9.5.1 – Exemplo nº 12

O programa apresentado neste exemplo soma dois números de 16 bits armazenados em bytes consecutivos de memória. A primeira parcela tem o byte menos significativo no endereço PARC1 e a segunda parcela em PARC2. O resultado da operação fica armazenado em PARC1. A lógica do programa está mostrada na figura 9.11, e o programa na linguagem Assembly na figura 9.12.

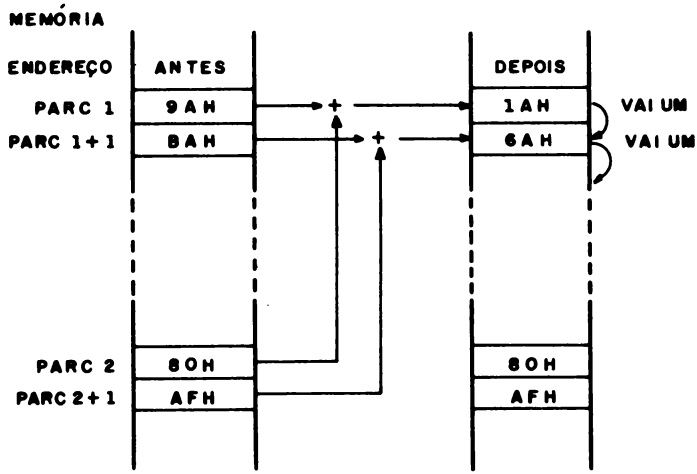


Fig. 9.11 – Soma de Multiprecisão

```
PARC1       EQU     1000H
PARC2       EQU     2000H
            ORG     100H
            LD      BC, PARC1     ; BC ← PARC1
            LD      HL, PARC2     ; HL ← PARC2
            LD      A, (BC)
            ADD     (HL)          ; SOMA PARTE MENOS SIGN.
            LD      (BC),A
            INC     L
            INC     C
            LD      A, (BC)
            ADC     (HL)
            LD      (BC),A
            HALT
            END
```

Fig. 9.12 – Programa da Soma de Multiprecisão

Observe que ao atualizarmos os ponteiros com as instruções INC L e INC C, não há passagem de "vai um" da parte menos significativa para a mais significativa de cada ponteiro.

9.6 – SUBTRAÇÃO DE MULTIPRECISÃO

A partir do exemplo nº 12, para se implementar um programa de subtração de multiprecisão basta substituir a instrução ADD por SUB e a instrução ADC por SBC.

9.7 – COMPARAÇÃO COM O 8080

O microprocessador 8080 tem as mesmas funções aritméticas e lógicas de 8 bits que o Z-80.

No entanto, o Z-80 tem as vantagens de poder designar operações pelo endereçamento indexado e de gerar os bits de condição V e N.

9.8 – EXERCÍCIOS DE FIXAÇÃO

9.8.1 – Escreva as operações apresentadas a seguir em Assembly e monte o código de máquina com endereço inicial 0100H.

a) Some o conteúdo do registrador H com o conteúdo do registrador L e armazene o resultado na posição 1000H de memória.

b) Subtraia do conteúdo da posição 2000H de memória o conteúdo do registrador C e armazene o resultado na mesma posição (2000H).

c) Realize a operação lógica "ou" entre os conteúdos dos registradores H e E e armazene o resultado no registrador L.

d) Some o conteúdo do par BC com o conteúdo do par DE e armazene o resultado no par HL.

e) Incremente de duas unidades o conteúdo da posição 13FFH de memória.

9.8.2 – Como se comporta o bit P/V nas operações lógicas?

9.8.3 – Determine o valor do acumulador para cada uma das instruções apresentadas abaixo:

a) SUB A
b) AND A
c) ADD A,A
d) XOR A
e) OR A
f) XOR FFH

9.8.4 – Informe os valores finais dos bits de condição e do acumulador para cada trecho de programa apresentado a seguir:

a)
```
LD   HL, 10FFH
LD   A,H
ADD  A,L
```

d)
```
LD   BC, 23C5H
LD   A,B
AND  C
```

b) LD A,32H
SUB 1FH

e) SUB A
ADD A,30H

c) LD A,1CH
ADD A,A
ADD A,A

f) LD A,55H
AND C3H
OR F0H

9.8.5 – Cite uma razão para o fato das instruções INC e DEC não modificarem o bit carry.

9.8.6 – Faça um programa que resete os bits 2 e 3 do registrador D.

9.8.7 – Faça um programa que sete os quatro bits menos significativos da posição 80D3H.

9.8.8 – Faça um programa que resete os bits 0 e 1 do registrador D e sete os bits 6 e 7 deste mesmo registrador.

9.8.9 – Faça um programa que divide o conteúdo do registrador C em duas partes. Os quatro bits menos significativos de C devem ser transferidos para os quatro bits menos significativos do registrador B e os demais bits deste registrador devem ser zerados. Os quatro bits mais significativos de C tem que ser transferidos para os quatro bits menos significativos do registrador H e os demais bits deste registrador devem ser zerados.

9.8.10 – Faça um programa que multiplique o conteúdo do par HL por dez.

9.8.11 – Antes da execução de uma instrução SUB E, o conteúdo dos registradores A e E representam números positivos na representação complemento a dois sem o sinal. Diga se as seguintes afirmações são falsas ou verdadeiras e justifique.

Após a execução da instrução SUB E:

(i) se (A) $\geqslant$ (E) então CY = 0

(ii) se (A) < (E) então CY = 1
(iii) se (A) = (E) então Z = 1
(iv) se (A) ≠ (E) então Z = 0

9.8.12 — Faça um programa que subtraia o conteúdo do registrador D do acumulador e adicione este resultado ao bit carry anterior à subtração. O resultado deve ser armazenado no acumulador.

9.8.13 — Faça um programa que some dois números com a precisão de três bytes. Cada número está armazenado em três posições de memória consecutivas com suas partes mais significativas nos endereços de maior valor. O endereço de menor valor de cada número é passado em IX e IY. O resultado deve ser colocado no lugar do operando indicado por IY.

9.8.14 — Faça o exercício anterior com a operação de subtração. Assuma que IX indica o minuendo.

9.8.15 — Verifique se as afirmativas do exercício 9.8.11 são válidas caso os números armazenados nos registradores A e E venham representar números na notação complemento a 2 com o sinal.

9.8.16 — Verifique se as afirmativas do exercício 9.8.11 são válidas caso os números representados nos registradores A e E possuam sinais contrários.

9.8.17 — Dado o seguinte programa que compara dois números binários positivos de 16 bits, contidos nos registradores DE e HL respectivamente, explique como avaliar o sinal do resultado.

```
LD    A,E
SUB   L
LD    A,D
SBC   A,H
```

10 INSTRUÇÕES ARITMÉTICAS DE 16 BITS, ARITMÉTICAS ESPECIAIS E DE CONTROLE

10.1 – APRESENTAÇÃO

Após termos apresentado no capítulo anterior as instruções aritméticas de 8 bits, estudaremos as operações aritméticas de 16 bits e as aritméticas especiais, as instruções do bit carry e as instruções de controle do Z-80.

10.2 – INSTRUÇÕES ARITMÉTICAS DE 16 BITS

O grupo de *instruções aritméticas de 16 bits* realiza operações de adição e subtração com os pares de registradores BC, DE ou HL ou com os registradores de 16 bits, IX, IY ou SP.

Da mesma forma que as instruções do grupo de operações lógicas e aritméticas de 8 bits, as instruções deste grupo classificam-se em dois subgrupos. O *primeiro subgrupo* envolve operações com dois operandos. O *segundo subgrupo* envolve operações com um operando.

Os operandos em todas as instruções dos dois subgrupos só podem ser especificados pelo *endereçamento por registrador.*

A tabela 10.1 apresenta as instruções aritméticas de 16 bits, com um resumo sucinto das suas principais características.

TABELA 10.1 – INSTRUÇÕES ARITMÉTICAS DE 16 BITS

Mnemônico 8080/85	Mnemônico Z-80	Operação Simbólica	Bits de Condição C	Z	P/V	S	N	H	Código de Máquina binário 76	543	210	Nº de Bytes	Nº de M ciclos	Nº de T ciclos	SUB-GRUPO
OPERAÇÕES ARITMÉTICAS DE 16 BITS COM DOIS OPERANDOS															
DAD ss	ADD HL,ss	HL ← HL + ss	↕	●	●	●	0	X	00	ss1	001	1	3	11	
–	ADD IX,pp	IX ← IX + pp	↕	●	●	●	0	X	11	011	101	2	4	15	
									01	pp1	001				
–	ADD IY,rr	IY ← IY + rr	↕	●	●	●	0	X	11	111	101	2	4	15	
									00	rr1	001				1
–	ADC HL,ss	HL ← HL + ss + CY	↕	↕	V	↕	0	X	11	101	101	2	4	15	
									01	ss1	010				
–	SBC HL,ss	HL ← HL – ss – CY	↕	↕	V	↕	1	X	11	101	101	2	4	15	
									01	ss0	010				
OPERAÇÕES ARITMÉTICAS DE 16 BITS COM UM OPERANDO															
INX ss	INC ss	ss ← ss + 1	●	●	●	●	●	●	00	ss0	011	1	1	6	
–	INC IX	IX ← IX + 1	●	●	●	●	●	●	11	011	101	2	2	10	
									00	100	011				
–	INC IY	IY ← IY + 1	●	●	●	●	●	●	11	111	101	2	2	10	
									00	100	011				2
DCX ss	DEC ss	ss ← ss – 1	●	●	●	●	●	●	00	ss1	011	1	1	6	
–	DEC IX	IX ← IX – 1	●	●	●	●	●	●	11	011	101	2	2	10	
									00	101	011				
–	DEC IY	IY ← IY – 1	●	●	●	●	●	●	11	111	101	2	2	10	
									00	101	011				

NOTAS:

1 – "ss" simboliza qualquer um dos seguintes registradores de 16 bits: BC(00), DE(01), HL(10) e SP(11).
2 – "pp" simboliza qualquer um dos seguintes registradores de 16 bits: BC(00), DE(01), IX(10) e SP(11).
3 – "rr" simboliza qualquer um dos seguintes registradores de 16 bits: BC(00), DE(01), IY(10) e SP(11).
4 – Notação dos bits de condição: ● = não afetado, 0 = resetado, 1 = setado.
X = indefinido.
↕ = o bit é afetado em função do resultado da operação.

Tabela 10.1 – Instruções Aritméticas de 16 bits

10.2.1 — Subgrupo 1 — Operações com dois operandos

As instruções do subgrupo 1 operam dois números de 16 bits. Os seus mnemônicos são compostos do código de operação (ADD, ADC ou SBC) e dos dois operandos separados por uma vírgula. No operando da esquerda é armazenado o resultado da operação e o da direita é um registrador de 16 bits do Z-80.

As operações de adição podem ser *sem carry* (ADD) ou *com carry* (ADC) e são realizadas sempre na aritmética complemento a dois.

Os resultados das operações de adição sem carry são depositados no par HL (ADD HL, ss), no registrador IX (ADD IX, pp) ou no registrador IY (ADD IY, rr).

Na instrução ADD HL,ss o par HL pode ser adicionado aos pares BC, DE, ao próprio HL e ao registrador SP.

Na instrução ADD IX,pp o registrador IX pode ser adicionado aos pares BC e DE, ao registrador SP e ao próprio IX.

Na instrução ADD IY,rr o registrador IY pode ser adicionado aos pares BC e DE, ao registrador SP e ao próprio IY.

Observe que as operações de adição *sem carry* só permitem testar o *carry bit* dos seus resultados.

Nas operações de adição com carry (ADC HL,ss) o par HL pode ser adicionado aos pares BC, DE e HL e ao registrador SP. O resultado destas operações é sempre depositado no registrador HL.

Observe que as operações de adição *corn carry* permitem testar os bits S, Z, C e o overflow.

As operações de subtração com operandos de 16 bits (SBC HL,ss) são sempre *com carry* e são realizadas sempre na aritmética complemento a dois. Nestas operações o par HL é sempre um dos operandos e dele pode ser subtraído o carry bit e os pares BC, DE, o próprio HL e o registrador SP.

As operações de subtração com operandos de 16 bits possibilitam testar os bits, S, Z, C e o overflow.

10.2.1.1 – Exemplo nº 1

Assuma que o registrador SP contém F3FFH e o registrador HL = 0000H, então, a instrução ADD HL, SP realiza a operação de adição conforme mostra a figura 10.1.

Observe que o exemplo nº 1 efetivamente transfere o conteúdo de SP para HL.

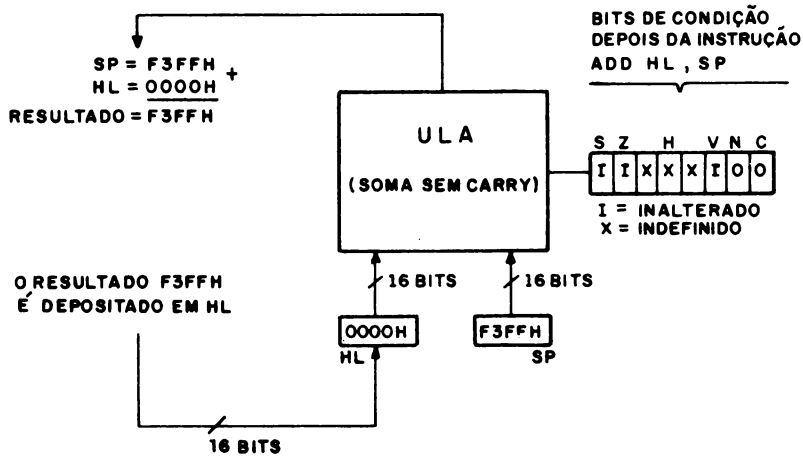


Fig. 10.1 – A instrução ADD HL, SP

10.2.1.2 – Exemplo nº 2

Assuma que o par BC contém 2000H, o par HL = 07FFH e o bit C = 1, então, a instrução ADC HL, BC realiza a operação de adição com carry conforme mostra a figura 10.2.

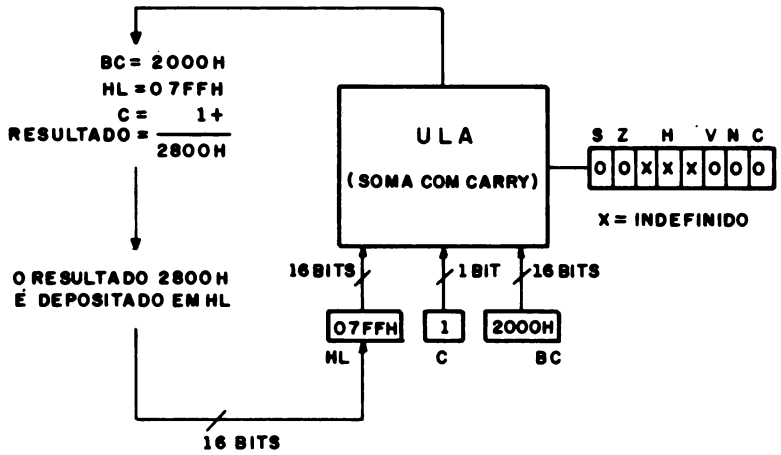


Fig. 10.2 – A Instrução ADC HL, BC

10.2.1.3 – Exemplo nº 3

Assuma que o par DE contém 03FFH, o par HL = A3FFH e o bit C = O, então, a instrução SBC HL, DE realiza a operação de subtração com carry conforme mostra a figura 10.3.

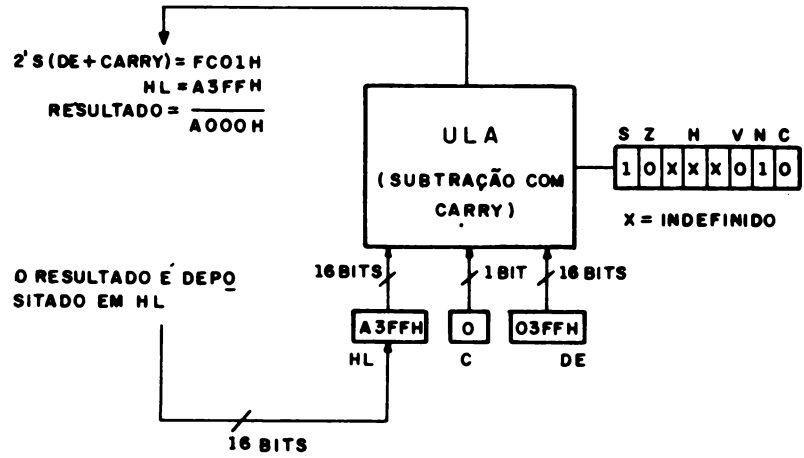


Fig. 10.3 – A Instrução SBC HL, DE

10.2.2 – Subgrupo 2 – Operações com um operando

As instruções do subgrupo 2 operam um único número de 16 bits, diferente do subgrupo 1 que opera dois números. Os seus mnemônicos são compostos do código de operação (INC ou DEC) e do operando.

As operações de incremento (INC) adicionam uma unidade ao operando e as operações decremento (DEC) subtraem uma unidade. Ambas as operações são realizadas na aritmética complemento a dois.

Os operandos das subtrações de incremento e decremento podem ser os pares BC, DE e HL e os registradores SP, IX e IY.

Observe que as insttruções do subgrupo 2 *não afetam* os bits de condição.

10.2.2.1 – Exemplo nº 4

- Assuma que o registrador IX contém FFFFH, então, a instru-

ção INC IX realiza a operação de incremento conforme mostra a figura 10.4.

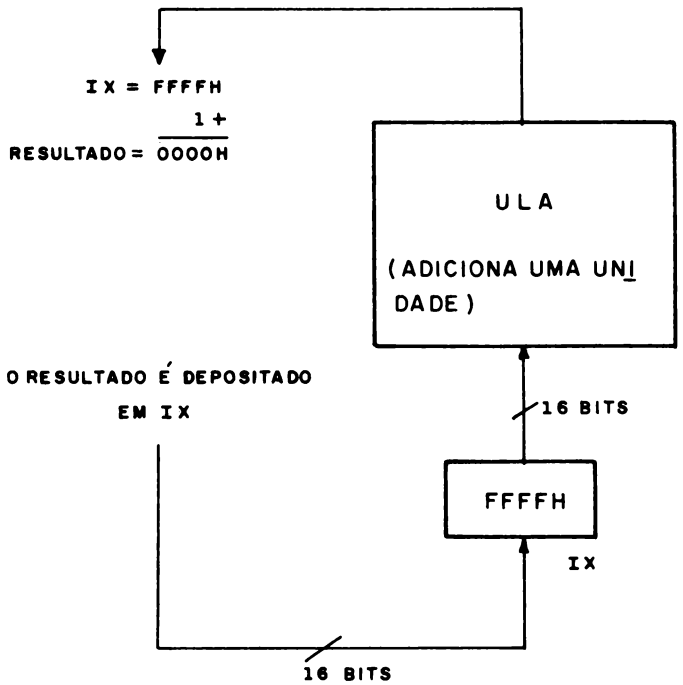


Fig. 10.4 – A Instrução INC IX

10.3 – INSTRUÇÕES ARITMÉTICAS ESPECIAIS

O grupo de instruções artiméticas especiais realiza operações de complemento a um (CPL), complemento a dois (NEG) e ajuste de número na representação BCD (DAA).

A tabela 10.2 apresenta as instruções aritméticas especiais.

Todas as instruções deste grupo utilizam o endereçamento implícito, envolvendo sempre o acumulador como operando.

TABELA 10.2 – INSTRUÇÕES ARITMÉTICAS ESPECIAIS

Mnemônico		Operação Simbólica	Bits de Condição						Código de Máquina binário			Nº de Bytes	Nº de M ciclos	Nº de T ciclos
8080/85	Z-80		C	Z	P/V	S	N	H	76	543	210			
CMA	CPL	$A \leftarrow \overline{A}$	●	●	●	●	1	1	00	101	111	1	1	4
–	NEG	$A \leftarrow 0 - A$	↕	↕	V	↕	1	↕	11 01	101 000	101 100	2	2	8
DAA	DAA	Ajuste Decimal	↕	↕	P	↕	●	↕	00	100	111	1	1	4

NOTAS:

1 – Notação dos bits de condição: ● = não afetado, 0 = resetado, 1 = setado.
X = indefinido.
↕ = o bit é afetado em função do resultado da operação.

Tabela 10.2 – Instruções Aritméticas Especiais

A instrução CPL determina o complemento a um do conteúdo do acumulador pela troca de todos os bits zero para um, e vice-versa. A figura 10.5 ilustra a instrução CPL.

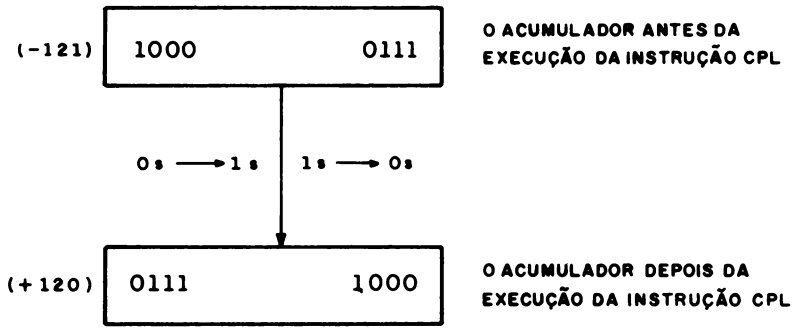


Fig. 10.5 – A Instrução CPL

A instrução NEG determina o complemento a dois do conteúdo do acumulador, pela adição de uma unidade ao seu complemento a um. A figura 10.6 ilustra a instrução NEG.

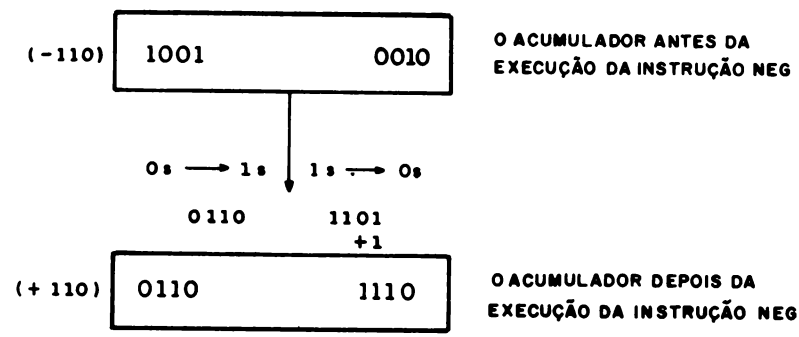


Fig. 10.6 – A Instrução NEG

A instrução DAA realiza o ajuste do acumulador após operações com números BCD. Lembramos ao leitor que o funcionamento da instrução DAA foi estudado no capítulo 3.

A instrução DAA, usada imediatamente após as instruções ADD, ADC, INC, SUB, DEC ou NEG, ajusta o resultado obtido para BCD.

10.4 – INSTRUÇÕES DO BIT CARRY

O grupo de instruções do bit carry realiza operações que manipulam o bit carry.

A tabela 10.3 apresenta as instruções do bit carry.

TABELA 10.3 – INSTRUÇÕES DO BIT CARRY

Mnemônico		Operação Simbólica	Bits de Condição						Código de Máquina binário			Nº de Bytes	Nº de M ciclos	Nº de T ciclos
8080/85	Z-80		C	Z	P/V	S	N	H	76	543	210			
CMC	CCF	CY ← $\overline{CY}$	↕	●	●	●	0	X	00	111	111	1	1	4
STC	SCF	CY ← 1	1	●	●	●	0	0	00	110	111	1	1	4

NOTAS:

1 – CY simboliza o flip-flop do carry bit.
2 – Notação dos bits de condição: ● = não afetado, 0 = resetado.
1 = setado, X = indefinido.
↕ = o bit é afetado em função do resultado da operação.

Tabela 10.3 – Instruções do Bit Carry

A primeira instrução (CCF) complementa o bit carry e a segunda (SCF) seta o bit carry.

O valor do bit carry pode ser testado após a instrução CCF.

O programa da figura 10.7 armazena o valor zero no bit carry.

```
SCF     ; SETA O BIT CARRY
CCF     ; COMPLEMENTA O BIT CARRY
HALT
```

Fig. 10.7 – Programa que reseta o bit carry

10.5 – INSTRUÇÕES DE CONTROLE

As instruções de controle do Z-80 provocam a sua entrada no estado operacional NOP ou no estado HALT e, ainda, determinam os modos de atendimento de interrupções.

O estado operacional NOP é destituído de qualquer operação e o estado HALT é caracterizado pela paralisação da busca de instrução.

Para efeitos de estudo, dividiremos este grupo em dois outros subgrupos. O *subgrupo 1* envolve as instruções de controle dos estados operacionais do Z-80 e o *subgrupo 2* envolve as instruções relativas ao atendimento de interrupções.

A tabela 10.4 apresenta as instruções de controle do Z-80.

TABELA 10.4 – INSTRUÇÕES DE CONTROLE DO Z-80

Mnemônico		Operação Simbólica	Bits de Condição						Código de Máquina binário			Nº de Bytes	Nº de M ciclos	Nº de T ciclos	SUB-GRUPO
8080/85	Z-80		C	Z	P/V	S	N	H	76	543	210				
		OPERAÇÕES DE CONTROLE DE ESTADOS OPERACIONAIS													
NOP	NOP	Sem operação	●	●	●	●	●	●	00	000	000	1	1	4	1
HLT	HALT	Estado de halt	●	●	●	●	●	●	01	110	110	1	1	4	
		OPERAÇÕES DE CONTROLE DE INTERRUPÇÕES													
DI	DI	IFF ← 0	●	●	●	●	●	●	11	110	011	1	1	4	
EI	EI	IFF ← 1	●	●	●	●	●	●	11	111	011	1	1	4	
–	IM 0	Ativa modo 0 de interrupção	●	●	●	●	●	●	11 01	101 000	101 110	2	2	8	2
–	IM 1	Ativa modo 1 de interrupção	●	●	●	●	●	●	11 01	101 010	101 110	2	2	8	
–	IM 2	Ativa modo 2 de interrupção	●	●	●	●	●	●	11 01	101 011	101 110	2	2	8	

Notação dos bits de condição: ● = não afetado.

Tabela 10.4 – Instruções de Controle do Z-80.

10.5.1 – Subgrupo 1 – Instruções de controle operacional

O Z-80 possui cinco estados operacionais: WAIT, NOP, HOLD, HALT e NORMAL. A passagem para os estados de WAIT ou HOLD se faz por *Hardware* e, para os estados NOP e HALT por *Software.*

O estado de WAIT permite que o Z-80 troque informações com periféricos lentos.

O estado de HOLD é próprio para a realização de operações de DMA (Acesso Direto à Memória).

O estado NORMAL é aquele em que o Z-80 fica no processo contínuo de busca, interpretação e execução de instruções. Consideramos o atendimento de interrupções como parte integrante do estado NORMAL do Z-80. Podemos ter ainda estados de WAIT ou HOLD intercalados ao estado normal.

O Z-80 entra nos estados de NOP e HALT por software e pelas instruções do mesmo nome, que fazem parte do subgrupo que estamos estudando.

A figura 10.8 mostra o diagrama de transição entre os estados NORMAL, HALT e NOP.

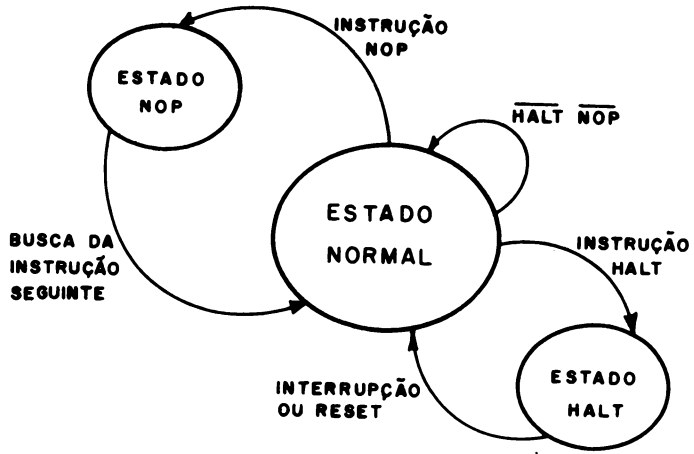


Fig. 10.8 – Estados NORMAL, HALT e NOP

O Z-80 se mantém no estado normal até que ocorra uma busca de uma instrução HALT ou NOP.

A *instrução HALT* coloca o Z-80 no estado HALT, que caracteriza-se por paralisar o avanço do contador de programa e a busca de instruções. A saída do estado de HALT só é possível através de uma interrupção ou do reset.

A *instrução NOP* coloca o Z-80 no estado NOP, que caracteriza-se por executar uma instrução destituída de qualquer operação. O *estado NOP* é um estado quase-estável, pois finda a execução da instrução NOP, uma nova instrução é apanhada e executada, retornando o Z-80, desse modo, ao estado normal.

Geralmente, usa-se a instrução NOP quando se deseja provocar atrasos de tempo ("delay") em um programa ou reservar posições de memória intercaladas entre programas.

10.5.2 – Subgrupo 2 – Instruções de controle de interrupções

A *instrução DI* bloqueia qualquer solicitação de interrupção que chega à entrada $\overline{\text{INT}}$ (pino 16) do Z-80. Efetivamente, esta instrução reseta o flip-flop IFF1, interno ao Z-80.

A *instrução EI* habilita as interrupções que chegam à entrada $\overline{\text{INT}}$, setando o flip-flop IFF1.

As interrupções geradas pela entrada $\overline{\text{NMI}}$ (pino 17) não são afetadas pelas instruções EI ou DI e, ficam sempre habilitadas.

As *instruções IM0, IM1 e IM2* habilitam o Z-80 a atender interrupções INT nos modos 0, 1 e 2, respectivamente.

Após o reset o *Modo 0* é automaticamente ativado ("default").

Recomendamos o estudo do capítulo 6 do nosso livro de HARDWARE, para o melhor entendimento dos modos de interrupção do Z-80.

10.6 – EXERCÍCIOS DE FIXAÇÃO

10.6.1 – Realize os programas a seguir em Assembly e monte o código objeto a partir do endereço 0000H.

a) Some o conteúdo do par HL com o conteúdo do registrador SP e armazene o resultado no par BC.

b) Subtraia do conteúdo do par DE o conteúdo do par HL e armazene o resultado no par BC.

10.6.2 – Cite uma razão para que os bits de condição não sejam afetados após as instruções INC e DEC.

10.6.3 – Porque é necessário a utilização da instrução DAA para atualizar uma operação de soma ou subtração de nºs BCD?

10.6.4 – Como é possível a instrução DAA ajustar resultados de subtração?

10.6.5 – Caracterize o estado de NOP. Cite uma aplicação para a utilização desta instrução.

10.6.6 – Explique o diagrama de transição entre os estados NORMAL, NOP e HALT.

10.6.7 – Como ocorre a entrada do Z-80 nos estados de HOLD e WAIT?

10.6.8 – Porque o 8080 não possui as instruções IMO, IMI e IM2?

10.6.9 – O Z-80 aceita interrupções provenientes da entrada $\overline{INT}$ após a liberação do RESET?

10.6.10 – Qual o modo de interrupção do Z-80 que é compatível com a interrupção no 8080?

10.6.11 – Faça um programa que some o conteúdo do topo da pilha com o conteúdo do par DE, sendo que o resultado desta operação deve ficar no topo da pilha.

10.6.12 — Faça um programa que determine o complemento a dois de um número de 16 bits armazenado na memória.
A parte mais significativa está na posição 8001H e a menos significativa está na posição 8000H. O resultado deve ser armazenado nas posições originais.

10.6.13 — Explique o funcionamento da instrução HALT e cite uma aplicação para a sua utilização.

10.6.14 — Cite duas aplicações para a instrução NOP.

10.6.15 — O que acontece com o Z-80 se forem executadas as instruções DI e HALT em seqüência?

10.6.16 — Faça um programa que multiplique por 20D o conteúdo do par HL.

10.6.17 — Faça um programa que transfira quatro bytes em posições consecutivas na memória para outras quatro posições na memória, também consecutivas. O par DE endereça o primeiro byte do bloco de bytes de origem e o par BC endereça o primeiro byte do bloco de destino.

10.6.18 — Faça um programa que transfira para o acumulador o conteúdo da posição de memória, cujo endereço é dado pela soma do par HL e o topo da pilha.

10.6.19 — Faça um programa que some dois números positivos representados na notação BCD. O endereço do primeiro operando está no par HL e o segundo operando está no registrador C.

10.6.20 — Faça o exercício 9.8.13 do capítulo anterior assumindo que os dois operandos estão representados em BCD.

10.6.21 — Faça o exercício 9.8.14 do capítulo anterior assumindo que os dois operandos estão representados em BCD.

10.6.22 — Faça um programa que subtraia 1000H do ponteiro de pilha.

11 INSTRUÇÕES DE DESLOCAMENTO

11.1 – APRESENTAÇÃO

Neste capítulo estudaremos as instruções de deslocamento do microprocessador Z-80.

Estas instruções são muito usadas em rotinas aritméticas e em conversão de códigos na comunicação com dispositivos de entrada e saída.

11.2 – INTRODUÇÃO

Este grupo de instruções realiza *deslocamentos lineares ("shift")* e *deslocamentos circulares ("rotate")*, para a direita ou para a esquerda, nos bytes armazenados em registradores do Z-80 e em posições de memória. Além disso, o Z-80 tem instruções que realizam o deslocamento de números na representação BCD.

Os deslocamentos circulares podem também ser *através do bit carry* e os deslocamentos lineares podem ser *lógicos* ou *aritméticos.*

Para fins didáticos, subdividiremos o grupo de instruções de deslocamento nos quatro subgrupos abaixo:

SUBGRUPO 1 – Deslocamentos circulares compatíveis com o 8080;

SUBGRUPO 2 – Deslocamentos circulares específicos do Z-80;

SUBGRUPO 3 – Deslocamentos lineares;

SUBGRUPO 4 – Deslocamentos para operandos em BCD.

A tabela 11.1 apresenta as instruções de deslocamento do Z-80, com um resumo sucinto das suas principais características. Esta tabela está divida nos mesmos subgrupos apresentados.

TABELA 11.1 – INSTRUÇÕES DE DESLOCAMENTO

Mnemônico 8080/85	Mnemônico Z-80	Operação Simbólica	Bits de Condição C	Z	P/V	S	N	H	Código de Máquina Binário 76	543	210	Nº de Bytes	Nº de M ciclos	Nº de T ciclos	SUB-GRUPO
OPERAÇÕES DE DESLOCAMENTO CIRCULAR COMPATÍVEIS COM O 8080															
RLC	RLCA	Veja pg. 160	↕	●	●	●	0	0	00	000	111	1	1	4	
RAL	RLA	Veja pg. 161	↕	●	●	●	0	0	00	010	111	1	1	4	
RRC	RRCA	Veja pg. 161	↕	●	●	●	0	0	00	001	111	1	1	4	1
RAR	RRA	Veja pg. 162	↕	●	●	●	0	0	00	011	111	1	1	4	
OPERAÇÕES DE DESLOCAMENTO CIRCULAR ESPECÍFICAS DO Z-80															
–	RLC r		↕	↕	P	↕	0	0	11 00	001 000	011 r	2	2	8	
	RLC (HL)		↕	↕	P	↕	0	0	11 00	001 000	011 110	2	4	15	
–	RLC (IX + d)	Veja pg. 163	↕	↕	P	↕	0	0	11 11 ←	011 001 d	101 011 →	4	6	23	
–	RLC (IY + d)		↕	↕	P	↕	0	0	00 11 11 ←	000 111 001 d	110 101 011 →	4	6	23	2
–	RL s	Veja pg. 163 S = r, (HL), (IX + d) e (IY + d)	↕	↕	P	↕	0	0	00	000 010	110				
–	RRC s	Veja pg. 163 S = r,(HL),(IX + d), (IY + d)	↕	↕	P	↕	0	0		001					

–	RR s	Veja pg. 163 S = r,(HL),(IX + d), IY + d)	↕	↕	P	↕	0	0		011				
		OPERAÇÕES DE DESLOCAMENTO LINEAR												
–	SLA s	Veja pg. 164 S = r,(HL), (IX + d), (IY + d)	↕	↕	P	↕	0	0		100				**3**
–	SRA s	Veja pg. 165 S = r,(HL),(IX + d), (IY + d)	↕	↕	P	↕	0	0		101				
–	SRL s	Veja pg. 166 S = r,(HL), (IX + d), (IY + d)	↕	↕	P	↕	0	0		111				
		OPERAÇÕES DE DESLOCAMENTO EM NÚMEROS NA REPRESENTAÇÃO BCD												
–	RLD	Veja pg. 167	●	↕	P	↕	0	0	11 01	101 101	101 111	2	5	18 **4**
–	RRD	Veja pg. 168	●	↕	P	↕	0	0	11 01	101 100	101 111	2	5	18

NOTAS:

1 – "r" simboliza qualquer um dos seguintes registradores: B(000), C(011), D(010), E(011), H(100), L(101) e A(111).
2 – O formato, número de bytes e ciclos para as instruções RL, RRC, RR, SLA, SRA e SRL são iguais aos da instrução RLC. Para formar os novos códigos de máquina binário basta substituir 000 em RLC pelo código mostrado em cada instrução.
3 – Notação do bits de condição: ● = não afetado, 0 = resetado, 1 = setado.
X = indefinido.
↕ = o bit é afetado em função do resultado da operação.

Tabela 11.1 – Instruções de Deslocamento.

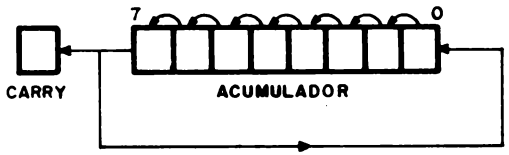
11.3 – SUBGRUPO 1 – DESLOCAMENTOS CIRCULARES COMPATÍVEIS COM O 8080

O subgrupo 1 inclui as instruções de deslocamento circular cujos códigos objetos são compatíveis com os do microprocessador 8080, que opera somente o acumulador.

Como se infere da tabela 11.1, as instruções deste subgrupo têm os seguintes mnemônicos: RLCA, RLA, RRCA e RRA. A primeira letra destes mnemônicos origina-se da palavra *"rotate"*, a segunda letra simboliza o sentido do deslocamento circular *"left"* (para a esquerda) e *"right"* (para a direita) e, a última letra tem a sua origem na palavra *acumulador*. A ausência do "C" nos mnemônicos caracteriza que o deslocamento se faz através do bit carry.

Observe que o único bit de condição afetado pelas instruções do subgrupo 1 é o *bit carry*.

A *instrução RLCA* realiza o deslocamento circular para a esquerda. O bit mais significativo do acumulador é transferido para o bit menos significativo do acumulador. A figura 11.1 mostra esta operação e um exemplo ilustrando o estado do acumulador antes e depois da sua execução.

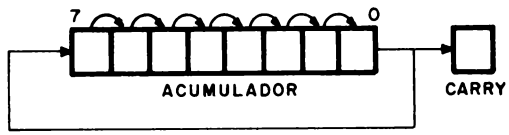


EXEMPLO

ANTES	DEPOIS
A = 01010101 B	A = 10101010 B
CARRY = 1	CARRY = 0

Fig. 11.1 – A Instrução RLCA

A *instrução RRCA* realiza operação similar a da instrução RLCA, sendo o deslocamento circular para a esquerda. A figura 11.2 ilustra a instrução RRCA e um exemplo.

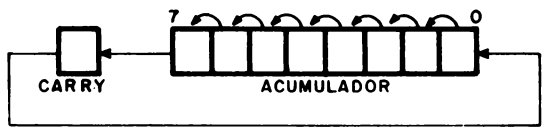


EXEMPLO

ANTES	DEPOIS
A= 11110000 B	A= 01111000 B
CARRY = 1	CARRY = 0

Fig. 11.2 – A Instrução RRCA

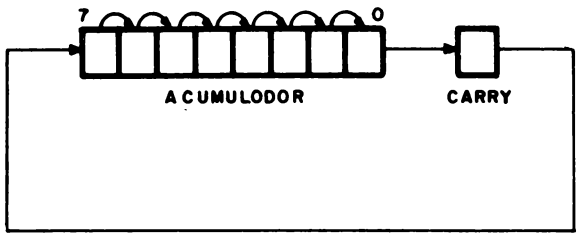
As *instruções RLA e RRA* realizam o deslocamento através do carry; como se o acumulador estivesse concatenado com o carry bit formando um registrador de nove bits. Nestas operações o conteúdo do bit carry é transferido para o acumulador e, o bit deslocado para fora do acumulador é transferido para o bit carry. A figura 11.3 ilustra a operação da instrução RLA e a figura 11.4 ilustra a instrução RRA.



EXEMPLO

ANTES	DEPOIS
A = 00001111 B	A= 000 11111 B
CARRY = 1	CARRY = 0

Fig. 11.3 – A Instrução RLA



EXEMPLO

ANTES	DEPOIS
A = 11000101 B	A = 01100010 B
CARRY = 0	CARRY = 1

Fig. 11.4 – A Instrução RRA

11.4 – SUBGRUPO 2 – DESLOCAMENTOS CIRCULARES ESPECÍFICOS DO Z-80

As instruções do *subgrupo 2* incluem as mesmas operações de deslocamento circular do subgrupo 1. No entanto, as instruções do subgrupo 2 realizam deslocamentos circulares em todos os registradores internos do Z-80, pela técnica de endereçamento por registrador, e em bytes armazenados na memória, pelos endereçamentos por registrador indireto e indexado. No caso do endereçamento por registrador indireto só é possível usar o HL como *ponteiro.*

Os mnemônicos do subgrupo 2 são iguais aos do subgrupo 1 retirando-se a letra A, para descaracterizar que os deslocamentos só se fazem no acumulador.

A figura 11.5 ilustra a ação das instruções deste grupo.

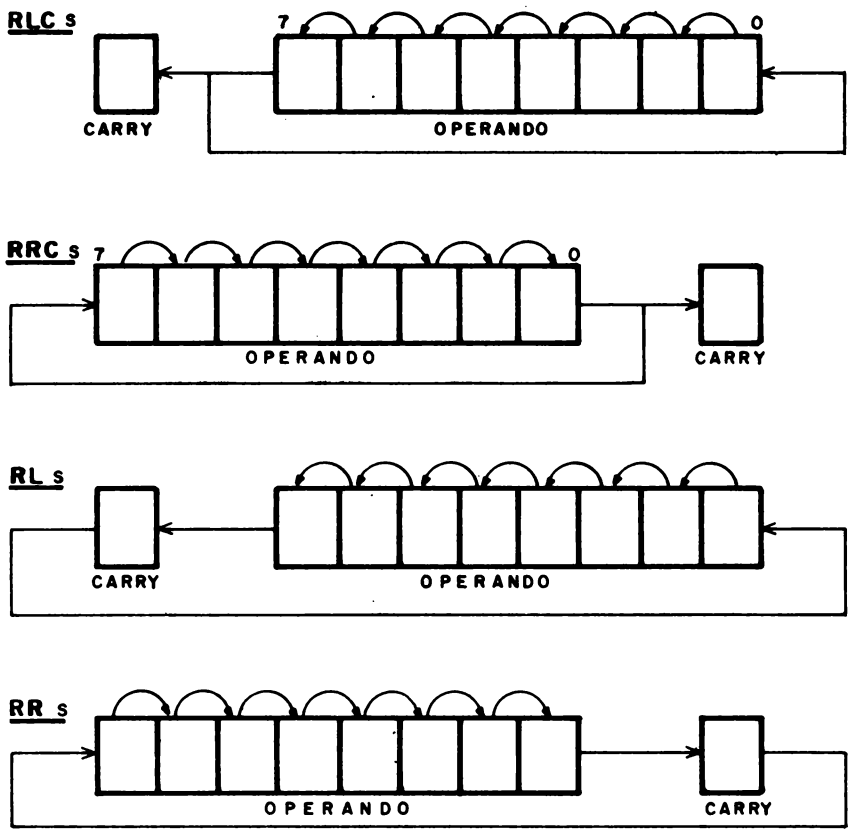


Fig. 11.5 – Deslocamentos circulares específicos do Z-80.

Convém ressaltar ainda que as instruções do subgrupo 2 afetam os bits de condição C, Z e S, e o bit P/V fornece a condição de paridade.

11.5 – SUBGRUPO 3 – DESLOCAMENTOS LINEARES

As instruções deste subgrupo realizam deslocamentos aritméticos para a direita e para a esquerda, e deslocamento lógico para a direita. Estas operações de deslocamento podem ser feitas em todos os registradores de uso geral do Z-80, pela técnica de endereçamento por registrador, ou em operandos na memória, usando-se o HL como *ponteiro* ou, ainda, pelo endereçamento indexado.

Os mnemônicos das instruções de deslocamento linear começam pela letra "S" de "shift" (deslocar). A segunda letra indica o sentido do deslocamento "left" (para a esquerda) e "right" (para a direita). A última letra caracteriza a natureza aritmética ou lógica do deslocamento, através das letras "A" e "L" respectivamente.

As instruções de deslocamento linear afetam os bits de condição C, Z e S, e o bit P/V fornece a condição de paridade.

11.5.1 — Deslocamento Aritmético à Esquerda — SLA

As operações aritméticas, pela sua própria natureza, necessitam preservar o sinal. No caso das operações de deslocamento aritmético para a esquerda, tanto para deslocamentos em registradores como em posições de memória, o bit mais significativo (o sinal) do byte é transferido para o bit carry e, um zero é automaticamente inserido no lugar do bit menos significativo. Esta operação está representada graficamente na figura 11.6.

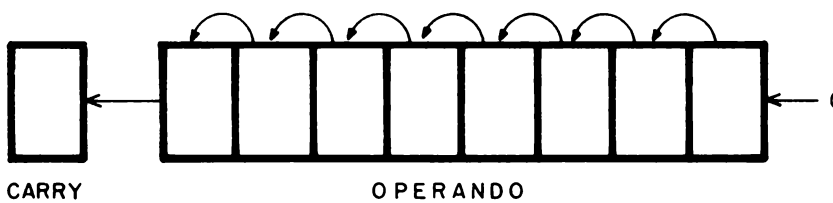


Fig. 11.6 – A Instrução SLA s

11.5.1.1 — Exemplo nº 1

Antes da execução da instrução SLA (IX + 20H) temos: IX= A000H, (AO20H) = 01011000B e o bit carry = 1. Então, após a

sua execução temos: (A020H) = 10110000B, bit carry = 0, bit P = 0 e bit S = 1.

11.5.2 – Deslocamento Aritmético à Direita – SRA

No caso do deslocamento artimético à direita o sinal é preservado na sua posição original, isto é, no bit 7. E, também, transferido para o bit 6. O bit 0 é transferido para o bit carry e ali preservado. A figura 11.7 ilustra esta operação.

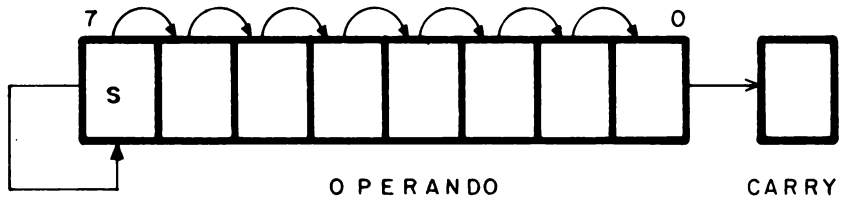


Fig. 11.7 – A Instrução SRA s

11.5.2.1 – Exemplo nº 2

Suponha que antes da execução da instrução SRA H temos: H = 01011000B e o bit carry = 1. Então, após a sua execução temos: H = 00101100B, bit carry = 0, bit Z = 0, bit P = 0 e o bit S = 0.

11.5.3 – Deslocamento lógico à Direita – SRL

De um modo geral, um operando em uma operação aritmética algébrica necessita do sinal. No entanto, para operações lógicas, via de regra o sinal não tem significado. No Z-80, o deslocamento à direita, automaticamente, insere um zero no bit mais significativo do operando. O bit menos significativo é transferido para o bit carry e ali preservado. Esta operação está representada na figura 11.8.

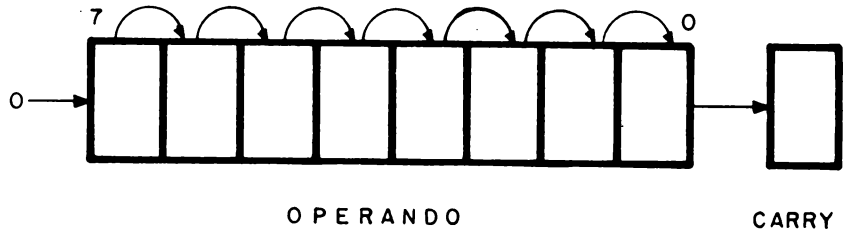


Fig. 11.8 – A Instrução SRL s

11.5.3.1 – Exemplo nº 3

Suponha que antes da execução da instrução SRL B temos: B = 11000000B e o bit carry = 1. Então, após sua execução temos: H = 01100000, bit carry = 0, bit Z = 0, bit P = 1 e bit S = 0.

11.6 – SUBGRUPO 4 – DESLOCAMENTOS PARA OPERANDOS EM BCD

Os deslocamentos deste *subgrupo* – RLD e RRD – operam o acumulador e o conteúdo de posição de memória endereçada pelo par HL e, deslocam 4 bits de uma só vez. Na verdade, estes dois deslocamentos foram implementados para facilitar o deslocamento de números na representação BCD, uma vez que cada dígito BCD é composto de 4 bits.

A primeira letra do mnemônico destas instruções origina-se de "rotate", a segunda indica o sentido do deslocamento e a última, letra "D", indica que o deslocamento é para números em BCD.

As instruções deste subgrupo afetam os bits de condição Z e S, e o bit P/V fornece a condição de paridade.

A instrução RLD realiza o deslocamento para a esquerda

transferindo os quatro bits menos significativos do acumulador para os quatro menos significativos da posição de memória endereçada pelo par HL. E estes últimos quatro bits para os quatro bits mais significativos desta mesma posição de memória, que por sua vez são transferidos para os 4 bits menos significativos do acumulador.

A figura 11.9 ilustra a execução desta instrução.

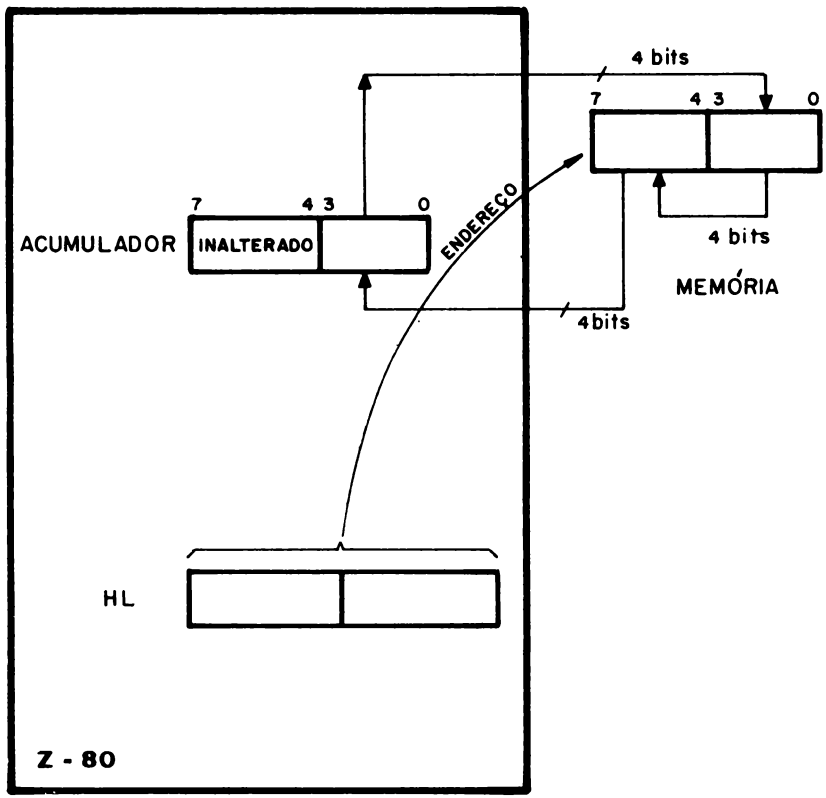


Fig. 11.9 – A Instrução RLD

A instrução RRD realiza o deslocamento para a direita. A figura 11.10 mostra a sua ação.

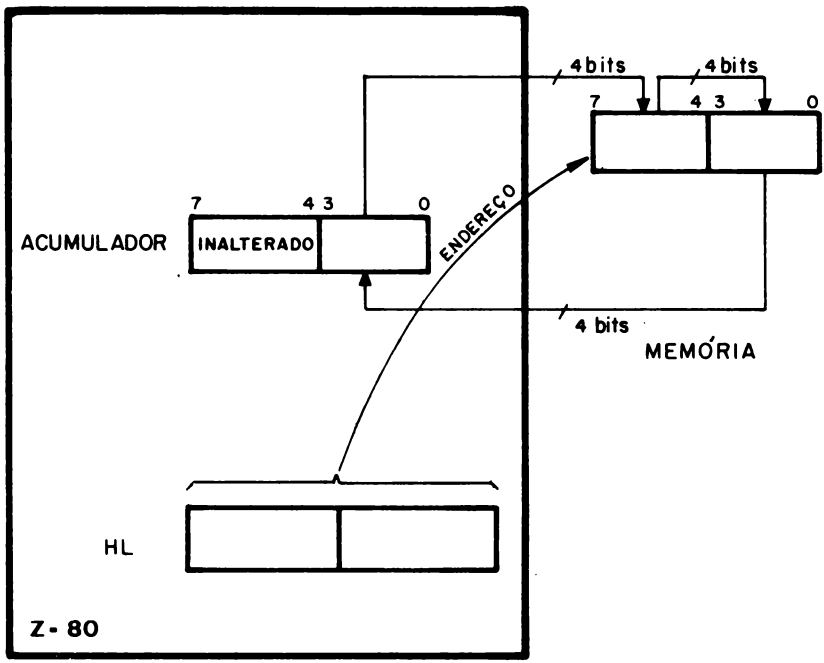


Fig. 11.10 – A Instrução RRD

11.6.1 – Exemplo nº 4

Suponha que antes da execução da instrução RLD temos: HL = 2F3DH, (2F3DH) = F0H, A = ADH. Então, após a sua execução temos: (2F3DH) = 0DH, A = AFH, bit S = 1, bit Z = 0 e bit P = 1.

Observe que os bits de condição são atualizados em função do resultado no acumulador.

11.6.2 – Exemplo nº 5

Suponha que antes da execução da instrução RRD temos: HL = 1000H, (1000H) = F4H e A = 05H. Então, após a sua execução temos: (1000H) = 5FH, A = 04H, bit S = 0, bit Z = 0 e bit P = 0.

11.7 – EXERCÍCIOS DE FIXAÇÃO

11.7.1 – Identifique as instruções do grupo de operações de deslocamento circular específicas do Z-80 compatíveis com as operações do 8080. Verifique se os bits de condição são alterados da mesma forma.

11.7.2 – Antes de executar a instrução RLC H o registrador H contém F6H e o bit de condição C está setado. Como ficam os registradores H e F após a execução desta instrução.

11.7.3 – Com relação a questão anterior, obtenha o resultado após a execução das instruções RLH, RRC H, RR H, SRA H e SRL H.

11.7.4 – Faça um programa que teste os bits 6 e 7 da posição de memória 2000H. Se bit 6 = bit 7 = 1 então finalize o programa com o bit carry igual a 1, caso contrário faça carry igual a 0.

11.7.5 – Faça um programa que realize o deslocamento aritmético à direita de um número em BCD de quatro algarismos armazenado em DE.

11.7.6 – Realize um programa que multiplique um número armazenado em L por 5 usando operações de deslocamento.

11.7.7 – Realize um programa que divida um número armazenado em A por 6 usando operações de deslocamento.

11.7.8 – Escreva um programa que faça um deslocamento à direita de um nº binário formado por 4 posições consecutivas de memória. O byte mais significativo do nº está localizado na posição 2000H e o bit menos significativo da posição 2003H deve ser armazenado no bit carry.

11.7.9 — Determine o valor final do acumulador e do bit carry.

a)
```
LD   A, 55H
OR   30H
RLCA
```

b)
```
LD   B, 45H
STC
RR   B
RRC B
```

c)
```
LD   A, F0H
XOR C3H
RRA
RRA
```

11.7.10 — Faça um programa que teste o bit 0 do registrador C. Sendo igual a '0', escreva FFH na posição 1FFFH. Caso contrário, escreva F0H nesta mesma posição.

12 INSTRUÇÕES DE MANIPULAÇÃO E TESTE DE BITS

12.1 – APRESENTAÇÃO

Neste capítulo estudaremos as instruções de manipulação e teste de bits

Estas instruções são muito importantes para o programador pois dispensam a utilização da técnica de mascaramento, na manipulação e teste de bits, que é necessária, por exemplo, nos microprocessadores 8080 e 8085.

12.2 – INTRODUÇÃO

As instruções de manipulação e teste de bits podem setar, resetar ou testar, separadamente, qualquer um dos bits de um byte armazenado em um registrador interno do Z-80 ou na memória. Operandos na memória podem ser acessados pelo endereçamento por registrador indireto, somente através do par HL, e pelo endereçamento indexado.

As instruções deste grupo são uma grande virtude do Z-80, pois para se fazer tais operações no 8080 são necessárias instruções lógicas que, normalmente, realizam rotações (carry bit) ou implementam a técnica de mascaramento ("and", "or")

Para fins de estudo, dividiremos as instruções deste grupo nos três subgrupos abaixo:

SUBGRUPO 1 – Operações de setar um bit
SUBGRUPO 2 – Operações de resetar um bit
SUBGRUPO 3 – Operações de teste de um bit

A tabela 12.1 apresenta as instruções de manipulação e teste de bits, com um resumo sucinto das suas principais características. Esta tabela está dividida nos mesmos subgrupos anteriormente apresentados.

TABELA 12.1 – INSTRUÇÕES DE MANIPULAÇÃO E TESTE DE BITS

Mnemônico 8080/85	Mnemônico Z-80	Operação Simbólica	Bits de Condição C	Z	P/V	S	N	H	Código de Máquina binário 76	543	210	Nº de Bytes	Nº de M ciclos	Nº de T ciclos	SUB-GRUPO
		OPERAÇÕES DE SETAR UM BIT													
–	SET b,r	$r_b \leftarrow 1$	●	●	●	●	●	●	11	001	011	2	2	8	
									11	b	r				
–	SET b,(HL)	$(HL)_b \leftarrow 1$	●	●	●	●	●	●	11	001	011	2	4	15	
									11	b	110				
–	SET b,(IX + d)	$(IX + d)_b \leftarrow 1$	●	●	●	●	●	●	11	011	101	4	6	23	1
									11	001	011				
									←	d	→				
									11	b	110				
–	SET b,(IY + d)	$(IY + d)_b \leftarrow 1$	●	●	●	●	●	●	11	111	101	4	6	23	
									11	001	011				
									←	d	→				
									11	b	110				
		OPERAÇÕES DE RESETAR UM BIT													
–	RES b,s	$s_b \leftarrow 0$ s = r,(HL), (IX + d), (IY + d)	●	●	●	●	●	●	10						2

OPERAÇÕES DE TESTE DE UM BIT

–	BIT b,r	$Z \leftarrow \overline{r_b}$	●	↕	X	X	0	1	11 01	001 b	011 r	2	2	8	
–	BIT b,(HL)	$Z \leftarrow \overline{(HL)_b}$	●	↕	X	X	0	1	11 01	001 b	011 110	2	3	12	
–	BIT b,(IX + d)	$Z \leftarrow \overline{(IX + d)_b}$	●	↕	X	X	0	1	11 11 ← 01	011 001 d b	101 011 → 110	4	5	20	3
–	BIT b,(IY + d)	$Z \leftarrow \overline{(IY + d)_b}$	●	↕	X	X	0	1	11 11 ← 01	111 001 d b	101 011 → 110	4	5	20	

NOTAS:

1 – "r" simboliza qualquer um dos seguintes registradores: B(000), C(001), D(010), E(011), H(100), L(101) e A(111).
2 – "b" simboliza os bits de um byte: 0(000), 1(001), 2(010), 3(011), 4(100), 5(101), 6(110) e 7(111).
3 – O formato, número de bytes, ciclos e bits de condição afetados para a instrução RES são iguais aos da instrução SET. Para formar os códigos de máquina binário de RES basta substituir 11 (bits 7 e 6) do último byte da SET por 10.
4 – Notação dos bits de condição: ● = não afetado, 0 = resetado, 1 = setado.
X = indefinido.
↕ = o bit é afetado em função do resultado da operação.

Tabela 12.1 – Instruções de Manipulação e Teste de Bits

12.3 – SUBGRUPO 1 – OPERAÇÕES DE SETAR UM BIT

Como se infere da tabela 12.1, as instruções deste subgrupo têm o mnemônico "SET". Os seus operandos designam o bit (b) que se deseja setar e o seu registrador r, (HL), (IX + d) e (IY + d).

Observe que nas operações SET os bit de condição não são afetados.

12.3.1 – Exemplo nº 1

Suponha que antes da execução da instrução SET 2, (IX + 10H) temos IX = 2000H e (2010H) = 00010000B. Então, após a sua execução temos IX = 2000H, (2010H) = 00010100B e os bits de condição inalterados.

12.4 – SUBGRUPO 2 – OPERAÇÕES DE RESETAR UM BIT

As instruções deste subgrupo têm o mnemônico "RES". Da mesma forma que para as instruções SET, os operandos das instruções RES designam o registrador e o bit que se deseja resetar. As instruções do subgrupo 2 também não afetam os bits de condição.

12.4.1 – Exemplo nº 2

Admita que antes da execução da instrução RES 0, E temos: 00001111B. Então, após a sua execução temos: E = 00001110B e os bits de condição inalterados.

12.5 – SUBGRUPO 3 – OPERAÇÕES DE TESTE DE UM BIT

As instruções deste subgrupo transferem o conteúdo de um determinado bit de um registrador para o bit de condição Z, com o seu estado invertido. Após esta operação o bit Z pode ser testado pelas instruções condicionais JUMP, CALL E RETURN. Desta maneira podemos testar qualquer bit separadamente.

As instruções deste subgrupo têm o mnemônico "BIT" e, da mesma maneira que nas instruções SET e RES, os seus operandos designam o registrador e o bit que se deseja testar.

Observe que as instruções "BIT" atuam apenas no bit de condição Z. Os demais bits alterados não têm significado para testes.

12.5.1 – Exemplo nº 3

Suponha que antes da execução da instrução BIT 7, (HL) temos: HL = F536H, (F536H) = 01001010B e bit Z = 0. Então, após a sua execução temos: HL = F536H, (F536H) = 01001010B e bit Z = 1.

A título de exemplo, a figura 12.1 mostra como transcorre as instruções SET 5, C, RES 6, (HL) e BIT 0, (IX + d).

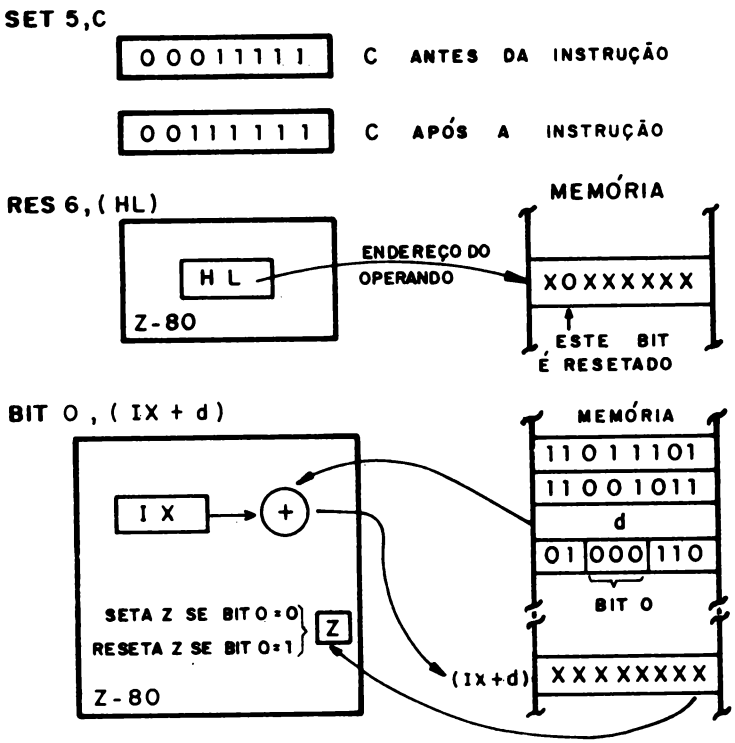


Fig. 12.1 – Instruções SET 5, C, RES 6, (HL) e BIT 7, (IX + d).

12.6 – EXERCÍCIOS DE FIXAÇÃO

12.6.1 – Sabendo-se que o registrador C contém 55H, como fica este registrador após as instruções SET C e RES C.

12.6.2 – Faça um programa que sete o bit 1 e resete o bit 6 da posição de memória 1000H.

12.6.3 – Faça um programa que teste se o bit 5 do registrador H é igual a 1. Se bit 5 = 1 então termine o programa com z = 0, em caso contrário com z = 1.

12.6.4 – Realize o exercício 11.7.4 usando instruções de teste.

12.6.5 – Realize o exercício 11.7.10 usando instruções de teste.

12.6.6 – Implemente rotinas que simulem as operações Test, Set e Reset de um determinado bit do acumulador usando operações lógicas.

13 INSTRUÇÕES DE MUDANÇA DE SEQÜÊNCIA

13.1 – APRESENTAÇÃO

Neste capítulo estudaremos as instruções de mudança de seqüência – JUMP'S –.

Estas instruções são fundamentais no desenvolvimento de qualquer programa, principalmente as que executam desvios condicionais.

13.2 – INTRODUÇÃO

As instruções deste grupo, basicamente, alteram a seqüência incremental de busca de instruções, determinada pelo contador de programa, *sem preservar o seu valor na pilha.* Isto, do ponto de vista do programador, é uma *mudança de seqüência* ou um *desvio no curso natural do programa.* Efetivamente, estas operações se fazem pela transferência de um novo valor ao contador de programa, diferente daquele automaticamente estabelecido pela unidade de controle do Z-80. A figura 13.1 ilustra uma operação de mudança de seqüência. Note que se a instrução armazenada na posição de memória 1000H não fosse de desvio teríamos PC = 1003H.

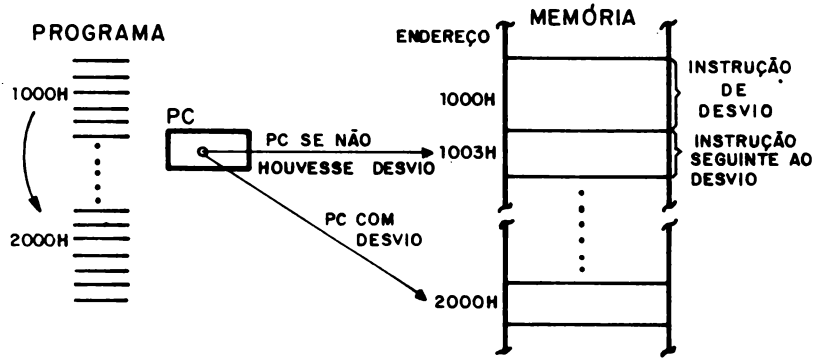


Fig. 13.1 – Operação de Mudança de Seqüência

Daqui para frente usaremos no nosso texto os termos "desvio" e "mudança de seqüência", indistintamente.

As instruções de mudança de seqüência classificam-se quanto a sua forma de atuação em relação aos bits de condição em: *desvios condicionais e incondicionais.*

Os *desvios* são denominados *condicionais* quando a sua realização depende do estado de um dos bits de condição Z, C, P/V ou S. E, *incondicionais* quando o desvio se faz independentemente de qualquer condição. A figura 13.2 ilustra um desvio condicional com base no bit carry.

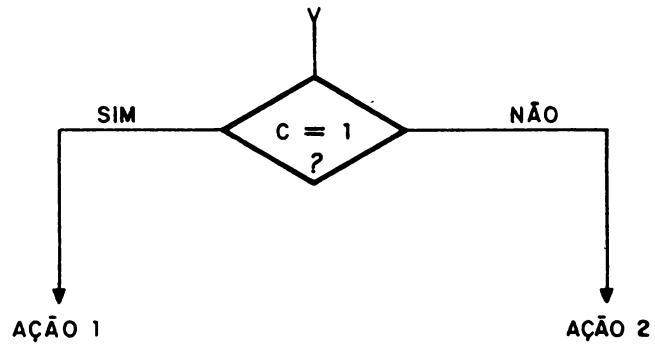


Fig. 13.2 – Desvio Condicional com base no Bit Carry.

A inteligência das máquinas de computação reside na sua capacidade de decisão. E, esta peculiaridade fundamenta-se, basicamente, na ação das instruções que alteram o curso de um programa com base no estado dos bits de condição. A modificação do estado dos bits de condição se faz, na maioria das vezes, através de ação das operações lógicas e aritméticas. As instruções condicionais de desvio e de chamada e retorno de subrotina testam estes resultados e, em função disto, decidem sobre o curso de um programa.

Para fins didáticos classificaremos as instruções de mudança de seqüência, quanto ao seu tipo de endereçamento, nos três subgrupos abaixo:

SUBGRUPO 1 — Desvios com endereçamento estendido

SUBGRUPO 2 — Desvios com endereçamento relativo.

SUBGRUPO 3 — Desvios com endereçamento por registrador.

A tabela 13.1 apresenta as instruções de mudança de seqüência com um resumo sucinto das suas principais características. Esta tabela está dividida nos mesmos subgrupos anteriormente apresentados.

TABELA 13.1 – INSTRUÇÕES DE MUDANÇA DE SEQUÊNCIA – JUMP'S –

Mnemônico 8080/85	Mnemônico Z-80	Operação Simbólica	Bits de Condição C	Z	P/V	S	N	H	Código de Operação Binário 76	543	210	Nº de Bytes	Nº de M ciclos	Nº de T ciclos	SUB-GRUPO
		OPERAÇÕES DE DESVIO COM ENDEREÇAMENTO ESTENDIDO													
JMP end	JP nn	PC ← nn	●	●	●	●	●	●	11	000	011	3	3	10	1
									←	n	→				
									←	n	→				
VEJA A TABELA 13.2	JP cc,nn	Se a condição CC for verdadeira então PC ← nn, em caso contrário continue	●	●	●	●	●	●	11	cc	010	3	3	10	
									←	n	→				
									←	n	→				
		OPERAÇÕES DE DESVIO COM ENDEREÇAMENTO RELATIVO													
–	JR e	PC ← PC + e	●	●	●	●	●	●	00	011	000	2	3	12	
									←	e – 2	→				
–	JR C,e	Se C = 0, continua	●	●	●	●	●	●	00	111	000	2	2	7	
									←	e – 2	→				
		Se C = 1, PC ← PC + e										2	3	12	
–	JR NC,e	Se C = 1, continua	●	●	●	●	●	●	00	110	000	2	2	7	
									←	e – 2	→				
		Se C = 0, PC ← PC + e										2	3	12	2
–	JR Z,e	Se Z = 0 continua	●	●	●	●	●	●	00	101	000	2	2	7	
									←	e – 2	→				
		Se Z = 1, PC ← PC + e										2	3	12	

–	JR NZ,e	Se Z = 1, continua	●	●	●	●	●	●	00 ←	100 e – 2 →	000	2	2	7
		Se Z = 0, PC ← PC + e										2	3	12
–	DJNZ,e	B ← B – 1 Se B = 0, continua	●	●	●	●	●	●	00 ←	010 e – 2 →	000	2	2	8
		Se B ≠ 0, PC ← PC + e										2	3	13

OPERAÇÕES DE DESVIO COM ENDEREÇAMENTO POR REGISTRADOR

PCHL	JP (HL)	PC ← HL	●	●	●	●	●	●	11	101	001	1	1	4
	JP (IX)	PC ← IX	●	●	●	●	●	●	11	011	101	2	2	8
									11	101	001			
–	JP (IY)	PC ← IY	●	●	●	●	●	●	11	111	101	2	2	8
									11	101	001			

3

NOTAS:

1 – Para se obter os códigos de máquina binário das instruções JP cc,nn, basta substituir "cc" da seguinte forma: NZ(000), Z(001), NC(010), C(011), PO(100), PE(101), P(110) e M(111).

2 – As instruções condicionais com endereçamento relativo têm a duração de 7 estados se não houver o desvio e 12 estados se o mesmo ocorrer, exceto para a instrução DJNZ, que têm a duração de 8 estados se B = 0 e 13 estados se B = 1.

3 – Nas instruções com endereçamento relativo, "e" representa o deslocamento sendo um número na representação 2'S. A página móvel varia de – 126 a + 129 bytes em relação ao primeiro byte de instrução, uma vez que o PC fica incrementado de 2 quando o valor do deslocamento é adicionado.

4 – Notação dos bits de condição: ● = não afetado.

Tabela 13.1 – Instruções de Mudança de Seqüência

13.3 – SUBGRUPO 1 – DESVIOS COM ENDEREÇAMENTO ESTENDIDO

De um modo geral, as instruções de desvio com endereçamento estendido transferem o endereço especificado no seu campo de operando para o contador de programa. É neste endereço que é apanhada a próxima instrução.

Na linguagem assembly do Z-80, as instruções deste subgrupo são JP nn e JP cc,nn. O mnemônico "JP" tem a sua origem na palavra "JUMP". Do operando fazem parte o endereço de desvio "nn" e a condição "cc", no caso de desvios condicionais. Na efetivação do desvio, as instruções JP nn e JP cc, nn carregam o endereço "nn" no contador de programa.

Estas instruções existem no *microprocessador 8080,* e se as considerarmos como se fossem simples operações de transferência para o contador de programa podemos, então, classificá-las como endereçamento imediato estendido, onde "nn" é o operando de 16 bits que faz parte do código de instrução.

A instrução JP nn é um desvio incondicional para o endereço "nn", enquanto que JP cc, nn, desvia para o endereço "nn" se a condição "cc" for satisfeita. A tabela 13.2 mostra as instruções JP cc, nn, seus mnemônicos para o 8080 e os bits de condição testados para cada instrução com suas respectivas condições para favorecer a ocorrência do desvio.

MNEMÔNICO DO Z-80	MNEMÔNICO DO 8080	CONDIÇÃO PARA OCORRER O DESVIO
JP NZ, nn	JNZ nn	Z = 0
JP Z, nn	JZ nn	Z = 1
JP NC, nn	JNC nn	C = 0
JP C, nn	JC nn	C = 1
JP PO, nn	JPO nn	P = 0
JP PE, nn	JPE nn	P = 1
JP P, nn	JP nn	S = 0
JP M, nn	JM nn	S = 1

Tabela 13.2 – Instruções JP cc, nn

Note que os bits de condição N e H não podem ser testados. E, o resultado de overflow (bit P/V) pode ser testado através das instruções JP PO, nn e JP PE, nn.

13.3.1 – Exemplo nº 1

O trecho de programa da figura 13.3 mostra uma comparação entre dois números positivos; um no acumulador e outro no registrador B. Se o resultado da comparação for maior ou igual a zero o programa prossegue dobrando o valor do acumulador (ADD A.A). E, se o resultado for negativo o programa prossegue realizando um deslocamento aritmético à direita no acumulador (SRA A).

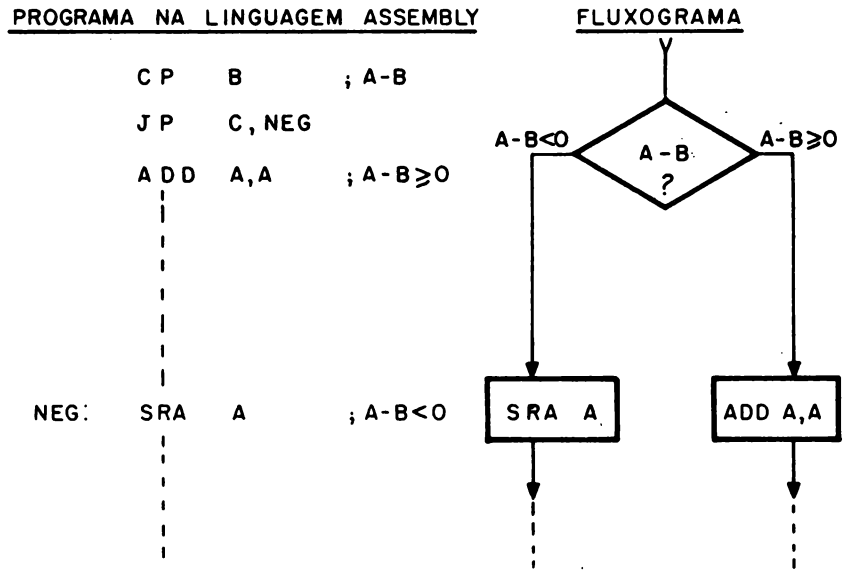


Fig. 13.3 – Comparação de dois Números Positivos .

13.4 – SUBGRUPO 2 – DESVIOS COM ENDEREÇAMENTO RELATIVO

Nas operações de desvio com endereçamento relativo, o segundo byte é chamado de *deslocamento* (ou *"displacement"*) e representa um número algébrico na notação 2's, na faixa de – 128D a + 127D (10000000B a 01111111B). Tal valor adicionado ao valor

do contador de programa gera o endereço efetivo da instrução que deve dar continuidade ao programa.

Tendo em vista que as instruções com endereçamento relativo ocupam dois bytes, o endereço efetivo fica situado na faixa de — 126D a + 129D em relação ao 1º byte do código da instrução.

Na linguagem assembly do Z-80 as instruções deste subgrupo têm mnemônico "JR", que tem a sua origem nas palavras "JUMP" e "RELATIVE", exceto a instrução DJNZ que estudaremos mais adiante.

A instrução "JR e" é um desvio incondicional onde "e" é o seu *deslocamento.*

O Z-80 tem quatro instruções com desvio condicional e endereçamento relativo. Estas instruções só podem testar os bits Z e C. A tabela 13.3 mostra estas instruções com os seus mnemônicos e os bits de condição testados para cada instrução com sua respectiva condição de desvio.

MNEMÔNICO	CONDIÇÃO PARA OCORRER O DESVIO
JR Z, e	Z = 1
JR NZ, e	Z = 0
JR C, e	C = 1
JR NC, e	C = 0

Tabela 13.3 – Instruções JR Condicionais.

Como se infere da tabela 13.1 as instruções JR condicionais possuem dois tempos diferentes de execução. Se o teste for realizado com sucesso estas instruções têm a duração de 12 estados. E, se não houver sucesso no teste a duração é de 7 estados.

Na elaboração de um programa o programador tem que fazer a escolha entre desvios com endereçamento estendido ou relativo, que é sempre uma solução de compromissos. De um modo geral, deve-se optar pelas instruções com endereçamento relativo, desde

que os seus endereços efetivos de desvios recaiam no escopo da instrução. Isto porque os códigos das instruções JR utilizam dois bytes, enquanto que as instruções JP utilizam três bytes, apesar destas serem mais velozes (exceto quando uma condição não for satisfeita).

Via de regra, os montadores do Z-80 fornecem uma indicação na sua listagem, se o escopo de uma instrução de desvio relativo for ultrapassado. Neste caso, o programador tem que substituir a instrução JR por JP. No entanto, as estatísticas mostram que na maioria dos casos é possível o uso do endereçamento relativo.

13.4.1 – Exemplo nº 2

A figura 13.4 mostra um exemplo com a instrução "JR NZ,e" em que o bit 3 do acumulador é testado. Se o bit 3 for igual a 1 o programa prossegue no endereço PROS. Observe que o valor do segundo byte da instrução JR NZ, PROS na memória é a diferença entre o endereço do primeiro byte da instrução OR B e o primeiro byte da instrução JR NZ, PROS, somado de duas unidades.

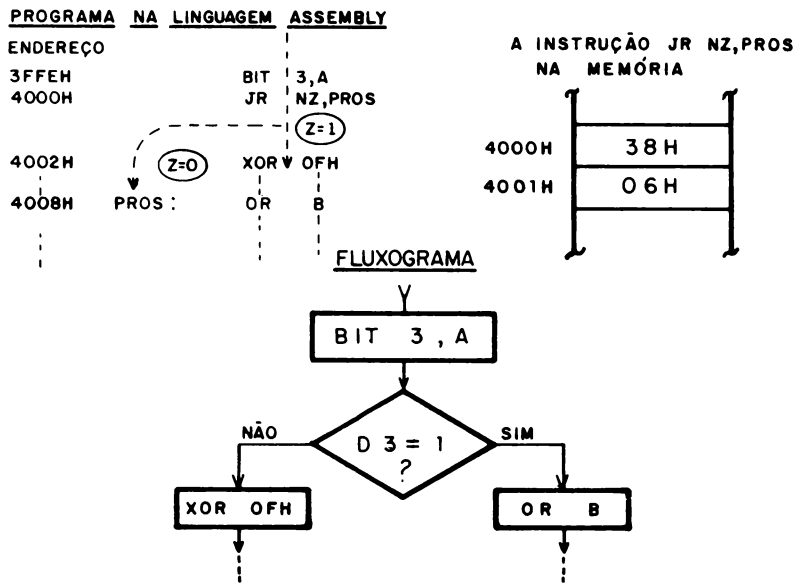


Fig. 13.4 – Exemplo de Desvio Relativo

A instrução "DJNZ, e" também utiliza endereçamento relativo e tem dois bytes. Esta instrução decrementa o conteudo do registrador B de uma unidade. Se o resultado desta operação for diferente de zero é realizado o desvio para o endereço PC + e. E, se o resultado for zero é realizada a instrução seguinte na seqüência normal do contador de programa. Esta instrução também tem dois tempos de execução: 8 estados se B = 0 e 13 estados se B ≠ 0.

13.4.2 – Exemplo nº 3

A instrução DJNZ, LOOP substitui o trecho de programa mostrado na figura 13.5.

```
                .
                .
                .
LOOP:   DEC     B               ; DECREMENTA B
        JR      NZ, LOOP:       ; DESVIA PARA LOOP SE B ≠ 0
```

Fig. 13.5 – Programa equivalente a DJNZ, LOOP

13.5 – SUBGRUPO 3 – DESVIOS COM ENDEREÇAMENTO POR REGISTRADOR

O Z-80 possui três instruções de desvio incondicional com endereçamento por registrador: JP (HL), JP (IX) e JP (IY). Nestas instruções, o desvio se faz pelo carregamento do contador de programa com o conteúdo dos registradores HL, IX ou IY, respectivamente.

As instruções deste subgrupo têm o mnemônico "JP" e o operando é o registrador de 16 bits entre parênteses. O conteúdo deste registrador é o endereço de desvio.

A instrução JP (HL) tem 1 byte e 4 estados, e existe no repertório do 8080. As instruções JP (IX) e JP (IY) ocupam 2 bytes, têm a duração de 8 estados e não existem no 8080.

13.5.1 -- Exemplo nº 4

A instrução JP (IX) está armazenada nas posições de memória 134AH e 134BH e antes da sua execução temos IX = 145AH. Então, após a sua execução o contador de programa assume o valor 145AH.

As instruções de desvio com endereçamento por registrador mostram-se muito úteis em situações de múltiplas opções de desvio, em que a partir de um ponto do programa existem vários destinos (mais do que dois) para onde o desvio pode ser efetivado.

13.5.2 – Aplicação nº 1

Suponha que em um programa, a partir do endereço 2000H da memória, temos uma série de endereços contíguos de outros programas, que denominamos *"tabela de endereços"*, conforme mostra a figura 13.6.

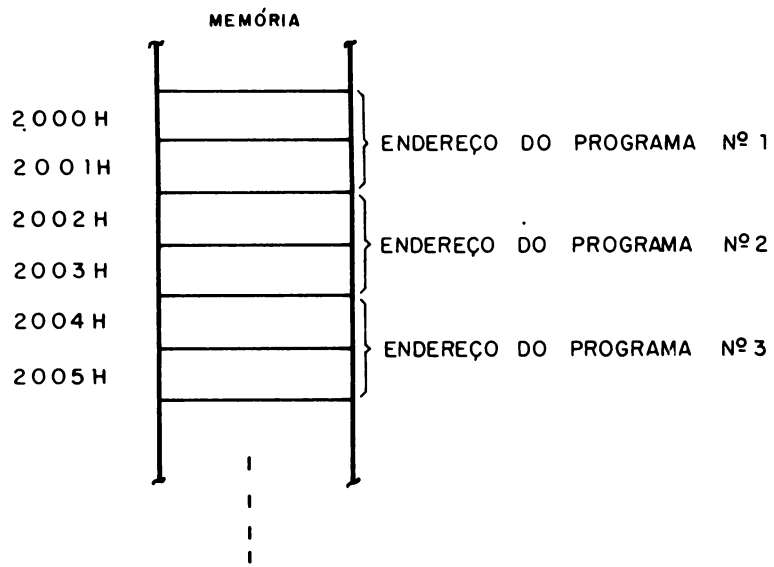


Fig. 13.6 – Tabela de Endereços

No programa da figura 13.7 a ordem de entrada na "tabela de endereços" é passada no acumulador e a transferência para o programa se faz através da instrução JP (HL).

```
                .
                .
                .
ADD     A, A            ; DOBRA O VALOR DO ACC
LD      HL, 2000H       ; CARREGA HL COM O ENDEREÇO
                        ; INICIAL DA TABELA DE END

; DETERMINA O ENDEREÇO DE ENTRADA NA TABELA

ADD     A, L
LD      L, A
LD      A, H
ADC     A, 00
LD      H, A            ; ATUALIZA HL
LD      A, (HL)         ; APANHA A PARTE MENOS
                        ; SIGNIFICATIVA DO ENDEREÇO
INC     HL
LD      H, (HL)         ; APANHA A PARTE MAIS
                        ; SIGNIFICATIVA
LD      L, A            ; COLOCA PARTE MENOS
                        ; SIGNIFICATIVA
                        ; EM L
JP      (HL)            ; DESVIA
```

Fig. 13.7 – Acesso à Tabela de Endereços.

13.6 – EXERCÍCIOS DE FIXAÇÃO

13.6.1 – Escreva um programa que realize a transferência de 255 bytes na memória. O endereço inicial de origem é 2000H e o inicial de destino é 3000H.

13.6.2 – Monte o código de máquina do exercício anterior considerando a origem em 0100H.

13.6.3 – Compare as soluções do exercício 13.6.1 usando instruções de desvio com endereçamento estendido e com o endereçamento relativo. Monte o código de máquina para cada solução.

13.6.4 – Faça um programa que escreva 00 em todas as posições de memória compreendidas entre 1000H e 1FFFH.

13.6.5 – Escreva um programa que escreva FFH e 00H alternadamente nas posições de memória compreendidas entre 1030H e 10FFH.

13.6.6 – Seja uma área de memória compreendida entre as posições 200H e 20FFH. Determine (reg. C) o nº de posições cujo conteúdo tenha o bit 7 igual a zero. Monte o código de máquina com endereço inicial em 0100H.

13.6.7 – Seja uma área de memória compreendida entre as posições 2000H e 2FFEH. Determine (reg. C) o nº de posições de memória com paridade par.

13.6.8 – Faça um programa que determine o endereço da primeira posição de memória compreendida entre 8000H e 80FEH cujo conteúdo seja ímpar (bit 0 = 1).

13.6.9 – Sejam dois números binários de 4 bytes cada um armazenados na memória com endereços iniciais 1000H e 1010H menos significativo encontra-se no endereço inicial. Faça um programa que some estes dois nºs armazenando o resultado a partir do endereço 1020H.

13.6.10 – Considere um nº BCD de 10 dígitos armazenado em 5 bytes na memória a partir do endereço 2000H. O dígito mais significativo ocupa a parte alta da posição 2000H. Escreva um programa que transforme estes 10 dígitos BCD nos correspondentes códigos ASCII. Armazene os 10 bytes em código ASCII a partir do endereço inicial 2005H.

13.6.11 – Antes da execução de uma instrução SUB E, o conteúdo dos registradores A e E representam nºs positivos na representação complemento a 2 sem sinal. Diga se as seguintes afirmações são falsas ou verdadeiras e justifique:

Após a execução da instrução SUB E:

(i) Se (A) $\geqslant$ (E) então CY = 0
(ii) Se (A) $<$ (E) então CY = 1
(iii) Se (A) = (E) então Z = 1
(iv) Se (A) $\neq$ (E) então Z = 0

Explique como você faria para testar estes resultados.

14 INSTRUÇÕES DE CHAMADA DE SUBROTINA E DE RETORNO

14.1 – APRESENTAÇÃO

Iniciaremos este capítulo apresentando conceitualmente a subrotina e, em seguida, passaremos a estudar as instruções que permitem a chamada e retorno de subrotinas e, também, aquelas que possibilitam o retorno das rotinas de tratamento de interrupção.

14.2 – O CONCEITO DE SUBROTINA

Considere uma operação como a multiplicação, para qual o Z-80 não dispõe de uma única instrução capaz de implementá-la. Torna-se, portanto, necessário um grupo de instruções, as quais ao serem executadas em seqüência realizam a operação de multiplicação.

Um grupo de instruções que executa uma tarefa específica é chamada de *rotina.*

Dentro de um programa uma mesma rotina pode ser requisitada várias vezes, o que ocasiona a repetição de trechos idênticos de instruções, provocando um gasto excessivo de memória. A figura 14.1 ilustra uma rotina requisitada três vezes.

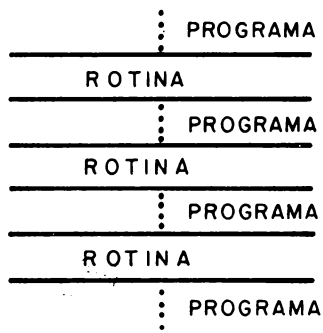


Fig. 14.1 – Uso de Rotina

Uma forma mais eficiente de implementar rotina é armazená-la uma só vez na memória e encontrar uma forma adequada de acessá-la. A figura 14.2 ilustra este mecanismo.

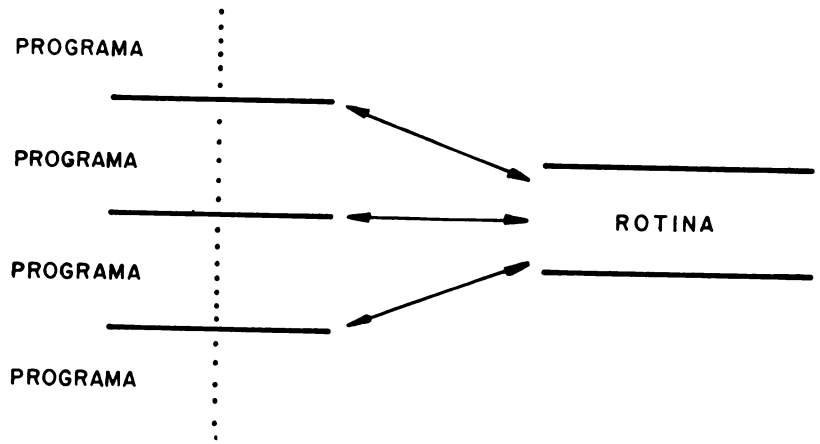


Fig. 14.2 – Rotina armazenada uma só vez na memória.

Uma rotina implementada desta forma é chamada de *subrotina.*

Sempre que for necessária a execução das instruções que constituem a *subrotina,* usa-se uma *instrução de chamada de subrotina – CALL –*, cujo efeito é o de mudar a seqüência normal do programa, desviando-o para o endereço inicial da subrotina e preservando o valor do contador de programa na pilha. Esta operação é denominada *chamada de subrotina.*

Assim a subrotina é executada até a sua última instrução que é obrigatoriamente uma *instrução de retorno – RET* -- sendo também uma instrução de mudança de seqüência. Esta instrução faz com que a execução do programa volte ao ponto imediatamente após àquele no qual a subrotina foi chamada. Isto é conseguido pela transferência do topo da pilha para o contador de programa. A figura 14.3 ilustra a seqüência de eventos na chamada e retorno de subrotina. As setas indicam a seqüência de execução do programa.

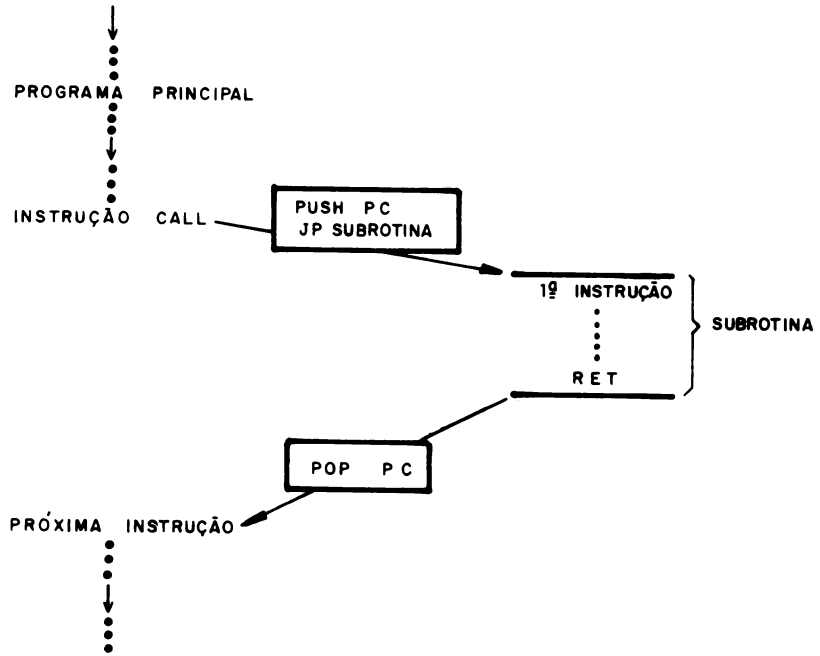


Fig. 14.3 – Chamada e Retorno de Subrotina

Quando uma instrução CALL é executada, o endereço da próxima instrução, ora armazenado no contador de programa, é transferido para o topo da pilha, como se fosse uma operação PUSH PC. A execução do programa é transferida para a primeira instrução da subrotina; pela transferência do seu endereço inicial para o contador de programa, como se fosse uma operação JUMP.

A instrução RET transfere o conteúdo do topo da pilha para o contador de programa, como se fosse uma operação POP PC. Isto faz com que o Z-80 execute a instrução seguinte ao CALL.

A figura 14.4 ilustra a execução de uma subrotina. Observe que o endereço na instrução CALL obedece a mesma convenção aplicada as demais instruções: primeiro a parte menos significativa e depois a parte mais significativa.

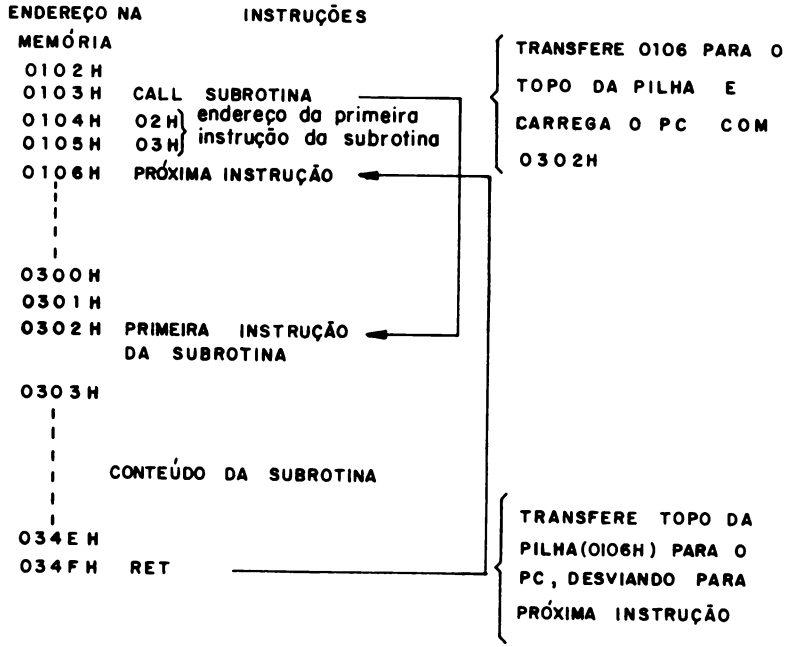


Fig. 14.4 – Execução de uma Subrotina

O repertório de instruções do microprocessador Z-80 conta com instruções de chamada e retorno de subrotina, e estas instruções podem ser *incondicionais* e *condicionais.* Existe, também, uma instrução especial – RESTART –, cuja finalidade principal é a de prover as facilidades necessárias para o processamento de interrupções.

Além da vantagem de economia de memória, as subrotinas facilitam a compreensão dos programas; pela própria redução do número de instruções e pela forma mais organizada que o programa passa a assumir.

Subrotinas podem chamar outras subrotinas, o que denominamos de *aninhamento* ("nesting"). O número máximo de subrotinas aninhadas, em um dado momento, é limitado pela quantidade de memória reservada à pilha.

As subrotinas que chamam a si próprias são denominadas de *subrotinas recursivas.*

14.3 -- GRUPO DE INSTRUÇÕES DE CHAMADA DE SUBROTINA E DE RETORNO

As instruções deste grupo, basicamente, transferem a seqüência de um programa para uma subrotina ou realizam o retorno desta.

Dentro deste grupo também estão incluídas as instruções de retorno de interrupção, cujo comportamento é idêntico ao retorno de subrotinas.

Para fins didáticos classificaremos as instruções deste grupo nos dois subgrupos abaixo:

SUBGRUPO 1 — Chamadas de Subrotinas

SUBGRUPO 2 — Retornos

A tabela 14.1 apresenta as instruções de chamada de subrotina e retornos, com um resumo sucinto das suas principais características. Esta tabela está dividida nos mesmos subgrupos anteriormente apresentados.

TABELA 14.1 – INSTRUÇÕES DE CHAMADA DE SUBROTINA E RETORNOS

Mnemônico 8080/85	Mnemônico Z-80	Operação Simbólica	Bits de Condição C	Z	P/V	S	N	H	Código de Máquina binário 76	543	210	Nº de Bytes	Nº de M ciclos	Nº de T ciclos	SUB-GRUPO
		OPERAÇÕES DE CHAMADA DE SUBROTINAS INCONDICIONAIS													
CALL end	CALL nn	$(SP-1) \leftarrow PC_H$ $(SP-2) \leftarrow PC_L$ $SP \leftarrow SP-2$ $PC \leftarrow nn$	●	●	●	●	●	●	11	001	101	3	5	17	1
									←	n	→				
									←	n	→				
RST const	RST p	$(SP-1) \leftarrow PC_H$ $(SP-2) \leftarrow PC_L$ $SP \leftarrow SP-2$ $PC_H \leftarrow 0$ $PC_L \leftarrow P$	●	●	●	●	●	●	11	t	111	1	3	11	
		OPERAÇÕES DE CHAMADA DE SUBROTINAS CONDICIONAIS													
VEJA A TABELA 14.3	CALL cc, nn	Se a condição CC for falsa continue, caso contrário proceda como o CALL	●	●	●	●	●	●	11	cc	100	3	3	10	
									←	n	→				
									←	n	→	3	5	17	
		OPERAÇÕES DE RETORNO INCONDICIONAIS													
RET	RET	$PC_L \leftarrow (SP)$ $PC_H \leftarrow (SP+1)$ $SP \leftarrow SP+2$	●	●	●	●	●	●	11	001	001	1	3	10	2
–	RETI	Retorno de interrupção	●	●	●	●	●	●	11 01	101 001	101 101	2	4	14	
–	RETN	Retorno de interrupção não mascarável	●	●	●	●	●	●	11 01	101 000	101 101	2	4	14	

OPERAÇÕES DE RETORNO CONDICIONAIS

VEJA A TABELA 14.4	RET cc	Se a condição CC for falsa continue, caso contrário proceda como o RET	●	●	●	●	●	●	11	cc	000	1 1	1 3	5 11

NOTAS:

1 — Para se obter os códigos de máquina binário das instruções RST p basta substituir "t" segundo a tabela abaixo:

t	P
000	00H
001	08H
010	10H
011	18H
100	20H
101	28H
110	30H
111	38H

2 — Para se obter os códigos de máquina binário das instruções CALL cc, nn basta substituir "cc" segundo a tabela abaixo:

cc		CONDIÇÃO
000	NZ	Resultado diferente de zero
001	Z	Resultado igual a zero
010	NC	Não houve carry
011	C	Ocorrência de carry
100	PO	Paridade ímpar
101	PE	Paridade par
110	P	Resultado positivo
111	M	Resultado negativo

3 — As instruções CALL cc, nn têm a duração de 17 estados caso ocorra a chamada de subrotina e 10 estados se não ocorrer a mudança de seqüência no programa.

4 — As instruções RET cc têm a duração de 11 estados caso ocorra o retorno e 5 estados caso não ocorra o retorno.

5 — Notação dos bits de condição: ● = não afetado.

Tabela 14.1 — Instruções de Chamada de Subrotina e Retornos

14.3.1 — SUBGRUPO 1 — Chamadas de Subrotinas

As instruções de chamada de subrotina classificam-se, quanto a sua forma de atuação com base nos bits de condição, em chamadas de subrotinas *condicionais* e *incondicionais.*

As chamadas de subrotinas são classificadas como incondicionais quando a efetivação da chamada da subrotina independe do estado dos bits de condição. As instruções deste subgrupo classificadas como chamadas incondicionais de subrotinas são: CALL nn e RST p.

As chamadas de subrotinas são denominadas condicionais quando a sua realização depende do estado de um dos bits de condição: Z, C, P/V ou S. As instruções classificadas como chamadas condicionais de subrorinas são identificadas por CALL cc, nn.

14.3.1.1 — CALL Incondicional

A instrução CALL nn transfere o conteúdo do contador de programa para a pilha e, em seguida, transfere o endereço especificado no seu campo de operando (nn) para o contador de programa que, na realidade, é o endereço inicial da subrotina.

14.3.1.2 — Exemplo nº 1

A instrução CALL 2000H encontra-se armazenada nas posições de memória 1000H, 1001H e 1002H conforme ilustra a figura 14.5. Então, logo após a sua execução temos o valor 1003H transferido para o topo da pilha e o contador de programa fica com o valor 2000H. Nesta última posição de memória fica armazenada a primeira instrução da subrotina. Finda a execução da subrotina e encontrada a instrução RET, o conteúdo do topo da pilha vai para o contador de programa, ficando, deste modo, com 1003H, que é o endereço da próxima instrução a ser apanhada e executada.

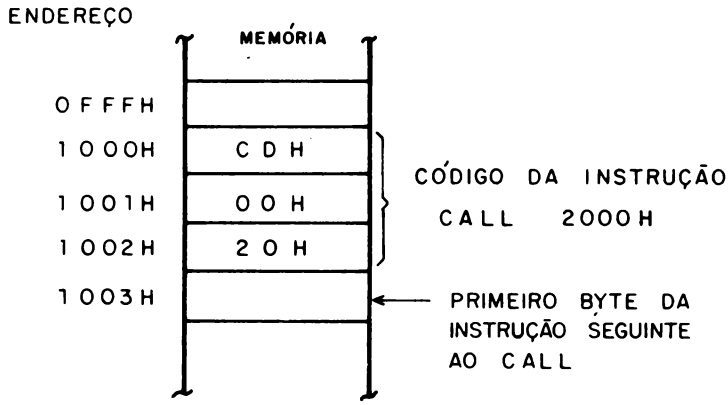


Fig. 14.5 – A Instrução CALL 2000H

14.3.1.3 – Restart

As instruções RST p, que vimos quando estudados o endereçamento página zero modificado, efetuam uma chamada de subrotina de *endereço fixo.* O endereço da instrução que vem imediatamente após a instrução RST p é guardado no topo da pilha, da maneira descrita para a instrução CALL nn. As instruções RST p são de um único byte e compreendem sete espécies distintas, definidas pelo endereço fixo do início da subrotina.

A tabela 14.2 mostra os tipos de instruções RST e o endereço fixo da chamada de cada uma delas. O mnemônico do código de operação é RST, que tem sua origem na palavra "RESTART". O operando é composto da parte menos significativa do endereço fixo da subrotina.

INSTRUÇÃO	ENDEREÇO FIXO DE INÍCIO DA SUBROTINA
RST 00H	0000H
RST 08H	0008H
RST 10H	0010H
RST 18H	0018H
RST 20H	0020H
RST 28H	0028H
RST 30H	0030H
RST 38H	0038H

Tabela 14.2 – Instruções RST p

O conjunto de instruções do microprocessador 8080 também conta com as instruções RST, cuja finalidade, conforme já mencionamos, é a de prover as facilidades necessárias para o processamento de interrupções. As instruções RST p desempenham este mesmo papel peculiar para ò microprocessador Z-80, quando o mesmo estiver operando no Modo 0 de atendimento de interrupções.

Quando o microprocessador Z-80 recebe uma solicitação de interrupção através do seu pino 16 ($\overline{INT}$) e, se não estiver nos estados de HOLD ou WAIT, estiver operando no Modo 0 e, ainda, as interrupções estiverem habilitadas, a interrupção solicitada será aceita e processada. Caso o Z-80 esteja com a execução de alguma instrução em procedimento, ele prossegue na execução da mesma até terminá-la, ativando então os sinais $\overline{M1}$ e $\overline{IORQ}$. Estes sinais indicam que a interrupção foi cientificada e servem para sinalizar ao dispositivo que solicitou a interrupção para introduzir uma instrução RST p diretamente no registrador de instrução do Z-80.

O uso de instruções RST p para indicar a rotina de atendimento da interrupção facilita a implementação do hardware, uma vez que estas instruções têm somente um byte.

Além de sua função específica no atendimento e processamento das interrupções, as instruções do tipo RST p podem ser usadas, eventualmente, para a chamada normal de subrotina, levando-se em conta que tal chamada se dirigirá para os endereços fixos

da tabela 14.2. Embora esse procedimento não seja muito usual, traz a vantagem de economicar espaço na memória, uma vez que as instruções RST p são de um único byte, contra os três que são utilizados em uma instrução normal (CALL) de chamada de subrotina.

É importante acrescentar, também, que a subrotina chamada por uma instrução RSTp deve terminar com uma instrução de retorno – RET –, de forma que o programa volte para o ponto onde a sua seqüência anterior possa ser devidamente retomada.

14.3.1.4 – Exemplo nº 2

A instrução RST 18H está armazenada na posição de memória 0100H. Então, logo após a sua execução temos o valor 0101H transferido para o topo da pilha, que é o endereço da instrução seguinte ao RST 18H na memória. O contador de programa fica carregado com o valor 0018H, onde inicia-se a execução da subrotina chamada. Terminada a execução da subrotina e encontrada a instrução RET, o conteúdo do topo da pilha é transferido para o contador de programa ficando, desse modo, com 0101H.

14.3.1.5 – Call's condicionais

As instruções representadas por CALL cc, nn realizam uma chamada de subrotina de endereço "nn", da maneira descrita para a instrução CALL nn, se a condição "cc" for satisfeita. Caso contrário, após a instrução CALL cc, nn o programa prossegue na instrução seguinte a esta na memória. A figura 14.6 ilustra o funcionamento de uma chamada de subrotina condicional.

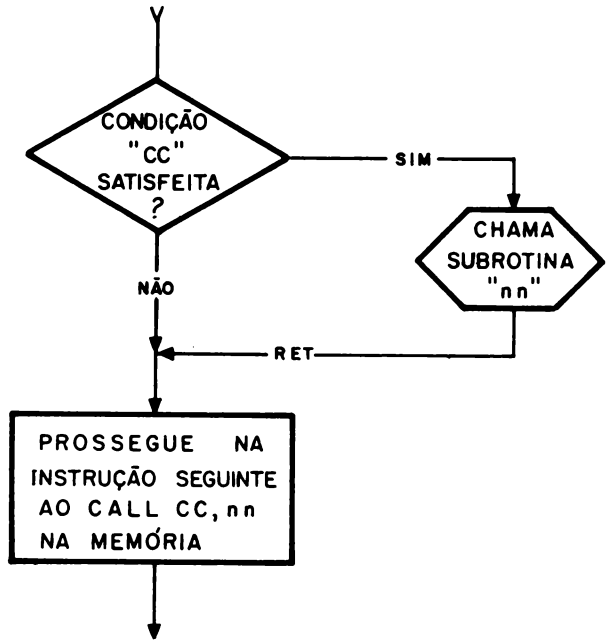


Fig. 14.6 – Chamada Condicional

A tabela 14.3 mostra as instruções CALL cc, nn, seus mnemônicos para o 8080 e os bits de condição testados para cada instrução, com suas respectivas condições para que ocorra a chamada da subrotina.

MNEMÔNICO DO Z-80	MNEMÔNICO DO 8080	CONDIÇÃO PARA OCORRER A CHAMADA DA SUBROTINA
CALL NZ, nn	CNZ nn	Z = 0
CALL Z, nn	CZ nn	Z = 1
CALL NC, nn	CNC nn	C = 0
CALL C, nn	CC nn	C = 1
CALL PO, nn	CPO nn	P = 0
CALL PE, nn	CPE nn	P = 1
CALL P, nn	CP nn	S = 0
CALL M, nn	CM nn	S = 1

Tabela 14.3 – Instruções CALL cc, nn

É importante ressaltar que os bits de condição N e H não podem ser testados. O overflow é testado através das instruções CALL PO, nn e CALL PE, nn.

14.3.1.6 -- Exemplo nº 3

O trecho de programa da figura 14.7 ilustra o teste do bit 4 do registrador B. Se o bit 4 for igual a 1 a subrotina SET é chamada, e se o bit 4 for igual a zero, o programa prossegue na instrução SRL B.

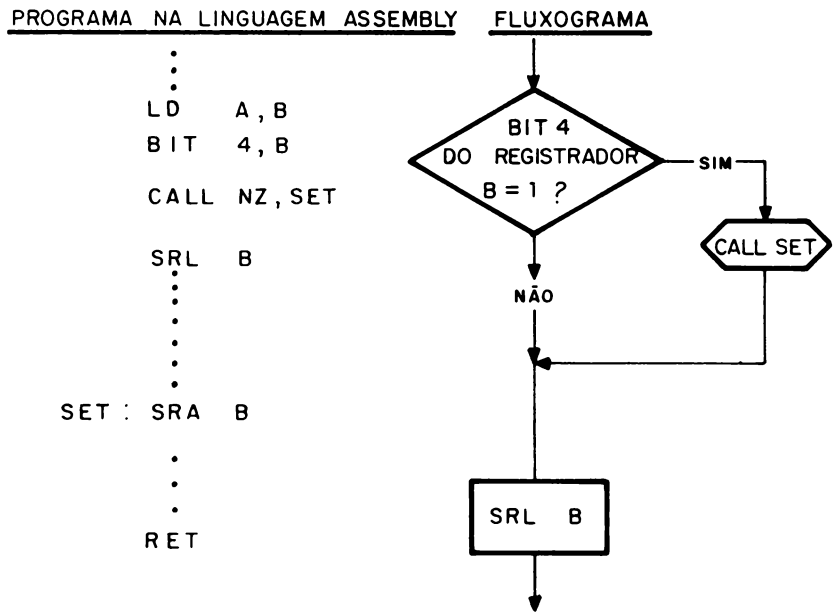


Fig. 14.7 – Teste do bit 4 do Registrador B

14.3.2 – SUBGRUPO 2 – Retornos

Da mesma forma que as instruções de chamada de subrotina, as instruções de retorno classificam-se, quanto a sua forma de atuação com base nos bits de condição, em retornos de subrotina *condicionais e incondicionais.*

Neste subgrupo estudaremos, também, os retornos de interrupção, RET I e RETN, que são classificados como incondicionais,

pois as suas consecuções independem do estado dos bits de condição.

14.3.2.1 – Retornos Incondicionais

As instruções de retorno incondicional do Z-80 são as instruções RETI e RETN, que são próprias para o retorno de interrupções, e a instrução RET, que é usada no retorno de subrotina.

A *instrução RETN* além de transferir o conteúdo do topo da pilha para o PC, afeta os flip-flops IFF1 e IFF2, que indicam o *status* do sistema de interrupção.

O flip-flop IFF1 resetado indica que o Z-80 está bloqueado à solicitações de interrupções através de seu pino 16 ($\overline{INT}$). Por outro lado, enquanto IFF1 estiver setado o Z-80 atende tais solicitações.

O flip-flop IFF2 armazena o estado de IFF1 durante o atendimento de interrupção NMI, que é solicitada através do pino 17 ($\overline{NMI}$). Tais interrupções não podem ser bloqueadas.

O atendimento da interrupção NMI tem a peculiaridade de armazenar IFF1 em IFF2 e, além disso, resetar IFF1, bloqueando assim a ocorrência da interrupção INT.

A instrução RETN além de transferir o conteúdo do topo da pilha para o contador de programa, transfere, também, o estado de IFF2 para IFF1, restaurando assim o estado original de IFF1 antes da ocorrência da interrupção NMI.

Se o programa de tratamento da interrupção NMI permitir o atendimento de interrupções INT durante o seu transcurso, basta executar uma instrução EI após o salvamento do contexto.

No Z-80, *a instrução RETI* apenas restaura o valor do contador de programa da pilha. No entanto, a família de periféricos do Z-80 (PIO, CTC. . .) tem a peculiaridade de decodificar o código de operação da instrução RETI diretamente da barra de dados, quando este estiver sendo apanhado da memória. Isto permite que

estes periféricos identifiquem o término do atendimento de interrupções quando estiverem operando no MODO 2 de interrupção do Z-80.

A *instrução de retorno incondicional de subrotina* do Z-80 é a RET, que transfere o conteúdo do topo da pilha para o contador de programa.

Aproveitamos para observar que o programador pode utilizar a pilha durante o processamento de subrotinas. No entanto, no momento da execução da instrução RET o endereço do retorno tem que estar no topo da pilha. Para tanto, qualquer operação de PUSH dentro da subrotina tem que ser compensada com uma operação de POP, antes da execução da instrução.

14.3.2.2. – Exemplo nº 4

Se antes da execução da instrução RET tivermos o endereço 20F6H no topo da pilha, então, após a sua execução teremos este mesmo endereço transferido para o contador de programa. Desse modo, o Z-80 buscará a próxima instrução em 20F6H.

14.3.2.3 – Retornos Condicionais

As instruções RET cc realizam o retorno de subrotina, da forma descrita para a instrução RET se a condição "cc" for satisfeita. Caso contrário, o retorno da subrotina não se realiza e o programa prossegue na instrução seguinte a esta na memória. A figura 14.8 ilustra o funcionamento da instrução RET cc.

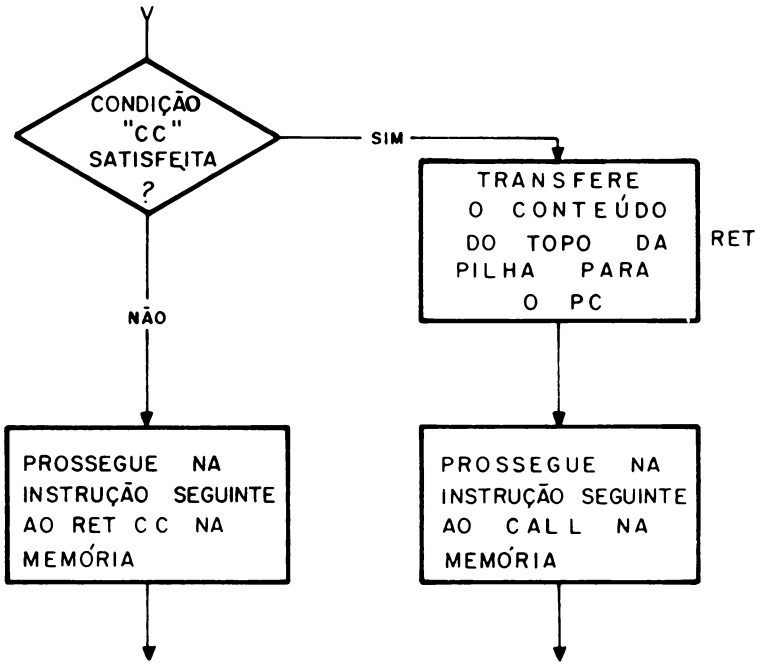


Fig. 14.8 – Retorno Condicional

A tabela 14.4 mostra as instruções RET cc, seus mnemônicos para o 8080 e os bits de condição testados para cada instrução com suas respectivas condições para que ocorra o retorno.

MNEMÔNICO DO Z-80	MNEMÔNICO DO 8080	CONDIÇÃO PARA OCORRER O RETORNO DE SUBROTINA
RET NZ	RNZ	Z = 0
RET Z	RZ	Z = 1
RET NC	RNC	C = 0
RET C	RC	C = 1
RET PO	RPO	P = 0
RET PE	RPE	P = 1
RET P	RP	S = 0
RET M	RM	S = 1

Tabela 14.4 – Instruções RET cc

Da mesma forma que nas instruções CALL cc, nn, os bits de condição N e H não podem ser testados pelas instruções RET cc. Por outro lado, o overflow é testado através das instruções RET PO e RET PE.

14.4 – APLICAÇÃO Nº 1

A subrotina da figura 14.9 incrementa de uma unidade um número de 16 bits armazenado em dois bytes consecutivos de memória. O endereço da parte menos significativa do número é passado à subrotina em HL, e este opera como ponteiro.

```
INCRE:  INC     (HL)        ; INCREMENTA PARTE MENOS SIGNIF.
        RET     NZ          ; SE NÃO HOUVE CARRY, RETORNE
        INC     HL          ; INCREMENTA PONTEIRO
        INC     (HL)        ; INCREMENTA PARTE MAIS SIGNIF.
        RET                 ; RETORNA
```

Fig. 14.9 – Aplicação nº 1

Observe que o teste do carry da parte menos significativa se faz pelo teste da nulidade do resultado, uma vez que as instruções de incremento não afetam o bit-carry.

14.5 – APLICAÇÃO Nº 2

A subrotina da figura 4.10 troca os conteúdos dos pares BC e DE entre si, utilizando o registrador L. No entanto, não se deseja que o conteúdo do registrador L seja alterado por esta subrotina. Para tanto, o conteúdo de HL é salvo na pilha antes do início da subrotina pela instrução PUSH HL e restaurado no final da subrotina pela instrução POP HL. Observe que esta última instrução também compensa a pilha, fazendo com que o endereço de retorno volte ao seu topo.

```
TROCA:  PUSH  HL      ; SALVA HL
        LD    L, D    ; TROCA B e D
        LD    D, B
        LD    B, L
        LD    L, E    ; TROCA C e E
        LD    E, C
        LD    C, L
        POP   HL      ; RESTAURA HL
        RET           ; RETORNA
```

Fig. 14.10 – Aplicação nº 2

14.6 – EXERCÍCIOS DE FIXAÇÃO

14.6.1 – Explique como é a seqüência de chamada de uma subrotina.

14.6.2 – Explique o mecanismo de funcionamento das subrotinas, e cite duas vantagens do seu uso.

14.6.3 – Utilizando as instruções de deslocamento, faça uma subrotina que teste o bit 4 do registrador C. Se o bit 4 = 1 então retorne com CY = 0, caso contrário retorne com CY = 1.

14.6.4 – Faça uma subrotina que troque o conteúdo do par BC com conteúdo do topo da pilha.

14.6.5 – Faça uma subrotina que transfira o conteúdo do topo da pilha para as posições de memória 8F00H e 8F01H.

14.6.6 – Faça uma subrotina que teste se os bits 5 e 6 da posição de memória 801DH são iguais a 1. Se os bits 5 e 6 forem iguais a 1 então retorne com CY = 0, caso contrário retorne com CY = 1.

14.6.7 – Faça uma subrotina que compare os conteúdos dos registradores E e L. Assuma que seus conteúdos representam números positivos na notação 2's sem o sinal.

a) Se E > L, então sete o carry bit
b) Se E < L, então resete o carry bit
c) Se E = L, então carregue FFH no acumulador.

14.6.8 — Escreva uma subrotina que determine o complemento a dois de conteúdo do par HL.

14.6.9 — Faça uma subrotina que transforme o conteúdo do registrador C em dois bytes correspondentes aos códigos ASCII das partes alta e baixa deste registrador. Armazene o resultado em DE.

14.6.10 — Escreva uma subrotina que ordene em ordem crescente os valores contidos em uma área de memória definida pelos endereços fornecidos nos pares HL e DE. O par HL contém o endereço inicial.

14.6.11 — Seja uma área de memória definida pelos endereços fornecidos via HL e DE. Admita que os conteúdos destas posições estão representados na notação 2's sem sinal (0 até 255). Escreva uma subrotina que forneça no registrador C o maior valor armazenado nesta área.

14.6.12 — Faça uma subrotina que determine a paridade do conteúdo do par HL. Forneça o resultado no bit P.

14.6.13 — Escreva uma subrotina que compare o conteúdo dos pares DE e HL. Se HL = DE então B = FF, caso contrário faça B = 00.

14.6.14 — Seja uma área de memória definida pelos endereços fornecidos pelos pares HL e DE. Faça uma subrotina que determine a soma em módulo 8 ("check sum") dos bytes desta área, armazenando o resultado em B.

14.6.15 — Escreva uma subrotina que implemente um deslocamento circular à esquerda no número formado pelos registradores DEHL. O bit mais significativo do D deve estar armazenado no bit carry ao final da subrotina.

14.6.16 — Faça os exercícios anteriores com endereçamento relativo e com endereçamento estendido e compare o nº de bytes montando o código de máquina com endereço inicial 000H.

15 INSTRUÇÕES DE TRANSFERÊNCIA DE BLOCOS E DE PESQUISA

15.1 – APRESENTAÇÃO

No capítulo anterior estudamos as instruções de chamada de subrotina e de retorno.

Agora apresentaremos as instruções que realizam transferência de blocos de memória e, também, pesquisam um determinado byte em uma área de memória.

15.2 – GRUPO DE INSTRUÇÕES DE TRANSFERÊNCIA DE BLOCOS

As instruções de transferência de blocos são quatro: LDI, LDIR, LDD e LDDR. Estas instruções usam os pares de registradores BC, DE e HL na consecução de suas operações e podem transferir blocos *contínuos* de 1 a 64K bytes entre posições diferentes na memória.

O par BC tem que ser previamente carregado com o número de bytes envolvidos na transferência. O par HL opera como *ponteiro* da área de *memória de origem* e tem que ser previamente carregado com o endereço da posição de memória de onde vai ser retirado o primeiro byte. O par DE opera como *ponteiro* da área de *memória de destino* e tem que ser previamente carregado com o endereço da posição de memória que vai receber o primeiro byte na transferência.

A tabela 15.1 apresenta as instruções de *transferência de blocos,* com um resumo sucinto das suas principais características.

TABELA 15.1 – INSTRUÇÕES DE TRANSFERÊNCIA DE BLOCOS

Mnemônico		Operação Simbólica	Bits de Condição						Código de Operação binário			Nº de Bytes	Nº de M ciclos	Nº de T ciclos
8080/85	Z-80		C	Z	P/V	S	N	H	76	543	210			
–	LDI	(DE) ← (HL) DE ← DE + 1 HL ← HL + 1 BC ← BC – 1	●	●	(1) ↕	●	0	0	11 10	101 100	101 000	2	4	16
–	LDIR	(DE) ← (HL) DE ← DE + 1 HL ← HL + 1 BC ← BC – 1 Repetir até BC = 0	●	●	0	●	0	0	11 10	101 110	101 000	2 2	5 4	21 16
–	LDD	(DE) ← (HL) DE ← DE – 1 HL ← HL – 1 BC ← BC – 1	●	●	(1) ↕	●	0	0	11 10	101 101	101 000	2	4	16
–	LDDR	(DE) ← (HL) DE ← DE – 1 HL ← HL – 1 BC ← BC – 1 Repetir até BC = 0	●	●	0	●	0	0	11 10	101 111	101 000	2 2	5 4	21 16

NOTA:

(1) Se BC-1 = 0, então o flag P/V = 0.

Notação dos bits de condição:

● = não afetado.
↕ = afetado de acordo com o resultado da operação BC-1.
o = resetado.

Tabela 15.1 – Instruções de Transferência de Blocos

A instrução LDI – "LOAD and INCREMENT" – procede da seguinte maneira:

1 – Um byte é transferido do bloco de origem para o bloco de destino utilizando-se HL e DE como ponteiros.

2 – Os pares HL e DE são incrementados de uma unidade, endereçando as posições de memória subseqüentes de cada bloco.

3 – O conteúdo do par BC é decrementado de uma unidade. Se este resultado for diferente de zero o bit de condição P/V é setado (P/V = 1).

A instrução LDD – "LOAD and DECREMENT" – realiza as mesmas operações efetuadas pela instrução LDI, exceto que no passo 2 os ponteiros HL e DE são *decrementados* de uma unidade. Assim, a instrução LDI realiza a transferência a partir do menor endereço do bloco, enquanto que a instrução LDD realiza a partir do maior. A atuação das instruções LDD e LDI está esquematizada na figura 15.1.

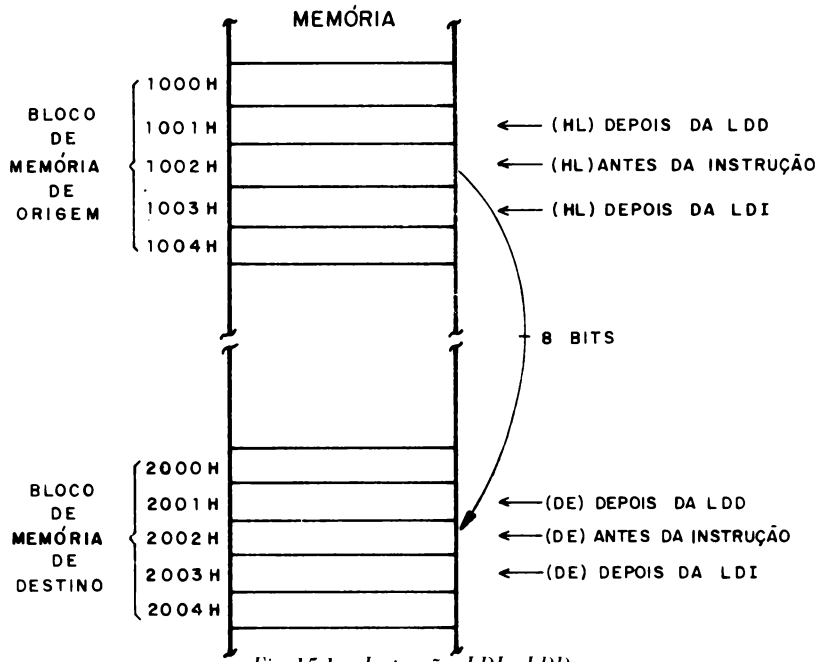


Fig. 15.1 – Instruções LDI e LDD

As instruções LDIR e LDDR além de realizarem as mesma operações das instruções LDI e LDD, também testam o conteúd de BC, que indica o saldo de bytes ainda por transferir. Nest teste, se BC−1 ≠ 0, isto é, P/V = 1, a transferência prossegue automaticamente, decrementando BC de uma unidade e atualizan do os *ponteiros* (HL e DE), até que BC atinja o valor zero.

Isto significa que a instrução LDIR (ou LDDR) uma vez iniciada será executada N vezes, onde N é o valor inicial de BC.

As instruções LDIR e LDDR transferem um bloco de bytes, *automaticamente,* enquanto que as instruções LDI e LDD realizam tais operações de forma *semi-automática,* permitindo o teste do bit de condição P/V após a transferência de cada byte.

A instrução LDIR atualiza os ponteiros incrementando seus conteúdos a cada transferência, enquanto que a LDDR atualiza seus ponteiros decrementando-os. A figura 15.2 ilustra a atuação das instruções LDIR e LDDR.

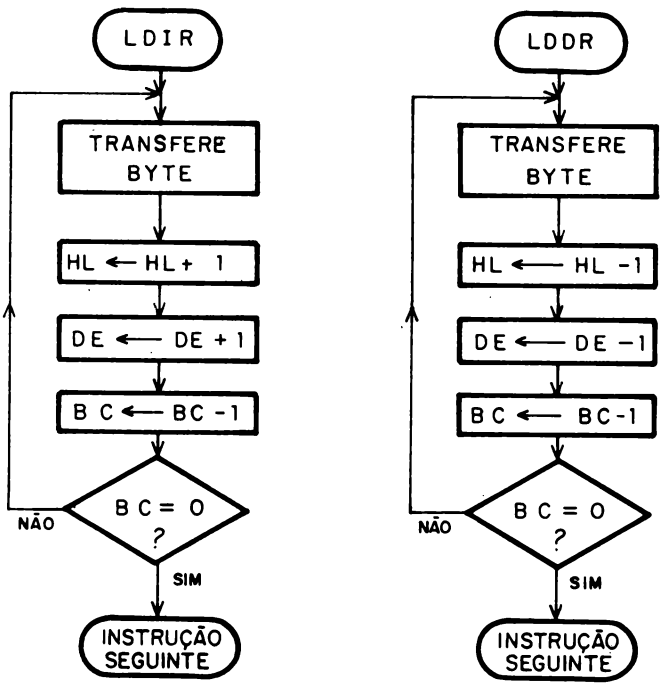


Fig.15.2 – Instruções LDIR e LDDR

15.2.1 — Transferências Automáticas e Semi-automáticas

As instruções de transferência de blocos oferecem a facilidade de mover até 64K bytes de dados automaticamente ou semi-automaticamente. No entanto, o uso destas instruções é uma solução de compromissos.

Via de regra, não é recomendável a transferência de blocos de dados de uma área de memória para outra; a não ser em casos inevitáveis. A principal razão desta recomendação é o tempo dispendido em tais transferências, quando comparado com outras operações de processamento.

Dentre as formas de se evitar movimentos de blocos, recomendamos que *dados de entrada e saída* sejam armazenados em áreas de memória ("buffers") nas quais possam ser processados. O uso de *tabelas* e *listas encadeadas* também evita tais movimentações.

As transferências *automáticas* movimentam o número de bytes especificados em BC, com uma única instrução.

As instruções de transferências *semi-automáticas* transferem apenas um byte cada vez que são executadas, possibilitando, assim, um processamento intermediário entre a transferência de cada byte. Portanto, é necessário utilizar uma instrução de teste condicional para identificar o final da transferência. Isto é, a instrução de transferência semi-automática precisa ser executada tantas vezes quantos forem os bytes a serem transferidos.

Podemos citar duas aplicações para a utilização das instruções de transferências semi-automáticas. A primeira é no caso de transferências que devem começar ou terminar quando for detectado um certo código, ou seja, até a localização de uma *chave.* Uma segunda aplicação é no caso de transferências de áreas descontínuas de memória.

Quando a transferência de um bloco de bytes não necessita de processamento intermediário, deve ser utilizada uma instrução que realiza a transferência *automaticamente.*

15.2.2. – Aplicação nº 1

A aplicação nº 1 mostrada na figura 15.3, apresenta um trecho de programa que testa cada byte antes da sua transferência. Se for encontrado um valor nulo a transferência é encerrada.

Primeiramente, os ponteiros e o número máximo de bytes são iniciados, em seguida, começa o processo de transferência.

```
                .
                .
                .
        LD      HL, 5000H   ; PONTEIRO DE ORIGEM
        LD      DE, 7000H   ; PONTEIRO DE DESTINO
        LD      BC, 100D    ; NÚMERO DE BYTES
PRÓXIMO: LDI                ; TRANSFERE BYTE
        JP      PO, FIM     ; NÚMERO MÁXIMO?
        LD      A, (HL)     ; TESTE DO PRÓXIMO BYTE
        OR      A
        JP      NZ,
                PRÓXIMO     ; BYTE ZERO?
FIM:
```

Fig. 15.3 – Aplicação nº 1

Neste exemplo, utilizamos duas instruções de desvio condicional: JP PO, FIM e JP NZ, PRÓXIMO.

A instrução JP PO, FIM testa o bit P/V, que após a execução da instrução LDI indica BC-1 $\neq$ 0. Caso contrário, se P/V = 0 significa que os 100D bytes do bloco foram transferidos e o programa prossegue no endereço FIM. Se P/V = 1 significa que todos os bytes do bloco ainda não foram transferidos e o programa testa se o próximo byte é nulo.

O teste do próximo byte se faz com o "ou lógico" do acumulador com ele mesmo, através da instrução OR A. Se o resultado desta operação for zero indica que o próximo byte da transferência é nulo.

A instrução JP NZ, PRÓXIMO testa o bit de condição Z após a instrução OR A. Se Z = 1 significa que o próximo byte é nulo e o programa prossegue no endereço FIM. Se Z = 0 significa que o próximo byte não é nulo e o programa prossegue no endereço PRÓXIMO.

15.3 – GRUPO DE INSTRUÇÕES DE PESQUISA

As instruções de pesquisa são quatro: CPI, CPIR, CPD e CPDR. Estas instruções pesquisam os conteúdos de posições consecutivas de memória, comparando-os com o conteúdo do acumulador.

O conteúdo do acumulador é denominado de *chave de pesquisa* ("search key"). Como no grupo de transferência de blocos, o par BC armazena a dimensão do bloco e o par HL opera como ponteiro. No caso das instruções CPI e CPIR ele armazena o menor endereço do bloco, enquanto que para as instruções CPD e CPDR armazena o maior endereço.

A tabela 15.2 apresenta as *instruções de pesquisa,* com um resumo sucinto das suas principais características.

TABELA 15.2 – INSTRUÇÕES DE PESQUISA

Mnemônico		Operação Simbólica	Bits de Condição						Código de Operação binário			Nº de Bytes	Nº de M ciclos	Nº de T ciclos
8080/85	Z-80		C	Z	P/V	S	N	H	76	543	210			
–	CPI	A – (HL) HL ← HL + 1 BC ← BC – 1	•	(2) ↕	(1) ↕	↕	1	↕	11 10	101 100	101 001	2	4	16
–	CPIR	A – (HL) HL ← HL + 1 BC ← BC – 1 Repetir até A = (HL) ou BC = 0	•	(2) ↕	(1) ↕	↕	1	↕	11 10	101 110	101 001	2 2	5 4	21 16
–	CPD	A – (HL) HL ← HL – 1 BC ← BC – 1	•	(2) ↕	(1) ↕	↕	1	↕	11 10	101 101	101 001	2	4	16
–	CPDR	A – (HL) HL ← HL – 1 BC ← BC – 1 Repetir até A = (HL) ou BC = 0	•	(2) ↕	(1) ↕	↕	1	↕	11 10	101 111	101 001	2 2	5 4	21 16

NOTAS:

(1) se BC-1 ≠ 0, então flag P/V = 0.
(2) se A = (HL), então flag Z = 1.

Notação dos bits de condição:

• = não afetados.
↕ = afetado de acordo com o resultado das operações BC-1 e A-(HL).

Tabela 15.2 – Instruções de Pesquisa

A *instrução CPI* compara o conteúdo da posição de memória endereçada pelo par HL com o acumulador. Se forem iguais, o bit de condição Z é setado e, caso contrário, ele é resetado. Os conteúdos dos pares BC e HL são, respectivamente, decrementados e incrementados de uma unidade, endereçando a próxima posição de memória a ser testada.

A *instrução CPD* opera de maneira semelhante a instrução CPI, com a diferença de que o par HL é decrementado de uma unidade a cada teste.

As instruções CPI e CPD pesquisam o bloco de memória de forma *semi-automática,* uma vez que para determinar se a *chave de pesquisa* foi encontrada, o bit de condição Z tem que ser testado toda vez que estas instruções forem executadas.

As *instruções CPIR e CPDR* são similares às instruções CPI e CPD, exceto que as primeiras são totalmente *automáticas.* Se a cada término da execução destas instruções o conteúdo de BC-1 não for igual a zero e o byte de memória testado for diferente da *chave de pesquisa,* uma nova transferência é realizada. Estas instruções são continuamente executadas até que BC-1 atinja o valor zero, o que significa que o bloco foi todo pesquisado, ou quando o conteúdo da posição de memória pesquisada coincidir com a *chave de pesquisa.*

15.3.1 – Aplicação nº 2

A aplicação nº 2, mostrada na figura 15.4, apresenta um trecho de programa que pesquisa uma seqüência de caracteres ("string") em código ASCII para buscar um caractere EOT, que em hexadecimal é representado por 04H. O string possui 30H caracteres e inicia-se no endereço 1000H. No término da pesquisa, o endereço do caractere EOT fica no par HL.

```
        .
        LD      HL, 1000H       ; PONTEIRO
        LD      BC, 30H         ; NÚMERO DE BYTES
        LD      A, 04H          ; CARACTERE EOT

CHEOT:  CPI                     ; PESQUISA BYTE

        JP      PO, FIM         ; FIM?

        JP      NZ, CHEOT       ; CHEOT?

FIM:
```

Fig. 15.4 – Aplicação nº 2

Neste exemplo, usamos as mesmas instruções de desvio condicional da aplicação nº 1.

A instrução JP PO, FIM testa se todo o bloco foi pesquisado. E, a instrução JP NZ, CHEOT testa se o caractere EOT foi encontrado.

Observe que a aplicação nº 2 poderia ser implementada com a instrução CPIR. Neste caso, as instruções JP não seriam necessárias. A figura 15.5 ilustra esta implementação.

```
        .
        .
        .
LD      HL, 1000H       ; INICIA PONTEIRO
LD      BC, 100D        ; NÚMERO DE BYTES
LD      A, 04H          ; CARACTERE EOT
CPIR                    ; PESQUISA

        .
        .
        .
```

Fig. 15.5 – Aplicação nº 2 com CPIR

Observe que a instrução CPIR reduz o número de bytes do programa e o seu tempo de execução. No entanto, se houver a necessidade de algum processamento entre as pesquisas dos bytes torna-se imprescindível o uso de instrução CPI.

15.4 – EXERCÍCIOS DE FIXAÇÃO

15.4.1 – Observe os exercícios dos capítulos 13 e 14 e tente simplificá-los usando as instruções deste capítulo.

15.4.2 – Usando as instruções anteriormente estudadas implemente através de uma subrotina a instrução LDIR.

15.4.3 – Da mesma maneira que no exercício anterior implemente a instrução LDDR.

15.4.4 – Faça agora uma subrotina com as instruções estudadas anteriormente e implemente a instrução CPIR.

15.4.5 – Como no exercício anterior implemente a instrução CPDR.

15.4.6 – Faça uma subrotina que pesquise 100 bytes consecutivos a partir de 0100H. Se um código 0AH for encontrado coloque o seu endereço em HL e sete o carry-bit.

15.4.7 – Faça uma subrotina que conte o número de vezes que o código 0AH está presente em um bloco de 256 bytes a partir de 1000H.

16 INSTRUÇÕES DE ENTRADA E SAÍDA

16.1 – APRESENTAÇÃO

Neste capítulo completaremos a apresentação das instruções do Z-80, estudando as instruções de entrada e saída de dados.

Estas instruções são as responsáveis pela comunicação do microprocessador com os dispositivos de entrada e saída.

16.2 – INTRODUÇÃO

Para melhor compreendermos as instruções de entrada e saída, faremos uma divisão em dois sub-grupos:

SUBGRUPO 1 – Instruções de Entrada de Dados
SUBGRUPO 2 – Instruções de Saída de Dados

Na tabela 16.1 apresentamos todas as instruções de entrada e saída com suas respectivas características. As instruções estão agrupadas nos dois subgrupos mencionados anteriormente.

TABELA 16.1 – INSTRUÇÕES DE ENTRADA E SAÍDA

Mnemônico 8080/85	Mnemônico Z-80	Operação Simbólica	Bits de Condição C	Z	P/V	S	N	H	Código de Máquina binário 76	543	210	Nº de Bytes	Nº de M ciclos	Nº de T ciclos	SUB-GRUPO
			INSTRUÇÕES DE ENTRADA DE DADOS												
IN n	IN A,(n)	A ← (n)	●	●	●	●	●	●	11	011	011	2	3	11	
		(2)							←	n	→				
–	IN r,(C)	r ← (C)	●	↕	P	↕	0	↕	11	101	101	2	3	12	
									01	r	000				
	INI	(HL) ← (C)	●	(1) ↕	X	X	1	X	11	101	101	2	4	16	
		B ← B – 1							10	100	010				
		HL ← HL + 1													
–	INIR	(HL) ← (C)	●	1	X	X	1	X	11	101	101	2	5	21	
		B ← B – 1							10	110	010		(Se B ≠ 0)		1
		Repetir até B = 0										2	4	16	
													(Se B ≠ 0)		
–	IND	(HL) ← (C)	●	(1) ↕	X	X	1	X	11	101	101	2	4	16	
		B ← B – 1							10	101	010				
		HL ← HL – 1													
–	INDR	(HL) ← (C)	●	1	X	X	1	X	11	101	101	2	5	21	
		B ← B – 1							10	111	010		(Se B ≠ 0)		
		HL ← HL – 1										2	4	16	
		Repetir até B = 0											(Se B = 0)		
			INSTRUÇÕES DE SAÍDA DE DADOS												
OUT n	OUT (n),A	(n) ← A	●	●	●	●	●	●	11	010	011	2	3	11	
									←	n	→				
–	OUT (C),r	(C) ← r	●	●	●	●	●	●	11	101	101	2	3	12	
									01	r	001				

–	OUTI	(C) ← (HL) B ← B – 1 HL ← HL + 1	●	(1) ↕	X	X	1	X	11 10	101 100	101 011	2	4	16
–	OTIR	(C) ← (HL) B ← B – 1 HL ← HL + 1 Repetir até B = 0	●	1	X	X	1	X	11 10	101 110	101 011	2 2	5 (Se B ≠ 0) 4 (Se B = 0)	21 16
–	OUTD	(C) ← (HL) B ← B – 1 HL ← HL – 1	●	(1) ↕	X	X	1	X	11 10	101 101	101 011	2	4	16
–	OTDR	(C) ← (HL) B ← B – 1 HL ← HL – 1 Repetir até B = 0	●	1	X	X	1	X	11 10	101 111	101 011	2 2	5 (Se B ≠ 0) 4 (Se B = 0)	21 16

NOTAS:

1 – Se o resultado de B – 1 = 0, então Z = 1. Caso contrário, Z = 0.
2 – Se r = 110, então somente os bits de condição serão afetados.
3 – r = 000 (B), r = 001 (C), r = 010 (D), r = 011 (E), r = (H), r = 101 (L), r = 111 (A).
4 – Notação dos bits de condição: ● = não afetado, 0 = resetado, 1 = setado.
X = indefinido.
↕ = o bit é afetado em função do resultado da operação.

Tabela 16.1 – Instruções de Entrada e Saída

16.3 – SUBGRUPO 1 – Instruções de Entrada de Dados

Este subgrupo inicia-se com duas instruções – IN A,(n) e IN r,(c) – que realizam, respectivamente, a entrada de dados para o acumulador e para um registrador r de 8 bits.

A primeira é compatível com a instrução de entrada de dados do 8080. Aliás, o 8080 só possui uma instrução de entrada, o que limita a sua performance na comunicação com dispositivos de entrada. Principalmente, porque o modo de endereçamento desta instrução é o direto, não permitindo variações do endereço dentro do processamento.

Na instrução IN A,(n) o endereço da *porta de entrada* é fornecido no byte seguinte ao do código de operação. Por outro lado, na instrução IN r,(c) o endereço da porta é fornecido pelo conteúdo do registrador C.

As *portas de entrada* são dispositivos eletrônicos conectados na barra de dados, sendo *ativadas* (ou *selecionadas*) quando são executadas instruções de entrada, cujo campo de endereço coincide com o endereço da porta. Este último é definido por hardware, e pode ser melhor entendido estudando-se o hardware do Z-80.

Como exemplo, seja uma porta de entrada cujo endereço é OFH. Então a execução da instrução IN A,(OFH) transfere para o acumulador o conteúdo desta porta.

As quatro instruções restantes deste subgrupo – INI, INIR, IND e INDR – são semelhantes àquelas estudadas no grupo de transferência de blocos.

A *origem* é a porta cujo endereço é igual ao conteúdo do registrador C e o *destino* é a posição de memória endereçada pelo par HL.

As instruções INI e IND realizam a transferência *semi-automática,* sendo que a primeira incrementa o par HL e a segunda decrementa-o. Em ambos os casos o registrador B é decrementado de uma unidade.

Conforme já estudamos, estas instruções de transferência semi-automática permitem um processamento entre cada byte a ser transferido.

Nas instruções INIR e INDR temos transferências *automáticas,* onde o registrador B contém o número de bytes a ser transferido. Na primeira instrução (INIR) o par HL é incrementado e na segunda ele é decrementado. Observe que o endereço da porta (C), ao contrário do par HL, não é atualizado após cada transferência. Isto significa, que estas instruções realizam a transferência de n bytes (máx de 255) de uma única porta de entrada para uma área de memória definida pelo par HL.

Voltando à tabela 16.1, podemos observar que na primeira instrução nenhum dos bits de condição é afetado. Na segunda temos, com excessão dos bits C e N, uma alteração dos outros bits de acordo com o byte armazenado no registrador r. Isto facilita alguns testes de dados de entrada, pois dispensa o uso de operações lógicas para o teste destes bits.

Nas instruções INI e IND somente o bit Z varia de acordo com o resultado, pois vai indicar se B−1 = 0.

A instrução IN A,(n) possui um byte para o código de operação e o segundo é a constante n, que é o *endereço direto* da porta de entrada.

A segunda instrução também possui dois bytes, sendo que no segundo os bits 5, 4 e 3 variam de acordo com o registrador r. Quando a combinação destes bits assumir o valor 110B, que não representa nenhum registrador de 8 bits, esta pode ser usada somente para ativar os bits de condição sem afetar nenhum dos registradores.

As instruções INIR e INDR possuem um número total de T-ciclos (estados) que varia de acordo com o conteúdo do registrador B.

16.3.1 – Exemplo nº 1

Na figura 16.1 temos uma chave conectada à entrada D1 da porta OEH. Nada é informado sobre o que está conectado nos outros bits desta porta.

Temos também nesta figura um trecho de programa que armazena no B o valor 00H se a chave estiver fechada. Caso contrário, B assume o valor FFH.

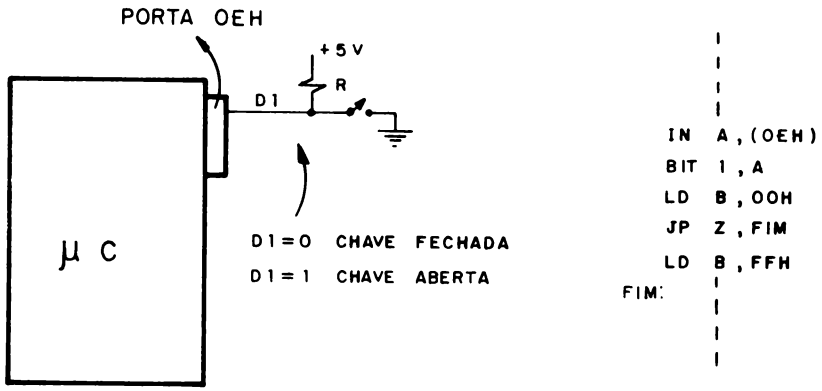


Fig. 16.1 – Teste do Estado de uma Chave

A primeira instrução transfere os oito bits de entrada da porta OEH para o acumulador. Em seguida, realizamos o teste do bit D1 do acumulador, pois nesta configuração do hardware é o único bit com significado para a análise. Após esta instrução (BIT 1,A), usamos instruções de carregamento e de desvio condicional para realizarmos a tarefa desejada.

16.4 – SUBGRUPO 2 – Instruções de Saída de Dados

As *portas de saída* são dispositivos eletrônicos conectados na barra de dados, sendo ativadas quando são executadas instruções de saída, cujo campo de endereço coincide com o endereço da porta, que é definido por hardware.

As instruções deste subgrupo realizam operações opostas àquelas executadas pelas instruções do subgrupo 1.

Agora, a origem da transferência é o acumulador (OUT(n),A), um registrador de 8 bits qualquer (OUT(c),r) ou uma posição de memória cujo endereço é definido pelo par HL(OUTI, OTIR, OUTD e OTDR). O destino é uma *porta de saída,* cujo endereço é um operando imediato (OUT(n),A), ou o conteúdo do registrador C, que é o caso de todas as outras instruções de saída.

A instrução OUT(n),A é compatível com a instrução de saída do 8080, que, aliás, possui somente uma única instrução de saída.

A segunda instrução (OUT(c),r) deste subgrupo transfere o conteúdo de um registrador de 8 bits qualquer para a porta de saída cujo endereço é igual ao conteúdo do registrador C. Como podemos observar, esta instrução é muito mais flexível do que a primeira.

As outras quatro instruções (OUTI, OTIR, OUTD e OTDR) são análogas as instruções INI, INIR, IND e INDR estudadas no subgrupo anterior. Temos as que realizam saídas de dados *semi-automáticas,* que são OUTI e OUTD, e as que realizam saídas de dados *automáticas,* que são OTIR e OTDR.

Os bits de condição não são afetados nas duas primeiras instruções deste subgrupo e, nas instruções OUTI e OUTD somente o bit Z varia de acordo com o resultado de B-1.

A formação do código de máquina obedece aos mesmos princípios já apresentados no subgrupo 1.

16.4.1 – Exemplo nº 2

Na figura 16.2 temos uma porta de saída na qual conectamos oito circuitos que acionam oito LED'S. Os LED'S são diodos emissores de luz, que são ativados quando recebem uma certa corrente elétrica. Comercialmente existem inúmeros tipos de LED'S, nos mais diversos tamanhos e nas mais variadas cores.

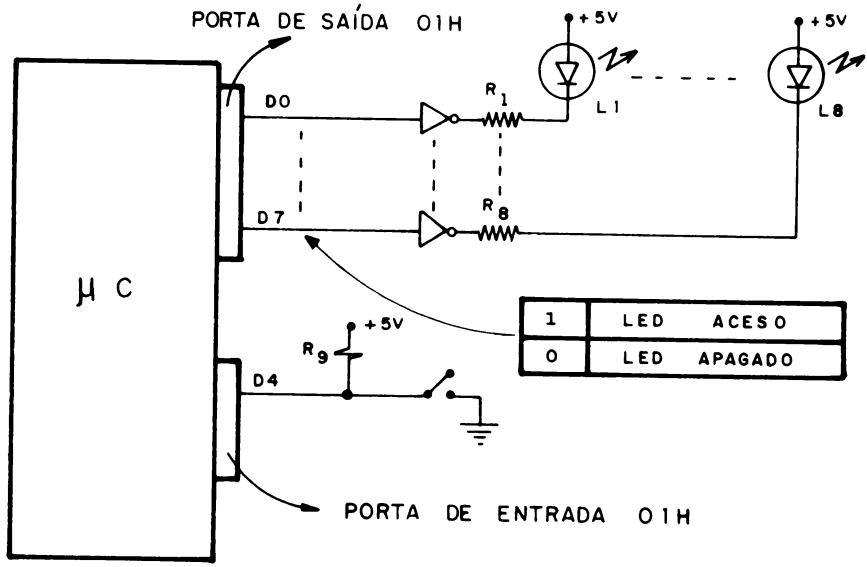


Fig. 16.2 – Exemplo de Acionamento de LED'S

Temos também, uma porta de entrada que contém uma chave. Quando esta chave estiver aberta (OFF) desejamos acender os LED'S 1, 2, 3 e 4, e apagar os restantes. Caso contrário, isto é, com a chave fechada (ON), desejamos apagar os LED's 1, 2, 3 e 4 e acender os restantes.

Na figura 16.3 temos o programa que executa as tarefas desejadas, de acordo com o estado da chave.

```
SAÍDA   EQU   01H
ENTDA   EQU   01H
        ORG   0000H
        IN    A, (ENTDA)    ; LEITURA CHAVE
        BIT   4, A          ; CHAVE ON/OFF?
        JP    Z, CHON       ; ON, DESVIA
        LD    A, OFH
        OUT   (SAÍDA), A    ; ACENDE LED's 1, 2, 3 e 4
        HALT
CHON:   LD    A, FOH
        OUT   (SAÍDA), A    ; ACENDE LED's 5, 6, 7 e 8
        HALT
        END
```

Fig. 16.3 – Programa do exemplo nº 2

Inicialmente é realizada a leitura e o teste da chave de controle. Se Z = 1, significa que a chave está acionada (D4 = 0) e, então, ocorre o desvio definido pela instrução JP Z, CHON. Completando, temos as instruções LD A,OFH e LD A,FOH que preparam o conteúdo do acumulador para ser transferido para a porta de saída.

16.5 – APLICAÇÃO Nº 1

Na figura 16.4 temos a comunicação de um microcomputador com uma leitora de fita de papel.

Temos 8 linhas de dados e duas linhas de controle LP e MP. Estas linhas indicam, respectivamente, leitora e microcomputador prontos.

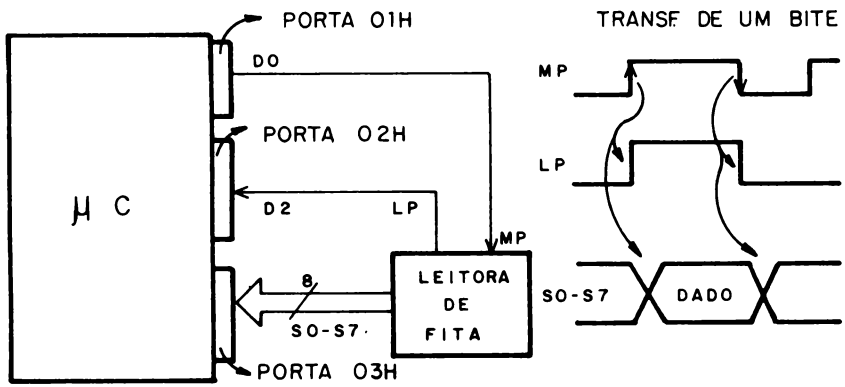


Fig. 16.4 – Interface com Leitora de Fita de Papel

No diagrama de transferência de um byte, tudo começa quando o microcomputador solicita, através de MP = 1, um byte da leitora. Em caso afirmativo, ela sinaliza com LP = 1 e coloca o byte nas saídas SO-S7. Após a leitura deste byte por parte do microcomputador, este recoloca a linha MP em zero (MP = 0). Com isso, a leitora desativa a linha LP (LP = 0) e retira o byte de dados das linhas de saída.

O programa da figura 16.5 armazena 100D bytes na memória, a partir do endereço 1000H.

```
ENCOM     EQU    01H
SAICOM    EQU    02H
ENDADO    EQU    03H
NBYTES    EQU    100D
INMEM     EQU    1000H
          ORG    0000H
          LD     HL, INMEM
          LD     C, ENDADO
          LD     B, NBYTES
LOOP1:    SET    O, A          ; BIT DO = 1
          OUT    (SAICOM), A   ; MP = 1
LOOP2:    IN     A, (ENCOM)
          BIT    2, A          ; LP = 1?
          JP     Z, LOOP2      ; NÃO, VOLTA
          INI                  ; LEITURA
          JP     Z, FIM        ; SE B = 0, FIM
          RES    O, A          ; BIT DO = 0
          OUT    (SAICOM), A   ; MP = 0
          JP     LOOP1
FIM:      HALT                 ; FIM
          END
```

Fig. 16.5 – Programa de Entrada Seqüencial de Dados

Observando a lógica do programa, podemos notar que o diagrama da figura 16.2 foi implementado por software. Inicialmente, a linha MP é ativada (MP = 1) e o LOOP2 é executado até que LP seja igual a 1. Após estes dois eventos (MP = 1 e LP = 1) é realizada a transferência através da instrução INI.

Em seguida a esta transferência, o bit Z é testado. Não sendo igual a 1 (B ≠ 00) o programa prossegue na instrução RES O,A que juntamente com OUT (SAICOM),A desativa a linha MP (MP = 0).

Todos estes eventos são repetidos até que B seja igual a zero e, assim, os 100D bytes são transferidos da leitora de fita de papel para a memória do microcomputador.

16.6 – EXERCÍCIOS DE FIXAÇÃO

16.6.1 – Realize um programa que armazene na posição 1000H de memória o conteúdo da porta de entrada F0H.

16.6.2 – Faça um programa que transfira o conteúdo da porta 02H de entrada para as portas 03H e 04H de saída.

16.6.3 – Admita que na porta 01H de saída temos 8 LED's. Faça um programa que acenda-os seqüencialmente do bit 0 para o bit 7. O intervalo entre dois LED's consecutivos deve ser de aproximadamente 0,5 segundo. Admita um clock de 2,5 MHz.

16.6.4 – Faça um programa que envie seqüencialmente os 8 bits do registrador B para o bit 3 da porta 0FH de saída. Transmita primeiro o bit 0 e por último o bit 7 do registrador B. O tempo de duração de cada bit deve ser igual a aproximadamente 1ms. (Z-80 com 2,5 MHZ).

16.6.5 – Admita que no bit 1 da porta 03H de entrada temos uma chave que possui um tempo de estabilização de 0 → 1 de aproximadamente 1 ms. Faça um programa que resolva este problema de ruído ("bounce") da chave.

17 APLICAÇÕES

17.1 – APRESENTAÇÃO

Reservamos este capítulo para apresentar uma coletânea de subrotinas aplicativas na linguagem ASSEMBLY do Z-80, visando ampliar e consolidar os conhecimentos adquiridos.

Para cada subrotina apresentamos um resumo de suas funções, dos seus parâmetros de entrada e do formato dos seus resultados.

A variedade das aplicações aqui apresentadas tem por objetivo fornecer ao leitor soluções de diferentes tipos de problemas, completando, assim, uma base mínima para o domínio do Sofware do Microprocessador Z-80.

A montagem dos códigos de máquina de cada aplicação deve ser exercitada pelo leitor, pois é extremamente fácil e importante, tanto na manutenção como no desenvolvimento de programas em ASSEMBLY.

17.2 – INCREMENTA NÚMERO NA MEMÓRIA – INCRE

A subrotina INCRE incrementa de uma unidade um número de 16 bits armazenado em dois bytes consecutivos de memória.

Antes da chamada da subrotina INCRE, o endereço da parte menos significativa do número de 16 bits tem que estar armazenado no registrador de índice IX.

Observe que o teste do carry se faz através do bit Z, uma vez que as instruções de incremento não afetam o bit carry.

```
INCRE:   INC    (IX + 0)    ; INCREM. PARTE MENOS SIGN.
         RNZ                 ; SE CARRY = 0 (RETORNA)
         INC    (IX + 1)    ; INCREM. PARTE MAIS SIG.
         RET
```

Fig. 17.1 – Subrotina INCRE

17.3 – MÚLTIPLOS DESLOCAMENTOS – MDESL

A subrotina MDESL realiza o deslocamento de um número de 16 bits armazenado na memória.

A quantidade de deslocamentos é um parâmetro de entrada da subrotina.

Os parâmetros são transferidos à subrotina MDESL através de um *buffer de controle* cujo endereço inicial é passado no registrador de índice IX.

O buffer de controle é composto de três bytes consecutivos de memória, sendo organizado da seguinte forma: O primeiro byte é a parte menos significativa do número de 16 bits a ser deslocado, o segundo byte é a parte mais significativa deste número e o terceiro byte especifica o número de deslocamentos.

O resultado do deslocamento fica armazenado nas posições iniciais de memória.

```
MDESL:   LD     L, (IX + 0)   ; L ← PARTE MENOS SIGNIFICATIVA
         LD     H, (IX + 1)   ; H ← PARTE MAIS SIGNIFICATIVA
         LD     B, (IX + 2)   ; B ← Nº DE DESLOCAMENTOS
MDESO:   SRL    H
         RR     L
         DJNZ   MDESO
         LD     (IX + 0), L
         LD     (IX + 1), H
         RET
```

Fig. 17.2 – Subrotina MDESL

17.4 – SOMA DE MULTIPRECISÃO BINÁRIA – SOMUL

A subrotina SOMUL realiza a soma de dois números binários armazenados na memória em posições consecutivas.

Os parâmetros são passados à subrotina SOMUL através de um buffer de controle cujo endereço inicial precisa ser previamente carregado no registrador de índice IY.

O buffer de controle é organizado da seguinte forma: os primeiros dois bytes contém o endereço do byte menos significativo da primeira parcela; os dois bytes seguintes contém o endereço do byte menos significativo da segunda parcela; e o último byte contém o número de bytes dos números binários constitutivos de ambas as parcelas.

O resultado da operação fica armazenado nas posições de memória reservadas, originalmente, à primeira parcela.

Note que para realizar a subtração de multiprecisão basta substituir a instrução ADC por SBC.

```
SOMUL:  LD    C, (IY + 0)
        LD    B, (IY + 1)   ; BC = END. PRIM. OPERANDO
        LD    L, (IY + 2)
        LD    H, (IY + 3)   ; HL = END. SEG. OPERANDO
        LD    E, (IY + 4)   ; E = NUM. DE BYTES
        XOR   A             ; AC = CY = 0
LOOP:   LD    A, (BC)
        ADC   A, (HL)       ; SOMA PARCELA 8 BITS
        LD    (BC), A
        DEC   E
        RZ                  ; E = 00, RETORNA
        INC   BC
        INC   HL            ; ATUALIZA PONTEIROS
        JR    LOOP
        RET
```

Fig. 17.3 – Subrotina SOMUL

17.5 – SUBTRAÇÃO DE MULTIPRECISÃO BCD – SMBCD

A subrotina SMBCD realiza a subtração de dois números na representação BCD armazenados na memória em posições consecutivas.

Através do par DE é passado o endereço da posição de memória do byte mais significativo do minuendo e, através do par HL o endereço do byte mais significativo do subtraendo. O registrador C contém o número total de dígitos BCD no minuendo e no subtraendo.

O resultado da operação fica armazenado nas posições de memórias reservadas, originalmente, ao minuendo.

Observe que para realizar a soma de multiprecisão com números na representação BCD basta substituir a instrução SBC por ADC.

```
SMBCD:  SRL   C          ; DIVIDE Nº DE DIG. POR 2
        LD    B,00
        DEC   BC         ; ATUALIZA INDICADOR DE POSIÇÃO
        ADD   HL, BC     ; ACESSA SUBTRAENDO
        EX    DE, HL
        ADD   HL, BC     ; ACESSA MINUENDO PELO HL
        EX    DE, HL
        LD    B, C
        INC   B          ; B INDICA NUM. POS. MEM.
        OR    A          ; CARRY = 0
SMBC1:  LD    A, (DE)
        SBC   A, (HL)    ; REALIZA SUBTRAÇÃO
        DAA              ; AJUSTE DECIMAL
        LD    (DE), A
        DEC   HL
        DEC   DE         ; ATUALIZA PONTEIROS
        DJNZ  SMBC1
        RET
```

Fig. 17.4 – Subrotina SMBCD

17.6 – CONVERSÃO BCD → BINÁRIO – BCDBN

A subrotina BCDBN converte números representados em BCD com até 8 dígitos para a representação binária com 24 bits.

Os dígitos em BCD ficam armazenados em bytes consecutivos na memória. Antes da chamada da subrotina BCDBN, o par DE deve estar carregado com endereço do dígito BCD menos significativo e o acumulador com a quantidade de dígitos do número BCD a ser convertido.

Ao término da rotina, o número binário fica armazenado nos registradores A, H e L, sendo que os 8 bits mais significativos ficam no acumulador e os 16 bits menos significativos no par HL.

O algoritmo usado nesta subrotina soma os dígitos BCD com os resultados parciais obtidos em HL e multiplica este total por dez. Observe, também, que a subrotina BCDBN utiliza a subrotina BCD que, basicamente, multiplica por dez.

```
BCDBN:      LD      C, A        ; TRANSFERE NÚMERO DE DÍGITOS BCD
                                ; PARA C
            LD      HL, 0       ; INICIA HL
            XOR     A           ; INICIA A COM 00
LOOP:       CALL    BCD         ; CHAMA SUBROTINA *10
            DEC     DE          ; APONTA O PRÓXIMO DÍGITO BCD
                                ; MENOS SIGNIFICATIVO
            JR      NZ, LOOP
            RET
;SUBROTINA BCD
BCD:        PUSH    DE
            PUSH    HL
            LD      B, A
            ADD     HL, HL
            ADC     A, A        ; TÉRMINO MULT POR 2
            ADD     HL, HL
            ADC     A, A        ; TÉRMINO MULT POR 4
            POP     DE
            ADD     HL, DE
            ADC     A, B        ; TÉRMINO MULT POR 5
            ADD     HL, HL
            ADC     A, A        ; TÉRMINO MULT POR 10
            LD      B, A
            POP     DE
            LD      A, (DE)     ; APANHA NOVO DÍGITO
            ADD     A, L        ; INÍCIO SOMA NOVO DÍGITO
            LD      L, A
            LD      A, H
            ADC     0
            LD      H, A
            LD      A, B
            ADC     0           ; TÉRMINIO SOMA NOVO DÍGITO
            RET
```

Fig. 17.5 – Subrotina BCDBN

17.7 – CONVERSÃO BINÁRIO → BCD – BNBCD

A subrotina BNBCD converte números binários de 16 bits para BCD.

Antes da chamada da subrotina BNBCD, o número binário tem que estar armazenado no par DE e o par HL deve conter o endereço da posição de memória onde será armazenado o dígito mais significativo do resultado.

O resultado da conversão fica armazenado em 5 bytes consecutivos na memória, sendo que cada dígito ocupa um byte. Observe que a subrotina BNBCD não altera nenhum registrador interno do Z-80.

O algorítmo utilizado por esta subrotina, na obtenção de cada dígito BCD, usa subtrações sucessivas da respectiva potência de 10. Note que para obter o resultado em ASCII, basta somar 30H a cada dígito, formando desta maneira o código numérico ASCII.

```
BNBCD:    PUSH   AF
          PUSH   BC
          PUSH   DE
          PUSH   HL
          EX     DE, HL          ; END. EM DE, NUM. EM HL
          LD     BC, - 10000 D
          CALL   NODEC           ; DETERMINA DÍGITO MAIS
                                 ; SIGNIFICATIVO
          LD     BC, - 1000 D
          CALL   NODEC
          LD     BC, - 100 D
          CALL   NODEC
          LD     BC, - 10 D
          CALL   NODEC
          LD     A, L
          LD     (DE), A         ; ARMAZENA DÍGITO MENOS SIGN.
          POP    HL
          POP    DE
          POP    BC
          POP    AF
          RET
; SUBROTINA NODEC

NODEC:    XOR    A               ; A ← 00 (FAÇA A ← 30H SE USAR
                                 ; CÓDIGO ASCII)
          PUSH   DE
          LD     E,L
          LD     D, H
          INC    A               ; INCREMENTA DÍGITO BCD
          ADD    HL, BC
          JR     C, NODEC + 2    ; CY = 1, DESVIA
          DEC    A               ; COMPENSA RESULTADO
          LD     L, E
          LD     H, D
          POP    DE
          LD     (DE), A         ; ARMAZENA DÍGITO
          INC    DE              ; ATUALIZA ENDEREÇO
          RET
```

Fig. 17.6 – Subrotina BNBCD

17.8 – MULTIPLICAÇÃO – MULT

A subrotina MULT multiplica um número de 16 bits por outro de 8 bits e obtém um resultado de 24 bits.

Antes da chamada da subrotina MULT o multiplicando de 16 bits tem que estar armazenado no par DE e, o multiplicador de 8 bits no acumulador.

O produto é armazenado nos registradores A, H e L, ficando a parte mais significativa no acumulador.

O algorítmo usado pela subrotina MULT na obtenção do produto é o seguinte:

PASSO 1: Teste do bit menos significativo do multiplicador; se for zero desvie para o passo 2, se for um some o multiplicando ao resultado parcial do produto e vá para o passo 2.

PASSO 2: Desloque o resultado parcial à esquerda

PASSO 3: Faça os passos 1 e 2 até que os 8 bits do multiplicador estejam testados.

```
MULT:   LD    B, 8          ; CONTADOR
        LD    HL, O         ; INICIA HL
MULT1:  ADD   HL, HL        ; DESLOCA A, H e L À ESQUERDA
        RLA
        JR    NC, MULT2     ; CY =1 SOMA MULTIPLICANDO
        ADD   HL, DE        ; CY =O DESLOCA SEM SOMAR
        ADC   A, O          ; SOMA MULTIPLICANDO
MULT2:  DJNZ  MULT1         ; ATUALIZA B
        RET
```

Fig. 17.7 – Subrotina MULT

17.9 – DIVISÃO DE NÚMEROS BINÁRIOS – DIVID

A subrotina DIVID divide um número binário de 16 bits por outro número binário de 8 bits. Os números são considerados positivos sem o sinal.

Antes da chamada da subrotina DIVID o dividendo tem que estar carregado em HL e o divisor em C. Ao finalizar a execução, o quociente fica armazenado em HL e o resto no acumulador.

O algorítmo usado nesta divisão, basicamente, desloca o par HL (dividendo) um bit à esquerda para o acumulador, em cada

uma das 16 iterações. E, subtrai o registrador C (divisor) do dividendo parcial produzido no acumulador. Se o resultado desta subtração for positivo, o valor um é colocado no bit 0 do par HL. Se o resultado for negativo, o valor zero é colocado no bit 0 do par HL e, o valor original em A é restaurado. O quociente vai ocupando o par HL, da direita para à esquerda, à medida que o resto é obtido pelo deslocamento à esquerda do acumulador.

```
DIVID:   LD    B, 16D        ; INICIA CONTADOR
         XOR   A             ; REG. DE EXTENSÃO = 0
DIV01:   ADD   HL, HL        ; DESLOCA HL UM BIT À ESQ.
         ADC   A, A          ; DESLOCA O ACUM UM BIT
                             ; À ESQ COM CARRY
         INC   L             ; SETA O BIT 0 DE L
         SUB   C             ; SUBTRAIA O DIVISOR DO
                             ; DIVIDENDO PARCIAL
         JR    NC, DIV02     ; SE O RESULTADO FOR NEG
                             ; ENTÃO RESTAURA O DIVIDENDO
                             ; PARCIAL E ZERA O BIT 0 DE L
         ADD   A, C
         DEC   L
DIV02:   DJNZ  DIV01         ; REALIZA O LOOP 16 VEZES
         RET
```

Fig. 17.8 – Subrotina DIVID

17.10 – ATRASO -- ATRAS

A determinação do atraso é obtida consultando-se na tabela correspondente a cada instrução, o nº de períodos do relógio que cada um leva para ser executada. Assim, multiplicando-se este nº pelo valor do período, que será considerado igual a 500 ns (2 MHZ), temos o tempo de execução de cada instrução.

Após a determinação do tempo de cada instrução, precisamos identificar quantas vezes cada uma delas é executada. Para esta rotina ATRAS, identificada na fig. 17.9, temos:

T = t(LD DE,1) + HL (t(LD B, C) + C *t (DJNZ) + t (SBC HL, DE) + t (JR)) + t (RET)

Substituindo-se os tempos de cada instrução, chegamos na expressão abaixo que fornece o tempo da subrotina em micro segundos:

T (ATRAS) = 6 + HL (7, 8 *C + 15,6)

Os conteúdos de C e do par HL são parâmetros de entrada da subrotina ATRAS. O conteúdo de C estipula o atraso do "LOOP" interno e o conteúdo do par HL o atraso do "LOOP" externo.

Por exemplo, se C = 20D, HL = 5827D e o Z-80 estiver operando com um relógio de 2MHZ, teremos um atraso de aprox. 1 seg.

```
ATRAS:   LD     DE, 1          ; PREPARA SUBTRAÇÃO DO HL
TEMP1:   LD     B, C
TEMP2:   DJNZ   TEMP2          ; DECREMENTA B ATÉ 00
         SBC    HL, DE         ; DECREMENTA HL
         JR     NZ, TEMP1      ; SE HL ≠ 0, DESVIA
         RET
```

Fig. 17.9 – Subrotina ATRAS

APÊNDICES

A INSTRUÇÕES DO Z-80 NÃO DIVULGADAS PELOS MANUAIS

Neste apêndice apresentamos algumas instruções que não são divulgadas nos manuais dos fabricantes do Z-80.

Acreditamos que a não divulgação destas instruções se deve ao fato de que a descrição de um conjunto excessivamente grande de instruções poderia dificultar a fabricação de novos produtos compatíveis com o Z-80. Além disso, estas instruções podem ser utilizadas para proteger programas.

As instruções aqui descritas foram testadas em microprocessadores Z-80 fabricados pela ZILOG, MOSTEK, SGS e NEC.

A.1 – INSTRUÇÕES DE DESLOCAMENTO

Os códigos de máquina hexadecimais compreendidos entre CB30H e CB37H fazem com que o conteúdo de um registrador de uso geral de 8 bits ou de uma posição de memória endereçada pelo par HL seja deslocado de 1 bit à esquerda. O conteúdo do bit 7 vai para o carry e o bit 0 assume o estado lógico 1. Do ponto de vista aritmético esta instrução multiplica o valor do registrador por 2 e incrementa 1. O mnemônico deste tipo de deslocamento é SLI ("shift left inverted"). Estas instruções são as seguintes:

SLI B	CB30H	SLI C	CB31H
SLI D	CB32H	SLI E	CB33H
SLI H	CB34H	SLI L	CB35H
SLI (HL)	CB36H	SLI A	CB37H

A.2 – INSTRUÇÕES QUE ENVOLVEM IX e IY

As instruções que envolvem os registradores IX e IY possuem os mesmos códigos de máquina que as instruções que lidam com o HL, precedidos pelos bytes DDH e FDH. Estes bytes acionam os registradores IX e IY no lugar de HL.

Em resumo, colocando-se o byte DDH precedendo uma instrução que envolve HL estaremos substituindo HL por IX e (HL) por (IX + d). E, precedendo-se uma instrução por FDH a mesma substituição se faz para o IY. Com excessão das instruções EX, DE, HL e EXX, além das instruções iniciadas pelo byte EDH.

Exemplos:

CB 36	–	SLI (HL)
DD CB dd 36	–	SLI (IX + dd)
CB 38	–	RES 0,B
FD CB dd 80	–	RES 0, (IY + dd)
46	–	LD B, (HL)
DD 46 dd	–	LD B, (IX + dd)

A.3 – INSTRUÇÕES QUE ENVOLVEM HX, HY, LX e LY

Se estudarmos o procedimento descrito em A.2 para H e L separadamente, teremos instruções envolvendo apenas 8 bits dos registradores IX e IY. Em outras palavras, colocando-se o byte DDH precedendo uma instrução que envolva H ou L, mas não envolvendo HL, como LD H, (HL), estaremos efetuando operações com o byte mais significativo (HX) ou o byte menos significativo de IX (LX) separadamente. Analogamente, teremos, HY e LY.

Exemplos:

DD 24 – INC HX	FD A5 – AND LY
DD 55 – LD D, LX	DD 95 – SUB LX

B CÓDIGOS DE MÁQUINA DO Z-80

(No código objeto, temos: d = XX, e = 2E, nn = 84XX e n = 20)

CÓDIGO OBJETO	COMANDO FONTE		CÓDIGO OBJETO	COMANDO FONTE	
8E	ADC	A,(HL)	A3	AND	E
DD8EXX	ADC	A,(IX+d)	A4	AND	H
FD8EXX	ADC	A,(IY+d)	A5	AND	L
8F	ADC	A,A	E620	AND	n
88	ADC	A,B	CB46	BIT	0,(HL)
89	ADC	A,C	DDCBXX46	BIT	0,(IX+d)
8A	ADC	A,D	FDCBXX46	BIT	0,(IY+d)
8B	ADC	A,E	CB47	BIT	0,A
8C	ADC	A,H	CB40	BIT	0,B
8D	ADC	A,L	CB41	BIT	0,C
CE20	ADC	A,n	CB42	BIT	0,D
ED4A	ADC	HL,BC	CB43	BIT	0,E
ED5A	ADC	HL,DE	CB44	BIT	0,H
ED6A	ADC	HL,HL	CB45	BIT	0,L
ED7A	ADC	HL,SP	CB4E	BIT	1,(HL)
86	ADD	A,(HL)	DDCBXX4E	BIT	1,(IX+d)
DD86XX	ADD	A,(IX+d)	FDCBXX4E	BIT	1,(IY+d)
FD86XX	ADD	A,(IY+d)	CB4F	BIT	1,A
87	ADD	A,A	CB48	BIT	1,B
80	ADD	A,B	CB49	BIT	1,C
81	ADD	A,C	CB4A	BIT	1,D
82	ADD	A,D	CB5B	BIT	1,E
83	ADD	A,E	CB4C	BIT	1,H
84	ADD	A,H	CB4D	BIT	1,L
85	ADD	A,L	CB56	BIT	2,(HL)
C620	ADD	A,n	DDCBXX56	BIT	2,(IX+d)
09	ADD	HL,BC	FDCBXX56	BIT	2,(IY+d)
19	ADD	HL,DE	CB57	BIT	2,A
29	ADD	HL,HL	CB50	BIT	2,B
39	ADD	HL,SP	CB51	BIT	2,C
DD09	ADD	IX,BC	CB52	BIT	2,D
DD19	ADD	IX,DE	CB53	BIT	2,E

(No código objeto, temos: d = XX, e = 2E, nn = 84XX e n = 20)

CÓDIGO OBJETO	COMANDO FONTE		CÓDIGO OBJETO	COMANDO FONTE	
DD29	ADD	IX,IX	CB54	BIT	2,H
DD39	ADD	IX,SP	CB55	BIT	2,L
FD09	ADD	IY,BC	CB5E	BIT	3,(HL)
FD19	ADD	IY,DE	DDCBXX5E	BIT	3,(IX+d)
FD29	ADD	IY,IY	FDCBXX5E	BIT	3,(IY+d)
FD39	ADD	IY,SP	CB5F	BIT	3,A
A6	AND	(HL)	CB58	BIT	3,B
DDA6XX	AND	(IX+d)	CB59	BIT	3,C
FDA6XX	AND	(IY+d)	CB5A	BIT	3,D
A7	AND	A	CB5B	BIT	3,E
A0	AND	B	CB5C	BIT	3,H
A1	AND	C	CB5D	BIT	3,L
A2	AND	D	CB66	BIT	4,(HL)
DDCBXX66	BIT	4,(IX+d)	E484XX	CALL	PO,nn
FDCBXX66	BIT	4,(IY+d)	CC84XX	CALL	Z,nn
CB67	BIT	4,A	CD84XX	CALL	nn
CB60	BIT	4,B	3F	CCF	
CB61	BIT	4,C	BE	CP	(HL)
CB62	BIT	4,D	DDBEXX	CP	(IX+d)
CB63	BIT	4,E	FDBEXX	CP	(IY+d)
CB64	BIT	4,H	BF	CP	A
CB65	BIT	4,L	B8	CP	B
CB6E	BIT	5,(HL)	B9	CP	C
DDCBXX6E	BIT	5,(IX+d)	BA	CP	D
FDCBXX6E	BIT	5,(IY+d)	BB	CP	E
CB6F	BIT	5,A	BC	CP	H
CB68	BIT	5,B	BD	CP	L
CB69	BIT	5,C	FE20	CP	n
CB6A	BIT	5,D	EDA9	CPD	
CB6B	BIT	5,E	EDB9	CPDR	
CB6C	BIT	5,H	EDB1	CPIR	
CB6D	BIT	5,L	EDA1	CPI	
CB76	BIT	6,(HL)	2F	CPL	
DDCBXX76	BIT	6,(IX+d)	27	DAA	
FDCBXX76	BIT	6,(IY+d)	35	DEC	(HL)
CB77	BIT	6,A	DD35XX	DEC	(IX+d)
CB70	BIT	6,B	FD35XX	DEC	(IY+d)
CB71	BIT	6,C	3D	DEC	A

(No código objeto, temos: d = XX; e = 2E, nn = 84XX e n = 20)

CÓDIGO OBJETO	COMANDO FONTE		CÓDIGO OBJETO	COMANDO FONTE	
CB72	BIT	6,D	05	DEC	B
CB73	BIT	6,E	0B	DEC	BC
CB74	BIT	6,H	0D	DEC	C
CB75	BIT	6,L	15	DEC	D
CB7E	BIT	7,(HL)	1B	DEC	DE
DDCBXX7E	BIT	7,(IX+d)	1D	DEC	E
FDCBXX7E	BIT	7,(IY+d)	25	DEC	H
CB7F	BIT	7,A	2B	DEC	HL
CB78	BIT	7,B	DD2B	DEC	IX
CB79	BIT	7,C	FD2B	DEC	IY
CB7A	BIT	7,D	2D	DEC	L
CB7B	BIT	7,E	3B	DEC	SP
CB7C	BIT	7,H	F3	DI	
CB7D	BIT	7,L	102E	DJNZ	e
DC84XX	CALL	C,nn	FB	EI	
FC84XX	CALL	M,nn	E3	EX	(SP),HL
D484XX	CALL	NC,nn	DDE3	EX	(SP),IX
C484XX	CALL	NZ,nn	FDE3	EX	(SP),IY
F484XX	CALL	P,nn	08	EX	AF,AF'
EC84XX	CALL	PE,nn	EB	EX	DE,HL
D9	EXX		382E	JR	C,e
76	HALT		302E	JR	NC,e
ED46	IM	0	2o2E	JR	NZ,e
ED56	IM	1	282E	JR	Z,e
ED5E	IM	2	182E	JR	e
ED78	IN	A,(C)	02	LD	(BC),A
ED40	IN	B,(C)	12	LD	(DE),A
ED48	IN	C,(C)	77	LD	(HL),A
ED50	IN	D,(C)	70	LD	(HL),B
ED58	IN	E,(C)	71	LD	(HL),C
ED60	IN	H,(C)	72	LD	(HL),D
ED68	IN	L,(C)	73	LD	(HL),E
34	INC	(HL)	74	LD	(HL),H
DD34XX	INC	(IX+d)	75	LD	(HL),L
FD34XX	INC	(IY+d)	3620	LD	(HL),n
3C	INC	A	DD77XX	LD	(IX+d),A
04	INC	B	DD70XX	LD	(IX+d),B
03	INC	BC	DD71XX	LD	(IX+d),C

(No código objeto, temos: d = XX, e = 2E, nn = 84XX e n = 20)

CÓDIGO OBJETO	COMANDO FONTE		CÓDIGO OBJETO	COMANDO FONTE	
0C	INC	C	DD72XX	LD	(IX+d),D
14	INC	D	DD73XX	LD	(IX+d),E
13	INC	DE	DD74XX	LD	(IX+d),H
1C	INC	E	DD75XX	LD	(IX+d),L
24	INC	H	DD36XX20	LD	(IX+d),n
23	INC	HL	FD77XX	LD	(IY+d),A
DD23	INC	IX	FD70XX	LD	(IY+d),B
FD23	INC	IY	FD71XX	LD	(IY+d),C
2C	INC	L	FD72XX	LD	(IY+d),D
33	INC	SP	FD73XX	LD	(IY+d),E
DB20	IN	A,(n)	FD74XX	LD	(IY+d),H
EDAA	IND		FD75XX	LD	(IY+d),L
EDBA	INDR		FD36XX20	LD	(IY+d),n
EDA2	INI		3284XX	LD	(nn),A
EDB2	INIR		ED4384XX	LD	(nn),BC
C384XX	JP	nn	ED5384XX	LD	(nn),DE
E9	JP	(HL)	2284XX	LD	(nn),HL
DDE9	JP	(IX)	DD2284XX	LD	(nn),IX
FDE9	JP	(IY)	FD2284XX	LD	(nn),IY
DA84XX	JP	C,nn	ED7384XX	LD	(nn),SP
FA84XX	JP	M,nn	0A	LD	A,(BC)
D284XX	JP	NC,nn	1A	LD	A,(DE)
C284XX	JP	NZ,nn	7E	LD	A,(HL)
F284XX	JP	P,nn	DD7EXX	LD	A,(IX+d)
EA84XX	JP	PE,nn	FD7EXX	LD	A,(IY+d)
E284XX	JP	PO,nn	3A84XX	LD	A,(nn)
CA84XX	JP	Z,nn	7F	LD	A,A
78	LD	A,B	1184XX	LD	DE,nn
79	LD	A,C	5E	LD	E,(HL)
7A	LD	A,D	DD5EXX	LD	E,(IX+d)
7B	LD	A,E	FD5EXX	LD	E,(IY+d)
7C	LD	A,H	5F	LD	E,A
ED57	LD	A,I	58	LD	E,B
7D	LD	A,L	59	LD	E,C
3E20	LD	A,n	5A	LD	E,D
ED5F	LD	A,R	5B	LD	E,E
46	LD	B,(HL)	5C	LD	E,H
DD46XX	LD	B,(IX+d)	5D	LD	E,L

(No código objeto, temos: d = XX, e = 2E, nn = 84XX e n = 20)

CÓDIGO OBJETO	COMANDO FONTE		CÓDIGO OBJETO	COMANDO FONTE	
FD46XX	LD	B,(IY+d)	1E20	LD	E,n
47	LD	B,A	66	LD	H,(HL)
40	LD	B,B	DD66XX	LD	H,(IX+d)
41	LD	B,C	FD66XX	LD	H,(IY+d)
42	LD	B,D	67	LD	H,A
43	LD	B,E	60	LD	H,B
44	LD	B,H	61	LD	H,C
45	LD	B,L	62	LD	H,D
0620	LD	B,n	63	LD	H,E
ED4B84XX	LD	BC,(nn)	64	LD	H,H
0184XX	LD	BC,nn	65	LD	H,L
4E	LD	C,(HL)	2620	LD	H,n
DD4EXX	LD	C,(IX+d)	2A84XX	LD	HL,(nn)
FD4EXX	LD	C,(IY+d)	2184XX	LD	HL,nn
4F	LD	C,A	ED47	LD	I,A
48	LD	C,B	DD2A84XX	LD	IX,(nn)
49	LD	C,C	DD2A84XX	LD	IX,nn
4A	LD	C,D	FD2A84XX	LD	IY,(nn)
4B	LD	C,E	FD2184XX	LD	IY,nn
4C	LD	C,H	6E	LD	L,(HL)
4D	LD	C,L	DD6EXX	LD	L,(IX+d)
0E20	LD	C,n	FD6EXX	LD	L,(IY+d)
56	LD	D,(HL)	6F	LD	L,A
DD56XX	LD	D,(IX+d)	68	LD	L,B
FD56XX	LD	D,(IY+d)	69	LD	L,C
57	LD	D,A	6A	LD	L,D
50	LD	D,B	6B	LD	L,E
51	LD	D,C	6C	LD	L,H
52	LD	D,D	6D	LD	L,L
53	LD	D,E	2E20	LD	L,n
54	LD	D,H	ED4F	LD	R,A
55	LD	D,L	ED7B84XX	LD	SP,(nn)
1620	LD	D,n	F9	LD	SP,HL
ED5B84XX	LD	DE,(nn)	DDF9	LD	SP,IX
FDF9	LD	SP,IY	FDCBXX86	RES	0,(IY+d)
3184XX	LD	SP,nn	CB87	RES	0,A
EDA8	LDD		CB80	RES	0,B
ED88	LDDR		CB81	RES	0,C

(No código objeto, temos: d = XX, e = 2E, nn = 84XX e n = 20)

CÓDIGO OBJETO	COMANDO FONTE		CÓDIGO OBJETO	COMANDO FONTE	
EDA0	LDI		CB82	RES	0,D
EDB0	LDIR		CB83	RES	0,E
ED44	NEG		CB84	RES	0,H
00	NOP		CB85	RES	0,L
B6	OR	(HL)	CB8E	RES	1,(HL)
DDB6XX	OR	(IX+d)	DDCBXX8E	RES	1,(IX+d)
FDB6XX	OR	(IY+d)	FDCBXX8E	RES	1,(IY+d)
B7	OR	A	CB8F	RES	1,A
B0	OR	B	CB88	RES	1,B
B1	OR	C	CB89	RES	1,C
B2	OR	D	CB8A	RES	1,D
B3	OR	E	CB8B	RES	1,E
B4	OR	H	CB8C	RES	1,H
B5	OR	L	CB8D	RES	1,L
F620	OR	n	CB96	RES	2,(HL)
EDBB	OTDR		DDCBXX96	RES	2,(IX+d)
EDB3	OTIR		FDCBXX96	RES	2,(IY+d)
ED79	OUT	(C),A	CB97	RES	2,A
ED41	OUT	(C),B	CB90	RES	2,B
ED49	OUT	(C),C	CB91	RES	2,C
ED51	OUT	(C),D	CB92	RES	2,D
ED59	OUT	(C),E	CB93	RES	2,E
ED61	OUT	(C),H	CB94	RES	2,H
ED69	OUT	(C),L	CB95	RES	2,L
D320	OUT	(n),A	CB9E	RES	3,(HL)
EDAB	OUTD		DDCBXX9E	RES	3,(IX+d)
EDA3	OUTI		FDCBXX9E	RES	3,(IY+d)
F1	POP	AF	CB9F	RES	3,A
C1	POP	BC	CB98	RES	3,B
D1	POP	DE	CB99	RES	3,C
E1	POP	HL	CB9A	RES	3,D
DDE1	POP	IX	CB9B	RES	3,E
FDE1	POP	IY	CB9C	RES	3,H
F5	PUSH	AF	CB9D	RES	3,L
C5	PUSH	BC	CBA6	RES	4,(HL)
D5	PUSH	DE	DDCBXXA6	RES	4,(IX+d)
E5	PUSH	HL	FDCBXXA6	RES	4,(IY+d)
DDE5	PUSH	IX	CBA7	RES	4,A

(No código objeto, temos: d = XX, e = 2E, nn = 84XX e n = 20)

CÓDIGO OBJETO	COMANDO FONTE		CÓDIGO OBJETO	COMANDO FONTE	
FDE5	PUSH	IY	CBA0	RES	4,B
CB86	RES	0,(HL)	CBA1	RES	4,C
DDCBXX86	RES	0,(IX+d)	CBA2	RES	4,D
CBA3	RES	4,E	DDCBXX16	RL	(IX+d)
CBA4	RES	4,H	FDCBXX16	RL	(IY+d)
CBA5	RES	4,L	CB17	RL	A
CBAE	RES	5,(HL)	CB10	RL	B
DDCBXXAE	RES	5,(IX+d)	CB11	RL	C
FDCBXXAE	RES	5,(IY+d)	CB12	RL	D
CBAF	RES	5,A	CB13	RL	E
CBA8	RES	5,B	CB14	RL	H
CBA9	RES	5,C	CB15	RL	L
CBAA	RES	5,D	17	RLA	
CBA8	RES	5,E	CB06	RLC	(HL)
CBAC	RES	5,H	DDCBXX06	RLC	(IX+d)
CBAD	RES	5,L	FDCBXX06	RLC	(IY+d)
CBB6	RES	6,(HL)	CB07	RLC	A
DDCBXXB6	RES	6,(IX+d)	CB00	RLC	B
FDCBXXB6	RES	6,(IY+d)	CB01	RLC	C
CBB7	RES	6,A	CB02	RLC	D
CBB0	RES	6,B	CB03	RLC	E
CBB1	RES	6,C	CB04	RLC	H
CBB2	RES	6,D	CB05	RLC	L
CBB3	RES	6,E	07	RLCA	
CBB4	RES	6,H	ED6F	RLD	
CBB5	RES	6,L	CB1E	RR	(HL)
CBBE	RES	7,(HL)	DDCBXX1E	RR	(IX+d)
DDCBXXBE	RES	7,(IX+d)	FDCBXX1E	RR	(IY+d)
FDCBXXBE	RES	7,(IY+d)	CB1F	RR	A
CBBF	RES	7,A	CB18	RR	B
CBB8	RES	7,B	CB19	RR	C
CBB9	RES	7,C	CB1A	RR	D
CBBA	RES	7,D	CB1B	RR	E
CBBB	RES	7,E	CB1C	RR	H
CBBC	RES	7,H	CB1D	RR	L
CBBD	RES	7,L	1F	RRA	
C9	RET		CB0E	RRC	(HL)
D8	RET	C	DDCBXX0E	RRC	(IX+d)

(No código objeto, temos: d = XX, e = 2E, nn = 84XX e n = 20)

CÓDIGO OBJETO	COMANDO FONTE		CÓDIGO OBJETO	COMANDO FONTE	
F8	RET	M	FDCBXX0E	RRC	(IY+d)
D0	RET	NC	CB0F	RRC	A
C0	RET	NZ	CB08	RRC	B
F0	RET	P	CB09	RRC	C
E8	RET	PE	CB0A	RRC	D
E0	RET	PO	CB0B	RRC	E
C8	RET	Z	CB0C	RRC	H
ED4D	RETI		CB0D	RRC	L
ED45	RETN		0F	RRCA	
CB16	RL	(HL)	ED67	RRD	
C7	RST	00H	DDCBXXD6	SET	2,(IX+d)
CF	RST	08H	FDCBXXD6	SET	2,(IY+d)
D7	RST	10H	CBD7	SET	2,A
DF	RST	18H	CBD0	SET	2,B
E7	RST	20H	CBD1	SET	2,C
EF	RST	28H	CBD2	SET	2,D
F7	RST	30H	CBD3	SET	2,E
FF	RST	38H	CBD4	SET	2,H
DE20	SBC	A,n	CBD5	SET	2,L
9E	SBC	A,(HL)	CBD8	SET	3,B
DD9EXX	SBC	A,(IX+d)	CBDE	SET	3,(HL)
FD9EXX	SBC	A,(IY+d)	DDCBXXDE	SET	3,(IX+d)
9F	SBC	A,A	FDCBXXDE	SET	3,(IY+d)
98	SBC	A,B	CBDF	SET	3,A
99	SBC	A,C	CBD9	SET	3,C
9A	SBC	A,D	CBDA	SET	3,D
9B	SBC	A,E	CBDB	SET	3,E
9C	SBC	A,H	CBDC	SET	3,H
9D	SBC	A,L	CBDD	SET	3,L
ED42	SBC	HL,BC	CBE6	SET	4,(HL)
ED52	SBC	HL,DE	DDCBXXE6	SET	4,(IX+d)
ED62	SBC	HL,HL	FDCBXXE6	SET	4,(IY+d)
ED72	SBC	HL,SP	CBE7	SET	4,A
37	SCF		CBE0	SET	4,B
CBC6	SET	0,(HL)	CBE1	SET	4,C
DDCBXXC6	SET	0,(IX+d)	CBE2	SET	4,D
FDCBXXC6	SET	0,(IY+d)	CBE3	SET	4,E
CBC7	SET	0,A	CBE4	SET	4,H

(No código objeto, temos: d = XX, e = 2E, nn = 84XX e n = 20)

CÓDIGO OBJETO	COMANDO FONTE		CÓDIGO OBJETO	COMANDO FONTE	
CBC0	SET	0,B	CBC5	SET	4,L
CBC1	SET	0,C	CBEE	SET	5,(HL)
CBC2	SET	0,D	DDCBXXEE	SET	5,(IX+d)
CBC3	SET	0,E	FDCBXXEE	SET	5,(IY+d)
CBC4	SET	0,H	CBEF	SET	5,A
CBC5	SET	0,L	CBE8	SET	5,B
CBCE	SET	1,(HL)	CBE9	SET	5,C
DDCBXXCE	SET	1,(IX+d)	CBEA	SET	5,D
FDCBXXCE	SET	1,(IY+d)	CBEB	SET	5,E
CBCF	SET	1,A	CBEC	SET	5,H
CBC8	SET	1,B	CBED	SET	5,L
CBC9	SET	1,C	CBF6	SET	6,(HL)
CBCA	SET	1,D	DDCBXXF6	SET	6,(IX+d)
CBCB	SET	1,E	FDCBXXF6	SET	6,(IY+d)
CBCC	SET	1,H	CBF7	SET	6,A
CBCD	SET	1,L	CBF0	SET	6,B
CBD6	SET	2,(HL)	CBF1	SET	6,C
CBF2	SET	6,D	CB2D	SRA	L
CBF3	SET	6,E	CB3E	SRL	(HL)
CBF4	SET	6,H	DDCBXX3E	SRL	(IX+d)
CBF5	SET	6,L	FDCBXX3E	SRL	(IX+d)
CBFE	SET	7,(HL)	CB3F	SRL	A
DDCBXXFE	SET	7,(IX+d)	CB38	SRL	B
FDCBXXFE	SET	7,(IX+d)	CB39	SRL	C
CBFF	SET	7,A	CB3A	SRL	D
CBF8	SET	7,B	CB3B	SRL	E
CBF9	SET	7,C	CB3C	SRL	H
CBFA	SET	7,D	CB3D	SRL	L
CBFB	SET	7,E	96	SUB	(HL)
CBFC	SET	7,H	DD96XX	SUB	(IX+d)
CBFD	SET	7,L	FD96XX	SUB	(IY+d)
CB26	SLA	(HL)	97	SUB	A
DDCBXX26	SLA	(IX+d)	90	SUB	B
FDCBXX26	SLA	(IY+d)	91	SUB	C
CB27	SLA	A	92	SUB	D
CB20	SLA	B	93	SUB	E
CB21	SLA	C	94	SUB	H
CB22	SLA	D	95	SUB	L

(No código objeto, temos: d = XX, e = 2E, nn = 84XX e n = 20)

CÓDIGO OBJETO	COMANDO FONTE		CÓDIGO OBJETO	COMANDO FONTE	
CB23	SLA	E	D620	SUB	n
CB24	SLA	H	AE	XOR	(HL)
CB25	SLA	L	DDAEXX	XOR	(IX+d)
CB2E	SRA	(HL)	FDAEXX	XOR	(IY+d)
DDCBXX2E	SRA	(IX+d)	AF	XOR	A
FDCBXX2E	SRA	(IY+d)	A8	XOR	B
CB2F	SRA	A	A9	XOR	C
CB28	SRA	B	AA	XOR	D
CB29	SRA	C	AB	XOR	E
CB2A	SRA	D	AC	XOR	H
CB2B	SRA	E	AD	XOR	L
CB2C	SRA	H	EE20	XOR	n

C RELAÇÃO ENTRE AS INSTRUÇÕES DO 8080 E Z-80

Conversão do 8080 para o Z-80

8080	Z-80	8080	Z-80
ACI[B2]	ADC A,n	IN[B2]	IN A,(n)
ADC M	ADC A,(HL)	INR M	INC (HL)
ADC r	ADC A,r	INR r	INC r
ADD M	ADD A,(HL)	INX B	INC BC
ADD r	ADD A,r	INX D	INC DE
ADI[B2]	ADD A,n	INX H	INC HL
ANA M	AND (HL)	INX SP	INC SP
ANA r	AND r	JC[B2][B3]	JP C,nn
ANI[B2]	AND n	JM[B2][B3]	JP M,nn
CALL	CALL nn	JMP[B2][B3]	JP nn
CC[B2][B3]	CALL CC,nn	JNC[B2][B3]	JP NC,nn
CM[B2][B3]	CALL M,nn	JNZ[B2][B3]	JP NZ,nn
CMA	CPL	JP[B2][B3]	JP P,nn
CMC	CCF	JPE[B2][B3]	JP PE,nn
CMP M	CP(HL)	JPO[B2][B3]	JP PO,nn
CMP r	CP r	JZ[B2][B3]	JP Z,nn
CNC[B2][B3]	CALL NC,nn	LDA[B2][B3]	LD A,(nn)
CNZ[B2][B3]	CALL NZ,nn	LDAX B	LD A,(BC)
CP[B2][B3]	CALL P,nn	LDAX D	LD A,(DE)
CPE[B2][B3]	CALL PE,nn	LHLD [B2][B3]	LD HL,(nn)
CPI[B2]	CP n	LXI B[B2][B3]	LD BC,nn
CPO[B2][B3]	CALL PO,nn	LXI D[B2][B3]	LD DE,nn
CZ[B2][B3]	CALL Z,nn	LXI H[B2][B3]	LD HL,nn
DAA	DAA	LXI SP[B2][B3]	LD SP,nn
DAD B	ADD HL,BC	MOV M,r	LD (HL),r
DAD D	ADD HL,DE	MOV r,M	LD r,(HL)
DAD H	ADD HL,HL	MOV ri,r2	LD r,r'
DAD SP	ADD HL,SP	MVI M	LD (HL),n
DCR M	DEC (HL)	MVI r[B2]	LD r,n
DCR r	DEC r	NOP	NOP
DCX B	DEC BC	ORA M	OR (HL)
DCX D	DEC DE	ORA r	OR r

8080	Z-80	8080	Z-80
DCX H	DEC HL	ORI[B2]	OR n
DCX SP	DEC SP	OUT[B2]	OUT (n),A
DI	DI	PCHL	JP (HL)
EI	EI	POP B	POP BC
HALT	HLT	POP D	POP DE
POP H	POP HL	RZ	RET Z
POP PSW	POP AF	SBB M	SBC A,(HL)
PUSH B	PUSH BC	SBB r	SBC A,r
PUSH D	PUSH DE	SBI[B2]	SBC A,n
PUSH H	PUSH HL	SHLD[B2][B3]	LD(nn),HL
PUSH PSW	PUSH AF	SPHL	LD SP,HL
RAL	RLA	STA[B2][B3]	LD(nn),A
RAR	RRA	STAX B	LD(BC),A
RC	RET C	STAX D	LD(DE),A
RET	RET	STC	SCF
RLC	RLCA	SUB M	SUB(HL)
RM	RET M	SUB r	SUB r
RNC	RET NC	SUI[B2]	SUB n
RNZ	RET NZ	XCHG	EX DE,HL
RP	RET P	XRA M	XOR(HL)
RPE	RET PE	XRA r	XOR r
RPO	RET PO	XRI[B2]	XOR n
RRC	RRCA	XTHL	EX(SP),HL
RST P	RST P		

D RELAÇÃO ENTRE AS INSTRUÇÕES DO Z-80 E 8080

Conversão do Z-80 para o 8080

Z-80	8080	Z-80	8080
ADC A,(HL)	ADC M	EX (SP),HL	XTHL
ADC A,n	ACI[B2]	HALT	HLT
ADC A,r	ADC r	IN A,(n)	IN[B2]
ADD A,(HL)	ADD M	INC BC	INX B
ADD A,n	ADI[B2]	INC DE	INX D
ADD A,r	ADD r	INC HL	INX H
ADD HL,BC	DAD B	INC r	INR r
ADD HL,DE	DAD D	INC SP	INX SP
ADD HL,HL	DAD H	INC (HL)	INR M
ADD HL,SP	DAD SP	JP C,nn	JC[B2][B3]
AND n	ANI[B2]	JP M,nn	JM[B2][B3]
AND r	ANA r	JP NC,nn	JNC[B2][B3]
AND (HL)	ANA M	JP nn	JMP[B2][B3]
CALL C,nn	CC[B2][B3]	JP NZ,nn	JNZ[B2][B3]
CALL M,nn	CM[B2][B3]	JP P,nn	JP[B2][B3]
CALL NC,nn	CNC[B][B3]	JP PE,nn	JPE[B2][B3]
CALL nn	CALL	JP PO,nn	JPO[B2][B3]
CALL NZ,nn	CNZ[B2][B3]	JP Z,nn	JZ[B2][B3]
CALL P,nn	CP[B2][B3]	JP (HL)	PCHL
CALL PE,nn	CPE[B2][B3]	LD A,(DE)	LDAX
CALL PO,nn	CPO[B2][B3]	LD A,(nn)	LDA[B2][B3]
CALL Z,nn	CZ[B2][B3]	LD DE,nn	LXID,[B2][B3]
CCF	CMC	LD SP,nn	LXI SP,[B2][B3]
CP r	CMP r	LD (BC),A	STAX B
CP (HL)	CMP M	LD (DE),A	STAX D
CPL	CMA	LD (HL),r	MOV M,r
CP n	CPI[B2]	LD (nn),A	STA[B2][B3]
DAA	DAA	LD (nn),HL	SHLD[B2][B3]
DEC BC	DCX B	LD A,(BC)	LDAX B
DEC DE	DCX D	LD BC,nn	LXIB,[B2][B3]
DEC HL	DCX H	LD HL,(nn)	LHLD [B2][B3]
DEC r	DCR r	LD HL,nn	LXI H[B2][B3]
DEC SP	DCX SP	LD r,(HC)	MOV i,M

Z-80	8080	Z-80	8080
DEC (HL)	DCR M	LD r,n	MVI r,[B2]
DI	DI	LD r,r'	MOV r1,r2
EI	EI	LD SP,HL	SPHL
EX DE,HL	XCHG	NOP	NOP
OR n	ORI[B2]	RET PE	RPE
OR r	ORA r	RET PO	RPO
OR (HL)	ORA M	RET Z	RZ
OUT (n),A	OUT[B2]	RLA	RAL
POP AF	POP PSW	RLCA	RLC
POP BC	POP B	RRA	RAR
POP DE	POP D	RRCA	RRC
POP HL	POP H	RST P	RST P
PUSH AF	PUSH PSW	SBC A,(HL)	SBB M
PUSH BC	PUSH B	SBC A,n	SBI[B2]
PUSH DE	PUSH D	SBC A,r	SBB r
PUSH HL	PUSH H	SCF	STC
RET	RET	SUB n	SUI[B2]
RET C	RC	SUB r	SUB r
RET M	RM	SUB (HL)	SUB M
RET NC	RNC	XOR n	XRI[B2]
RET NZ	RNZ	XOR r	XRA r
RET P	RP	XOR (HL)	XRA M

E CÓDIGO ASCII

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

BIBLIOGRAFIA

1. OLIVEIRA N. A. R, Rubens A. G.: "Microprocessador Z-80 Hardware", A e N Consultoria Projetos e Publicações Ltda, 1983.
2. SERRA, C. P.: "Prática de Programação do 8080A", Edições Monitor, SP, 1981.
3. DONOVAN J. J.: "Systens Programming", McGraw-Hill Book Company, 1972.
4. KNUTE E. D.: "The Art of Computer Programming", Fundamental Algorithms Vol. 1, Addison – Wesley Publishing Company, 1976.
5. GEAR W.: "Computer Organization and Programming", 2 nd edition, McGraw-Hill Book Company, 1974.
6. TURNER R. C.: "Real-Time Programming With Microcomputers", Lexington Books, 1978.
7. VISCONTI, A. C. J. F.: "Microprocessador 8080 e 8085", Livros Érica Editora Ltda, SP, 1982.
8. BARDEN, W. J.: "The Z-80 Microcomputer Handbook", Howard W. Sams & Co., Inc., Indiana, 1981.
9. BARDEN, W. J.: "Z-80 Microcomputer Design Projects", Howard W. Sams & Co., Inc. Indiana, 1982.
10. KANE, J., OSBORNE, A.: "An Indroduction to Microcomputers", Volume 3, Osborne & Associates. Inc., California, 1979.
11. LEVENTHAL, A. L.: "Z-80 Assembly Language Programming", Osborne/McGraw-Hill, California, 1979.
12. KHAMBATA, A. J.: "Introduction to the Z-80 Microcomputer", John Wiley & Sons, New York, 1982.
13. TAILLIAR, A., HARBERT, D.: "Les Tendances actuelles des microprocessurs 8 bits", Micro Systemes, Société Parisiense d'Edition, Paris, Nov./Dec. 1982.
14. ZILOG: "Data Book", Zilog Inc., 1981.

Impressão e Acabamento
GRAPHOS
Rua Santo Cristo, 70/78 — Tel.: 223-1272

OUTRAS PUBLICAÇÕES

MICROPROCESSADOR Z-80 HARDWARE

Autores: *Ney Acyr R. de Oliveira*
André Gil Rubens

Neste trabalho os autores apresentam de uma forma didática e com inúmeros exemplos e exercícios, a arquitetura, o funcionamento e o interfaceamento do Z-80 com módulos de memória e portas de entrada e saída.

Todos os conceitos necessários ao entendimento do hardware são apresentados gradativamente, de tal modo que o leitor não necessite de conhecimentos profundos de eletrônica.

PROJETO DE MICROMESTRE Z-80

Autores: *Ney Acyr R. de Oliveira*
André Gil Rubens

Este livro apresenta um projeto completo de um kit didático para o treinamento e desenvolvimento com o microprocessador Z-80.

Os autores apresentam a operação do kit, os diagramas elétricos, os lay-outs das placas, a lista de componentes e a listagem em assembly do programa monitor.

SOBRE OS AUTORES

André Gil Rubens

- Engenheiro Eletrônico formado pela PUC/RJ em dezembro de 1975.
- Mestre em Ciência de Engenharia Elétrica pela PUC/RJ em fevereiro de 1980.
- Engenheiro da Embratel desde 1976, onde trabalha no desenvolvimento de hardware e software de microcomputadores.
- Professor Titular da Faculdade Politécnica Estácio de Sá.
- Consultor Técnico da Revista Interface.

Ney Acyr Rodrigues de Oliveira

- Engenheiro Eletrônico formado pela Escola de Engenharia da UFRJ em dezembro de 1977.
- Pós-Graduação em Sistemas Digitais na COPPE/UFRJ.
- Engenheiro da Embratel desde 1978, onde trabalha no desenvolvimento de hardware e software de microcomputadores.
- Professor Titular da Faculdade Politécnica Estácio de Sá.
- Consultor Técnico da Revista Interface.

OUTRAS PUBLICAÇÕES

MICROPROCESSADOR Z-80 HARDWARE

Autores: Ney Acyr R. de Oliveira
André Gil Rubens

Neste trabalho os autores apresentam de uma forma didática e com inúmeros exemplos e exercícios, a arquitetura, o funcionamento e o interfaceamento do Z-80 com módulos de memória e portas de entrada e saída.

Todos os conceitos necessários ao entendimento do hardware são apresentados gradativamente, de tal modo que o leitor não necessite de conhecimentos profundos de eletrônica.

PROJETO DO MICROMESTRE Z-80

Autores: Ney Acyr R. de Oliveira
André Gil Rubens

Este livro apresenta um projeto completo de um kit didático para o treinamento e desenvolvimento com o microprocessador Z-80.

Os autores apresentam a operação do kit, os diagramas elétricos, os layouts das placas, a lista de componentes e a listagem em assembly do programa monitor.

ISBN 85-85039-01-9