

PROGRAMAÇÃO ASSEMBLER

280
6502
8080
8085

**PROGRAMAÇÃO
ASSEMBLER
PARA MICROPROCESSADORES
8080/85, Z80 e 6502**

PROGRAMAÇÃO ASSEMBLER PARA MICROPROCESSADORES 8080/85, Z80 e 6502

TAMIO SHIMIZU

Professor Adjunto da Escola Politécnica-USP e da FEI-Faculdade de Engenharia Industrial.

Analista de Sistemas de Software. Livre-docente e Doutor em Ciências pela Escola Politécnica da USP.

Mestre em Ciências pelo ITA-Instituto Tecnológico de Aeronáutica.

Diretor da Mega Systems Hardware e Software Ltda.

McGraw-Hill

São Paulo

Rua Tabapuã, 1.105, Itaim-Bibi

CEP 04533

(011) 881-8604 e (011) 881-8528

Rio de Janeiro • Lisboa • Porto • Bogotá • Buenos Aires • Guatemala • Madrid • México •
New York • Panamá • San Juan • Santiago

Auckland • Hamburg • Kuala Lumpur • London • Milan • Montreal • New Delhi • Paris •
Singapore • Sydney • Tokyo • Toronto

Programação Assembler para Microprocessadores 8080/85, Z80 e 6502

Copyright ©1987 da Editora McGraw-Hill, Ltda.

Todos os direitos para a língua portuguesa reservados pela Editora McGraw-Hill, Ltda.

Nenhuma parte desta publicação poderá ser reproduzida, guardada pelo sistema "retrieval" ou transmitida de qualquer modo ou por qualquer outro meio, seja este eletrônico, mecânico, de fotocópia, de gravação, ou outros, sem prévia autorização, por escrito, da Editora.

Editor: Laércio Bento

Coordenadora de Revisão: Daisy Pereira Daniel

Capa: Layout: Cyro Giordano

Arte Final: Ademir Aparecido Alves

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

Shimizu, Tamio, 1938 –
S559p Programação Assembler para microprocessadores 8080/85, Z80 e 6502 /
Tamio Shimizu. – São Paulo : McGraw-Hill, 1987.

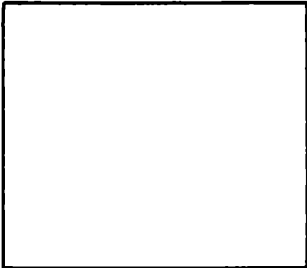
1. Assembler (Linguagem de programação para computadores) I. Título.

86-2303

CDD-001.6424

Índices para catálogo sistemático:

1. Assembler : Linguagem de programação : Computadores :
Processamento de dados 001.6424



SUMÁRIO

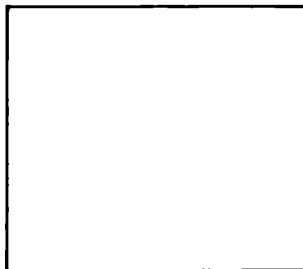
Parte I Noções Gerais de Microprocessador e Linguagem Assembler

Capítulo 1	Noções Gerais de Arquitetura da UCP do Computador e Microcomputador . A Linguagem de Máquina	3
Capítulo 2	Noções Gerais de Linguagem Assembler Elementos Básicos	15

Parte II Linguagem Assembler dos Microprocessadores 8080/85 e Z80

Capítulo 3	Elementos Básicos da Arquitetura do Microprocessador 8080/85 e Z80	31
Capítulo 4	Instruções de Movimento ou Transferência de Dados (MOVE, LOAD ou STORE) do 8080/85 e Z80.	39
Capítulo 5	Instruções de Operações Aritméticas (Increment, Decrement, Complement, ADD e Subtract) para 8080/85 e Z80	59
Capítulo 6	Instruções de Comparação e Desvios para 8080/85 e Z80	80
Capítulo 7	Instruções para Operações Lógicas “AND”, “OR” e “EXCLUSIVE OR” para 8080/85 e Z80	91
Capítulo 8	Aplicações: Programas-Exemplos com Contador de Repetições (LOOPS) . . .	99
Capítulo 9	Instruções de Giros e Deslocamentos (ROTATE e SHIFT).	113
Capítulo 10	Instruções de Desvio (CALL) e Retorno (RET) da Sub-Rotina	128
Capítulo 11	Instruções de testes de Bits, Interrupções, Pilhas e Entrada/Saída	137

Capítulo 12	Exemplo de Assembler: Leitura e Escrita de Caracteres através de uma Máquina TTY (TELETYPEWRITER) ou Terminal de TV	143
Parte III Linguagem Assembler do Microprocessador 6502		
Capítulo 13	Elementos Básicos da Arquitetura do Microprocessador 6502	159
Capítulo 14	Instruções de Movimento ou Transferência de Dados do 6502: LOAD e STORE	167
Capítulo 15	Instruções de Operações Aritméticas do 6502: INCREMENT, DECREMENT, ADD e SUBTRACT	177
Capítulo 16	Instruções de Comparação e Desvios do 6502: COMPARE, BRANCH e JUMP	189
Capítulo 17	Instruções de Operações Lógicas do 6502: AND, BIT TEST, OR e EXCLUSIVE OR	197
Capítulo 18	Aplicações: Programas com Contador de Repetições (LOOPS)	205
Capítulo 19	Instruções de Deslocamentos e Giros do 6502: SHIFT e ROTATE	217
Capítulo 20	Instruções de Desvios (JSR) para Sub-Rotina e Retorno da Sub-Rotina (RTS) para 6502	226
Capítulo 21	Instruções com Flags, Pilhas e Interrupções do 6502	232
Capítulo 22	Exemplo de Assembler: Leitura e Escrita de caracteres através de uma Máquina TTY (TELETYPEWRITER) ou Terminal de TV	236
Parte IV Apêndices		
Apêndice A	O Sistema Binário de Representação de Dados	247
Apêndice B	Instruções do Microprocessador 8080/85 e Z80 em Ordem Alfabética	263
Apêndice C	Instruções do Microprocessador 6502 em Ordem Alfabética	279
Referências		285



P R E F Á C I O

Este livro destina-se a um curso introdutório da linguagem Assembler dos microprocessadores de oito bits:

- INTEL 8080/85,
- ZILOG Z80 e
- APPLE 6502.

Serve também como manual de consulta no uso das instruções desses microprocessadores.

O livro é formado das seguintes partes:

PARTE I – NOÇÕES BÁSICAS DE ARQUITETURA, LINGUAGEM DE MÁQUINA E LINGUAGEM ASSEMBLER DE UM MICROPROCESSADOR.

Apresenta noções e terminologias básicas sobre UCP e linguagem Assembler.

PARTE II – LINGUAGEM ASSEMBLER DOS MICROPROCESSADORES 8080/85 e Z80.

As instruções do 8080/85 e Z80 são apresentadas simultaneamente, lado a lado, onde houver identidade dos códigos de máquinas. Entretanto, os programas-exemplo são apresentados nas duas versões e o texto serve tanto para um curso sobre 8080/85 como para Z80 de modo independente.

PARTE III – LINGUAGEM ASSEMBLER DO MICROPROCESSADOR 6502

As instruções do 6502 são apresentadas de modo independente, porém seguindo a mesma ordem didática de apresentação das instruções do 8080/85 e Z80. Os programas-exemplo são os mesmos da PARTE 2.

Sendo assim, se desejar, o leitor pode efetuar um aprendizado ou estudo comparativo das instruções e programas-exemplo dos três microprocessadores, utilizando o conhecimento e habilidade de programação que já possui em um dos microprocessadores, para conhecer um novo microprocessador.

APÊNDICE – Contém as seguintes informações:

- A** – O SISTEMA BINÁRIO DE REPRESENTAÇÃO DE DADOS;
- B** – INSTRUÇÕES EM ORDEM ALFABÉTICA DO 8080/85 e Z80;
- C** – INSTRUÇÕES EM ORDEM ALFABÉTICA DO 6502.

O uso da linguagem de máquina e Assembler dos microprocessadores torna-se cada vez mais interessante e obrigatório para os programadores mais sofisticados dos microcomputadores ou também para os projetistas de sistemas industriais ou científicos que utilizam esses microprocessadores em suas aplicações.

O livro apresenta, de modo gradual e didático, o uso das instruções de máquina e Assembler, sempre acompanhado de exemplos ilustrados e programas-exemplos comentados, de modo que o leitor ou estudante possa utilizá-lo como texto de consulta complementar de programação, juntamente com um texto sobre o **HARDWARE** desses microprocessadores.

Para aplicações mais sofisticadas, principalmente usando as instruções de Pilha, Entrada/Saída ou Interrupções que requerem conhecimentos adicionais sobre o hardware, o leitor deverá consultar textos mais detalhados como:

- 6502 ASSEMBLY LANGUAGE PROGRAMMING de Lance A. Leventhal,
 - 8080/85 ASSEMBLY LANGUAGE PROGRAMMING do mesmo autor e
 - Z80 ASSEMBLY LANGUAGE PROGRAMMING também do mesmo autor,
- publicados pela OSBORNE/MCGRAW-HILL.

Para tornar compatível o uso deste livro com as referências acima citadas, muitos dos programas-exemplos são adaptações comentadas e ilustradas de exemplos neles citados, efetuadas com a devida permissão da Editora McGraw-Hill do Brasil Ltda.

Gostaríamos de agradecer à Editora McGraw-Hill do Brasil que incentivou e forneceu todo o apoio necessário para a publicação deste livro.

Agradecemos, também, a colaboração dos srs. engenheiros/analistas Carlos Sanchez Favalli, Marcos Ishikawa e Roberto Kiyoshi Yamada nos testes dos programas-exemplos e na utilização do texto original em seus cursos de programação.

O AUTOR

PARTE

I

**NOÇÕES GERAIS DE MICROPROCESSADOR E
LINGUAGEM ASSEMBLER**

CAPÍTULO

I

NOÇÕES GERAIS DE ARQUITETURA DA UCP DO COMPUTADOR E MICROCOMPUTADOR A LINGUAGEM DE MÁQUINA

I.1 ARQUITETURA OU ORGANIZAÇÃO INTERNA DA UCP

É formada por diversas peças ou registradores que manipulam valores binários e desempenham todas as operações previstas pela UCP – Unidade Central de Processamento.

Registradores ou registros são, portanto, locais especiais do *hardware* que servem para guardar um ou vários *bits* para determinada operação.

Uma arquitetura típica e simples contém os seguintes componentes ou registros:

- contador de instruções (CI) ou Program counter (PC);
- memória principal;
- acumulador (AC) ou diversos registradores de dados A, B etc.;
- unidade lógico-aritmética (ULA);
- registro de índices X, Y;
- registro de código de condição ou Status Flag;
- decodificador de instrução;
- circuitos que acionam a execução de cada instrução decodificada;
- Pilha ou Stack.

Muitos registros possuem denominação especial como acumulador, decodificador etc. A memória pode ser considerada como registro especial que contém número elevado de *bits*.

A arquitetura dos computadores e microcomputadores existentes é uma versão ampliada ou modificada da arquitetura básica.

1.2 INSTRUÇÃO DE MÁQUINA

É o conjunto de valores binários que foram codificados de tal modo que representem todas as operações possíveis executadas pela UCP. Cada instrução de máquina ocupa um ou mais *bytes* (conjunto de oito *bits*) da memória e é formada de duas partes principais:

- *código* de operação, que identifica o tipo de operação a ser executada;
- *operandos* ou *endereços*, que contêm os valores ou endereços dos dados que entram na operação.

O conjunto de instruções varia de UCP para UCP, mas basicamente é formado por:

- instruções de transferência de dados da memória para registradores e vice-versa (MOVE, LOAD ou STORE),
- instruções de operações aritméticas e lógicas (INCREMENT, DECREMENT, ADD, SUBTRACT, AND, OR, EOR) etc.;
- instruções de comparação (COMPARE);
- instruções de desvios (JUMP, BRANCH, CALL etc.);
- instruções de deslocamentos e giros (SHIFT e ROTATE);
- instruções de entrada e saída de dados (entre memória e terminais).

1.3 OS PRINCIPAIS COMPONENTES DA UCP

Contador de instrução ou de programa (CI ou PC = Program Counter)

É o registro ou componente que contém o valor binário do endereço numérico ou posição da instrução de máquina que está sendo executado. Após a execução da instrução, o contador muda automaticamente de valor e indica o endereço da próxima instrução a ser executada.

Acumulador (AC)

O valor do dado que foi trazido da memória passa para o acumulador para receber um dos tratamentos previstos pela instrução.

O acumulador usa o dado para:

- deslocamentos (*shifts*) ou rotação de *bits* para a direita ou esquerda;
- adição ou subtração: contém uma parcela e recebe o resultado da operação aritmética ou lógica;
- escrever ou ler: guardar um valor (*byte*) que foi lido ou será escrito pelo Terminal.

Unidade Lógico-Aritmética (ULA)

É o componente binário que executa, em conjunto com AC ou outros registros, as operações aritméticas (adição e subtração binária) e operações lógicas (verificação de sinais e composição de *bits* através de operação do tipo AND, OR, NOT etc.).

A multiplicação ou divisão binária geralmente é efetuada por circuitos (hardware) ou sub-rotinas (software) adicionais.

Registro de Dados A, B etc.

São registros menos utilizados do que o acumulador e servem, em geral, para efetuar operações que auxiliam a manipulação de dados pelo acumulador. Por exemplo, esses registros podem guardar temporariamente os dados do acumulador ou memória, ou podem ser incrementados ou decrementados, servindo como contador etc. Não são, em geral, usados para operações aritméticas ou lógicas, deslocamentos etc.

Se existir o registro A, geralmente trata-se do próprio acumulador.

Registro de Índices X, Y

São registros que contêm uma das partes (denominada ÍNDICE) que entram na formação (dizemos INDEXAÇÃO) do endereço final ou efetivo da memória a ser usado como operando de uma instrução. Podem também ser usados em instrução de carregar, armazenar, somar valores que serão usados como endereços.

Decodificador

É um componente ou circuito que recebe os bits do código de operação da instrução de máquina, identifica o tipo de operação a ser executada, acionando com valor UM o bit ou sinal que comanda a execução dessa instrução.

Pilha ou Stack

É formado por porção da memória (o tamanho aumenta ou diminui de acordo com o uso das instruções CARREGA ou ARMAZENA valor da pilha), cujo endereço está acessível através do registro chamado PONTEIRO DA PILHA OU STACK POINTER (SP). Serve para receber valores através da operação LAST-IN-FIRST-OUT ou LIFO, e que é apropriada para guardar endereço de retorno após a execução de uma sub-rotina, salvar conteúdo de registros etc.

Ponteiro da Pilha ou Stack Pointer (SP)

É um registro da UCP e que guarda o endereço ou posição corrente disponível na pilha definida na memória. Pode ser decrementado ou incrementado através de instruções apropriadas.

COMPONENTES DA UCP

Um esquema de relacionamento simplificado dos principais componentes da UCP encontra-se na figura I.1 seguinte:

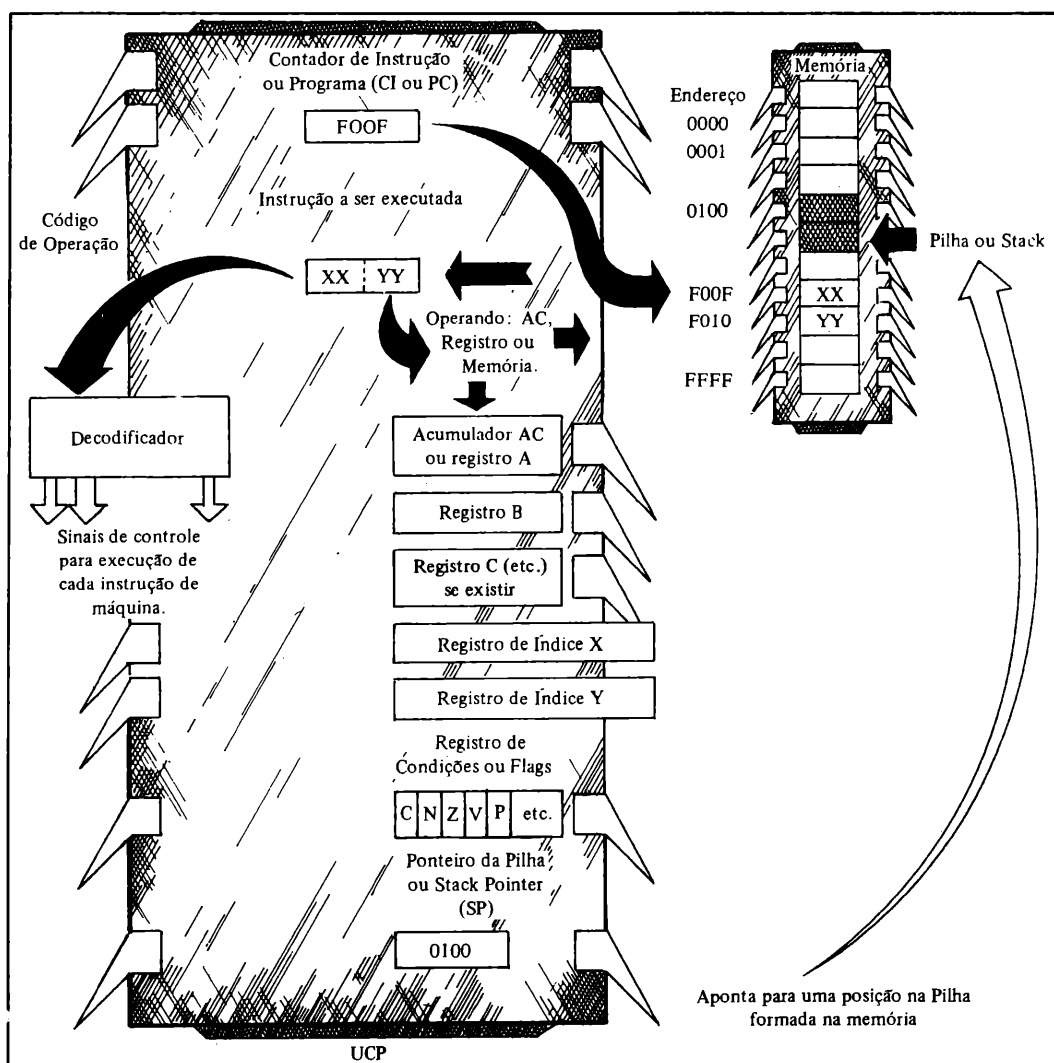


Figura I.1 Componentes de uma UCP do microprocessador.

I.4 REGISTRO DE CÓDIGOS DE CONDIÇÃO OU BANDEIRA DE "STATUS" (STATUS FLAGS)

É um registro onde cada um dos seus bits (chamados de FLAGS) indica, através do valor UM ou ZERO, o estado ou condição de um determinado fenômeno ocorrido durante a execução de uma instrução.

Os principais códigos ou Flags de condição são:

- Flag ou código de condição C=1 ou 0 — indica que ocorreu OU não ocorreu o fenômeno CARRY ou VAI-UM.
- Flag ou código de condição V=1 ou 0 — indica que ocorreu OU não ocorreu o transbordo ou OVERFLOW na operação executada.
- Flag N ou S=1 ou 0 — indica que o valor obtido pela instrução é negativo (bit 7 do valor obtido é UM) OU positivo (bit 7 do valor obtido é ZERO).
- Flag Z=1 ou 0 — indica que o valor obtido pela instrução é zero OU diferente de zero.
- Flag P=1 ou 0 — indica que o valor obtido pela instrução contém um número PAR (PARITY EVEN) de bits OU um número ÍMPAR (PARITY ODD) de bits.
- Flag D=1 ou 0 — Se D=1, toda operação aritmética (ADD e SUBTRACT) é executada no MODO DECIMAL, isto é, os valores que entram na operação são usados em formato BCD (Binary Coded Decimal).

I.5 CONFIGURAÇÃO DE UM MICROCOMPUTADOR SIMPLES: EXEMPLO

O microcomputador estudado é bastante simples e possui a seguinte estrutura básica:

- a) UCP ou Unidade Central de Processamento que possui um Acumulador ou Registrador de Operação Aritmética com capacidade para 8 bits e onde são efetuadas as operações aritméticas e as comparações;
- b) Memória — com 256 palavras de 8 bits cada uma, e numeradas de 000 a 255 ou de 00 a FF em notação hexadecimal;
- c) terminal (teclado com tela de TV);
- d) impressora (Figura I.2).

Este microcomputador trabalha com dois tipos de instruções de máquina:

- **instrução de uma palavra:** que contém apenas o código de operação;
- **instrução de duas palavras:** que contém o código de operação e um operando variável, que é um endereço de memória (de 00 a FF).

Conjunto de instruções de máquina: exemplo

O conjunto de operações desta máquina simples é:

INSTRUÇÃO DE UMA PALAVRA			O que faz
Mnemônico	Binário	Hexa	
ZERA	0000 0001	01	– coloca valor zero no Acumulador
STOP	0000 0010	02	– suspende a execução do programa
CHS	0000 0011	03	– muda sinal (CHANGE SIGN) do valor do Acumulador
INSTRUÇÃO DE DUAS PALAVRAS			O que faz
LDA	0000 0100	04	– carrega (LOAD) no Acumulador o valor que está no endereço indicado pelo operando
STO	0000 0101	05	– operação inversa (STORE) do LDA
ADD	0000 0110	06	– soma valor do Acumulador com valor que está no endereço indicado pelo operando
SUB	0000 0111	07	– <i>idem</i> , subtração
BRA	0000 1000	08	– desvia (BRANCH) a execução do programa para o endereço indicado pelo operando
BZER	0000 1001	09	– <i>idem</i> , só se o conteúdo do Acumulador for zero, caso contrário, prossegue na instrução seguinte
BPOS	0000 1010	0A	– <i>idem, idem</i> , se o Acumulador for positivo
BNEG	0000 1011	0B	– <i>idem, idem</i> , se o Acumulador for negativo

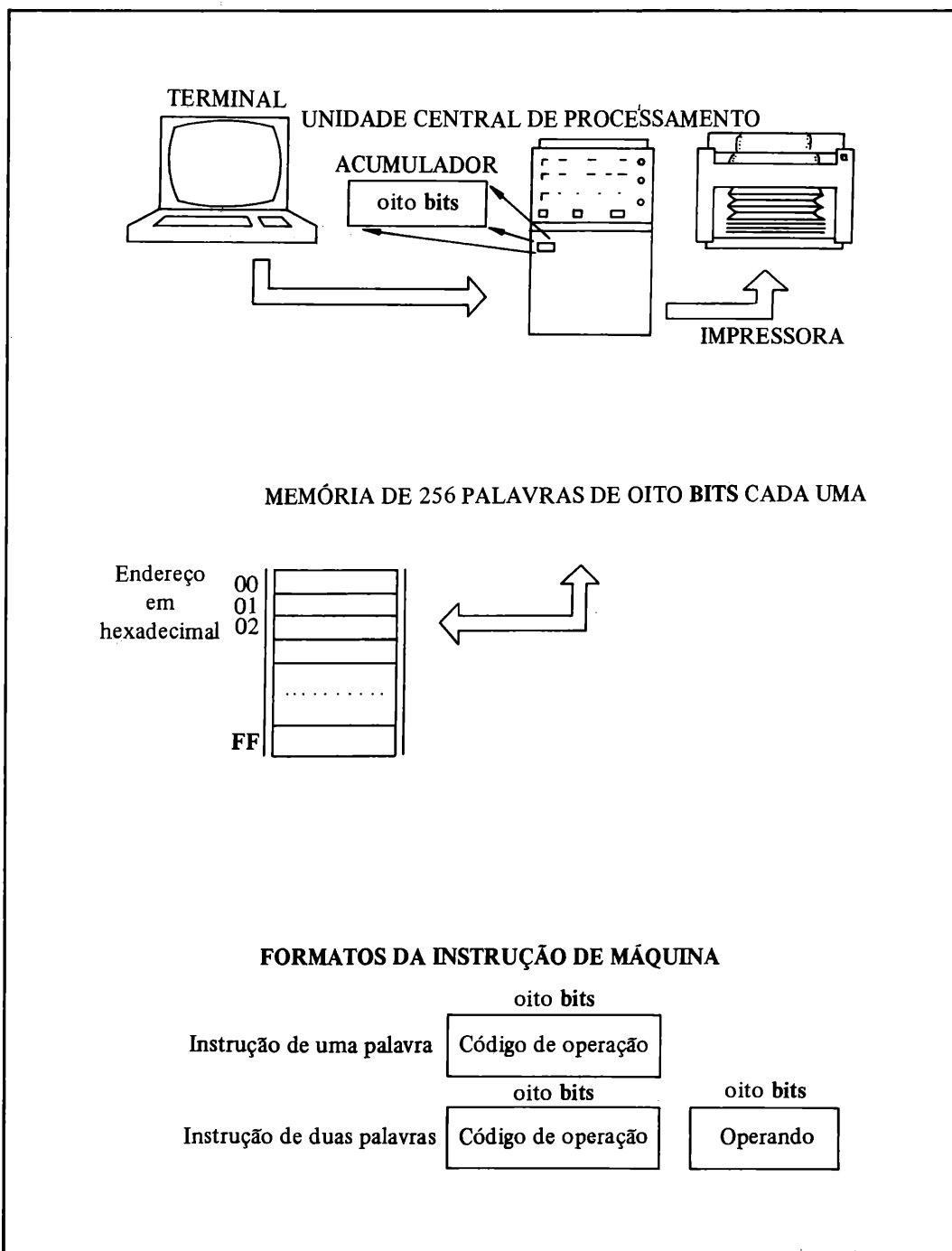


Figura I.2 Configuração de um microcomputador simples.

Uso das instruções de máquina (notação hexadecimal)

Analisemos os seguintes exemplos de instrução de máquina do microcomputador estudado.

Instrução de máquina	Significado	Resultados: antes da execução	e	Acumulador (AC) depois da instrução	Memória afetada
00	Coloca zero no AC	F1		00	não afeta
03	Troca sinal de AC (valor negativo ou complemento de dois)	01		FF	não afeta
04 F1	Carrega AC com o que está no endereço F1 da memória (por exemplo: F1 contém 10)	FF		10	Endereço F1 = 10 não se altera
05 F1	Guarda valor do AC: endereço F1 recebe o valor que está em AC	FF		FF	Endereço F1 = FF
06 F1	Soma conteúdos de AC e do endereço F1; o resultado fica em AC (exemplo: F1 contém 30) e AC = 10; soma 10 + 30 = 40 fica em AC.	10		40	Endereço F1 = 30
08 30	Desvia para o endereço 30, não afetando AC				(não se altera)

Exercício

Explicar o que fazem as seguintes instruções:

07 F1 com (F1 contendo 01 e AC contendo valor 03) *.
 07 F1 *idem* (F1 contendo 08 e AC contendo FF)
 03 *idem*, considerando valor de AC da instrução anterior
 05 F2
 04 F1
 0A 25

* Subtração equivale a somar “complemento de dois” do valor do operando ao AC.

1.6 TIPOS BÁSICOS DE ENDEREÇAMENTO DOS OPERANDOS

Já vimos que uma instrução de máquina possui o código de operação e o operando opcional.

O operando de uma instrução de máquina fornece à UCP a “informação” necessária (localização na memória, valor etc.) de um valor ou dado que participa na execução dessa instrução.

Essa “informação” para obter o valor definitivo do dado que entra na execução de uma instrução chama-se *modo de endereçar (ou localizar) um dado*.

As formas básicas de endereçamento são:

Endereçamento direto

A instrução de máquina contém como operando o *endereço real* (também chamado *endereço absoluto*) da memória onde está o valor do dado desejado. O conjunto de instrução da máquina dada só permite endereçamento direto. *Exemplo:*

LDA	F1
04	F1

- Carregar (LOAD) o Acumulador com o valor que está no endereço F1 da memória.

Endereçamento imediato

O próprio valor que está na instrução de máquina deve ser usado como operando. Se esse tipo de endereçamento for permitido, a máquina deve ter códigos próprios de instrução. *Exemplo:*

LDIM \$F1

- Carregar (LOAD IMMEDIATE) (\$) significa operando imediato Acumulador com o próprio valor F1.

Endereçamento indireto

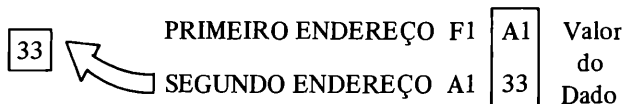
O operando da instrução contém um endereço de memória (por exemplo, F1) cujo conteúdo vai ser usado como segundo endereço (por exemplo, F1 contém o valor A1).

O segundo endereço (A1) será o *endereço final ou efetivo* onde está localizado o dado (por exemplo, 33). *Exemplo:*

- LDIN: LOAD INDIRECT
LDIN F1

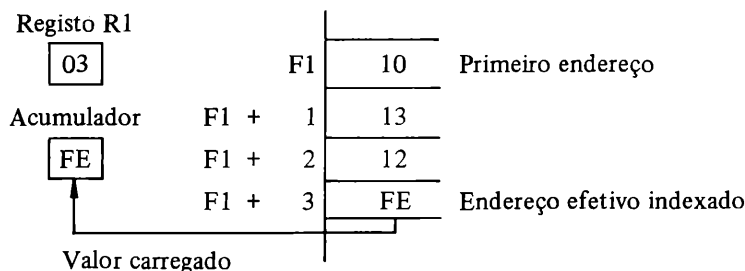
Endereço

ACUMULADOR

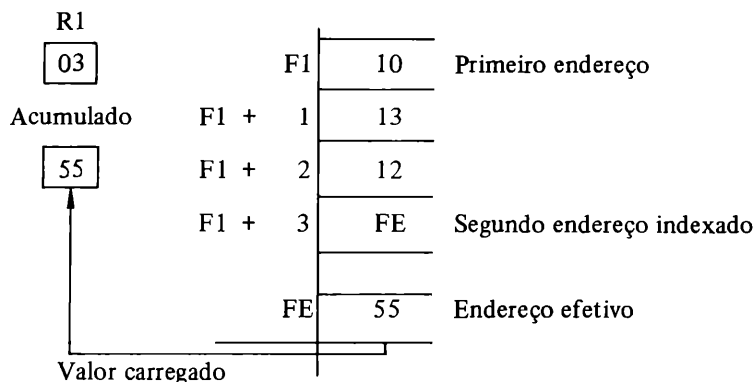
**Endereçamento indexado**

O endereço (direto ou indireto) que está na instrução é *indexado* ou *modificado* pelo conteúdo de um *registrador* próprio para esse fim e chamado *registro de índice*. Em geral, o conteúdo do registro é somado ao endereço (direto ou indireto) para se obter o endereço efetivo do dado. *Exemplos:*

- a) LDA F1, R1 • Carregar Acumulador com o conteúdo do endereço F1 indexado (ou modificado) pelo registro de índice R1.



- b) LDIN F1, R1 • Indexação de um endereço indireto.



Algumas UCP permitem outra alternativa: indexado-indireto, isto é, indexar o conteúdo do endereço (no caso, conteúdo de F1 = 10) e obter o endereço efetivo $10 + 3 = 13$.

Outras formas ou formas básicas modificadas de endereçamento podem ser encontradas nas diferentes UCP e microprocessadores.

Os operandos imediatos não permitem a indexação.

EXERCÍCIOS

De recapitulação

1. Dar o conceito e exemplos ilustrativos das seguintes expressões:

Arquitetura da UCP
Instrução de máquina
Conjunto de instruções
Código de operação
Operando
Contador de instrução
AC – Acumulador
ULA – Unidade Lógico-Aritmética
Registro de instruções e índices
Decodificador
Endereçamento direto
Endereçamento imediato
Endereçamento indireto
Endereçamento indexado

De aplicação

(Nota – Utilizar as instruções do computador simples dado como exemplo.)

1. O endereço F1 e F2 da memória contém respectivamente os valores AA e BB em hexadecimal. Dizer o que faz cada uma das instruções de máquina exibindo o conteúdo do Acumulador e posições afetadas da memória:

ZERA
CHS
LDA F1
STO F3
ADD F2
SUB F2
STO F3
BPOS F7

2. Os endereços de memória contêm os seguintes valores:

Endereço F1 contém E1
Endereço F2 contém E3
Endereço F3 contém 03
Endereço F4 contém 07
Endereço E1 contém 05
Endereço E2 contém 0A
Endereço E3 contém FF

Dizer o que faz cada uma das seguintes instruções executadas em seqüência.
Exibir os conteúdos da memória e acumulador:

LDA F1
ADDIN F2 (indireto)
STO F2, R1 (R1 contém valor 2)
LDIM 02 (imediato)
ADD F2, R1
STO F2

3. Projeto: estudar e escrever um relatório sobre a arquitetura (mencionando as características do AC, PC, registradores principais), o conjunto de instruções (até 30 instruções) e os modos de endereçamentos de uma UCP ou microprocessador que você conhece. Exemplo:

Z-80	INTEL 8085,	INTEL 8086
M6800,	M6900,	MC68000 (Motorola)
IBM,	Burroughs,	PDP

CAPÍTULO

II

NOÇÕES GERAIS DE LINGUAGEM ASSEMBLER ELEMENTOS BÁSICOS

II.1 INSTRUÇÕES EM LINGUAGEM ASSEMBLER (OU ASSEMBLY)

É trabalhoso manipular instruções de máquinas que usam códigos binários. Usamos normalmente uma versão mnemônica ou *simbólica* dessas instruções.

Essa versão é denominada Linguagem ASSEMBLER, ASSEMBLY ou de Montador, pois, antes da execução, é preciso traduzir ou montar cada instrução simbólica em instrução de máquina. Exemplos:

<i>Instrução de máquina</i>	<i>Instrução mnemônica ou ASSEMBLER</i>
1011 1111	ADD X
1111 0000	STOP
1010 0011	SHIFT 3
0010 0000	CLEAR

Nota: “ASSEMBLY” significa montagem e indica a característica da linguagem, “ASSEMBLER” significa montador e indica o programa tradutor. Ambos os termos são usados sem distinção.

II.2 ELEMENTOS DE UMA INSTRUÇÃO ASSEMBLER: RÓTULO, CÓDIGO DE INSTRUÇÃO, OPERANDOS E COMENTÁRIOS

Temos os seguintes elementos em uma instrução ASSEMBLER.

- a) *Endereço simbólico*
ou Rótulo ou Label: identificador simbólico de localização da instrução. É formado por 1 a 6 letras e usado opcionalmente quando o programador quer voltar a essa instrução ou usá-la como operando.

- b) *Código mnemônico de operação:* formado por um conjunto de letras que indicam o significado da instrução.
- c) *Operandos:* elementos usados na execução da instrução. Pode ser endereço simbólico ou rótulo de outra instrução (X1, Y2, VAI) ou valor constante (5 ou 1). Há instruções que não usam operando (como o ZERA, END etc.). Uma instrução pode usar mais de um operando.
- d) *Comentários:* de uso opcional para orientar o programador.

Exemplo:

Rótulo	Código mnemônico	Operando	Comentário
INÍCIO	LDA	X	Carrega Acumulador com valor da posição X

II.3 TIPOS DE INSTRUÇÃO ASSEMBLER: INSTRUÇÕES DE MÁQUINA EM FORMA SIMBÓLICA, "DIRECTIVES" OU PSEUDO-INSTRUÇÕES E MACROINSTRUÇÕES

Podemos identificar três tipos de instrução:

- a) **Instrução de uma palavra:** quando traduzida fornece instrução de máquina que ocupa uma só palavra ou **byte** na memória. Possui somente o código da operação.
- Instrução de duas ou mais palavras:** possui um código de operação e um operando e será convertida em duas ou mais palavras ou **bytes** de instrução de máquina.
- b) **Pseudo-instrução ou instrução diretiva (Directive):** não é convertida em instrução de máquina, mas define valor ou certa função especial necessária ao programa, tais como:
- definir o valor de uma constante (instrução DB ou DC) em uma **byte** da memória;
 - indica fim do programa (instrução END).
- c) **Macroinstrução:** instrução definida pelo próprio programador que codifica certas operações novas, usando um conjunto de instruções assembler simples. **Exemplo:**
- MADD M1, M2, M3: macroinstrução que soma três valores.

Para definir macroinstrução, o tradutor deve ser um *Macroassembler* e não um ASSEMBLER comum.

II.4 CARACTERÍSTICAS DAS DIFERENTES VERSÕES DE PROGRAMAS EM INSTRUÇÃO DE MÁQUINA E EM ASSEMBLER.

Vamos analisar as seguintes versões diferentes de um mesmo programa.

Os códigos de linguagem de máquina, os códigos mnemônicos e as instruções em Assembler são de um microprocessador ou computador hipotético. A lógica ou raciocínio do programa não deve ser levado em consideração, pois queremos discutir apenas as formas diferentes de programas em Assembler.

Versão 1 Programa escrito em linguagem de máquina: códigos de operação e operandos em notação binária hexadecimal.

Endereço Absoluto da Memória	Código Objeto
0000	01
0001	02
0002	0F
0003	01
0004	05
0005	1A
0006	00
0007	04
0008	03
0009	00
000A	40
000B	C9

Notação — em código hexadecimal.

Características deste programa

- O programa está na forma executável pelo microprocessador e também colocado em endereços absolutos definitivos da memória.
- É uma forma difícil de ser entendida pelo programador, pois não há uma distinção visível entre os códigos de operação e os operandos da instrução.

Versão 2 Programa assembler apenas com códigos de operação em forma mnemônica.

Características deste programa

- Já é possível saber claramente o significado de cada instrução, através dos respectivos códigos mnemônicos e também saber o tamanho de cada instrução.
- Ainda depende muito da posição ou endereço onde é carregado, pois mudando o posicionamento do programa na memória, há alterações em certas instruções e o programador precisa controlar essa alteração.

Endereço Absoluto	Código Objeto	Instrução Assembler		Comentários	Número de bytes que ocupa
		Mnemônico	Operandos		
0000	01	CLR	A	— limpa Acumulador A	1
0001	02 OF 01	LD	A,F01H	— conteúdo do endereço F01 (em hexadecimal) vai para Acumulador	3
0004	05	DCR	A	— decrementa de um o valor do Acumulador	1
0005	1A 00 04	JNZ	0004H	— se operação anterior não for zero desvia para endereço 0004	3
0008	03 00 40	ADD	A,40H	— soma ao acumulador o conteúdo do endereço 40 (em hexadecimal)	3
000B	C9	RET		— retorna ao ponto onde o programa foi chamado	1
0040	06			— endereço a ser usado como operando em ADD. valor inicial é 6	1
0F01	0A			— idem, operando usado na instrução LD, valor inicial é 0A	1

Exemplo: se o programa for carregado a partir da posição F000, a instrução JNZ 0004H deve ser mudada para JNZ F004H.

- Usa operandos em posições fixas da memória e o programador precisa saber se realmente essas posições estão disponíveis ou não.

Exemplo: LD A,F01H e ADD A,40H

- O programador pode programar, instrução por instrução, esta versão do Assembler (código mnemônico e operando absoluto), converter em código objeto e executá-lo diretamente no microprocessador. A conversão pode ser feita manualmente ou através de um programa tradutor de Assembler passo a passo, pois todos os endereços usados são absolutos e fixos.
- Esta versão não é recomendável para programas mais extensos.

Versão 3 Programa em Assembler usando endereços e operandos simbólicos

Endereço Absoluto Atribuído	Código Objeto	Endereço Simbólico ou Rótulo	Código Mnemônico	Operandos	Comentários
0000	01		CLR	A	— limpa acumulador A
0001	02 00 0C		LD	A,VALOR	— carrega acumulador com o conteúdo do endereço simbólico VALOR
0004	05	REPETE	DCR	A	— decrementa conteúdo do Acumulador A
0005	1A 00 04		JNZ	REPETE	— se A não for zero desvia para o endereço simbólico REPETE
0008	03 00 0D		ADD	A,VALOR 1	— se A for zero, soma conteúdo de VALOR 1 em A
000B	C9		RET		— Retorno
000C	06	VALOR	DB	06	— Instrução diretiva (ou pseudo-instrução) que define valor constante 06 nesta posição da memória, ou endereço simbólico VALOR
000D	0A	VALOR 1	DB	0A	— <i>idem</i> , define valor 0A e é atribuído ao endereço simbólico VALOR 1
000E	—		END		— fim do programa em Assembler

Características deste programa

- O programador não se preocupa mais com o endereço definitivo dos desvios (como em JNZ REPETE) ou dos operandos (como VALOR e VALOR 1), os quais são atribuídos automaticamente pelo programa tradutor ASSEMBLER.

Exemplos:

- a) O endereço simbólico REPETE será colocado no endereço 0004, pois o Assembler inicia a montagem da instrução a partir de 0000.
- b) Os endereços de VALOR e VALOR 1 *não* serão mais F01 e 40, pois o Assembler atribui automaticamente os endereços disponíveis que vêm a seguir à instrução RET. Sendo assim,

VALOR será colocado no endereço 000C (após RET) e

VALOR 1 será colocado no endereço seguinte 000D.

- A PSEUDO-INSTRUÇÃO (“DIRECTIVE” em inglês) ou INSTRUÇÃO DIRETIVA é uma instrução especial de *orientação* ao tradutor ASSEMBLER e serve para designar determinadas operações de montagem da instrução ou código de máquina.

Exemplos: VALOR DB OFH – reserva na posição da memória com o nome VALOR o conteúdo inicial OF em hexadecimal.

END – indica fim do programa ASSEMBLER

IMPORTANTE – A pseudo-instrução ou “diretiva” NÃO faz parte da instrução de máquina do microprocessador ou do computador, sendo apenas uma instrução de orientação ao tradutor Assembler.

- O código-objeto ainda será montado a partir do endereço 000, fornecendo códigos idênticos ao da versão 2, com exceção para VALOR e VALOR 1.

Versão 4 Programa em Assembler usando instrução diretiva *ORG* (define origem) para montar o programa-objeto a partir de um endereço desejado.

INSTRUÇÃO DIRETIVA “ORG XXXX” – indica ao Assembler que as instruções subseqüentes serão colocadas a partir do endereço XXXX. Esta instrução diretiva não será traduzida em código objeto.

Endereço Absoluto Atribuído	Código Objeto	Rótulo	Código Mnemônico	Operandos	Comentários
–	–		ORG	500H	– Instruções serão colocadas a partir do endereço 500 (em hexadecimal)
0500	01		CLR	A	–
0501	02 OF 00		LD	A, VALOR	–
0504	05	REPETE	DCR	A	–
0505	1A 05 04		JNZ	REPETE	–
0508	03 OF 01		ADD	A, VALOR 1	–
050B	C9		RET		–
			ORG	FOOH	– Próximas instruções serão colocadas de F00 em diante
0F00	06	VALOR	DB	06	–
0F01	0A	VALOR1	DB	OAH	–
			END		

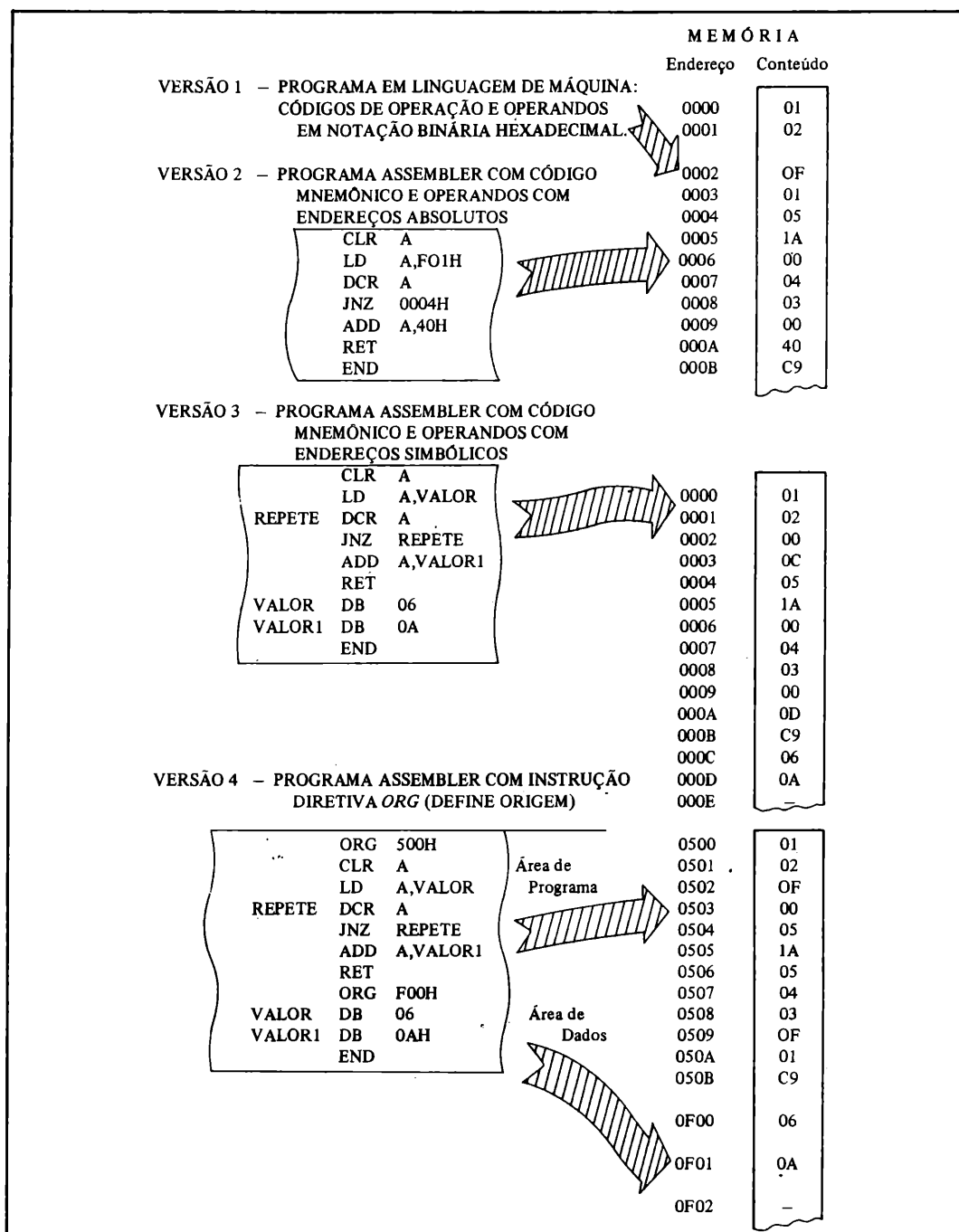


Figura II.1 CARACTERÍSTICAS DAS DIFERENTES VERSÕES DE PROGRAMA ASSEMBLER E EM LINGUAGEM DE MÁQUINA.

Características deste programa

- É possível montar instruções de máquinas ou dados do programa a partir de qualquer endereço da memória, separando o programa em ÁREA DE PROGRAMA, DE LEITURA, DE CÁLCULO, DE DADOS etc.
- Sempre é possível utilizar endereços absolutos caso seja necessário.
- Todos os endereços simbólicos são automaticamente atribuídos e ajustados pelo Assembler.

II.5 ALGUNS TIPOS DE PSEUDO-INSTRUÇÕES (“DIRECTIVES”)

Até agora tivemos a oportunidade de usar apenas algumas pseudo-instruções. As principais pseudo-instruções são:

DB, DC ou DW: Definir valor constante **k** na memória.

Define **Byte**: DB 5 • Coloca valor 5 em um **byte** da memória.

Define **word**: DW 33 • Coloca valor 33 em uma palavra de 16 bits.

Define constante: DC 15 • Análogo a DB ou DW.

END: Definir fim do programa.

DS: Definir **n** espaços na memória ou definir área (define **spaces**).

Exemplo:

AREAX DS 50 • A partir do local AREAX define ou reserva 50 posições ou **bytes** na memória.

AREAZ DS FAH • Idem, FA (hexadecimal) posições.

ORG: Definir origem ou novo valor do contador de instruções.

Exemplo:

ORG FOH • Iniciar contador com valor hexadecimal F0.
LDA X Esta instrução será montada na posição F0.

...

...

ORG FBH • Dá novo valor FB ao contador. Instruções
ADD X seguintes serão montadas a partir da posição
STO Y FB.

EQU: Definir equivalência entre nomes ou endereços.

Exemplo:

LABEL EQU XIS • LABEL recebe mesmo endereço de XIS.

XIS EQU 100 • XIS recebe endereço 100.

```

      ...
INÍCIO EQU AAH • INÍCIO recebe valor hexadecimal AA.
      ORG INÍCIO • Programa será montado a partir do endereço
                        INÍCIO ou AA.

START LDA X

```

Pergunta:

Qual será o endereço a ser atribuído ao rótulo START?

Exemplos de instruções diretivas em microprocessadores

6502	8080/85	Z80	6800 / 6809	Explicação
BYT ou .BYTE 10 (*)	DB 10	DEFB 10	FCB 10	– Define ou coloca na memória o valor decimal 10 de UM BYTE
BYT ou .BYTE \$10	DB 10H	DEFB 10H	FCB \$10	– <i>Idem</i> , valor hexadecimal 10
DBY ou .DBYTE \$FA10	DW FA10H	DEFW FA10H	FDB \$FA10	– Define ou coloca na memória valor hexadecimal FA10 ocupando DOIS BYTES ou UMA PALAVRA
.END	END	END	END	– Define o fim do programa em Assembler
ORG ou *=\$F000	ORG F000H	ORG F000H	ORG \$F000	– Define ORIGEM do programa a ser montado; todas as instruções seguintes serão traduzidas e colocadas a partir do endereço F000, a menos que outra instrução do tipo ORG seja utilizada
Z1 EQU ou *=\$F000	Z1 EQU F000H	Z1 EQU F000H	Z1 EQU \$F000	– Rótulo ou endereço simbólico Z1 será colocado, obrigatoriamente, no endereço F000
DFS ou *=+ 10	DS 10	DEFS 10	RMB 10	– Define ou reserva a partir deste ponto na memória, uma área de dez bytes.

Nota: (*) Dependendo da versão do programa Montador ou Assembler, ambas ou apenas uma das formas de instrução diretiva é aceita. O programador deve consultar o Manual Assembler 6502 que está sendo usado.

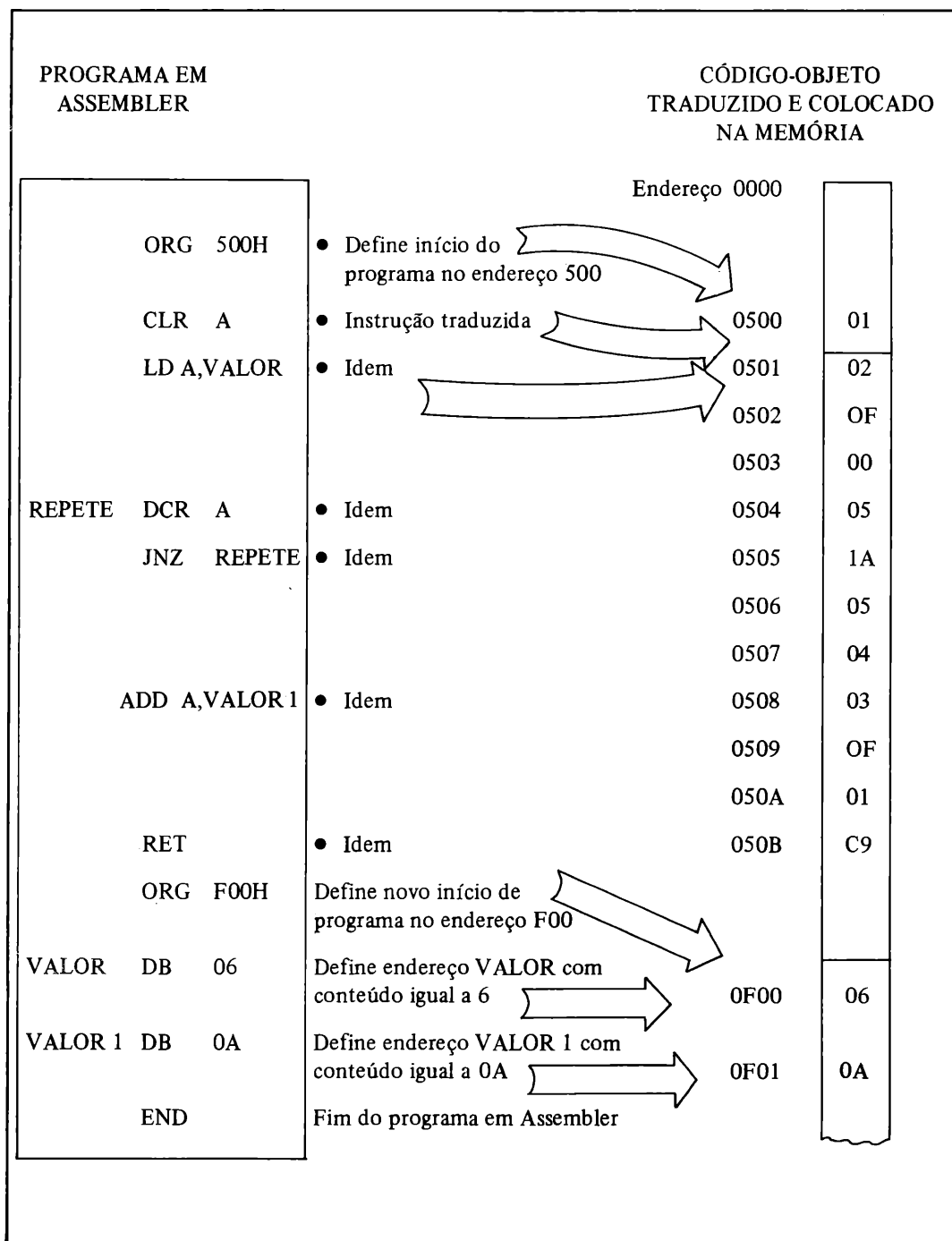


Figura II.2 INSTRUÇÕES SIMBÓLICAS E DIRETIVAS DE UM PROGRAMA ASSEMBLER.

II.6 MACROASSEMBLERS

São versões expandidas de um tradutor ou montador Assembler capazes de ler e traduzir, além das instruções Assembler e as pseudo-instruções, as macroinstruções.

Macroinstruções

São instruções simbólicas definidas pelo próprio usuário e que serão traduzidas pelo tradutor em mais de uma instrução (simbólica ou de máquina). Assim, o usuário pode definir e utilizar, em forma de uma única macroinstrução, determinada operação composta por mais de uma instrução de computador. O conceito de macroinstrução equivale ao de uma sub-rotina.

Definição de uma macroinstrução

O exemplo seguinte ilustra a definição e uso de macroinstrução.

Exemplo:

Definir uma macroinstrução que efetue a soma de três valores da memória.

Formato da macroinstrução:

Rótulo	Código	Operandos
	MADD	X, Y, Z

Significado: uma macroinstrução que soma (MADD) os valores *X*, *Y* e *Z* e coloca resultado em *X*.

Uso: em qualquer ponto do programa ASSEMBLER basta usar essa macroinstrução para somar três valores quaisquer:

```

MADD          VALOR1, VALOR2,  VALOR3
...
MADD          X + 1, CÓDIGO, A
...
```

Definição da parte executável de macroinstrução

O próprio usuário deve escrever (uma única vez no programa) a parte executável da macroinstrução utilizando as instruções em Assembler comum. Essas definições são geralmente colocadas no fim do programa em Assembler.

A definição da macroinstrução MADD X, Y, Z seria:

MADD	MACRO X, Y, Z	• indica início da definição, nome e parâmetros da macroinstrução
	LDA X ADD Y ADD Z STO X MEND	• instruções de execução • fim da definição

Como é efetuada a tradução das macroinstruções

Se forem utilizadas macroinstruções, o tradutor Assembler requer uma passagem adicional de tradução a fim de converter todas as macroinstruções utilizadas em instruções Assembler simples, após o que a tradução ou montagem normal é efetuada. *Exemplo:*

Programa com uso de macroinstrução

```

STO .
MOVE
MADD V1, V2, V3 (primeira macroinstrução)
ADD CODIGO
...
MADD X + 1, Z, VALOR (segunda macroinstrução)
...
MADD MACRO X, Y, Z (definição da macroinstrução MADD)
LDA X
ADD Y
ADD Z
STO X
MEND
(outras definições de macroinstrução)
END

```

Programa Assembler comum (após traduzir as macroinstruções)

```

...
STO
MOVE
{ LDA V1
  ADD V2
  ADD V3
  STO V1
}
ADD CODIGO
...
{ LDA X + 1
  ADD Z
  ADD VALOR
  STO X + 1
  END
}

```

EXERCÍCIOS

De recapitulação

1. Quais são os elementos de uma instrução em Assembler?
2. Quais são os tipos de instrução Assembler?
3. Dizer o que faz e de que tipo são as instruções ADD, CLR, DCR, LD, JNZ, END, ORG e DB utilizadas nos exemplos.
4. Dar exemplos de instrução Assembler, pseudo-instrução e macroinstrução.

De aplicação

1. Dizer, instrução por instrução, o que faz cada um dos programas seguintes, citando as variações dos conteúdos do Acumulador e dos endereços afetados:

a) INIC ZERA
 ADD CINC
 ADD UM
 CHS
 STOP
 CINC DB 5
 UM DB 1
 END

b) INIC LDA DEZ
 BPOS SIM
 Y STOP
 SIM SUB X
 BNEG Y
 CHS
 STOP
 DEZ DB 10
 X DB 15
 END

c) ORG 2500H
 INIC LDA Z1
 ADD Z2
 STO TEMP
 SUB QNZE
 BPOS SEG
 STOP
 SEG LDA TEMP
 STO TOTAL
 STOP

 Z1 DB 8
 Z2 DB 9
 TOTAL DB 1
 QNZE DB 15
 TEMP DB 0
 END

2. Traduzir, para linguagem de máquina com os respectivos endereços absolutos, os programas em Assembler a, b e c da questão 1. Utilizar código-objeto das instruções de um COMPUTADOR SIMPLES, dado como exemplo, no Capítulo I.

PARTE

II

**LINGUAGEM ASSEMBLER DOS
MICROPROCESSADORES 8080/85 E Z80**

CAPÍTULO

III

ELEMENTOS BÁSICOS DA ARQUITETURA DO MICROPROCESSADOR 8080/85 E Z80

III.1 ARQUITETURA OU ORGANIZAÇÃO INTERNA DO 8080/85 E Z80

Os principais elementos que interessam aos programadores em linguagem de máquina e Assembler, no 8080/85 e Z80, são:

8080/85

bit 7	PSW						0	7		0
S	Z		AC		P		C		A	
B								C		
D								E		
H								L		

bit 15																0
PROGRAM COUNTER (PC)																
STACK POINTER (SP)																

Z-80

bit 7	F						0	7		0
S	Z		AC		PO	N	C		A	
B								C		
D								E		
H								L		

bit 15																0
PROGRAM COUNTER (PC)																
STACK POINTER (SP)																
INDEX REGISTER IX																
INDEX REGISTER IY																

bit	7	0				7	0			
	F'				A'					
	B'				C'					
	D'				E'					
	H'				L'					

Registros Alternativos

8080/85	Z -80
– Registrador de status PSW com os flags: - S ou Sinal: indica resultado negativo ou não-negativo. - Z ou Zero: indica resultado zero - AC (Aux. Carry): indica vai-um ocorrido entre bits 3 e 4 - P ou Paridade: indica número par de bits no resultado - C ou Carry: indica a ocorrência de Carry no resultado da operação – Registro ou Acumulador A (8 bits) – Registros B, C, D, E, H e L (8 bits) – Contador de Programa PC (16 bits) – Ponteiro de Pilha SP (16 bits)	– Registrador de status F com os flags: - S ou Sinal: IDEM - Z ou Zero: IDEM - AC (Aux. carry): IDEM - P/O Paridade/Overflow: indica paridade ou overflow - N ou Subtração: indica ocorrência de subtração - C ou Carry: IDEM – Registro ou Acumulador A (8 bits) – Registro B, C, D, E, H e L (8 bits) – Contador de Programa PC (16 bits) – Ponteiro de Pilha SP (16 bits) – Registro de Índice IX ou X (16 bits) – Registro de Índice IY ou Y (16 bits) – Registros Alternativos contendo: - A', B', C', D', E', H', L' (8 bits) – Registro Alternativo de Status F' contendo S', Z', AC', P/O', C', N'

NOTA-Z80: Os REGISTRADORES ALTERNATIVOS são registros adicionais que podem ser utilizados como “reserva ou regra-três” através das instruções:

EX AF, AF' – Troca conteúdo de FA e FA' e, a partir daí, opera com o par FA' até que outra instrução EX AF, AF' seja usada, voltando portanto ao uso de FA.

EXX – Troca conteúdo de BC, DE e HL com respectivos pares alternativos, e a UCP passa a operar com estes registros alternativos, até que outra instrução EXX seja usada, voltando portanto a utilizar os registradores BC, DE e HL.

Notação sobre os bits de flags

Os bits de flags ou status podem ou não ser afetados pela instrução. Quando forem afetados pela instrução, a seguinte notação será usada:

- “X” — Indica que o flag é afetado pelo resultado da operação. Ver estados possíveis dos flags afetados, nos exemplos ilustrados.
- “1” — Indica que o flag assume valor um, após a execução da instrução.
- “0” — Indica que o flag assume valor zero, após a execução da instrução.
- “Ø” — Indica que o flag não é afetado pela instrução.

III.2 INSTRUÇÃO EM ASSEMBLER E MODOS DE ENDEREÇAMENTO DOS OPERANDOS

No Z80 ou 8080/85 temos as seguintes formas possíveis de instruções Assembly, diferenciadas através de seus operandos:

Instrução sem operando

Quando nenhuma forma normal (acumulador, registro ou memória) de operando é utilizada.

Exemplos:

- | | |
|-----|--|
| RET | — Retorna de Sub-rotina. |
| NOP | — Não executa operação servindo apenas para preencher espaço na memória. |

Operando é um ou dois registros

O uso de registros ou acumuladores pode ser indicado diretamente pelo código mnemônico da instrução, ou então mencionado explicitamente na instrução.

Exemplos:

- | 8080/85 | Z80 | |
|----------|------------|---|
| CMA | CPL | — Complementa conteúdo de A. |
| INR B | INC B | — Incrementa conteúdo de B. |
| MOV A, B | LD A, B | — Transfere conteúdo de B para A. |
| MOV M, B | LD (HL), B | — Transfere conteúdo de B para posição da memória cujo endereço está no par HL. |

Operando imediato

É um valor numérico (decimal, hexadecimal, binário ou octal) ou alfabético (em código ASCII) que aparece na instrução e é diretamente utilizado pela instrução.

- a) 8080/85 — Neste caso o operando imediato é identificado através do código mnemônico da instrução que utiliza em geral uma letra “I” adicional.

Exemplos:

MVI B,10 06 0A (objeto) — Valor decimal 10 vai para B.
ADI 10H C6 10 (objeto) — Soma com A o valor hexadecimal 10.

- b) Z80 — No Assembler, o valor imediato é identificado quando aparece SEM O USO DE PARÊNTESES (com o uso de parênteses indica endereço direto da memória). O código-objeto gerado é igual ao do 8080/85.

Exemplos:

LD B,10 06 0A (objeto) — Valor decimal 10 vai para B.
ADD A,10H C6 10 (objeto) — Soma A com o valor hexadecimal 10.

Endereço direto da memória

Podemos citar dois casos:

- a) ENDEREÇO DA MEMÓRIA ESTÁ NO PAR HL, BC OU DE

Exemplos:

8080/85	Z80	
MOV A, M	LD A, (HL)	— Valor da memória, cujo endereço está em HL, vai para A.
LDAX B	LD A, (BC)	— <i>Idem</i> , quando endereço está no par BC.
LDAX D	LD A, (DE)	— <i>Idem</i> , quando endereço está no par DE.

- b) ENDEREÇO DA MEMÓRIA É USADO NA INSTRUÇÃO

Exemplos:

LDA Z1 LD A, (Z1) — Conteúdo do endereço Z1 vai para A.

Endereço indexado (só para Z80)

O endereço efetivo é obtido pela soma do conteúdo do registro X ou Y e o valor ou endereço de oito bits (chamado deslocamento ou offset) que aparece na instrução.

Ilustração:**a) Operando Imediato**

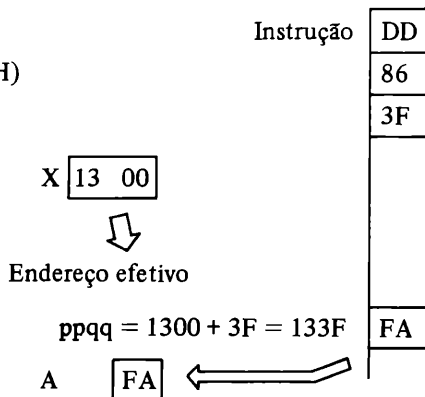
8080/85	Z80	
MVI B, 10H	LD B, 10H	B 10 ← valor hexadecimal 10

Endereço Direto da Memória

b)	MOV A, M	LD A, (HL)	HL	13 F0	memória
			A	10 ← Endereço 13F0	10
	LDA Z1	LD A, (Z1)	A	10 ← Endereço Z1	10

c) Endereço Indexado (só Z80)

— ADD A, (IX + 3FH)

**Figura III.1** EXEMPLOS DE ENDEREÇAMENTO DO 8080/85 e Z80.**Exemplos:**

ADD A, (IX + 3FH) DD 86 3F (objeto) — Soma A com conteúdo do endereço X + 3F.

ADD A, (IY + 3FH) FD 86 3F (objeto) — Soma A com conteúdo do endereço Y + 3F.

ADD A, (IX + Z1) — O Assembler verifica se o endereço simbólico Z1 está na posição 00 a FF. Consultar o Manual Assembler em uso.

Endereço relativo das instruções de desvio (só para Z80)

Além da forma comum de desvio para endereço ppqq mencionado na instrução, o Z80 permite também o uso de DESVIOS RELATIVOS, onde o tradutor Assembler transforma o endereço absoluto ou simbólico de desvio para um deslocamento ou offset ds de oito bits que aparece no código-objeto. O endereço de desvio deve estar dentro do limite de -126 a +129 bytes de posições a contar da posição corrente dessa instrução.

Exemplos:

JR Z1	18 ds	(objeto)	— Desvia para endereço Z1; deslocamento ds é calculado pelo Assembler.
JR C, \$F011	38 ds	(objeto)	— Desvia para endereço absoluto \$F011 que deve estar em posição adequada para possibilitar o cálculo correto do valor ds.

III.3 PRINCIPAIS INSTRUÇÕES DIRETIVAS DO ASSEMBLER

As principais instruções diretivas aceitas pela linguagem Assembler são:

a) DEFINE VALORES CONSTANTES

8080/85	Z80	
DB	DEFB	— Define na memória um valor constante de um byte.
DW	DEFW	— Define na memória um valor constante de uma palavra ou dois bytes.
	DEFM	— Define na memória uma série (strings) de caracteres em código ASCII.

b) DEFINE ÁREA OU POSIÇÕES VAGAS NA MEMÓRIA

DS	DEFS	— Define, a partir destas instruções, um certo número k de posições, na memória.
----	------	--

c) DEFINE IGUALDADE DE ENDEREÇOS OU SÍMBOLOS

EQU	EQU	— Atribui (EQUATE) o endereço dado ao símbolo ou rótulo indicado na instrução.
-----	-----	--

d) DEFINE ORIGEM DE PROGRAMA OU ÁREA

ORG	ORG	— Define endereço de origem (ORG) ou início para colocar códigos de máquina.
-----	-----	--

e) DEFINE FIM DO PROGRAMA ASSEMBLER

END	END
-----	-----

Exemplos:

8080/85			Z80			Explicação
INIC	EQU	1000H	INIC	EQU	1000H	— Endereço simbólico INIC será colocado no endereço absoluto 1000H em hexadecimal.
	ORG	INIC		ORG	INIC	— Todas as instruções seguintes serão colocadas a partir da origem ou endereço INIC = 1000H.
V1	DB	10	V1	DEFB	10	— Endereço V1 conterá valor decimal 10.
V2	DW	F1F2H	V2	DEFW	F1F2H	— Endereço V2 conterá o byte F2 e V2 + 1, o byte F1: primeiro, o byte menos significativo.
V3	DB	10101111B		DEFB	10101111B	— Endereço V3 conterá o valor binário 1010111 equivalente a AF em hexadecimal.
CODIG	DB	'FÓRMULA-1'	CODIG	DEFM	'FÓRMULA-1'	— Coloca, nas nove posições seguintes, a partir do endereço CODIG, os caracteres F, 0, R, M, U, L, A, —, e 1 em ASCII.
ÁREA	DS	9	ÁREA	DEFS	9	— Reserva nove bytes ou posições, sem nada colocar, a partir do endereço ÁREA.

EXERCÍCIOS

De recapitulação

1. Explicar para que servem cada um dos seguintes elementos básicos do microprocessador 8080/85:

- Registro A ou Acumulador A
- Registros B, C, D, E, H e L
- Contador de Programas PC
- Ponteiro de Pilha SP
- Flags ou bits S, Z, Ac, P e C.

Sugestão: consultar noções gerais dos elementos da arquitetura de um microprocessador, Capítulo I.

2. Exemplificar e explicar cada uma das seguintes formas de instrução Assembler do microprocessador 8080/85 ou Z80:

- Instrução sem operando
- Instrução que operam com registros
- Instruções com operandos imediatos
- Instruções com endereços diretos da memória
- Instruções com endereço indexado
- Instruções com endereço relativo

3. Explicar e exemplificar cada uma das principais instruções diretivas do Assembler 8080/85 ou Z80.

CAPÍTULO

IV

INSTRUÇÕES DE MOVIMENTO OU TRANSFERÊNCIA DE DADOS (MOVE, LOAD OU STORE) DO 8080/85 E Z80

IV.1 FORMAS GERAIS DAS INSTRUÇÕES

As instruções tratadas neste capítulo são:

CARREGA REGISTRO OU MEMÓRIA COM OPERANDO IMEDIATO dd: MOVE OU LOAD IMMEDIATE DATA.

FORMAS GERAIS:

8080/85 Z80

MVI reg, dd	LD reg, dd	— Valor dd vai para registro.
MVI M, dd	LD (HL), dd	— Valor dd vai para memória cujo endereço está no par HL.
	LD (IX + ds), dd	— Valor dd vai para memória, cujo endereço é obtido por IX + ds.
	LD (IY + ds), dd	— Idem, endereço é obtido por IY + ds.

MOVE OU TRANSFERE DADOS DE UM REGISTRO PARA OUTRO: MOVE OR LOAD REGISTER TO REGISTER

MOV reg _i , reg _j	LD reg _i , reg _j	— Valor do registro j vai para registro i.
---	--	--

TROCA CONTEÚDO DE PARES DE REGISTROS (EXCHANGE REGISTER PAIRS)

XCHG	EX DE, HL	— Troca conteúdos de HL e DE.
XTHL	EX (SP), HL	— Troca conteúdos de HL com topo da pilha.
	EX e EXX	— Troca conteúdos de HL, BC, DE com conteúdos de registros alternativos HL', BC', DE' (só Z80).

MOVE TRANSFERE DADOS DA MEMÓRIA PARA REGISTRO: MOVE OR LOAD MEMORY TO REGISTER.

MOV reg, M	LD reg, (HL)	— Conteúdo da memória, cujo endereço está em HL, vai para registro.
	LD reg, (IX + ds)	— Memória ppqq + ds para registro.
	LD reg, (IY + ds)	— Memória ppqq + ds para registro.
LDAX B	LD A, (BC)	— Memória com endereço em BC, para A.
LDAX D	LD A, (DE)	— Memória com endereço em DE, para A.
LDA endereço	LD A, (endereço)	— Conteúdo de “endereço” vai para A.

MOVE OU TRANSFERE DADOS DO REGISTRO PARA MEMÓRIA: MOVE OU STORE MEMORY.

MOV M, reg	LD (HL), reg	— Registro para memória, com endereço em HL.
	LD (IX + ds), reg	— <i>Idem</i> para memória ppqq + ds, ppqq em X.
	LD (IY + ds), reg	— <i>Idem</i> , para memória ppqq + ds, ppqq em Y.
STAX B	LD (BC), A	— De A para memória com endereço em BC.
STAX D	LD (DE), A	— De A para memória com endereço em DE.

MOVE DADOS DE 16 BITS

LXI regs, ddee	LD regs, ddee	— Valor ddee vai para o par de registros.
LHLD endereço	LD regs, (endereço)	— Valor em endereço e endereço + 1 para par de registros.
SHLD endereço	LD (endereço), regs	— Operação inversa da anterior.

TRANSFERÊNCIA DE DADOS DE MEMÓRIA PARA MEMÓRIA — só Z80.

—	LDI	— Conteúdo da memória endereçada por HL vai para memória endereçada por DE, incrementando HL, DE.
	LDIR	— Repetição de LDI, decrementando BC até zero.
	LDD	— Análogo a LDI, mas decrementando HL e DE.
	LDDR	— Repetição de LDD, decrementando BC até zero.

— PROGRAMAS-EXEMPLOS

Além dos exemplos ilustrados de cada instrução, o capítulo contém os seguintes programas-exemplos:

- TRANSFERÊNCIA DE DADOS USANDO ENDEREÇOS ABSOLUTOS
- TRANSFERÊNCIA DE DADOS USANDO ENDEREÇOS SIMBÓLICOS
- TRANSFERÊNCIA DE DADOS USANDO ENDEREÇOS CONSECUTIVOS DA MEMÓRIA

IV.2 MOVE OR LOAD IMMEDIATE

CARREGAR REGISTRO OU MEMÓRIA COM OPERANDO IMEDIATO

Valor imediato dd vai para registro R_i ou memória.

Formas gerais:	8080/85	Z80
	MVI reg_i , dd	LD reg_i , dd
	MVI M, dd	LD (HL), dd
		LD (IX + ds), dd
		LD (IY + ds), dd

8080/85 Assembler	Código objeto	Z80 Assembler	Código objeto	Explicação
MVI A, dd	3E dd	LD A, dd	3E dd	– Valor imediato dd vai para A.
MVI B, dd	06 dd	LD B, dd	06 dd	– <i>Idem</i> , para B.
MVI C, dd	0E dd	LD C, dd	0E dd	– <i>Idem</i> , para C.
MVI D, dd	16 dd	LD D, dd	16 dd	– <i>Idem</i> , para D.
MVI E, dd	1E dd	LD E, dd	1E dd	– <i>Idem</i> , para E.
MVI H, dd	26 dd	LD H, dd	26 dd	– <i>Idem</i> , para H.
MVI L, dd	2E dd	LD L, dd	2E dd	– <i>Idem</i> , para L.
MVI M, dd	36 dd	LD (HL), dd	36 dd	– <i>Idem</i> , para endereço ppqq; HL contém ppqq.
		LD (IX + ds), dd	DD 36 ds dd	– <i>Idem</i> , para endereço ppqq + ds; ppqq está em X.
		LD (IY + ds), dd	FD 36 ds dd	– <i>Idem</i> , para endereço ppqq + ds; ppqq está em Y.

IV.3 MOVE OR LOAD REGISTER TO REGISTER

TRANSFERE (MOVE OU CARREGA) DADOS DE UM REGISTRO PARA OUTRO REGISTRO.

Formas gerais: 8080/85

Z80

MOV reg_j, reg_i

LD reg_j, reg_i

8080/85	Z80	Explicação
<i>De registro Ri para A:</i>		
MOV A, A 7F	LD A, A 7F	– Move (ou carrega) A para A.
MOV A, B 78	LD A, B 78	– Move B para A.
MOV A, C 79	LD A, C 79	– Move C para A.
MOV A, D 7A	LD A, D 7A	– Move D para A.
MOV A, E 7B	LD A, E 7B	– Move E para A.
MOV A, H 7C	LD A, H 7C	– Move H para A.
MOV A, L 7D	LD A, L 7D	– Move L para A.
<i>De registro Ri para B</i>		
MOV B, A 47	LD B, A 47	– Move conteúdo de A para B.
MOV B, B 40	LD B, B 40	
MOV B, C 41	LD B, C 41	
MOV B, D 42	LD B, D 42	
MOV B, E 43	LD B, E 43	
MOV B, H 44	LD B, H 44	
MOV B, L 45	LD B, L 45	
<i>De registro Ri para C</i>		
MOV C, A 4F	LD C, A 4F	– Move conteúdo de A para C.
MOV C, B 48	LD C, B 48	
MOV C, C 49	LD C, C 49	
MOV C, D 4A	LD C, D 4A	
MOV C, E 4B	LD C, E 4B	
MOV C, H 4C	LD C, H 4C	
MOV C, L 4D	LD C, L 4D	

8080/85	Z80	Explicação
<i>De registro Ri para D</i>		
MOV D, A 57	LD D, A 57	— Move conteúdo de A para D.
MOV D, B 50	LD D, B 50	
MOV D, C 51	LD D, C 51	
MOV D, D 52	LD D, D 52	
MOV D, E 53	LD D, E 53	
MOV D, H 54	LD D, H 54	
MOV D, L 55	LD D, L 55	
<i>De registro Ri para E</i>		
MOV E, A 5F	LD E, A 5F	— Move (ou transfere) conteúdo de A para E.
MOV E, B 58	LD E, B 58	
MOV E, C 59	LD E, C 59	
MOV E, D 5A	LD E, D 5A	
MOV E, E 5B	LD E, E 5B	
MOV E, H 5C	LD E, H 5C	
MOV E, L 5D	LD E, L 5D	
<i>De registro Ri para H</i>		
MOV H, A 67	LD H, A 67	— Move conteúdo de A para H.
MOV H, B 60	LD H, B 60	
MOV H, C 61	LD H, C 61	
MOV H, D 62	LD H, D 62	
MOV H, E 63	LD H, E 63	
MOV H, H 64	LD H, H 64	
MOV H, L 65	LD H, L 65	
<i>De registro Ri para L</i>		
MOV L, A 6F	LD L, A 6F	— Move conteúdo de A para L.
MOV L, B 68	LD L, B 68	
MOV L, C 69	LD L, C 69	
MOV L, D 6A	LD L, D 6A	
MOV L, E 6B	LD L, E 6B	
MOV L, H 6C	LD L, H 6C	
MOV L, L 6D	LD L, L 6D	

IV.4 EXCHANGE REGISTER PAIR

TROCA CONTEÚDO DOS PARES DE REGISTROS CITADOS. Não afeta flags.

8080/85	Z80		Explicação
	EX AF, AF'	08	– Troca conteúdo do par AF = (Acumulador + Registro F de Flags) com o par AF' = (Registros alternativos A' + F').
XCHG EB	EX DE, HL	EB	– Troca conteúdo do par HL com DE.
XTHL E3	EX (SP), HL	E3	– Troca conteúdo do par HL com posições no topo da pilha, troca L com topo da pilha e H com topo da pilha mais um.
	EX (SP), IX	DD E3	– Troca conteúdo de IX com o par de posições do topo da pilha.
	EX (SP), IY	FD E3	– Troca conteúdo de IY com o par de posições do topo da pilha.
	EX X	D9	– Troca conteúdo de BC, DE e HL com o conteúdo dos registros alternativos B'C', D'E e H'L', respectivamente.

Nota— Após o uso da instrução EX AF, AF' ou EXX, há a troca de conteúdo dos registros indicados e todas as instruções seguintes passam a operar com esses registros alternativos, até que uma outra instrução EX AF, AF' ou EXX seja usada, e os registros AF ou B, C, D, E, H, L passem a estar novamente disponíveis às instruções.

IV.5 MOVE OR LOAD REGISTER TO MEMORY

TRANSFERÊNCIA DE DADOS (MOVE ou LOAD) DE MEMÓRIA PARA REGISTRO

Formas gerais:	8080/85	Z80
	MOV reg _i , M	LD reg _i , (HL)
		LD reg _i , (IX + ds)
		LD reg _i , (IY + ds)
	LDAX B	LD A, (BC)
	LDAX D	LD A, (DE)
	LDA endereço	LD A, (endereço)

8080/85	Z80		Explicação
MOV A, M 7E	LD A, (HL) 7E		— Conteúdo do endereço ppqq vai para A; ppqq está em HL.
	LD A, (IX + ds) DD 7E ds		— Conteúdo do endereço ppqq + ds vai para A; ppqq está em X.
	LD A, (IY + ds) FD 7E ds		— Conteúdo de ppqq + ds vai para A; ppqq está em Y. — Demais casos são análogos ao do registro A.
MOV B, M 46	LD B, (HL) 46		— Conteúdo de endereço ppqq vai para B.
	LD B, (IX + ds) DD 46 ds		
	LD B, (IY + ds) FD 46 ds		
MOV C, M 4E	LD C, (HL) 4E		— Conteúdo do endereço ppqq vai para C.
	LD C, (IX + ds) DD 4E ds		
	LD C, (IY + ds) FD 4E ds		
MOV D, M 56	LD D, (HL) 56		— Conteúdo do endereço ppqq vai para D.
	LD D, (IX + ds) DD 56 ds		
	LD D, (IY + ds) FD 56 ds		
MOV E, M 5E	LD E, (HL) 5E		— Conteúdo do endereço ppqq vai para E.
	LD E, (IX + ds) DD 5E ds		
	LD E, (IY + ds) FD 5E ds		
MOV H, M 66	LD H, (HL) 66		— Conteúdo do endereço ppqq vai para H.
	LD H, (IX + ds) DD 66 ds		
	LD H, (IY + ds) FD 66 ds		
MOV L, M 6E	LD L, (HL) 6E		— Conteúdo do endereço ppqq vai para L.
	LD L, (IX + ds) DD 6E ds		
	LD L, (IY + ds) FD 6E ds		
LDAX B 0A	LD A, (BC) 0A		— Conteúdo do endereço ppqq vai para A; ppqq está em BC.
LDAX D 1A	LD A, (DE) 1A		— <i>Idem</i> , ppqq está em DE.
LDA endereço 3A qq pp	LD A, (endereço) 3A qq pp		— Conteúdo do endereço ppqq vai para A.

IV.6 MOVE OR LOAD (STORE) REGISTER TO MEMORY

TRANSFERÊNCIA DE DADOS (MOVE, LOAD ou STORE) DE UM DADO DO REGISTRO PARA A MEMÓRIA.

Formas Gerais:	<i>8080/85</i>	<i>Z80</i>
	MOV M, reg _i	LD (HL), reg _i LD (IX + ds), reg _i LD (IY + ds), reg _i
	STAX B	LD (BC), A
	STAX D	LD (DE), A
	STA endereço	LD (endereço), A

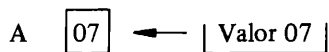
<i>8080/85</i>	<i>Z80</i>	<i>Explicação</i>
MOV M, A 77	LD (HL), A 77	— Move conteúdo de A para endereço ppqq; ppqq está em HL.
	LD (IX + ds), A DD 70 ds	— Move A para endereço ppqq + ds; ppqq está em X.
	LD (IY + ds), A FD 77 ds	— Move A para endereço ppqq + ds; ppqq está em Y.
MOV M, B 70	LD (HL), B 70	— Move B para memória ppqq.
	LD (IX + ds), B DD 70 ds	— (Demais casos são análogos a A.)
	LD (IY + ds), B FD 70 ds	
MOV M, C 71	LD (HL), C 71	— Move C para memória ppqq.
	LD (IX + ds), C DD 71 ds	
	LD (IY + ds), C FD 71 ds	
MOV M, D 72	LD (HL), D 72	— Move conteúdo de registro D para memória.
	LD (IX + ds), D DD 72 ds	
	LD (IY + ds), D FD 72 ds	
MOV M, E 73	LD (HL), E 73	— Move conteúdo de E para memória.
	LD (IX + ds), E DD 73 ds	
	LD (IY + ds), E FD 73 ds	

8080/85	Z80	Explicação
MOV M, H 74	LD (HL), H 74	– Move conteúdo de H para memória.
	LD (IX + ds), H DD 74 ds	
	LD (IY + ds), H FD 74 ds	
MOV M, L 75	LD (HL), L 75	– Move conteúdo de L para memória.
	LD (IX + ds), L DD 75 ds	
	LD (IY + ds), L FD 75 ds	
STAX B 02	LD (BC), A 02	– Conteúdo de A vai para endereço ppqq; ppqq está em BC.
STAX D 12	LD (DE), A 12	– <i>Idem</i> , ppqq está em DE.
STA endereço 32 qq pp	LD (endereço), A 32 qq pp	– Conteúdo de A vai para endereço ppqq.

Exemplos

– Carregar registro ou memória com operando imediato.

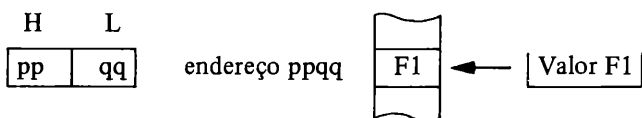
- 8080/85 Z80
- a) MVI A, 7 LD A, 7 • Operando imediato 7 vai para registro A.



- b) MVI L, 0F1H LD L, 0F1H • Valor hexadecimal F1 vai para registro L.



- c) MVI M, 0F1H LD (HL), 0F1H • Valor hexadecimal F1 vai para endereço ppqq da memória; ppqq está em HL.



– Mover valor de um registro para outro.

- d) MOV A, A LD A, A • Valor de A vai para A, isto é, nada acontece.

- e) MOV A, B LD A, B • Valor do registro B vai para registro A.

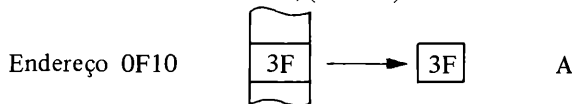


f) MOV H, L LD H, L • Valor de L vai para H.

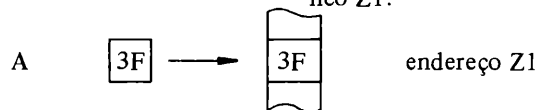


– Transferência de dados entre um registro e memória ou vice-versa (8 bits).

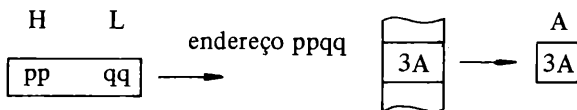
g) LDA 0F10H LD A, (0F10H) • Conteúdo do endereço 0F10 vai para A.



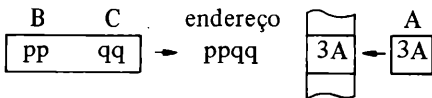
h) STA Z1 LD (Z1), A • Conteúdo do registro A vai para endereço simbólico Z1.



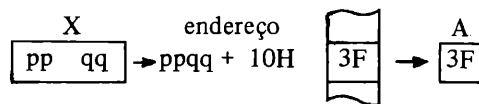
i) MOV A,M LD A, (HL) • Conteúdo do endereço ppqq vai para A.
HL contém ppqq.



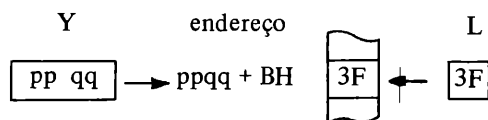
j) STAX B LD (BC), A • Conteúdo de A vai para endereço ppqq.
BC contém endereço ppqq.



k) (só Z80) LD A, (IX + 10H) • Conteúdo do endereço ppqq + 10H vai para A.
Registro X contém ppqq.



l) (só Z80) LD (IY + 11), L • Conteúdo do registro L vai para o endereço.
ppqq + BH (11 em decimal).
Registro Y contém ppqq.



IV.7 LOAD OR STORE 16 BITS

TRANSFERÊNCIA DE DADOS (MOVE, LOAD E STORE) DE 16 BITS.

Formas gerais:	8080/85	Z80
	LXI regs, ddee	LD regs, ddee
	LHLD endereço	LD regs, (endereço)
	SHLD endereço	LD (endereço), regs

CARREGA IMEDIATO PAR DE REGISTROS OU STACK POINTER COM OPERANDO IMEDIATO DE 16 BITS. Flags não são afetados.

8080/85	Z80	Explicação
LXI B, ddee 01 ee dd	LD BC, ddee 01 ee dd	— Valor ddee vai para par BC
LXI D, ddee 11 ee dd	LD DE, ddee 11 ee dd	— <i>Idem</i> , para DE.
LXI H, ddee 21 ee dd	LD HL, ddee 21 ee dd	— <i>Idem</i> , para HL.
LXI SP, ddee 31 ee dd	LD SP, ddee 31 ee dd	— <i>Idem</i> , stack pointer SP.
	LD IX, ddee DD 21 ee dd	— <i>Idem</i> , registro de índice X.
	LD IY, ddee FD 21 ee dd	— <i>Idem</i> , registro de índice Y.

TRANSFERÊNCIA DE DADOS (DE 16 BITS) entre PAR DE REGISTROS e DUAS POSIÇÕES ppqq e ppqq + 1 da MEMÓRIA. Flags não são afetados.

LHLD endereço 2A qq pp	LD HL, (endereço) 2A qq pp	— Conteúdo do endereço (ppqq) vai para L; conteúdo de endereço + 1 (ppqq + 1) para H.
	LD BC, (endereço) ED 4B qq pp	— Análogo com par BC.
	LD DE, (endereço) ED 5B qq pp	— Análogo com par DE.
	LD HL, (endereço) ED 6B qq pp	— Executa mesma operação de 2A qq pp.
	LD SP, (endereço) ED 7B qq pp	— Análogo com Stack Pointer SP.
	LD IX, (endereço) DD 2A qq pp	— Análogo com registro X.
	LD IY, (endereço) FD 2A qq pp	— Análogo com registro Y.

SHLD endereço 22 qq pp	LD (endereço), HL 22 qq pp	— Conteúdo de L vai para endereço (ou ppqq); conteúdo de H vai para endereço + 1 (ou ppqq + 1).
	LD (endereço), BC ED 43 qq pp	— Análogo para BC.
	LD (endereço), DE ED 53 qq pp	— Análogo para DE.
	LD (endereço), HL ED 63 qq pp	— Equivalente a 22 pp qq.
	LD (endereço), SP ED 73 qq pp	— Análogo com SP.
	LD (endereço), IX DD 22 qq pp	— Análogo com registro X.
	LD (endereço), IY FD 22 qq pp	— Análogo com registro Y.

Nota: No código-objeto, tanto o valor imediato ddee como o endereço absoluto ppqq ocupam posições invertidas de bytes.

Exemplos:

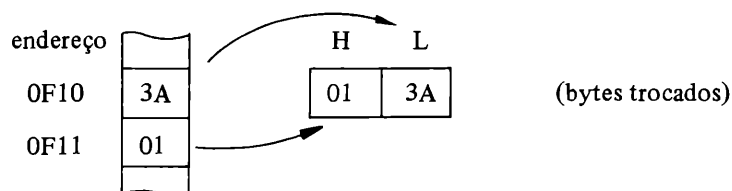
8080/85

Z80

a) LHLD 0F10H

LD HL, (0F10H)

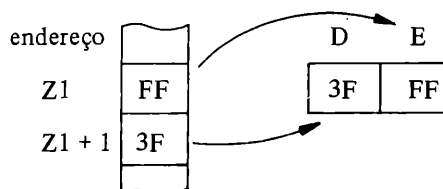
- Conteúdos dos endereços 0F10 e 0F11 vão para registros L e H, respectivamente.



b) --- só Z80

LD DE, Z1

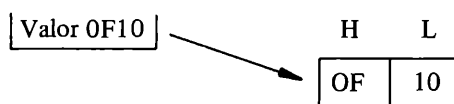
- Conteúdo do endereço simbólico Z1 vai para E e de Z1 + 1 vai para D. Só Z80. Bytes são trocados.



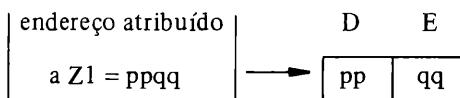
c) LXI H, 0F10H

LD HL, 0F10H

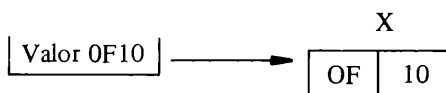
- Valor imediato hexadecimal 0F10H vai para HL.



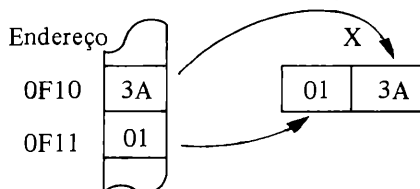
- d) LXI D, Z1 LD DE, Z1 • Valor do endereço ppqq atribuído ao endereço simbólico Z1 vai para o par DE (operando imediato).



- e) só Z80 LD IX, 0F10H • Valor imediato hexadecimal 0F10H vai para X.



- f) só Z80 LD IX, (0F10H) • Conteúdos dos endereços F10 e F11 vão para registro X, com bytes trocados.



IV.8 TRANSFER MEMORY TO MEMORY WITH DECREMENT/INCREMENT AND REPETION

TRANSFERÊNCIA DE DADOS MEMÓRIA A MEMÓRIA SIMPLES OU COM REPETIÇÕES (SÓ Z80)

LDI (TRANSFER MEMORY TO MEMORY WITH INCREMENT) – Transfere dado da memória (endereçado pelo par HL) para a memória (endereçada pelo par DE), com incremento nos pares HL, DE, após a operação e decremento no par BC, mas sem repetição, apenas uma posição da memória.

Flags: Z80: Ac = N = 0; PO = X e S = Z = C = ∅.

8080/85

Z80

— LDI ED A0 —

LDIR (TRANSFER MEMORY TO MEMORY WITH INCREMENT AND REPEAT) – Transfere k posições da memória. Análogo à instrução LDI, mas repete a operação k vezes, até que o par BC que contém inicialmente o valor k torne-se nulo. Após cada operação de transferência, o par HL e DE é incrementado e o par BC que contém o valor k é decrementado, até o valor zero.

Flags: idênticos a LDI.

— LDIR ED B0

LDD (TRANSFER MEMORY TO MEMORY WITH DECREMENT) – Análogo à instrução LDI, mas com decremento no par HL, DE e BC, após a operação de transferência.

Flags: idênticos a LDI.

— **LDD ED A8**

LDDR (TRANSFER MEMORY TO MEMORY WITH DECREMENT AND REPEAT) – Transfere k posições da memória, decrementando os endereços que estão no par HL e DE e decrementando o valor k que foi colocado no par BC. Corresponde à operação LDD repetida k vezes.

Flags: idênticos a LDI.

— **LDDR ED B8**

Exemplos ilustrativos:

a) **LDI** – Transferência simples, com incremento nos pares HL, DE e decremento no par BC.

Antes	Depois
HL = 1114H	HL = 1115H
DE = 2545H	DE = 2546H
BC = 0005H	BC = 0004H
Endereço 1114H xx →	Endereço 2545H xx

b) **LDIR** – Transferência repetida de k posições da memória, com incremento nos pares HL e DE e decremento no par BC.

Antes	Depois
HL = 1114H	HL = 1118H
DE = 2545H	DE = 2549H
BC = 0004H	BC = 0000H
Endereço 1114H xx →	Endereço 2545H xx Primeiro valor transferido
1115H yy →	2546H yy Segundo valor
1116H zz →	2547H zz Terceiro valor
1117H ww →	2548H ww Quarto valor

c) LDD – Transferência simples, com decremento nos pares HL, DE e no par BC.

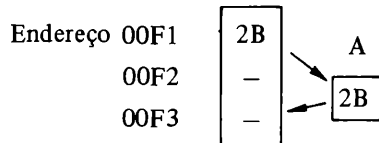
Antes		Depois
HL = 1114H		HL = 1113H
DE = 2545H		DE = 2544H
BC = 0005H		BC = 0004H
Endereço 1114H	xx	Endereço 2545H xx

d) LDDR – Transferência repetida k vezes, com decremento nos pares HL, DE e no par BC.

Antes		Depois
HL = 1114H		HL = 1110H
DE = 2545H		DE = 2541H
BC = 0004H		BC = 0000H
Endereço 1114H	xx	Endereço 2545H xx
1113H	yy	Endereço 2544H yy
1112H	zz	2543H zz
1111H	ww	2542H ww
		Primeiro valor transferido
		Segundo valor
		Terceiro valor
		Quarto valor

IV.9 PROGRAMA-EXEMPLO: TRANSFERÊNCIA DE DADOS USANDO ENDEREÇOS ABSOLUTOS

Transferir o valor 2B que está no endereço absoluto F1 para o endereço absoluto F3.



Versão para 8080/85

Código-Objeto Assembler

```

; TRANSFERENCIA DE DADOS USANDO
; ENDEREÇOS ABSOLUTOS.
;
0000 3AF100  EX8501: LDA    0F1H    ; 1)
0003 32F300      STA    0F3H    ; 2)
0006 C30600  PARE:  JMP    PARE    ; 3)
0009          END    EX8501    ; 4)

```

Comentários:

- 1) Conteúdo do endereço F1 vai para A. No código objeto, endereço 00F1 fica com bytes trocados.
- 2) Conteúdo do registro A vai para o endereço F3.
- 3) Pare. Instrução de repetição infinita PARE: JMP PARE usada no lugar de HALT.
- 4) Fim do programa Assembler.

VERSÃO PARA Z80

Código-Objeto

Assembler

```

; TRANSFERENCIA DE DADOS USANDO
; ENDEREÇOS ABSOLUTOS.
;
1000          ORG    1000H    ; 1)
1000 3AF100  EX01Z80: LD    A,(0F1H) ; 2)
1003 32F300          LD    (0F3H),A  ; 3)
1006 18FE    PARE:   JR     PARE     ; 4)
1008          END     EX01Z80      ; 5)

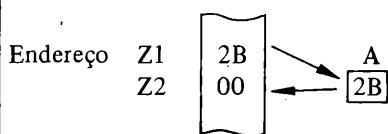
```

Comentários:

- 1) Programa será montado a partir do endereço hexadecimal 1000.
- 2) Conteúdo do endereço absoluto 00F1 vai para A. No código-objeto, endereço 00F1 fica com os bytes trocados.
- 3) Conteúdo do registro A vai para endereço absoluto 00F3.
- 4) Pare: instrução de repetição infinita da mesma instrução usada no lugar de HALT.
- 5) Fim do programa ASSEMBLER.

IV.10 PROGRAMA-EXEMPLO**TRANSFERÊNCIA DE DADOS USANDO ENDEREÇOS SIMBÓLICOS**

Transferir o valor hexadecimal 2B que está no endereço simbólico Z1 para o endereço simbólico Z2.



VERSÃO PARA 8080/85

Código-Objeto

Assembler

```

; TRANSFERENCIA DE DADOS USANDO
; ENDEREÇOS SIMBÓLICOS.
;
0000 3A0900  EX8502: LDA    Z1    ; 1)
0003 320A00      STA    Z2    ; 2)
0006 C30600  PARE:  JMP    PARE  ; 3)
0009 2B      Z1:    DB    2BH   ; 4)
000A 00      Z2:    DB    0     ; 5)
000B          END    EX8502

```

Comentários:

- 1) Conteúdo do endereço simbólico Z1 vai para A.
- 2) Conteúdo de A vai para endereço simbólico Z2.
- 3) Pare.
- 4) Endereço Z1 recebe valor 2B através da instrução diretiva DB.
- 5) Endereço Z2 recebe valor inicial 0, através da instrução diretiva DB.

VERSÃO PARA Z80

Código-Objeto

Assembler

```

; TRANSFERENCIA DE DADOS USANDO
; ENDEREÇOS SIMBÓLICOS.
;
1000          ORG    1000H    ; 1)
1000 3A0810  EX02Z80: LD    A,(Z1) ; 2)
1003 320910      LD    (Z2),A    ; 3)
1006 18FE    PARE:  JR    PARE    ; 4)
1008 2B      Z1:    DEFB  2BH    ; 5)
1009 00      Z2:    DEFB  0      ; 6)
100A          END    EX02Z80

```

Comentários:

- 1) Programa será montado a partir do endereço hexadecimal 1000.
- 2) Conteúdo do endereço simbólico Z1 vai para registro A.

- 3) Conteúdo do registro A vai para endereço simbólico Z2.
- 4) Pare.
- 5) Endereço simbólico Z1 recebe valor 2B, através da instrução diretiva DEFB.
- 6) Endereço simbólico Z2 recebe valor inicial 0.
- 7) Fim do programa.

Observações: Os endereços absolutos foram atribuídos aos endereços simbólicos Z1 e Z2 automaticamente pelo programa ASSEMBLER.

IV.11 PROGRAMA-EXEMPLO

TRANSFERÊNCIA DE DADOS PARA ENDEREÇOS CONSECUTIVOS DA MEMÓRIA

Como os endereços simbólicos Z1 e Z2, em geral, são atribuídos a endereços absolutos consecutivos da memória, podemos usar o par de registros HL para endereçar essas variáveis. Esta versão é mais vantajosa, quando se efetua transferência de valores para várias posições consecutivas da memória.

VERSÃO PARA 8080/85

Código-Objeto

Assembler

```

; TRANSFERENCIA DE DADOS POR ENDERECAMENTO INDIRETO
; DEFININDO O ENDEREÇO INICIAL DO PROGRAMA (1000).
;
1000      EX8504: ORG      1000H      ; 1)
1000 210910      LXI      H,Z1      ; 2)
1003 7E          MOV      A,M      ; 3)
1004 23          INC      H         ; 4)
1005 77          MOV      M,A      ; 5)
1006 C30610      PARE:    JMP      PARE ; 6)
1009 2B          Z1:      DB      2BH ; 7)
100A 00          Z2:      DB      0   ; 8)
100B          END      EX8504

```

Comentários:

- 1) Programa será montado a partir do endereço hexadecimal 1000.
- 2) Par HL recebe o endereço (imediato) atribuído a Z1.
- 3) Conteúdo de Z1 vai para A. Endereço de Z1 está em HL.
- 4) Incrementa HL para indicar o endereço seguinte, onde está Z2.
- 5) Conteúdo de A vai para Z2. Endereço de Z2 está em HL.

- 6) Pare.
- 7) Define valor 2B em Z1.
- 8) Define valor inicial 0 em Z2.

VERSÃO PARA Z80

Código-Objeto

Assembler

```

; TRANSFERENCIA DE DADOS POR ENDERECAAMENTO INDIRETO
; DEFININDO O ENDEREÇO INICIAL DO PROGRAMA (1000).
;
1000      EX04Z80: ORG      1000H
1000 210810      LD      HL,Z1      ; 1)
1003 7E          LD      A,(HL)     ; 2)
1004 23          INC     HL         ; 3)
1005 77          LD      (HL),A     ; 4)
1006 18FE      PARE: JR      PARE     ; 5)
1008 28          Z1:  DEFB     2BH     ; 6)
1009 00          Z2:  DEFB     0       ; 7)
100A          END      EX04Z80

```

Comentários:

- 1) Carrega par HL com o endereço de Z1. Operação imediata.
- 2) Carrega o conteúdo de Z1, cujo endereço está em HL, em A.
- 3) Incrementa HL, para indicar o endereço seguinte, onde está Z2.
- 4) Conteúdo de A vai para Z2. Endereço de Z2 está em HL.
- 5) Pare.
- 6) Define valor 2B para endereço Z1.
- 7) Define valor inicial 0 para endereço Z2.

EXERCÍCIOS

De Recapitulação

1. Descrever e citar exemplos de cada uma das formas possíveis das seguintes instruções:
 - MOVE ou LOAD REGISTERS
 - MOVE ou LOAD MEMORY
 - MOVE ou TRANSFER MEMORY TO MEMORY (só Z80)

De Aplicação

Escrever a versão Assembler e respectivo código-objeto dos programas abaixo citados. Todos os programas devem ser testados manualmente, instrução por instrução, através de um exemplo numérico ilustrado, mencionando os conteúdos dos registros e memórias afetadas.

1. Transferir um valor do endereço absoluto hexadecimal 1500 para o endereço hexadecimal 1601.
2. Transferir os bytes dos endereços hexadecimais 1500 e 1501 para 1601 e 1605, respectivamente.
3. Transferir quatro bytes dos endereços simbólicos V1, V2, V3 e V4 para os endereços simbólicos S1, S2, S3 e S4, respectivamente.
4. Modificar o programa do exercício anterior 3 de modo que:
 - os códigos de instrução (área de programa) sejam colocados a partir do endereço hexadecimal 1500;
 - os endereços simbólicos V1, V2, V3, V4 sejam colocados a partir do endereço hexa 1700 e os endereços S1, S2, S3 e S4, a partir do endereço hexa 1800.

Sugestão: utilizar instrução diretiva ORG.

5. ZERAR ENDEREÇOS. Modificar os programas dos exercícios 1, 2, 3, e 4 de maneira que os valores transferidos para os endereços destinos sejam sempre ZERO.
6. TRANSFERÊNCIA DE TEXTO. Colocar, a partir do endereço NOME, os caracteres do seu nome e sobrenome pessoal. Transferir esses caracteres para outra área que começa com o endereço simbólico NOVONOME.

CAPÍTULO

V

INSTRUÇÕES DE OPERAÇÕES ARITMÉTICAS (INCREMENT, DECREMENT, COMPLEMENT ADD E SUBTRACT) PARA 8080/85 E Z80

V.1 FORMAS GERAIS DAS INSTRUÇÕES

As instruções mencionadas neste capítulo são:

– INCREMENTA REGISTRO OU MEMÓRIA

8080/85	Z80	
INR reg	INC reg	– Incrementa registro.
INR M	INC (HL)	– Incrementa memória, com endereço em HL.
	INC (IX + ds)	– Incrementa memória ppqq + ds, ppqq em X.
	INC (IY + ds)	– Incrementa memória ppqq + ds, ppqq em Y.
INX regs	INC regs	– Incrementa registros de 16 bits ou Stack Pointer.

– DECREMENTA REGISTRO OU MEMÓRIA

DCR reg	DEC reg	– Decrementa registro.
DCR M	DEC (HL)	– Decrementa memória.
	DEC (IX + ds)	
	DEC (IY + ds)	
DCX regs	DEC regs.	– Decrementa registros de 16 bits ou Stack Pointer.

– COMPLEMENTA OU TORNA NEGATIVO

CMA	CPL	– Complementa conteúdo de A.
	NEG	– Negativo (complemento de dois) do valor em A.

– CONVERTE PARA ALGARISMO BCD (BINARY CODED DECIMAL)

DAA	DAA	– Valor em A é convertido em dois algarismos BCD.
-----	-----	---

– ADIÇÃO DE CONTEÚDO EM A, COM VALOR IMEDIATO; REGISTRO ou MEMÓRIA

ADI dd	ADD A, dd	– Soma A com valor imediato dd; resultado em A.
ADD reg	ADD A, reg	– Soma A com registro.
ADD M	ADD A, (HL)	– Soma A com memória ppqq, ppqq em HL.
	ADD A, (IX + ds)	– Soma A com memória ppqq + ds, ppqq em X.
	ADD A, (IY + ds)	– Soma A com memória ppqq + ds, ppqq em Y.
ACI dd	ADC A, dd	– Soma A, Carry e valor imediato.
ADC reg	ADC A, reg	– Soma A, Carry e registro.
ADC M	ADC A, (HL)	– Soma A, Carry e memória ppqq.
	ADC A, (IX + ds)	– Soma A, Carry e memória ppqq + ds.
	ADC A, (IY + ds)	– Soma A, Carry e memória ppqq + ds.

– SUBTRAÇÃO: CONTEÚDO EM A MENOS OPERANDO IMEDIATO, REGISTRO OU MEMÓRIA

SUI dd	SUB dd ou SUB A, dd	– Registro A menos valor imediato dd.
SUB reg	SUB reg ou SUB A, reg	– Conteúdo de A menos registro.
SUB M	SUB (HL) ou SUB A, (HL)	– A menos memória ppqq, ppqq em HL.
	SUB (IX + ds) ou SUB A, (IX + ds)	– A menos memória ppqq + ds.
	SUB (IY + ds) ou SUB A, (IY + ds)	– A menos memória ppqq + ds.

– SUBTRAÇÃO COM CARRY: CONTEÚDO EM A MENOS OPERANDO E CARRY.

SBI dd	SBC A, dd	– Conteúdo de A menos (dd mais Carry).
SBB reg	SBC A, reg	– Conteúdo de A menos (registro mais Carry).

SBB M	SBC A, (HL)	– Conteúdo de A menos (memória ppqq mais Carry).
	SBC A, (IX + ds)	– Conteúdo de A menos (memória ppqq + ds e Carry).
	SBC A, (IY + ds)	– Conteúdo de A menos (memória ppqq + ds e Carry).

– ADIÇÃO E SUBTRAÇÃO COM VALORES DE 16 BITS

DAD regs	ADD HL, regs	– Soma par HL com par de registro HL, BC ou DE ou SP.
	ADD IX, regs	– Soma X com par de registros ou SP.
	ADD IY, regs	– Soma Y com par de registros ou SP.
–		
–	SBC HL, regs	– Subtração de HL, do conteúdo de par de registro ou Stack Pointer mais o Carry.

– PROGRAMAS-EXEMPLOS

Além dos exemplos ilustrados de cada instrução, o capítulo contém os seguintes programas-exemplos:

- SOMA DE OITO BITS.
- SOMA DE DOIS VALORES DE 16 BITS.

V.2 INR, INX OU INC (INCREMENT)

INCREMENTA REGISTRADORES ou MEMÓRIA

INCREMENTA REGISTRO (8 bits)				
8080/85		Z80		Explicação
Assembler	Código-objeto	Assembler	Código-objeto	
INR A	3C	INC A	3C	– Incrementa A.
INR B	04	INC B	04	– Incrementa B.
INR C	0C	INC C	0C	– Incrementa C.
INR D	14	INC D	14	– Incrementa D.
INR E	1C	INC E	1C	– Incrementa E.
INR H	24	INC H	24	– Incrementa H.
INR L	2C	INC L	2C	– Incrementa L.

INCREMENTA MEMÓRIA

INR M	34	INC (HL)	34	– Incrementa memória ppqq; HL contém ppqq.
		INC (IX + ds)	DD 34 ds	– Incrementa endereço ppqq + ds; ppqq está em X.
		INC (IY + ds)	FD 34 ds	– <i>Idem</i> ppqq + ds; ppqq está em Y.

INCREMENTA REGISTROS ou STACK POINTER (16 bits)

INX B	03	INC BC	03	– Incrementa par de registros BC.
INX D	13	INC DE	13	– Incrementa par de registros DE.
INX H	23	INC HL	23	– Incrementa par de registros HL.
INX SP	33	INC SP	33	– Incrementa Stack Pointer SP.
		INC IX	DD 23	– Incrementa registro X.
		INC IY	FD 23	– Incrementa registro Y.

V.3 DCR, DCX OU DEC – DECREMENTA REGISTRO OU MEMÓRIA**DECREMENTA REGISTRO**

<i>8080/85</i>		<i>Z80</i>		<i>Explicação</i>
DCR A	3D	DEC A	3D	– Decrementa A.
DCR B	05	DEC B	05	– Decrementa B.
DCR C	0D	DEC C	0D	– Decrementa C.
DCR D	15	DEC D	15	– Decrementa D.
DCR E	1D	DEC E	1D	– Decrementa E.
DCR H	25	DEC H	25	– Decrementa H.
DCR L	2D	DEC L	2D	– Decrementa L.

DECREMENTA MEMÓRIA

DCR M	35	DEC (HL)	35	– Decrementa endereço ppqq; HL contém ppqq.
		DEC (IX + ds)	DD 35 ds	– Decrementa endereço ppqq + ds; ppqq está em X.
		DEC (IY + ds)	FD 35 ds	– Decrementa endereço ppqq + ds; ppqq está em Y.

DECREMENTA REGISTROS ou STACK POINTER (16 bits)				
DCX B	0B	DEC BC	0B	– Decrementa par de registros BC.
DCX D	1B	DEC DE	1B	– Decrementa par de registros DE.
DCX H	2B	DEC HL	2B	– Decrementa par de registros HL.
DCX SP	3B	DEC SP	3B	– Decrementa Stack Pointer SP.
		DEC IX	DD 2B	– Decrementa registro X.
		DEC IY	FD 2B	– Decrementa registro Y.

V.4 CMA, CPL OU NEG – COMPLEMENTA OU TORNA NEGATIVO VALOR NO REGISTRO A

CMA	2F	CPL	2F	– Complementa conteúdo do acumulador A.
		NEG	ED 44	– Inverte sinal (complemento de dois) do conteúdo de A.

V.5 DAA (DECIMAL ADJUST ACUMULATOR)

Converte conteúdo do Acumulador A em dois dígitos BCD (BINARY CODED DECIMAL). Exemplo: A = 80 (hexadecimal) fica A = 86 (em BCD).

DAA	27	DAA	27	– Converte para BCD.
-----	----	-----	----	----------------------

Exemplos

8080/85	Z80	Comentário	REGISTRO OU MEMÓRIA	
			ANTES	DEPOIS
a) INR A	INC A	. Incrementa A.	A = 03	A = 04
b) CMA	CPL	. Complementa A.	A = 0F	A = F0
c) –	NEG	. Negativo (complemento de dois).	A = 0F	A = F1
d) DAA	DAA	. Dígitos decimais em A: 80 em hexadecimal torna-se 86 em BCD (Binary Coded Decimal).	A = 80	A = 86
e) INX B	INC BC	. Incrementa par BC.	BC = 2A 00	BC = 2A 01
f) DCX SP	DEC SP	. Decrementa Stack Pointer SP.	SP = 3A FF	SP = 3A FE

- g) INR M INC (HL) . Incrementa memória endereçada por HL. HL contém ppqq. ppqq = 25 ppqq = 26
- h) --- DEC (IX + 5) . Decrementa conteúdo de ppqq + 5. Registro X contém ppqq. ppqq + 5 = 22 ppqq + 5 = 21

V.6 ADD E ADC – ADIÇÃO SEM CARRY (VAI UM) E COM CARRY (VAI UM)

ADD (ADD) – ADIÇÃO DO ACUMULADOR (REGISTRO A) COM REGISTRO, MEMÓRIA OU VALOR IMEDIATO; RESULTADO FICA EM A.

Flags afetados: 8080/85: Ac, P, S, C, Z = X
Z80: S, Z, Ac, PO = X e N = 0

8080/85		Z80		Explicação
ADI dd	C6 dd	ADD A, dd	C6 dd	– Soma A com valor imediato dd.
ADD A	87	ADD A, A	87	– Soma A com A.
ADD B	80	ADD A, B	80	– Soma A com B.
ADD C	81	ADD A, C	81	– Soma A com C.
ADD D	82	ADD A, D	82	– Soma A com D.
ADD E	83	ADD A, E	83	– Soma A com E.
ADD H	84	ADD A, H	84	– Soma A com H.
ADD L	85	ADD A, L	85	– Soma A com L.
ADD M	86	ADD A, (HL)	86	– Soma A com conteúdo de ppqq; HL contém ppqq.
		ADD A, (IX + ds)	DD 86 ds	– Soma A com conteúdo de ppqq + ds; ppqq está em X.
		ADD A, (IY + ds)	FD 86 ds	– Soma A com conteúdo de ppqq + ds; ppqq está em Y.

ADC (ADD WITH CARRY) – ADIÇÃO DO ACUMULADOR OU REGISTRO A COM REGISTRO, MEMÓRIA OU VALOR IMEDIATO MAIS O CARRY; RESULTADO FICA EM A.

Flags afetados: 8080/85: Ac, P, S, C, Z = X
Z80: S, Z, Ac, PO = X e N = 0

ACI dd	CE dd	ADC A, dd	CE dd	– Soma A com valor imediato dd e Carry.
ADC A	8F	ADC A, A	8F	– Soma A com A e Carry.

ADC B	88	ADC A, B	88	– Soma A com B e Carry.
ADC C	89	ADC A, C	89	– Soma A com C e Carry.
ADC D	8A	ADC A, D	8A	– Soma A com D e Carry.
ADC E	8B	ADC A, E	8B	– Soma A com E e Carry.
ADC H	8C	ADC A, H	8C	– Soma A com H e Carry.
ADC L	8D	ADC A, L	8D	– Soma A com L e Carry.
ADC M	8E	ADC A, (HL)	8E	– Soma A com conteúdo de ppqq e Carry; ppqq em HL.
		ADC A, (IX + ds)	DD 8E ds	– Soma A com conteúdo de ppqq + ds e Carry; ppqq em X.
		ADC A, (IY + ds)	FD 8E ds	– Soma A com conteúdo de ppqq + ds e Carry; ppqq em Y.

V.7 SUB, SBB OU SBC – SUBTRAÇÃO SEM OU COM CARRY

SUB (SUBTRACT FROM ACUMULATOR OU REGISTER A) – SUBTRAI DE A, o VALOR IMEDIATO, REGISTRO OU CONTEÚDO DA MEMÓRIA; RESULTADO FICA EM A.

8080/85		Z80		Explicação
SUI dd	D6 dd	SUB dd	D6 dd	– Subtrai valor dd de A, resultado em A.
SUB A	97	SUB A	97	– Subtrai A de A; A torna-se zero.
SUB B	90	SUB B	90	– Subtrai B de A, resultado fica em A.
SUB C	91	SUB C	91	– Resultado de A menos C vai para A.
SUB D	92	SUB D	92	– Resultado de A menos D vai para A.
SUB E	93	SUB E	93	– Resultado de A menos E vai para A.
SUB H	94	SUB H	94	– Resultado de A menos H vai para A.
SUB L	95	SUB L	95	– Resultado de A menos L vai para A.
SUB M	96	SUB (HL)	96	– Subtrai conteúdo de ppqq de A; ppqq está em HL.
		SUB (IX + ds)	DD 96 ds	– Subtrai conteúdo de ppqq + ds de A; X contém ppqq.
		SUB (IY + ds)	FD 96 ds	– Subtrai conteúdo de ppqq + ds de A; Y contém ppqq.

Nota: No Assembler Z80 as formas SUB A, dd ou SUB A, (HL) ou SUB A, reg_i podem ser aceites. O Manual Assembler em uso deve ser consultado.

SBB ou SBC (SUBTRACT WITH CARRY) – SUBTRAI DE A O VALOR IMEDIATO, REGISTRO OU CONTEÚDO DA MEMÓRIA ACRESCIDO DO CARRY; RESULTADO FICA EM A.

SBI dd	DE dd	SBC A,dd	DE dd	– Subtrai valor imediato dd mais Carry de A.
SBB A	9F	SBC A, A	9F	– Subtrai conteúdo de A mais Carry de A.
SBB B	98	SBC A, B	98	– Subtrai conteúdo de B mais Carry de A.
SBB C	99	SBC A, C	99	– Subtrai conteúdo de C mais Carry de A.
SBB D	9A	SBC A, D	9A	– Subtrai conteúdo de D mais Carry de A.
SBB E	9B	SBC A, E	9B	– Subtrai conteúdo de E mais Carry de A.
SBB H	9C	SBC A, H	9C	– Subtrai conteúdo de H mais Carry de A.
SBB L	9D	SBC A, L	9D	– Subtrai conteúdo de L mais Carry de A.
SBB M	9E	SBC A, (HL)	9E	– Subtrai conteúdo de ppqq mais Carry de A; ppqq em HL.
		SBC A, (IX + ds)	DD 9E ds	– Subtrai conteúdo de ppqq + ds e Carry de A; ppqq em X.
		SBC A, (IY + ds)	FD 9E ds	– Subtrai conteúdo de ppqq + ds e Carry de A; ppqq em Y.

V.8 DOUBLE ADD OU SUBTRACT – ADIÇÃO OU SUBTRAÇÃO DE 16 BITS

SOMA DE 16 BITS: SOMA PAR DE REGISTROS ou STACK POINTER SP.

<i>8080/85</i>		<i>Z80</i>		<i>Explicação</i>
DAD B	09	ADD HL, BC	09	– Soma conteúdo do par HL com BC; resultado em HL.
DAD D	19	ADD HL, DE	19	– Soma par HL com DE, resultado em HL.
DAD H	29	ADD HL, HL	29	– Soma par HL com HL, resultado em HL.
DAD SP	39	ADD HL, SP	39	– Soma par HL com Stack Pointer SP, resultado em HL.
		ADC HL, BC	ED 4A	– Soma HL com BC e Carry, resultado em HL.
		ADC HL, DE	ED 5A	– Soma HL com DE e Carry, resultado em HL.
		ADC HL, HL	ED 6A	– Soma HL com HL e Carry, resultado em HL.

ADC HL, SP	ED 7A	– Soma HL com Stack Pointer SP e Carry, resultado em HL.
ADD IX, BC	DD 09	– Soma registro X com par BC, resultado fica em X.
ADD IX, DE	DD 19	– Soma X com par DE, resultado em X.
ADD IX, IX	DD 29	– Soma X com X, resultado em X.
ADD IX, SP	DD 39	– Soma X com Stack Pointer SP, resultado em X.
ADD IY, BC	FD 09	– Soma Y com par BC, resultado em Y.
ADD IY, DE	FD 19	– Soma Y com par DE, resultado em Y.
ADD IY, IY	FD 29	– Soma Y com Y, resultado em Y.
ADD IY, SP	FD 39	– Soma Y com Stack Pointer SP, resultado em Y.

SUBTRAÇÃO DE 16 BITS: SUBTRAI DE HL O CONTEÚDO DO PAR DE REGISTROS OU STACK POINTER SP MAIS O CARRY, RESULTADO FICA EM HL.

SBC HL, BC	ED 42	– Subtrai conteúdo de BC mais Carry de HL.
SBC HL, DE	ED 52	– Subtrai conteúdo de DE mais Carry de HL.
SBC HL, HL	ED 62	– Subtrai de HL, conteúdo de HL mais Carry.
SBC HL, SP	ED 72	– Subtrai de HL, conteúdo de SP mais Carry.

Exemplos: (ADIÇÃO)

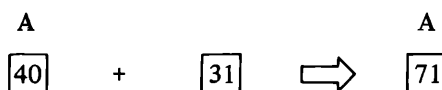
8080/85

Z80

a) ADI 31H

ADD A, 31H

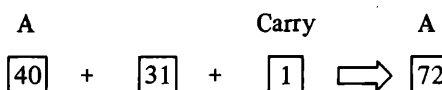
• Adição de valor imediato 31H com A.



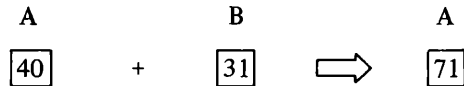
b) ACI 31H

ADC A, 31H

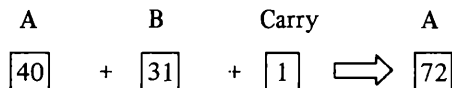
• Adição do valor imediato 31H e Carry com A.



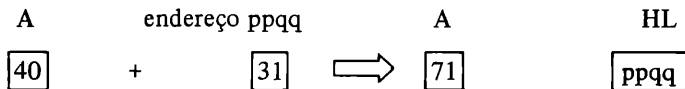
c) ADD B ADD A, B • Adição do valor em B com A.



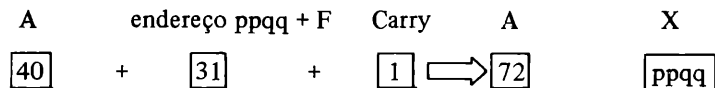
d) ADC B ADC A, B • Adição do valor em B e Carry com A.



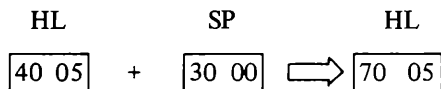
e) ADD M ADD A, (HL) • Adição de A com memória endereçada por HL.



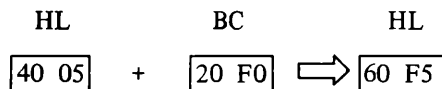
f) só Z80 ADC A, (IX + 0FH) • Adição de A e Carry, com conteúdo da memória endereçada por X e deslocamento F.



g) DAD SP ADD HL, SP • Adição de 16 bits do conteúdo de HL com SP (Stack Pointer)



h) DAD B ADD HL, BC • Adição do conteúdo de BC com HL.



i) só Z80 ADD IX, HL • Adição de conteúdo de X com par HL, resultado em X.

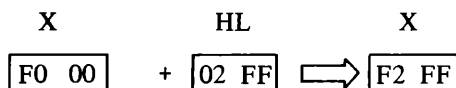
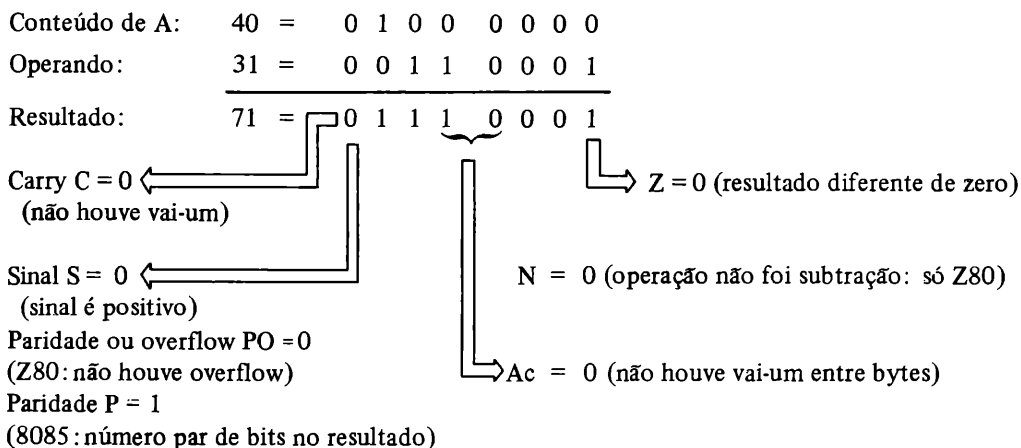
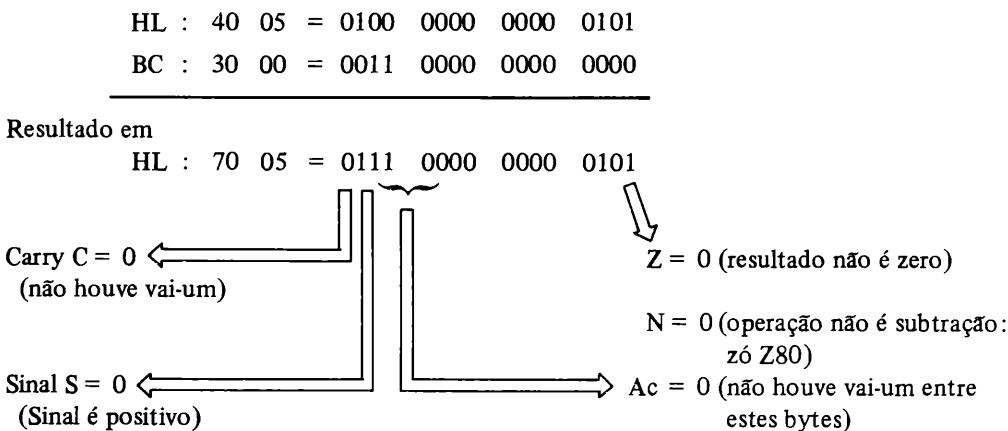


Ilustração: Bits de Flags afetados pela adição.

a) Somar conteúdo de A com operando com valor 31.



b) Somar conteúdo de HL (16 bits) com conteúdo de BC (16 bits).



Paridade ou overflow PO = não afetado

Exemplos: (SUBTRAÇÃO)

8080/85

Z80

a) SBI 36H

SBC A, 36H

• Subtração do operando imediato 36 e Carry, do valor em A.

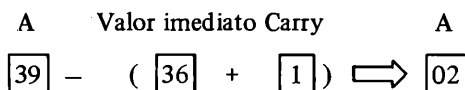


Ilustração: Subtração é soma com complemento de dois.

Valor imediato mais Carry :	36 + 1 = 37 : 0011 0111
Complemento de 37 :	1100 1000
Mais um :	<u>1</u>
Complemento de dois de 37:	1100 1001
Mais conteúdo de A: 39 :	<u>0011 1001</u>
Resultado da subtração: 02:	0000 0010 com Carry C = 1.

b) SUI 36H SUB 36H • Subtração do valor imediato 36 de A.

A	valor imediato	A
39	— 36	03

c) SUB B SUB B • Subtração do valor em B, de A.

A	B	A
39	— 36	03

d) SBB B SBC A, B • Subtrai de A, valor em B e Carry.

A	B	Carry	A
39	— (36 + 1)		02

e) SUB M SUB (HL) • Subtrai de A, conteúdo do endereço ppqq.
Endereço ppqq está em HL.

A	Endereço ppqq	A	HL
39	— 36	03	ppqq

f) só Z80 SBC A, (Y + 3) • Subtrai de A, conteúdo do endereço ppqq + 3 e Carry.

Endereço ppqq está em Y.				
A	Endereço ppqq + 3	Carry	A	Y
39	— (36 + 1)		02	ppqq

g) só Z80

SBC HL, BC

- Subtração de 16 bits do par HL com par BC e Carry.



Ilustração: Bits de Flags afetados na subtração: Conteúdo de A menos operando 36H.

Conteúdo de A : 39 : 0011 1001

menos operando : - 36 : 1100 1010 (complemento de dois)

Resultado : 03 : 0000 0011

Carry C = 0 $\leftarrow$ (não houve empresta-um) $\rightarrow$ Z = 0 (resultado não é zero)

Sinal S = 0 $\leftarrow$ (Sinal é positivo) N = 1 (operação é subtração: só Z80)

Paridade / Overflow PO = 0 $\rightarrow$ Ac = 0 (não houve empresta-um entre bytes)
(Z80: não houve overflow)

Paridade P = 1
(8085: número par de bits no resultado)

Nota: O “borrow ou empresta-um” ocorre quando o valor subtraído for maior que o valor em A. Este fato é indicado pelo valor do Carry C = 1. Quando C = 0, não houve “empresta-um”.

O mesmo fato se verifica com o flag Ac (Half Carry) que indica a ocorrência do “empresta-um” entre os bytes.

V.9 PROGRAMA-EXEMPLO: SOMA DE OITO BITS

Somar o conteúdo do endereço absoluto 00F1 (ou endereço simbólico Z1) com o conteúdo do endereço absoluto 1A00 (ou endereço simbólico Z2) e colocar o resultado no endereço absoluto 1B01 (ou endereço simbólico RESULT).

Algoritmo

Endereço absoluto 00F1 (ou Z1) contém : 21

Endereço absoluto 1A00 (ou Z2) contém : A3

Soma usando Registro A : 21 + A3 = C4

Endereço absoluto 1B01 (ou RESULT) conterá: C4

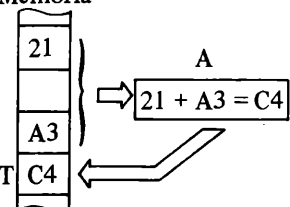
Endereços

00F1 ou Z1

1A00 ou Z2

1B01 ou RESULT

Memória



VERSÃO PARA 8080/85

Código-Objeto

Assembler

```
                ; SOMA DE OITO BITS USANDO
                ; ENDEREÇOS ABSOLUTOS 00F1, 1A00 E 1B01.
                ;
0000 3AF100    EX8505: LDA    00F1H ; 1)
0003 21001A          LXI    H,1A00H ; 2)
0006 86          ADD     M      ; 3)
0007 32011B          STA    1B01H ; 4)
000A C30A00    PARE:  JMP    PARE ; 5)
000D          END      EX8505
```

Comentários:

- 1) Carrega A com conteúdo de 00F1.
- 2) Carrega HL com o endereço imediato 1A00.
- 3) Soma conteúdo de A com conteúdo de 1A00.
- 4) Coloca resultado no endereço 1B01.
- 5) Pare.

Código-Objeto

Assembler

```
                ; SOMA DE OITO BITS USANDO
                ; ENDEREÇOS SIMBOLICOS Z1, Z2 E RESULT.
                ;
0000 3A0000    EX8506: LDA    Z1      ; 1)
0003 210E00          LXI    H,Z2    ; 2)
0006 86          ADD     M      ; 3)
0007 320F00          STA    RESULT ; 4)
000A C30A00    PARE:  JMP    PARE   ; 5)
000D 21          Z1:    DB      21H  ; 6)
000E A3          Z2:    DB      0A3H ; 7)
000F 00          RESULT: DB      00   ; 8)
0010          END      EX8506
```

Comentários:

- 1) Carrega A com o conteúdo de Z1.
- 2) Carrega HL com o endereço de Z2.
- 3) Soma conteúdo de A (valor em Z1) com o conteúdo de Z2.

- 4) Coloca resultado da soma no endereço simbólico RESULT.
- 5) Pare.
- 6) Define endereço simbólico Z1, onde é colocado valor 21H através da instrução diretiva DB (DEFINE UM BYTE).
- 7) *Idem*, endereço simbólico Z2 recebe valor 0A3H.
- 8) Endereço simbólico RESULT recebe valor inicial 00H.

VERSÃO PARA Z80

Código-Objeto

Assembler

```

; SOMA DE OITO BITS USANDO
; ENDEREÇOS ABSOLUTOS 00F1, 1A00 E 1B01.
;
1000          ORG      1000H
1000 3AF100  EX05280: LD      A,(0F1H)  ; 1)
1003 21001A          LD      HL,1A00H  ; 2)
1006 86          ADD      (HL)        ; 3)
1007 32011B          LD      (1B01H),A ; 4)
100A 76          HALT                ; 5)
100B          END      EX05280

```

Comentários:

- 1) Conteúdo do endereço 00F1 vai para A. (Primeira parcela da soma.)
- 2) Carrega endereço 1A00 (segunda parcela) em HL.
- 3) Soma A com endereço apontado por HL.
- 4) Resultado da soma vai para endereço 1B01.
- 5) Pare.

Código-Objeto

Assembler

```

; SOMA DE OITO BITS USANDO
; ENDEREÇOS SIMBÓLICOS Z1, Z2 E RESULT.
;
1000          ORG      1000H
1000 3A0B10  EX06280: LD      A,(Z1)    ; 1)
1003 210C10          LD      HL,Z2      ; 2)
1006 86          ADD      (HL)        ; 3)
1007 320D10          LD      (RESULT),A ; 4)
100A 76          HALT                ; 5)

```

```
100B 21      Z1:      DEFB  21H      ; 6)
100C A3      Z2:      DEFB  0A3H     ; 7)
100D 00      RESULT:  DEFB  00       ; 8)
100E                      END      EX06Z80
```

Comentários:

- 1) Carrega A com conteúdo de Z1.
- 2) Carrega HL com endereço de Z2.
- 3) Soma conteúdo de A (valor em Z1) com o conteúdo de Z2.
- 4) Coloca resultado da soma no endereço simbólico RESULT.
- 5) Pare.
- 6) Define endereço simbólico Z1, contendo primeira parcela da soma 21H.
- 7) Define endereço simbólico Z2, contendo segunda parcela da soma 0A3H.
- 8) Define endereço simbólico RESULT, com valor inicial zero.

V.10 PROGRAMA-EXEMPLO: SOMA DE DOIS VALORES COM 16 BITS

Somar dois valores de 16 bits cada: o primeiro valor está colocado nos endereços consecutivos F1 e F2 (ou simbólicos Z1 e Z1 + 1); o segundo valor está nos endereços F3 e F4 (ou Z2 e Z2 + 1); e o resultado será colocado nos endereços F5 e F6 (ou RESULT e RESULT + 1).

Algoritmo-Exemplo

Soma: 2315 (hexa) + 25F1 (hexa) = 4906 (hexadecimal)

Primeiro valor 2315 (hexa) foi colocado em:

00F1 ou Z1 = 15 (byte menos significativo)
00F2 ou Z1 + 1 = 23 (byte mais significativo)

Segundo valor 25F1 (hexa) foi colocado em:

00F3 ou Z2 = F1 (byte menos significativo)
00F4 ou Z2 + 1 = 25 (byte mais significativo)

Resultado da soma será colocado em:

00F5 ou RESULT = 06 (byte menos significativo)
00F6 ou RESULT + 1 = 49 (byte mais significativo)

Guardar e incluir o CARRY na soma de um byte para outro.

Endereços		Memória				
				CARRY:	1	0
Z1	ou	00F1	15	Primeiro valor:	23	15
		00F2	23			
Z2	ou	F3	F1	Segundo valor:	25	F1 +
		F4	25			
RESULT	ou	F5	06	Soma:	49	06
		F6	49			

VERSÃO PARA 8080/85

Código-Objeto	Assembler
	; SOMA DE 16 BITS USANDO
	; ENDEREÇOS ABSOLUTOS DE F1 A F6.
	;
0000 2AF100	EX8507: LHLD 0F1H ; 1)
0003 EB	XCHG ; 2)
0004 2AF300	LHLD 0F3H ; 3)
0007 19	DAD D ; 4)
0008 22F500	SHLD 0F5H ; 5)
0008 C30800	PARE: JNP PARE ; 6)
000E	END EX8507

Comentários.

- 1) Carrega par HL com valor de 16 bits dos endereços 00F1 e 00F2. (Primeira parcela.).
- 2) Troca conteúdo de HL com DE.
- 3) Carrega em HL, valor dos endereços 00F3 e 00F4 (Segunda parcela.).
- 4) Soma de 16 bits do valor em HL com valor em DE, resultado fica em HL.
- 5) Coloca resultado de HL, nos endereços 00F5 e 00F6.
- 6) Pare.

Código-Objeto	Assembler
	; SOMA DE 16 BITS USANDO
	; ENDEREÇOS SIMBÓLICOS Z1, Z2 E RESULT.
	;
0000 2A0E00	EX8508: LHLD Z1 ; 1)
0003 EB	XCHG ; 2)
0004 2A1000	LHLD Z2 ; 3)


```

0007 19          DAD    D      ; 4)
0008 221200      SHLD   RESULT ; 5)
0008 C30800      PARE:   JMP    PARE ; 6)
000E 1523        Z1:    DW     2315H ; 7)
0010 F125        Z2:    DW     25F1H ; 8)
0012 0000        RESULT: DW     00   ; 9)
0014             END     EX8508

```

Comentários:

- 1) a 6) Análogo ao programa anterior, trocando endereços absolutos por simbólicos.
- 7) Define endereço Z1 contendo valor da primeira parcela de 16 bits.
- 8) Define endereço Z2 contendo valor da segunda parcela de 16 bits.
- 9) Define endereço RESULT, onde será colocado o resultado da soma de 16 bits.

VERSÃO PARA Z80

Código-Objeto

Assembler

```

; SOMA DE 16 BITS USANDO
; ENDEREÇOS ABSOLUTOS DE F1 A F6.
;
1000          ORG     1000H
1000 2AF100    EX07Z80: LD     HL,(0F1H) ; 1)
1003 ED58F300 LD     DE,(0F3H) ; 2)
1007 19        ADD     HL,DE      ; 3)
1008 22F500    LD     (0F5H),HL ; 4)
1008 76        HALT                ; 5)
100C          END     EX07Z80

```

Comentários:

- 1) Carrega par HL com o valor da primeira parcela que está em 00F1 e 00F2.
- 2) Carrega par DE com o valor da segunda parcela que está em 00F3 e 00F4.
- 3) Soma par HL com DE; resultado fica em HL.
- 4) Coloca resultado da soma nos endereços 00F5 e 00F6.
- 5) Pare.

Código-Objeto

Assembler

```

; SOMA DE 16 BITS USANDO
; ENDEREÇOS SIMBÓLICOS Z1, Z2 E RESULT.
;
1000          ORG     1000H

```

```

1000 2A0C10    EX08Z80: LD    HL,(Z1)      ; 1)
1003 ED580E10          LD    DE,(Z2)      ; 2)
1007 19                ADD    HL,DE        ; 3)
1008 221010          LD    (RESULT),HL    ; 4)
1008 76                HALT                ; 5)
100C 1523    Z1:      DEFW    2315H        ; 6)
100E F125    Z2:      DEFW    25F1H        ; 7)
1010 0000    RESULT: DEFW    00           ; 8)
1012                END    EX08Z80
    
```

Comentários:

- 1) a 5) Análogo ao programa anterior, trocando endereços absolutos por simbólicos.
- 6) Define no endereço Z1 valor 2315 de 16 bits.
- 7) Define no endereço Z2 valor 25F1 de 16 bits.
- 8) Define endereço RESULT para receber valor de 16 bits.

V.11 COLOCAÇÃO DOS ENDEREÇOS SIMBÓLICOS Z1, Z2 E RESULT A PARTIR DO ENDEREÇO ABSOLUTO 00F1

Podemos colocar os endereços simbólicos em qualquer posição desejada da memória, por exemplo, a partir do endereço absoluto 00F1, o que deixa as duas versões do programa em Assembler gerar os mesmos códigos-objeto.

Basta utilizar, antes da definição dos endereços simbólicos, a instrução diretiva ORG F1H. As instruções da parte executável do programa permanecem as mesmas.

Para
8085

```

1)          LHLD  Z1
    ...
6) PARE:    JMP   PARE
          ORG   F1H
7) Z1:      DW    2315H          00F1  15  23
8) Z2:      DW    25F1H          00F3  F1  25
9) RESULT:  DW    00             00F5  00  00
10)         END
    
```

Para
Z80

```
1)          LD      HL, (Z1)

5)          HALT
           ORG      F1H
6) Z1:      DEFW    2315 H           00F1  15  23
7) Z2:      DEFW    25F1H           00F3  F1  25
8) RESULT:  DEFW    00              00F5  00  00
9)          END
```

EXERCÍCIOS

De Recapitulação

1. Descrever e citar exemplos de cada uma das formas possíveis das instruções:

- INCREMENT e DECREMENT
- ADD e SUBTRACT

De Aplicação

Escrever a versão Assembler e respectivo código-objeto dos programas abaixo citados. Todos os programas devem ser testados manualmente, instrução por instrução, através de um exemplo numérico ilustrado, mencionando os conteúdos dos registros e memórias afetadas.

1. Somar o valor que está no endereço hexadecimal 1500 com o valor do endereço hexadecimal 1601. Colocar o resultado no endereço hexadecimal 1602.
2. Somar os valores que estão nos endereços simbólicos V1 e V2 e colocar o resultado no endereço simbólico V3.
3. SOMA DE 32 BITS – Somar o valor de 32 bits que está nos endereços consecutivos V1, V2, V3 e V4 com o valor de 32 bits, que está nos endereços simbólicos S1, S2, S3 e S4. Colocar o resultado da soma nos endereços T1, T2, T3 e T4.
4. Modificar os programas dos exercícios 2 e 3 de modo que os códigos de instrução (área de programa) sejam colocados a partir do endereço hexadecimal 1500 e os valores utilizados (área de dados) sejam colocados a partir do endereço hexadecimal 1800. Sugestão: utilizar a instrução diretiva ORG.

5. INCREMENTAR OU DECREMENTAR VALORES – Modificar os programas dos exercícios 1 e 2 de modo que o resultado das somas sejam de incremento (ou decremento) simples de valor 1 em cada posição da memória.
6. SUBTRAÇÃO – Modificar os programas dos exercícios 1 e 2 de modo que a operação executada seja de subtração.
7. SUBTRAÇÃO DE 32 BITS – Subtrair, de um valor de 32 bits, colocado nos endereços V1, V2, V3 e V4, outro valor de 32 bits, que está nos endereços S1, S2, S3 e S4. Colocar o resultado da subtração nos endereços T1, T2, T3 e T4.

CAPÍTULO

VI

INSTRUÇÕES DE COMPARAÇÃO E DESVIOS PARA 8080/85 E Z80

VI.1 FORMAS GERAIS DAS INSTRUÇÕES

As instruções estudadas neste capítulo são:

8080/85

Z80

COMPARAÇÃO DE A COM OUTRO REGISTRO, MEMÓRIA OU OPERANDO IMEDIATO

CPI dd	CP dd	— Compara A com valor imediato dd.
CMP reg	CP reg	— Compara A com registros.
CMP M	CP (HL)	— Compara A com memória ppqq; ppqq em HL.
	CP (IX + ds)	— Compara A com memória ppqq + ds; ppqq em X.
	CP (IY + ds)	— Compara A com memória ppqq + ds; ppqq em Y.

COMPARAÇÃO DE A COM CONTEÚDO DA MEMÓRIA, COM REPETIÇÃO INCREMENTADA OU DECREMENTADA

—	CPD	— Compara A com memória ppqq em HL; decrementa HL.
—	CPDR	— Repetição de CPD, decrementando BC até zero.
	CPI	— Compara A com memória ppqq em HL; incrementa HL.
	CPIR	— Repetição de CPI, decrementando BC até zero.

DESVIO SIMPLES OU CONDICIONAL COM ENDEREÇO LONGO OU RELATIVO

JMP endereço	JP endereço	— Desvia para endereço longo.
Jcc endereço	JP cc, endereço	— Desvia para endereço longo, se cc for verdade.
PCHL	JP (HL)	— Desvia para endereço ppqq, que está em HL.

- JP (IX + ds) – Desvia para endereço ppqq + ds; ppqq em X.
- JP (IY + ds) – Desvia para endereço ppqq + ds; ppqq em Y.
- JR endereço – Desvio relativo para endereço.
- JR cc, endereço – Desvio relativo para endereço, se cc for verdade.
- DJNZ endereço – Desvio relativo se B for zero, após decrementar B.
- PROGRAMA-EXEMPLO

Além dos exemplos de cada instrução, este capítulo contém o programa-exemplo: “COMPARAÇÃO E DESVIO: ACHAR O MAIOR DE DOIS VALORES DA MEMÓRIA”.

VI.2 COMPARE

Compara conteúdo do operando (Imediato, Memória ou Registro) com Acumulador afetando bits de flags Z, Ac, S, N, P/O, de acordo com o resultado da operação ((A) – operando). Conteúdos de A, operando e CARRY não são afetados.

8080/85		Z80		Explicação
Assembler	Código-Objeto	Assembler	Código-Objeto	
CPI dd	FE dd	CP dd	FE dd	– Compara A com valor imediato dd.
CMP A	BF	CP A	BF	– Compara A com A.
CMP B	B8	CP B	B8	– Compara A com B.
CMP C	B9	CP C	B9	– Compara A com C.
CMP D	BA	CP D	BA	– Compara A com D.
CMP E	BB	CP E	BB	– Compara A com E.
CMP H	BC	CP H	BC	– Compara A com H.
CMP L	BD	CP L	BD	– Compara A com L.
CMP M	BE	CP (HL)	BE	– Compara A com endereço ppqq; HL contém ppqq.
		CP (IX+ds) DD	BE ds	– Compara A com endereço ppqq + ds; ppqq está em X.
		CP (IY+ds) FD	BE ds	– Compara A com endereço ppqq + ds; ppqq está em Y.

Exemplos:

- | | | |
|------------|--------|--|
| 8080/85 | Z80 | |
| a) CPI 36H | CP 36H | <ul style="list-style-type: none"> • Compara A com operando imediato 36H. Efetua operação (A) – 36H e aciona bits de flags conforme o resultado dessa subtração. Não afeta conteúdo de A. |

Ilustração: Flags de Status.

Conteúdo de A: 39
 Operando : 36
 Resultado $39 - 36 = 03$: 00000011

Carry C = 0 (não houve empresta um)
 Sinal S = 0 (resultado é positivo)
 Paridade/overflow = 0 (Z80: não houve overflow)
 Paridade P = 1 (8085: número par de bits no resultado)
 Z = 0 (resultado não é zero)
 N = 1 (operação executada é subtração: só Z80).
 Ac = 0 (não houve empresta-um entre bytes)

b) CMP B CP B • Compara conteúdo de A com B.
 A B
 39 > = < 36 (obtem mesmos resultados do caso a)

c) CMP M CP (HL) • Compara A com conteúdo do endereço ppqq, que está em HL.
 A endereço ppqq HL
 39 > = < 36 ppqq

d) só Z80 CP (IX + 5) • Compara A com conteúdo do endereço ppqq + 5; ppqq está em X.
 A endereço ppqq + 5 X
 39 36 ppqq

VI.3 COMPARE ACUMULADOR COM MEMÓRIA COM DECREMENTO OU INCREMENTO DE HL E REPETIÇÃO (SÓ PARA Z80)

a) COM DECREMENTO DE HL E DECREMENTO DE BC

8080/85 Z80
 – CPD ED A9 – Comparação simples de A com conteúdo de memória apontada por HL. Decrementa HL e BC após a operação.

b) COM INCREMENTO DE HL E DECREMENTO DE BC

– CPI ED A1 – Comparação simples de A com conteúdo da memória apontada por HL. Incrementa HL e decrementa BC após a operação.

c) COMPARE ACUMULADOR COM REPETIÇÃO DE k BYTES DA MEMÓRIA ATÉ ACHAR BYTE IGUAL OU CONTEÚDO DE BC FICAR ZERO

COM DECREMENTO DE HL E DECREMENTO DE BC

- CPDR ED B9 – Comparação com repetição, análoga a CPD, terminando a comparação quando achar valor igual ou quando BC que contém valor k tornar-se nulo.

COM INCREMENTO DE HL E DECREMENTO DE BC

- CPIR ED B1 – Análogo à CPDR.

Nota-Exemplos ilustrativos são análogos aos das instruções LDI, LDIR, LDD e LDDR (transferência, memória a memória, com repetição).

Exemplos: Comparação de A com memória com incremento/decremento e repetição (só Z80).

- a) CPD • Compara A com conteúdo do endereço ppqq. Decrementa HL e BC.

A	ppqq	HL	BC	
39	36	ppqq	0004	(antes)
		ppqq - 1	0003	(depois)

(resultados dos flags são os mesmos dos casos anteriores)

- b) CPI • Compara A com conteúdo do endereço ppqq. Incrementa HL e decrementa BC.

A	ppqq	HL	BC	
39	36	ppqq	0004	(antes)
		ppqq + 1	0003	(depois)

(resultados dos flags são os mesmos dos casos anteriores)

- c) CPDR • Repetição de CPD: compara A com endereço em HL, decrementando HL e BC até achar valor igual ou BC ficar nulo.

A	Memória	HL	BC	
39	ppqq 30	ppqq	0005	(antes)
	ppqq - 1 31			
	ppqq - 2 37			
	ppqq - 3 39	⇒ Achou valor igual a A:		
	ppqq - 4 36	ppqq - 4	0001	(depois)
	ppqq - 5 4A			

- d) CPIR • Repetição de CPI: compara A com endereço em HL, incrementando HL e decrementando BC até achar valor igual ou BC ficar zero.

VI.4 JUMP TO ADDRESS – DESVIO PARA ENDEREÇO DE MEMÓRIA

<i>8080/85</i>	<i>Z80</i>	<i>Explicação</i>
JMP endereço C3 qq pp	JP endereço C3 qq pp	Desvio incondicional para endereço ppqq.
JNZ endereço C2 qq pp	JP NZ, endereço C2 qq pp	Desvia para ppqq se flag Z = 0 (NOT ZERO).
JZ endereço CA qq pp	JP Z, endereço CA qq pp	Desvia para ppqq se flag Z = 1 (ON ZERO).
JNC endereço D2 qq pp	JP NC, endereço D2 qq pp	Desvia para ppqq se flag C = 0 (NOT CARRY).
JC endereço DA qq pp	JP C, endereço DA qq pp	Desvia para ppqq se flag C = 1 (ON CARRY).
JPO endereço E2 qq pp	JP PO, endereço E2 qq pp	Desvia para ppqq se flag PO = 0 (PARITY ODD).
JPE endereço EA qq pp	JP PE, endereço EA qq pp	Desvia para ppqq se flag PO = 1 (PARITY EVEN).
JP endereço F2 qq pp	JP P, endereço F2 qq pp	Desvia para ppqq se flag S = 0 (PLUS).
JM endereço FA qq pp	JP M, endereço FA qq pp	Desvia para ppqq se flag S = 1 (MINUS).
PCHL E9	JP (HL) E9	Desvio incondicional para endereço em HL.
	JP (IX) DD E9	Desvio incondicional para endereço em X.
	JP (IY) FD E9	Desvio incondicional para endereço em Y.

VI.5

JR (JUMP RELATIVE) – DESVIO RELATIVO DE *ds* posições para frente ou para trás (só Z80) (ds = -126 a +129)

JR endereço	18 ds (+)	Desvio relativo incondicional.
JR NC, endereço	30 ds	Desvio relativo se flag C = 0 (NOT CARRY).

JR C, endereço	38 ds	– Desvio relativo se flag C = 1 (CARRY).
JR NZ, endereço	20 ds	– Desvio relativo se flag Z = 0 (NOT ZERO).
JR Z, endereço	28 ds	– Desvio relativo se flag Z = 1 (ZERO).

Nota (+) O valor ds do deslocamento ou offset do código objeto é calculado pelo tradutor Assembler do seguinte modo:
ds = endereço - (conteúdo de PC) - 2

VI.6

DJNZ (DECREMENT AND JUMP IF B NOT ZERO) – (só Z80)

DJNZ endereço	10 ds	– Decrementa B e desvio relativo se B não for zero.
---------------	-------	---

Exemplos:

8080/85	Z80	
a) JMP ROTX	JP ROTX	– Desvia para endereço simbólico ROTX.
b) JC Z1	JP C, Z1	– Desvia para endereço Z1 se Carry = 1.
c) JNZ Z1	JP NZ, Z1	– Desvia para endereço Z1 se Flag Z = 0.
d) PCHL	JP (HL)	– Desvia para endereço ppqq, que está em HL.
e) só Z80	JP (IX)	– Desvia para endereço ppqq, que está em X.
f) só Z80	JR Z1	– Desvia para endereço Z1, que deve estar dentro do limite de -126 a +127 bytes a contar da posição dessa instrução (desvio relativo curto).
g) só Z80	JR Z, Z1	– Desvio relativo para Z1 se flag Z = 1.
h) só Z80	DJNZ Z1	– Decrementa registro B (contador); se B não for nulo desvia para endereço Z1.

VI.7 PROGRAMA-EXEMPLO:

COMPARAÇÃO E DESVIO: ACHAR O MAIOR DE DOIS VALORES QUE ESTÃO NA MEMÓRIA

Comparar os valores que estão nos endereços absolutos 51 e 52 (ou endereços simbólicos Z1 e Z2). Coloque o maior valor no endereço absoluto 53 (ou endereço simbólico Z3).

Algoritmo:

Se conteúdo de 51 (ou de Z1) MAIOR QUE conteúdo de 52 (ou Z2)

ENTÃO conteúdo de 51 (ou de Z1) vai para endereço 53 (ou Z3);

CASO CONTRÁRIO, conteúdo de 52 (ou Z2) vai para endereço 53 (ou Z3).

VERSÃO PARA 8080/85

Código-Objeto

Assembler

```

; COMPARACAO E DESVIO USANDO
; ENDEREÇOS ABSOLUTOS 51, 52 E 53.
;
0000 215100  EX8509: LXI    H,51H    ; 1)
0003 7E      MOV     A,M        ; 2)
0004 23      INX     H          ; 3)
0005 8E      CMP     M          ; 4)
0006 D20A00  JNC     COLOC      ; 5)
0009 7E      MOV     A,M        ; 6)
000A 23      COLOC: INX     H          ; 7)
000B 77      MOV     M,A        ; 8)
000C C30C00  PARE:  JMP     PARE    ; 9)
000F                END    EX8509

```

Comentários:

Ver 1 a 9 do programa seguinte.

Código-Objeto

Assembler

```

; COMPARACAO E DESVIO USANDO
; ENDEREÇOS SIMBOLICOS Z1, Z2 E Z3.
;
0000 210F00  EX8510: LXI    H,Z1     ; 1)
0003 7E      MOV     A,M        ; 2)
0004 23      INX     H          ; 3)
0005 8E      CMP     M          ; 4)
0006 D20A00  JNC     COLOC      ; 5)
0009 7E      MOV     A,M        ; 6)
000A 23      COLOC: INX     H          ; 7)
000B 77      MOV     M,A        ; 8)
000C C30C00  PARE:  JMP     PARE    ; 9)

```


Comentários:

1 a 9) Os mesmos do programa seguinte.

Código-Objeto Assembler

```
                ; COMPARAÇÃO E DESVIO USANDO
                ; ENDEREÇOS SIMBÓLICOS Z1, Z2 E Z3.
                ;
1000            ORG      1000H
1000 210C10     EX10Z80: LD      HL,Z1      ; 1)
1003 7E         LD      A,(HL)             ; 2)
1004 23         INC     HL                  ; 3)
1005 8E         CP      (HL)               ; 4)
1006 3001       JR      NC,COLOC            ; 5)
1008 7E         LD      A,(HL)             ; 6)
1009 23         COLOC:  INC     HL          ; 7)
100A 77         LD      (HL),A             ; 8)
100B 76         HALT                       ; 9)
100C 5F         Z1:     DEFB    5FH         ;10)
101D 8A         Z2:     DEFB    0BAH       ;11)
101E 00         Z3:     DEFB    00         ;12)
101F            END      EX10Z80
```

Comentários:

- 1) Carrega imediato o endereço 51 (ou Z1) no par HL.
- 2) Carrega Acumulador AC com o valor do byte endereçado pelo par HL, isto é, valor em 51 (ou em Z1).
- 3) Incrementa par HL, que contém agora o endereço 52 (ou Z2).
- 4) Compara AC com o conteúdo de 52 (ou Z2).
- 5) Se for maior (NOT CARRY), desvia para COLOC.
- 6) Caso contrário, conteúdo de 52 (ou Z2) vai para AC.
- 7) Incrementa par HL, que contém agora 53 (ou endereço Z3).
- 8) Conteúdo de AC (maior valor) vai para endereço 53 (ou Z3).
- 9) Pare.
- 10) Valor em Z1 é 5FH.
- 11) Valor em Z2 é BAH.
- 12) Valor em Z3 é zero antes da execução do programa.

VI.8 OBSERVAÇÃO: USO DO BIT CARRY NA COMPARAÇÃO

Para comparar dois valores absolutos sem o sinal (unsigned numbers) usou-se o seguinte esquema:

SE $Z1 < Z2$ então $CARRY = 1$ (= 0 no 6502), uma vez que $Z1 - Z2 < 0$, e haverá necessidade de “empréstimo ou borrow” indicado no CARRY.

Se $Z1 \geq Z2$ então $CARRY = 0$ (= 1 no 6502), pois não há necessidade do “empréstimo ou borrow”.

Sendo assim, usou-se:

BCS COLOC (BRANCH IF CARRY SET) para 6502.

JR NC, COLOC (JUMP RELATIVE IF NOT CARRY) para Z80.

JNC COLOC (JUMP IF NOT CARRY) para 8080/85.

Se considerarmos os valores com o respectivo sinal no último bit à esquerda (e variando no intervalo -126 a $+127$), podemos verificar esse bit através dos bits de SINAL NEGATIVO do flag de status:

BPL COLOC (BRANCH IF PLUS) para 6502

JR P, COLOC (BRANCH IF PLUS) para Z80

JP COLOC (BRANCH IF PLUS) para 8080/85

Nota: No 6502, SOMENTE o flag de CARRY possui valor inverso ao dos demais microprocessadores.

EXERCÍCIOS

De Recapitulação

1. Descrever e citar exemplos de cada uma das formas possíveis das seguintes instruções:

- COMPARE MEMORY
- COMPARE REGISTER
- BRANCH OU JUMP

De Aplicação

Escrever a versão Assembly e respectivo código-objeto dos programas abaixo mencionados. Todos os programas devem ser testados manualmente, instrução por instrução, através de um exemplo numérico, mencionando os conteúdos dos registros e memórias afetadas.

1. Comparar os valores que estão nos endereços hexadecimais 1500 e 1600. Colocar o menor desses valores no endereço 1601 e o maior, no endereço 1602.

2. Comparar os valores que estão nos endereços simbólicos V1 e V2. Colocar o menor desses valores no endereço simbólico S1 e o maior no endereço simbólico S2.
3. Comparar, um por um, os bytes que estão nos endereços V1, V2, V3 e V4 com os bytes que estão nos endereços simbólicos S1, S2, S3 e S4. Colocar todos os bytes de maior valor a partir do endereço simbólico MAIOR e os bytes de menor valor a partir do endereço MENOR.
4. ORDENAÇÃO DE TRÊS BYTES – Comparar os bytes que estão nos endereços simbólicos V1, V2 e V3 e colocá-los em ordem crescente de seus valores.

Sugestão: Utilizar o seguinte algoritmo:

- a) Comparar V1 e V2: se $V1 > V2$ trocar conteúdos de V1 e V2; caso contrário nada se faz.
- b) Comparar V2 e V3: se $V2 > V3$ trocar conteúdos de V1 e V2, caso contrário nada se faz.
- c) Repetir as operações a e b e terminar.

CAPÍTULO

VII

INSTRUÇÕES PARA OPERAÇÕES LÓGICAS “AND”, “OR” E “EXCLUSIVE OR” PARA 8080/85 E Z80

VII.1 FORMAS GERAIS DAS INSTRUÇÕES

As instruções estudadas neste capítulo são:

– OPERAÇÃO LÓGICA “AND” OU “E”

8080/85 Z80

ANI dd	AND dd	– Operação AND de A com valor imediato dd
ANA reg _i	AND reg _i	– Operação AND de A com registro reg _i .
ANA M	AND (HL)	– Operação AND de A com memória, com endereço em HL.
	AND (IX + ds)	– <i>Idem</i> , endereço em IX + ds.
	AND (IY + ds)	– <i>Idem</i> , endereço em IY + ds.

– OPERAÇÃO LÓGICA “OR” ou “OU”

ORI dd	OR dd	– Operação OR de A com valor imediato dd.
ORA reg _i	OR reg _i	– Operação OR de A com registro reg _i .
ORA M	OR (HL)	– Operação OR de A com memória, com endereço em HL.
	OR (IX + ds)	– <i>Idem</i> , endereço em IX + ds.
	OR (IY + ds)	– <i>Idem</i> , endereço em IY + ds.

– OPERAÇÃO “OU EXCLUSIVO”

XRI dd	XOR dd	– Operação XOR de A com valor imediato dd.
XRA reg _i	XOR reg _i	– Operação XOR de A com registro reg _i .

XRA M	XOR (HL)	– Operação XOR de A com memória, com endereço em HL.
	XOR (IX + ds)	– <i>Idem</i> , com endereço em IX + ds.
	XOR (IY + ds)	– <i>Idem</i> , com endereço em IY + ds.

– PROGRAMA-EXEMPLO

Além dos exemplos ilustrados de cada instrução, o capítulo contém o programa-exemplo:

- SEPARAR POR MASCARAGEM “AND” OS QUATRO BITS MENOS SIGNIFICATIVOS DE UM BYTE.

VII.2 AND: OPERAÇÃO LÓGICA AND DO ACUMULADOR OU REGISTRO A COM OPERANDO IMEDIATO, REGISTRO OU CONTEÚDO DA MEMÓRIA, RESULTADO FICA EM A.

Operação lógica AND ou “E”: $0 + 0 = 0$; $0 + 1 = 1 + 0 = 0$ e $1 + 1 = 1$.

8080/85		Z80		Explicação
Assembler	Código-Objeto	Assembler	Código-Objeto	
ANI dd	E6 dd	AND dd	E6 dd	– AND do valor imediato dd com A.
ANA A	A7	AND A	A7	– AND de A com A.
ANA B	A0	AND B	A0	– AND de B com A.
ANA C	A1	AND C	A1	– AND de C com A.
ANA D	A2	AND D	A2	– AND de D com A.
ANA E	A3	AND E	A3	– AND de E com A.
ANA H	A4	AND H	A4	– AND de H com A.
ANA L	A5	AND L	A5	– AND de L com A.
ANA M	A6	AND (HL)	A6	– AND de conteúdo de ppqq com A; ppqq está em HL.
		AND (IX+ds) DD	A6 ds	– AND de conteúdo de ppqq + ds com A; X contém ppqq.
		AND (IY+ds) FD	A6 ds	– AND de conteúdo de ppqq + ds com A; Y contém ppqq.

VII.3 OR – OPERAÇÃO LÓGICA “OR” DO ACUMULADOR OU REGISTRO A COM OPERANDO IMEDIATO, REGISTRO OU CONTEÚDO DE MEMÓRIA, RESULTADO FICA EM A.

Operação lógica OR ou OU: $0 + 0 = 0$, $0 + 1 = 1 + 0 = 1$ e $1 + 1 = 1$.

ORI dd	F6 dd	OR dd	F6 dd	– OR do valor imediato dd com A.
ORA A	B7	OR A	B7	– OR de A com A.
ORA B	B0	OR B	B0	– OR de A com B.
ORA C	B1	OR C	B1	– OR de A com C.
ORA D	B2	OR D	B2	– OR de A com D.
ORA E	B3	OR E	B3	– OR de A com E.
ORA H	B4	OR H	B4	– OR de A com H.
ORA L	B5	OR L	B5	– OR de A com L.
ORA M	B6	OR (HL)	B6	– OR do conteúdo de ppqq com A, ppqq está em HL.
		OR (IX+ds)	DD B6 ds	– OR do conteúdo de ppqq + ds com A; X contém ppqq.
		OR (IY+ds)	FD B6 ds	– OR do conteúdo de ppqq + ds com A; Y contém ppqq.

VII.4 XOR – OPERAÇÃO LÓGICA “EXCLUSIVE OR” DO CONTEÚDO DO REGISTRO A COM O VALOR IMEDIATO, REGISTRO OU CONTEÚDO DA MEMÓRIA; RESULTADO FICA EM A.

Operação “OU EXCLUSIVO”: $0 + 0 = 1 + 1 = 0$ e $1 + 0 = 0 + 1 = 1$.

8080/85		Z80		Explicação
XRI dd	EE dd	XOR dd	EE dd	– XOR do valor imediato dd com A.
XRA A	AF	XOR A	AF	– XOR de A com A.
XRA B	A8	XOR B	A8	– XOR de A com B.
XRA C	A9	XOR C	A9	– XOR de A com C.
XRA D	AA	XOR D	AA	– XOR de A com D.
XRA E	AB	XOR E	AB	– XOR de A com E.
XRA H	AC	XOR H	AC	– XOR de A com H.
XRA L	AD	XOR L	AD	– XOR de A com L.

XRA M	AE	XOR (HL)	AE	— XOR do conteúdo de ppqq com A; ppqq está em HL.
		XOR (IX + ds)	DD AE ds	— XOR do conteúdo de ppqq + ds com A; ppqq está em X.
		XOR (IY + ds)	FD AE ds	— XOR do conteúdo de ppqq + ds com A; ppqq está em Y.

Exemplos: (Operação lógica AND)

8080/85 Z80

ANI 32H AND 32H • AND do valor em A com valor imediato 32H.

Ilustração: Flags de Status

Conteúdo de A = 39 : 0011 1001
 Operando = 32 : 0011 0010
 Resultado em A = 30 : 0011 0000

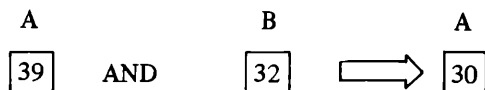
Flag S = 0 ← (resultado é positivo: bit 7 = 0)
 Flag Z = 0 → (resultado não é zero)

Paridade P/O = 1
 (Paridade par, resultado contém 2 bits um)

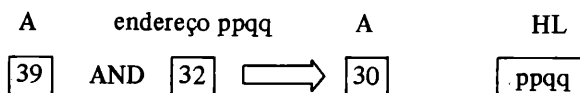
N = C = 0 (sempre)

Ac = 0 (8080) e 1 (8085 e Z80)

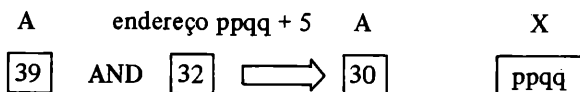
ANA B AND B • AND do valor em A com valor em B.



ANA M AND (HL) • AND do valor em A com valor no endereço ppqq da memória; ppqq está em HL.



só Z80 AND (IX + 5) • AND do valor em A com valor no endereço ppqq + 5; ppqq está em X.



Exemplos: (Operação lógica OR)

8080/85 Z80

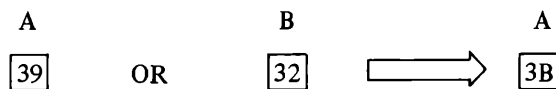
ORI 32H OR 32H • Operação OR do valor em A com valor 32H.

Ilustração: Flags de Status

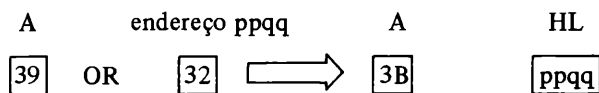
Conteúdo de A = 39 : 0011 1001
 Operando = 32 : 0011 0010
 Resultado em A = 3B : 0011 1011

Flags: S = 0 (resultado é positivo: bit 7 é zero)
 Z = 0 (resultado não é zero)
 P ou P/O = 0 (número ímpar de bits um no resultado)
 N e C sempre zero
 Ac = 0 (8080/85) e 1 (Z80)

ORA B OR B • Operação OR de A com valor em B.



ORA M OR (HL) • Operação OR de A com valor do endereço em ppqq; ppqq está em HL.



(Operação lógica EXCLUSIVE OR)

8080/85 Z80

XRI 1CH XOR 1CH • Operação EXCLUSIVE OR de valor em A com valor imediato 1CH.

Ilustração: Flags de Status

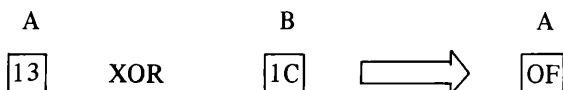
Conteúdo de A = 13 : 0001 0011
 Operando = 1C : 0001 1100
 Resultado em A = 0F : 0000 1111

Flags: S = 0 (resultado positivo: bit 7 é zero)
 Z = 0 (resultado não zero)

P ou P/0 = 1 (paridade par: quatro bits um no resultado)

$$N = C = 0 \text{ (sempre)}$$
$$Ac = 0 \text{ (8080/85)} \text{ e } 1 \text{ (Z80)}$$

XRA B XOR B • Operação EXCLUSIVE OR de A com B.



VII.5 PROGRAMA-EXEMPLO: USO DA OPERAÇÃO LÓGICA “AND”: SEPARAR (POR MASCARAGEM) OS QUATRO BITS MENOS SIGNIFICATIVOS DE UM BYTE

Separar os quatro bits menos significativos de um byte que está no endereço absoluto A1 (ou simbólico Z1) e colocar esses bits separados no endereço absoluto A2 (ou simbólico Z2).

Algoritmo ou operação:

- Apagar (por mascaramento com operação AND) os quatro bits mais significativos de A1 ou Z1.
- Colocar os bits que restaram no endereço A2 ou Z2.

Operação lógica AND: $0 + 0 = 0 + 1 = 1 + 0 = 0$ (Bit 0 apaga o outro bit.)
 $1 + 1 = 1$ (Bit 1 mantém o outro bit.)

VERSÃO PARA 8080/85

Código-Objeto

Assembler

```

; SEPARA OS QUATRO BITS MENOS SIGNIFICATIVOS DE
; UM BYTE USANDO A OPERACAO LOGICA "AND" COM
; ENDEREÇOS ABSOLUTOS 00A1 E 00A2.
;
0000 3AA100    EX8511: LDA    0A1H        ; 1)
0003 E60F      ANI     00001111B      ; 2)
0005 32A200      STA    0A2H        ; 3)
0008 C30800    PARE: JMP    PARE      ; 4)
000B           END    EX8511

```

Código-Objeto

Assembler

```

; SEPARA OS QUATRO BITS MENOS SIGNIFICATIVOS DE
; UM BYTE USANDO A OPERACAO LOGICA "AND" COM
; ENDEREÇOS SIMBOLICOS Z1 E Z2.
;
0000 3A0B00  EX8512: LDA    Z1      ; 1)
0003 E60F      ANI    00001111B ; 2)
0005 320C00      STA    Z2      ; 3)
0008 C30800  PARE:  JMP    PARE    ; 4)
0008 3C      Z1:   DB    3CH      ; 5)
000C 00      Z2:   DB    00      ; 6)
0000          END    EX8512

```

VERSÃO PARA Z80

Código-Objeto

Assembler

```

; SEPARA OS QUATRO BITS MENOS SIGNIFICATIVOS DE
; UM BYTE USANDO A OPERACAO LOGICA "AND" COM
; ENDEREÇOS ABSOLUTOS 00A1 E 00A2.
;
1000          ORG    1000H
1000 3AA100  EX11Z80: LD    A,(0A1H) ; 1)
1003 E60F      AND    0FH          ; 2)
1005 32A200      LD    (0A2H),A    ; 3)
1008 76      HALT                  ; 4)
1009          END    EX11Z80

```

Código-Objeto

Assembler

```

; SEPARA OS QUATRO BITS MENOS SIGNIFICATIVOS DE
; UM BYTE USANDO A OPERACAO LOGICA "AND" COM
; ENDEREÇOS SIMBOLICOS Z1 E Z2.
;
1000          ORG    1000H
1000 3A0910  EX12Z80: LD    A,(Z1)  ; 1)
1003 E60F      AND    0FH          ; 2)
1005 320A10      LD    (Z2),A      ; 3)
1008 76      HALT                  ; 4)
1009 3C      Z1:   DEFB  3CH      ; 5)
100A 00      Z2:   DEFB  00      ; 6)
100B          END    EX12Z80

```


Comentários (8080/85 e Z80): tanto para endereços absolutos como simbólicos.

- 1) Carrega A com conteúdo do endereço A1 ou Z1.
- 2) Operação AND do conteúdo de A com valor imediato OF : $3C + OF = AC$.
 $00111100 + 00001111 = 00001100 = 0C$.
- 3) Bits separados em A vão para endereço A2 ou Z2.
- 4) Pare.
- 5) Endereço Z1 contém valor 3C.
- 6) Endereço Z2 conterà valor 0C após a execução do programa.
- 7) Fim do programa.

EXERCÍCIOS

De Recapitulação

1. Descrever e citar exemplos de cada uma das formas possíveis das instruções seguintes: :
 - LOGICAL AND
 - LOGICAL OR
 - LOGICAL EXCLUSIVE OR

De Aplicação

Escrever a versão Assembler e respectivo código-objeto dos programas abaixo mencionados. Todos os programas devem ser testados manualmente, instrução por instrução, através de um exemplo numérico, mencionando os conteúdos dos registros e memórias afetadas, após cada instrução.

1. Apagar o bit de sinal (isto é, colocar zero no bit 7) de um byte, através da operação AND. Manter demais bits inalterados.
2. Colocar bit 1 nos quatro bits mais significativos de um byte, através de uma máscara de operação LOGICAL OR. Manter demais bits inalterados.
3. Colocar bit 1 (SET) nos bits das posições 0, 3 e 5 de um byte de memória, utilizando uma máscara apropriada. Manter demais bits inalterados.
4. Colocar bit 1 (SET) nos bits de posição 0 e 3 e bit 0 (RESET) nos bits de posição 5 e 6 de um byte da memória, utilizando operações lógicas e máscaras apropriadas. Manter demais bits inalterados.

CAPÍTULO

VIII

APLICAÇÕES: PROGRAMAS-EXEMPLOS COM CONTADOR DE REPETIÇÕES (LOOPS)

VIII.1 INTRODUÇÃO

Neste capítulo examinaremos vários exemplos de programas que utilizam praticamente todos os tipos de instruções (MOVE ou LOAD, ADD, COMPARE, JUMP e AND) examinados até aqui.

Todos os programas aplicam a técnica de repetição de um mesmo trecho de programa quantas vezes necessárias. Esta técnica denominada “LOOPS” ou REPETIÇÃO DE COMANDOS é bastante utilizada em qualquer tipo de programação utilizando linguagem de alto nível ou Assembler.

Nas linguagens de alto nível, esta técnica é usada nos seguintes tipos de comando de LOOPS ou REPETIÇÃO:

<i>BASIC</i>	<i>FORTRAN</i>	<i>PASCAL</i>
FOR I = 1 TO N STEP 2	DO 25 I = 1, N, 2	FOR I := 1 TO N DO BEGIN... END;
....	...	
NEXT I	25 CONTINUE	REPEAT BEGIN
		...
		END UNTIL I = N;
		WHILE I <= N DO BEGIN... END,

Os programas exemplos estudados neste capítulo são:

- SOMA DE N VALORES DE UM BYTE CADA;
- ACHAR O MAIOR VALOR DE UMA SÉRIE DE K NÚMEROS;
- CONTAR O NÚMERO DE CARACTERES DE UMA SEQUÊNCIA OU “STRING” DE CARACTERES;
- SOMA DE VALORES COM MÚLTIPLA PRECISÃO.

VIII.2 OPERAÇÃO DE REPETIÇÃO (LOOPS): SOMA DE N VALORES DE UM BYTE CADA

Calcular a soma de N valores de um byte cada.

O valor N está no endereço absoluto A1 (ou endereço simbólico TAM) e a série de valores começa no endereço absoluto A2 (ou endereço simbólico SÉRIE).

Colocar o valor da soma no endereço simbólico A0 (ou simbólico SOMA).

Ilustração:

Condição Inicial: Registro A ou Acumulador

00

Registro usado como contador

00 ou N = 3

Valores da memória:

Endereço SOMA

00

– Local onde será colocado o valor da soma.

TAM

03

– N = 3

SÉRIE

15

05

} Série de 3 valores

C1

Situação após a execução do programa:

Registro ou Acumulador A

DB

Registro usado como contador

03 ou 00

Valores da memória:

Endereço SOMA

DB

TAM

03

SÉRIE

15

05

} $15 + 05 + C1 = DB$

C1

Nota: O contador pode ser incrementado ou decrementado.

Algoritmo:

Início: Colocar valor N em um contador, zerar Acumulador.

Repetir a operação REPETE até o valor do contador ficar nulo.

Operação REPETE; Acumulador = Acumulador + um valor da série.

Contador = Contador – 1;

Se Contador = 0 então terminar.

Terminar: Colocar valor total do Acumulador no endereço A0 ou SOMA.

VERSÃO PARA 8080/85

Código-Objeto

Assembler

```

; SOMA DE N VALORES DE UM BYTE CADA
; USANDO ENDEREÇOS SIMBOLICOS.
;
0000 211200  EX8513: LXI    H,TAM    ; 1)
0003 46      MOV     B,H        ; 2)
0004 97      SUB     A          ; 3)
0005 23      REPET: INX    H      ; 4)
0006 86      ADD     M          ; 5)
0007 05      DCR     B          ; 6)
0008 C20500  JNZ     REPET    ; 7)
0008 321100  STA     SOMA      ; 8)
000E C30E00  PARE:  JMP     PARE ; 9)
0011 00      SOMA:  DB      00  ;10)
0012 03      TAM:   DB      03H ;11)
0013 15      SERIE: DB      15H ;12)
0014 05      DB      05H ;13)
0015 C1      DB      0C1H ;14)
0016        END     EX8513

```

VERSÃO PARA Z80

Código-Objeto

Assembler

```

; SOMA DE N VALORES DE UM BYTE CADA
; USANDO ENDEREÇOS SIMBOLICOS.
;
1000        ORG     1000H
1000 210F10  EX13Z80: LD     HL,TAM ; 1)
1003 46      LD     B,(HL)    ; 2)
1004 97      SUB     A        ; 3)
1005 23      REPET: INC    HL  ; 4)
1006 86      ADD     (HL)     ; 5)
1007 05      DEC     B        ; 6)
1008 20FB    JR     NZ,REPET  ; 7)

```

```
100A 320E10      LD      (SOMA),A      ; 8)
100D 76          HALT                    ; 9)
100E 00      SOMA:  DEFB      00        ;10)
100F 03      TAM:   DEFB      03H       ;11)
1010 15      SERIE: DEFB      15H       ;12)
1011 05          DEFB      05H         ;13)
1012 C1          DEFB      0C1H        ;14)
1013          END      EX13Z80
```

Comentários (8080/85 e Z80)

- 1) Valor imediato (endereço) TAM vai para par HL.
- 2) Valor N que está no endereço TAM vai para o contador B.
- 3) Acumulador menos acumulador igual a zero.
- 4) Incrementa par HL que contém o endereço de SÉRIE.
- 5) Acumulador = Acumulador + um valor da série.
- 6) Decrementa contador B.
- 7) Se contador não chegar a zero vai para REPET; e na repetição soma próximo valor.
- 8) Se contador chegar a zero, coloca valor total do acumulador no endereço SOMA.
- 9) Pare.
- 10) Endereço SOMA irá conter valor somado $15 + 05 + C1 = DB$ após a execução do programa.
- 11) TAM contém o valor $N = 3$.
- 12) Primeiro valor da série.
- 13) Segundo valor da série.
- 14) Terceiro valor da série.
- 15) Etc. até no máximo 256 valores podem ser somados.
- 16) Fim do programa.

Nota: Para atribuir endereços absolutos A0, A1 e A2 aos endereços simbólicos SOMA, TAM e SÉRIE, basta utilizar a instrução diretiva ORG:

```
                ORG      AOH
SOMA:          DB      ou DEFB 00
TAM:           DB      ou DEFB 03
SÉRIE:         DB      ou DEFB 15H
                DB      ou DEFB 05H
                DB      ou DEFB C1H
```

VIII.3 ACHAR MAIOR VALOR DE UMA SÉRIE DE K NÚMEROS

Achar o maior valor de uma série de K valores de um byte cada. Esses valores estão colocados, na memória, a partir do endereço SÉRIE. O valor K que indica o tamanho da série está no endereço COMPR.

O maior valor encontrado deve ser colocado no endereço simbólico chamado MAXIM.

Algoritmo:

Início: Contador = valor K;
Acumulador = zero.

Repetir a operação REPETE até que o valor do contador seja igual a zero.

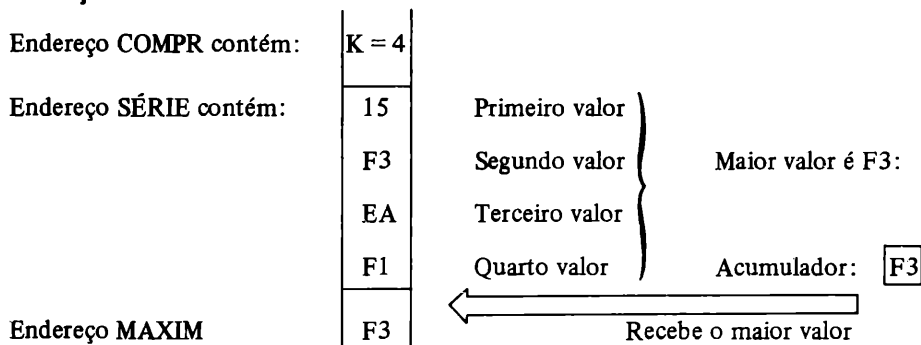
REPETE: Se Acumulador maior ou igual a um valor da SÉRIE então nada é feito; caso contrário, Acumulador = esse valor da SÉRIE.

Contador = Contador - 1;

Terminar: Coloca maior valor, que está no Acumulador, no endereço MAXIM.

Nota: O contador pode decrescer de valor K até zero, ou crescer do valor 0 até K, dependendo da maior conveniência de programação.

Ilustração:



VERSÃO PARA 8080/85

Código-Objeto

Assembler

```

; MAIOR VALOR DE UMA SERIE DE K NUMEROS.
;
0000 211600  EX8514: LXI    H,COMPR ; 1)
0003 46      MOV     B,H      ; 2)
0004 97      SUB     A        ; 3)
0005 23      REPET: INX    H   ; 4)

```

```

0006 BE          CMP      M      ; 5)
0007 D20B00      JNC      DIMIN  ; 6)
000A 7E          MOV      A,M    ; 7)
000B 05          DIMIN: DCR      B      ; 8)
000C C20500      JNZ      REPET  ; 9)
000F 321500      STA      MAXIM  ;10)
0012 C31200      PARE:  JMP      PARE  ;11)
0015 00          MAXIM: DB      00    ;12)
0016 04          COMPR: DB      04H   ;13)
0017 15          SERIE: DB      15H   ;14)
0018 F3          DB      0F3H   ;15)
0019 EA          DB      0EAH   ;16)
001A F1          DB      0F1H   ;17)
001B             END      EX8514

```

VERSÃO PARA Z80

Código-Objeto

Assembler

```

; MAIOR VALOR DE UMA SERIE DE K NUMEROS.
;
1000             ORG      1000H
1000 211110      EX14Z80: LD      HL,COMPR  ; 1)
1003 46          LD      B,(HL)  ; 2)
1004 97          SUB      A      ; 3)
1005 23          REPET: INC      HL      ; 4)
1006 8E          CP      (HL)    ; 5)
1007 3001        JR      NC,DIMIN  ; 6)
1009 7E          LD      A,(HL)  ; 7)
100A 10F9        DIMIN: DJNZ     REPET  ; 8)
100C 321010      LD      (MAXIM),A  ; 9)
100F 76          HALT          ;10)
1010 00          MAXIM: DEFB     00    ;11)
1011 04          COMPR: DEFB     04H   ;12)
1012 15          SERIE: DEFB     15H   ;13)
1013 F3          DEFB     0F3H   ;14)
1014 EA          DEFB     0EAH   ;15)
1015 F1          DEFB     0F1H   ;16)
1016             END      EX14Z80

```

Comentários (8080/85 e Z80)

- 1) Coloca no par HL endereço (imediato) COMPR.
- 2) Coloca no contador B o valor K = 04 que estava em COMPR.
- 3) Zera acumulador A: A – A = 0.

- 4) Incrementa HL, que passa a conter o endereço seguinte SÉRIE.
- 5) Compara acumulador com primeiro byte que está em SÉRIE.
- 6) Se o acumulador for maior, nada se faz, prosseguindo em DIMIN.
- 7) Se acumulador for menor, temos um novo valor maior, que é colocado no acumulador.
- 8) Decrementa contador B, e se resultado for maior que zero desvia para REPET.
- 9) Se contador B for zero, coloca maior valor que está em A, no endereço MAXIM.
- 10) Pare.
- 11) Define na memória, endereço MAXIM que receberá o maior valor encontrado.
- 12) Define na memória, endereço COMPR que contém o valor K = 04.
- 13) Define na memória, no endereço SÉRIE, o primeiro valor da série.
- 14) Define segundo valor.
- 15) Define terceiro valor.
- 16) Define quarto valor etc. até um máximo de 256 valores.
- 17) Fim do programa.

Nota – sobre as linhas 8 e 9:

No programa para Z80 foi utilizado o comando DJNZ (Decrementa B e JUMP) no lugar de DCR e JNZ. Sendo assim houve uma instrução a menos.

VIII.4 CONTAR NÚMERO DE CARACTERES ASCII EM UMA SEQUÊNCIA (STRINGS) DE CARACTERES

Determinar o tamanho (número de caracteres) de um string de caracteres. O string ou sequência de caracteres está colocado, na memória, a partir do endereço STRIN e termina com um caractere CR ou CARRIAGE RETURN (corresponde ao valor hexadecimal 0D em notação ASCII), que marca o fim da sequência.

Colocar o valor do número de caracteres obtido (sem incluir o caractere CR) no endereço COMPR.

Algoritmo:

Início: Contador de caracteres recebe valor zero.

Acumulador recebe o caractere Carriage Return = 'CR' = 0D (em hexadecimal).

Repetir: Compara próximo caractere do string com Acumulador: se for 'CR', Terminar, caso contrário, incrementa contador e vai para Repetir.

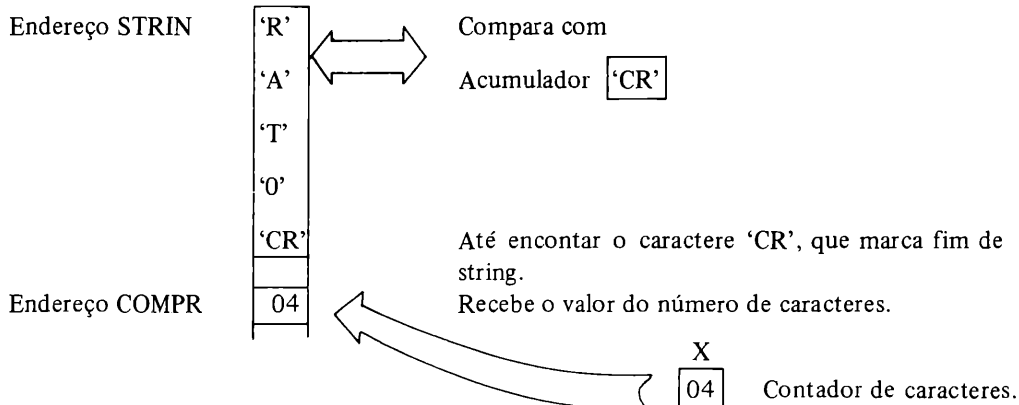
Terminar: Coloca valor do contador no endereço COMPR.

Ilustração:

Acumulador: contém caractere 'CR' marcador de fim de string.

Contador (Registro B): começa com 00 e termina com 04.

String de caracteres na memória:

**VERSÃO PARA 8080/85**

Código-Objeto

Assembler

```

; CONTADOR DE CARACTERES DE UM STRING.
;
0000 211800  EX8515: LXI    H,STRIN ; 1)
0003 0600      MVI    B,0      ; 2)
0005 3E0D      MVI    A,0DH    ; 3)
0007 BE       REPET: CMP    H      ; 4)
0008 CA1000    JZ      TERMI    ; 5)
000B 04        INR     B        ; 6)
000C 23        INX     H        ; 7)
000D C30700    JMP     REPET    ; 8)
0010 78        TERMI: MOV    A,B   ; 9)
0011 321700    STA     COMPR    ;10)
0014 C31400    PARE:  JMP     PARE ;11)
0017 00        COMPR: DB     00   ;12)
0018 52        STRIN: DB     'R'  ;13)
0019 41        DB     'A'        ;14)
001A 54        DB     'T'        ;15)
001B 4F        DB     'O'        ;16)
001C 0D        DB     0DH        ;17)
001D          END     EX8515

```

VERSÃO PARA Z80

Código-Objeto

Assembler

```

; CONTADOR DE CARACTERES DE UM STRING.
;
1000          ORG      1000H
1000 211410  EX15Z80: LD      HL,STRIN ; 1)
1003 0600          LD      B,0        ; 2)
1005 3E00          LD      A,0DH      ; 3)
1007 8E          REPET: CP      (HL)   ; 4)
1008 2804          JR      Z,TERMI    ; 5)
100A 04          INC      B           ; 6)
100B 23          INC      HL          ; 7)
100C 18F9          JR      REPET      ; 8)
100E 78          TERMI: LD      A,B    ; 9)
100F 321310       LD      (COMPR),A   ;10)
1012 76          HALT                ;11)
1013 00          COMPR: DEFB      00   ;12)
1014 52          STRIN: DEFB      'R'  ;13)
1015 41          DEFB      'A'        ;14)
1016 54          DEFB      'T'        ;15)
1017 4F          DEFB      'O'        ;16)
1018 0D          DEFB      0DH        ;17)
1019          END      EX15Z80

```

Comentários (8080/85 e Z80)

- 1) Endereço do início do string STRIN vai para par HL.
- 2) Registro B usado como contador recebe valor zero.
- 3) Acumulador A recebe caractere CR = 0DH, marcador de fim de string.
- 4) Compara acumulador com primeiro caractere do string.
- 5) Se for igual a 0DH, vai para TERMI.
- 6) Se for diferente, incrementa contador B.
- 7) Incrementa par HL, apontando para próximo caractere do string.
- 8) Vai para REPET.
- 9) Terminou, coloca valor do contador B no acumulador.
- 10) Que vai ser colocado no endereço COMPR.
- 11) Pare.
- 12) Define na memória o endereço COMPR, que vai receber o número de caracteres.
- 13) Início do string: primeiro caractere.

- 14) Segundo caractere.
- 15) Terceiro caractere.
- 16) Quarto caractere.
- 17) Caractere CR = 0DH que marca o fim do string.
- 18) Fim do programa.

VIII.5 SOMA DE VALORES COM MÚLTIPLA PRECISÃO

Somar dois valores binários formados por N bytes cada. Por exemplo, para N = 5:

Valor um :	3A	56	F1	45	1A
Valor dois:	A3	23	01	F0	E1
Soma :	DD	79	F3	35	FB

Ilustração:

Registro usado como contador

N = 5

Carry C

0

Valores da memória: VALOR UM

VALOR DOIS

Segunda Soma:

RESULTADO

$45 + F0 + \text{Carry} = 35$

Primeira soma:

$1A + E1 + \text{Carry} = FB$

DD
79
F3
35
FB

Nota: Se for conveniente para o microprocessador (8080/85, Z80), os bytes de VALOR UM, VALOR DOIS e RESULTADO podem ser colocados, na memória, na ordem inversa.
Nas versões para 8080/85, Z80, o RESULTADO da soma foi colocado na mesma área de VALOR UM e, conseqüentemente, o VALOR UM original foi perdido.

Algoritmo:**Condição Inicial:**

- Colocar no registro usado como contador o valor N = 5; Zerar Carry C = 0.

Repetir:

- Carregar acumulador A com o byte menos significativo do VALOR UM.
- Somar valor do acumulador com o byte menos significativo do VALOR DOIS, incluindo o Carry;
- Colocar o resultado da soma na posição mais à direita (menos significativo) da área RESULT, que receberá o resultado da soma.
- Decrementa o contador de valor.

Comparar:

- Verifica se contador chegou a valor zero; caso negativo voltar para Repetir.
- Termina se contador chegou a zero.

VERSÃO PARA 8080/85**Código-Objeto****Assembler**

```

; SOMA DE VALORES COM MULTIPLA PRECISAO.
;
0000 3A2100  EX8516: LDA    VALORN  ; 1)
0003 47      MOV     B,A      ; 2)
0004 211700  LXI     H,VALUM  ; 3)
0007 111C00  LXI     D,UDOIS  ; 4)
000A A7      ANA     A        ; 5)
000B 1A      REPET: LDAX   D    ; 6)
000C 8E      ADC     M        ; 7)
000D 77      MOV     M,A      ; 8)
000E 13      INX     D        ; 9)
000F 23      INX     H        ;10)
0010 05      DCR     B        ;11)
0011 C20B00  JNZ     REPET  ;12)
0014 C31400  PARE:  JMP     PARE ;13)
0017 1A      VALUM: DB     1AH  ;14)
0018 45      DB     45H  ;15)
0019 F1      DB     0F1H  ;16)
001A 56      DB     56H  ;17)
001B 3A      DB     3AH  ;18)

```

```

001C E1      VDOIS: DB      0E1H      ;19)
001D F0              DB      0F0H      ;20)
001E 01              DB      01H       ;21)
001F 23              DB      23H       ;22)
0020 A3              DB      0A3H      ;23)
0021 05      VALORN: DB      05H       ;24)
0022              END      EX8516

```

VERSÃO PARA Z80

Código-Objeto

Assembler

; SOMA DE VALORES COM MULTIPLA PRECISAO.

;

```

1000              ORG      1000H
1000 211110      EX16Z80: LD      HL,VALORN ; 1)
1003 46              LD      B,(HL) ; 2)
1004 23              INC     HL ; 3)
1005 111710      LD      DE,VDOIS ; 4)
1008 A7              AND     A ; 5)
1009 1A      REPET: LD      A,(DE) ; 6)
100A 8E              ADC     (HL) ; 7)
100B 77              LD      (HL),A ; 8)
100C 13              INC     DE ; 9)
100D 23              INC     HL ;10)
100E 10F9      DJNZ     REPET ;11)
1010 76              HALT    ;12)
1011 05      VALORN: DEFB    05H ;13)
1012 1A      VALUM:  DEFB    1AH ;14)
1013 45              DEFB    45H ;15)
1014 F1              DEFB    0F1H ;16)
1015 56              DEFB    56H ;17)
1016 3A              DEFB    3AH ;18)
1017 E1      VDOIS: DEFB    0E1H ;19)
1018 F0              DEFB    0F0H ;20)
1019 01              DEFB    01H ;21)
101A 23              DEFB    23H ;22)
101B A3              DEFB    0A3H ;23)
101C              END      EX16Z80

```

Comentários (8080/85 e Z80)

- 1) Carrega A com valor N = 5.
- 2) Coloca valor N = 5 no registro B que será o contador.
- 3) Coloca no par HL o endereço VALUM do byte menos significativo de VALOR UM.

- Nota:** Z80: DJNZ REPET é equivalente a DCR B
JNZ REPET

Escrever a versão Assembler e respectivo código-objeto dos programas abaixo mencionados. Todos os programas devem ser testados manualmente, instrução por instrução, através de um exemplo numérico, mencionando os conteúdos dos registos e memórias afetadas.

1. Subtrair sucessivamente os valores de N bytes de um único valor inicial colocado no endereço simbólico INÍCIO. Se o resultado final em INÍCIO for positivo, colocar 00 no endereço simbólico SINAL, caso contrário, colocar o valor FF.

Sugestão: modificar o programa para soma de N bytes.

2. Comparar uma série de N bytes com o valor hexadecimal 0F. Colocar os bytes acima desse valor, a partir do endereço MAIOR, e os bytes abaixo desse valor, a partir do endereço MENOR.

Sugestão: Modificar o programa para achar maior valor de uma série de N bytes. É preciso utilizar três contadores de endereços: um para ler a série de N bytes, outro para controlar os endereços dos valores maiores, e um terceiro contador, para os endereços dos valores menores.

3. Contar o número de caracteres vogais (A, E, I, O, U) e caracteres consoantes do seu nome e sobrenome. Colocar os dois resultados nos endereços VOGAIS e CONSAN, respectivamente.

4. ZERAR ÁREA DE N BYTES – Zerar uma área de N bytes a partir do endereço simbólico ÁREA.

5. Colocar o valor hexadecimal FF em cada um dos N bytes a partir do endereço simbólico NAREA.

6. Converter em instrução Assembler e respectivo código-objeto os seguintes trechos de programas em BASIC:

a) 05 LET A = 0 . 0

10 FOR I = 1 TO 10 STEP 2

20 LET A = A + 1

30 NEXT I

b) 10 DIM L (10)

20 ...

50 FOR I = 1 TO 10

60 IF L (I) < 0 . 0 THEN LET L (I) = 0 . 0

70 NEXT I

c) 10 DIM A (10, 10)

20 ...

50 LET SOMA = 0 . 0

60 FOR I = 1 TO 10

70 FOR J = 1 TO 10

75 IF I < > J THEN LET SOMA = SOMA + A (I, J)

80 NEXT J

90 NEXT I

CAPÍTULO

IX

INSTRUÇÕES DE GIROS E DESLOCAMENTOS (ROTATE E SHIFT)

IX.1 FORMATOS GERAIS DAS INSTRUÇÕES

As instruções estudadas neste capítulo são:

8080/85 Z80

– GIRO À DIREITA (ROTATE RIGHT)

RRC	RRCA ou RRC A	– Gira conteúdo de A para direita.
	RRC reg_i	– Gira conteúdo de registro reg_i (B até L).
	RRC (HL)	– Gira conteúdo da memória, cujo endereço está em HL.
	RRC (IX + ds)	– Gira conteúdo de memória; endereço em IX + ds.
	RRC (IY + ds)	– Gira conteúdo de memória; endereço em IY + ds.
RAR	RRA ou RR A	– Gira conteúdo de A, passando pelo CARRY.
	RR reg_i	– Gira conteúdo de registro reg_i e CARRY.
	RR (HL)	– Gira conteúdo de memória e CARRY, endereço em HL
	RR (IX + ds)	– <i>Idem</i> , endereço em IX + ds.
	RR (IY + ds)	– <i>Idem</i> , endereço em IY + ds.
	RRD	– Gira para direita um dígito BCD de A e memória.

– GIRO À ESQUERDA (ROTATE LEFT)

RLC	RLCA ou RLC A	– Gira A para esquerda.
	RLC reg_i	– Gira registro reg_i para esquerda.
	RLC (HL)	– Gira memória para esquerda; endereço em HL.

	RLC (IX + ds)	– <i>Idem</i> , endereço em IX + ds.
	RLC (IY + ds)	– <i>Idem</i> , endereço em IY + ds.
RAL	RLA ou RL A	– Gira A para esquerda passando pelo CARRY.
	RL reg _i	– Gira registro reg _i para esquerda com CARRY.
	RL (HL)	– Gira memória para esquerda; endereço em HL.
	RL (IX + ds)	– <i>Idem</i> , endereço em IX + ds.
	RL (IY + ds)	– <i>Idem</i> , endereço em IY + ds.
	RLD	– Gira para esquerda um dígito BCD de A e memória.

– DESLOCAMENTOS ARITMÉTICOS E LÓGICOS (SHIFT ARITHMETIC AND LOGIC)

SLA reg _i	– Desloca registro reg _i para esquerda.
SLA (HL)	– Desloca memória para esquerda, endereço em HL.
SLA (IX + ds)	– <i>Idem</i> , endereço em IX + ds.
SLA (IY + ds)	– <i>Idem</i> , endereço em IY + ds.
SRA reg _i	– Desloca registro reg _i para direita.
SRA (HL)	– Desloca memória para direita, endereço em HL.
SRA (IX + ds)	– <i>Idem</i> , endereço em IX + ds.
SRA (IY + ds)	– <i>Idem</i> , endereço em IY + ds.
SRL reg _i	– Deslocamento lógico (sem manter bit de sinal do registro reg _i para direita.
SRL (HL)	– Deslocamento lógico da memória, endereço em HL.
SRL (IX + ds)	– <i>Idem</i> , endereço em IX + ds.
SRL (IY + ds)	– <i>Idem</i> , endereço em IY + ds.

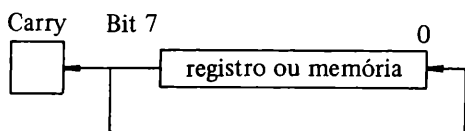
PROGRAMA-EXEMPLO

Além dos exemplos ilustrados de cada instrução, o capítulo apresenta o programa-exemplo:

IX.2 – ROTATES: GIROS

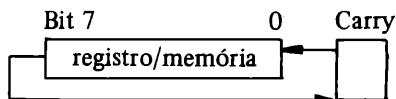
– MULTIPLICAÇÃO BINÁRIA DE DOIS VALORES DE OITO BITS CADA.

RLC (ROTATE LEFT CIRCULAR) Gira o conteúdo do registro ou memória de um bit à esquerda, SEM PASSAR pelo Carry, apenas fazendo Carry = Bit 7 e Bit 0 = Bit 7.



8080/85 Assembler	Código-Objeto	Z80 Assembler	Código-Objeto	Explicação
RLC	07	RLCA	07	– Gira conteúdo de A para esquerda.
		RLC A	CB 07	– Equivale a RLCA.
		RLC B	CB 00	– Gira conteúdo de B para esquerda.
		RLC C	CB 01	– Gira conteúdo de C para esquerda.
		RLC D	CB 02	– Gira conteúdo de D para esquerda.
		RLC E	CB 03	– Gira conteúdo de E para esquerda.
		RLC H	CB 04	– Gira conteúdo de H para esquerda.
		RLC L	CB 05	– Gira conteúdo de L para esquerda.
		RLC (HL)	CB 06	– Gira esquerda conteúdo de ppqq; HL contém ppqq.
		RLC (IX + ds)	DD CB ds 06	– Gira esquerda conteúdo de ppqq + ds; X contém ppqq.
		RLC (IY + ds)	FD CB ds 06	– Gira esquerda conteúdo de ppqq + ds; Y contém ppqq.

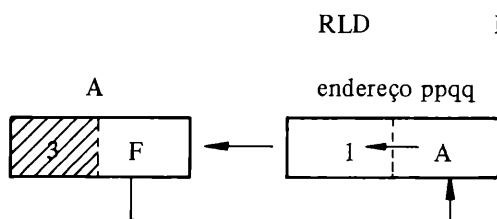
RL ou RAL (ROTATE LEFT) – Gira conteúdo de registro ou memória um bit para a esquerda, passando pelo CARRY, isto é, Bit 0 = Carry e Carry = Bit 7.



8080/85		Z80		Explicação
RAL	17	RLA	17	– Gira esquerda conteúdo de A passando pelo Carry.
		RL A	CB 17	– Equivale a RLA.
		RL B	CB 10	– Gira esquerda B passando pelo Carry.
		RL C	CB 11	– Gira esquerda C passando pelo Carry.
		RL D	CB 12	– Gira esquerda D passando pelo Carry.
		RL E	CB 13	– Gira esquerda E passando pelo Carry.
		RL H	CB 14	– Gira esquerda H passando pelo Carry.

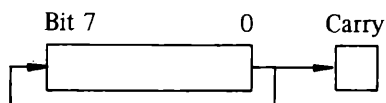
RL L	CB 15	– Gira esquerda L passando pelo Carry.
RL (HL)	CB 16	– Gira esquerda conteúdo de ppqq e Carry; ppqq em HL.
RL (IX + ds)	DD CB ds 16	– Gira esquerda conteúdo de ppqq + ds e Carry; ppqq em X.
RL (IY + ds)	FD CB ds 16	– Gira esquerda conteúdo de ppqq + ds e Carry; ppqq em Y.

RLD (ROTATE ONE DIGIT) – Gira para a esquerda os quatro bits (um dígito BCD) menos significativos de A com os oito bits do endereço ppqq da memória; ppqq está em HL; os quatro bits mais significativos de A não entram no giro. (só Z80)



RLD **ED 6F** – Acumulador A contém 3F e endereço ppqq contém 1A. Após a instrução, A contém 31 e ppqq contém AF, isto é, foi separado em A o dígito mais significativo que estava em ppqq; uma segunda execução traz para A o dígito A, pois A terá 3A.

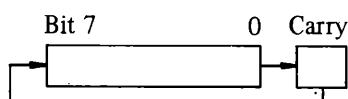
RRC (ROTATE RIGHT CIRCULAR) – Gira conteúdo de registro ou memória um bit para a direita sem passar pelo CARRY, apenas colocando Bit 0 no Carry e Bit 7 = Bit 0.



8080/85		Z80		Explicação
RRC	OF	RRCA	OF	– Gira conteúdo de A para direita.
		RRC A	CB 0F	– Equivale a RRCA.
		RRC B	CB 08	– Gira conteúdo de B para direita.
		RRC C	CB 09	– Gira conteúdo de C para direita.
		RRC D	CB 0A	– Gira conteúdo de D para direita.
		RRC E	CB 0B	– Gira conteúdo de E para direita.
		RRC H	CB 0C	– Gira conteúdo de H para direita.
		RRC L	CB 0D	– Gira conteúdo de L para direita.

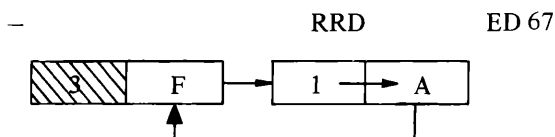
RRC (HL)	CB 0E	— Gira conteúdo de ppqq para direita; ppqq está em HL.
RRC (IX + ds)	DD CB ds 0E	— Gira conteúdo de ppqq + ds para direita; X contém ppqq.
RRC (IY + ds)	FD CB ds 0E	— Gira conteúdo de ppqq + ds para direita; Y contém ppqq.

RR ou RAR (ROTATE RIGHT THROUGH CARRY) — Gira conteúdo de registro ou memória um bit para a direita passando pelo CARRY, isto é, Bit 7 = Carry e Carry = Bit 0.



RAR	1F	RRA	1F	— Gira conteúdo de A para direita passando pelo CARRY.
		RR A	CB 1F	— Equivalente a RRA.
		RR B	CB 18	— Gira B para direita passando pelo Carry.
		RR C	CB 19	— Gira C para direita passando pelo Carry.
		RR D	CB 1A	— Gira D para direita passando pelo Carry.
		RR E	CB 1B	— Gira E para direita passando pelo Carry.
		RR H	CB 1C	— Gira H para direita passando pelo Carry.
		RR L	CB 1D	— Gira L para direita passando pelo Carry.
		RR (HL)	CB 1E	— Gira conteúdo de ppqq e Carry para direita; ppqq em HL.
		RR (IX + ds)	DD CB ds 1E	— Gira para a direita conteúdo de ppqq + ds e Carry; ppqq em X.
		RR (IY + ds)	FD CB ds 1E	— Gira para a direita conteúdo de ppqq + ds e Carry; ppqq em Y.

RRD (ROTATE RIGHT ONE DIGIT) — Gira para direita os quatro bits (um dígito BCD) menos significativo do Acumulador A com os oito bits da posição ppqq da memória; ppqq está em HL. (só Z80)



— Acumulador A contém 3F e endereço ppqq contém 1A. Após a instrução, A contém 3A e ppqq contém F1, isto é em cada execução, traz para A os quatro bits menos significativos de ppqq. Os quatro bits mais significativos de A não entram no giro.

Observação

FLAGS DE “ROTATES”

No 8080/85 — RLC, RAL, RRC e RAR: apenas o carry é afetado pela operação. Demais flags permanecem inalterados.

No Z80 — RLCA, RLA, RRCA e RRA (equivalentes a RLC, RAL, RRC e RAR do 8080/85): Ac = N = 0 e carry afetado pela operação.

Nas demais formas:

RL, RLC, RR e RRC os flags são

S = Z = PO = C = X; Ac = N = 0

e nas operações

RLD e RRD (ROTATE BCD DIGIT)

os flags afetados são:

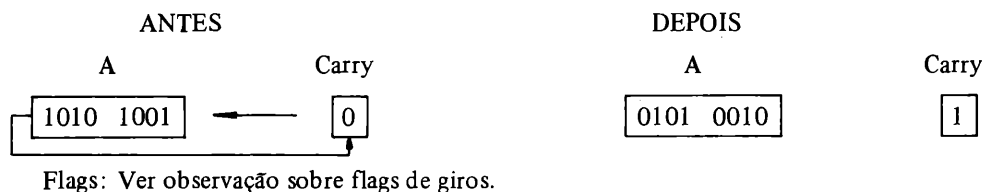
S = Z = PO = X; Ac = N = 0 e C = ∅.

Exemplos: (ROTATE ou GIRO)

8080/85 Z80

a) RAL RLA ou RL A

- Gira para a esquerda de um bit, o conteúdo de A, incluindo o Carry.



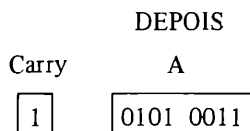
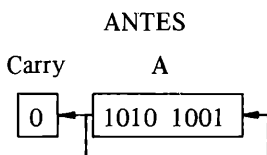
b) RAR RRA ou RR A

- Gira conteúdo de A para direita, em um bit, incluindo o Carry.



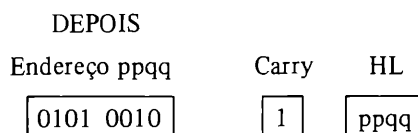
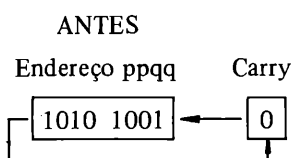
c) RLC RLCA ou RLC A

- Gira conteúdo de A para esquerda, sem incluir o Carry no giro, mas coloca bit 7 no Carry.



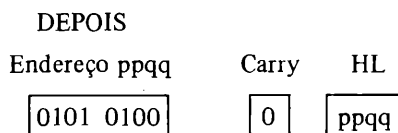
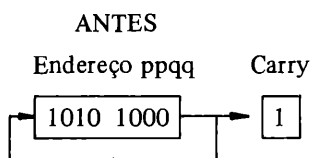
d) só Z80 RL (HL)

- Gira um bit para a esquerda, conteúdo do endereço ppqq, passando pelo Carry.



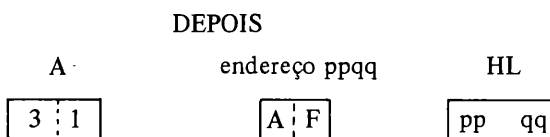
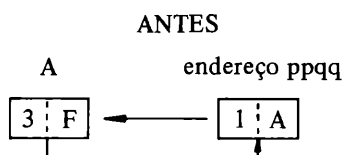
e) só Z80 RRC (HL)

- Gira conteúdo de ppqq para a direita sem incluir o Carry no giro, mas coloca bit 0 no Carry.



f) só Z80 RLD

- Gira para esquerda quatro bits (um dígito BCD) do endereço ppqq, junto com os quatro bits menos significativos de A. Os quatro bits mais significativos de A não são afetados. Resulta em colocar byte mais significativo de ppqq em A, e o menos significativo de A em ppqq.



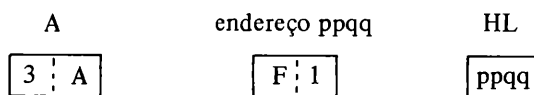
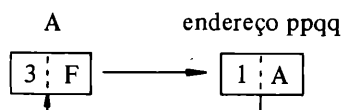
Flags: Z = 0 (resultado não nulo).

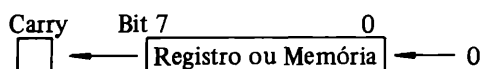
S = 0 (resultado não negativo: bit 7 = 0).

P/O = 1 (número par de bits um).

g) só Z80 RRD

- Operação análoga a RLD.



IX.3 SHIFTS: DESLOCAMENTOS (só Z80)

a) SLA – (SHIFT LEFT ARITHMETIC) – Desloca operando (registro ou memória) um bit para a esquerda, afetando flags CARRY, P/O, Z e S. Bit 0 torna-se zero.

8080/85 só Z80

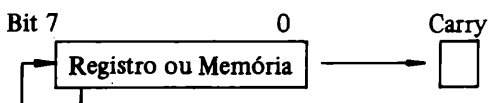
COM REGISTRO

–	SLA A	CB 27	– Desloca A para esquerda.
	SLA B	CB 20	– Desloca B para esquerda.
	SLA C	CB 21	– Desloca C para esquerda.
	SLA D	CB 22	– Desloca D para esquerda.
	SLA E	CB 23	– Desloca E para esquerda.
	SLA H	CB 24	– Desloca H para esquerda.
	SLA L	CB 25	– Desloca L para esquerda.

COM MEMÓRIA

	SLA (HL)	CB 26	– Desloca conteúdo de ppqq para esquerda; ppqq está em HL.
	SLA (IX + ds)	DD CB ds 26 (4 bytes)	– <i>Idem</i> , conteúdo de ppqq + ds; ppqq está em X.
	SLA (IY + ds)	FD CB ds 26 (4 bytes)	– <i>Idem</i> , conteúdo de ppqq + ds; ppqq está em Y.

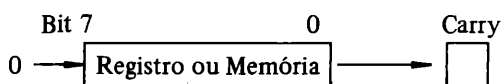
b) SRA – (SHIFT RIGHT ARITHMETIC) – Desloca operando (registro ou memória) um bit para a direita, afetando flags CARRY, P/O, Z. Mantém inalterado o bit 7 de sinal e o flag S.

**COM REGISTRO**

–	SRA A	CB 2F	– Desloca conteúdo de A para direita.
	SRA B	CB 28	– Desloca conteúdo de B para direita.
	SRA C	CB 29	– Desloca conteúdo de C para direita.
	SRA D	CB 2A	– Desloca conteúdo de D para direita.

SRA E	CB 2B	– Desloca conteúdo de E para direita.
SRA H	CB 2C	– Desloca conteúdo de H para direita.
SRA L	CB 2D	– Desloca conteúdo de L para direita.
COM MEMÓRIA		
SRA (HL)	CB 2E	– Desloca conteúdo de ppqq para direita; ppqq está em HL.
SRA (IX + ds)	DD CB ds 2E	– Desloca conteúdo de ppqq + ds para direita; X contém ppqq.
SRA (IY + ds)	FD CB ds 2E	– Desloca conteúdo de ppqq + ds para direita; Y contém ppqq.

c) SRL – (SHIFT RIGHT LOGICAL) – Desloca operando um bit para a direita, afetando flags CARRY, S, P/0 e Z. Bit 7 de sinal torna-se zero.

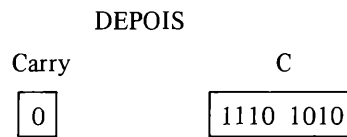
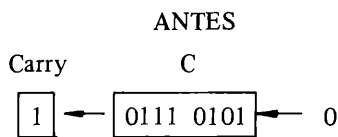


–	SRL A	CB 3F	– Deslocamento lógico de A para direita.
	SRL B	CB 38	– Deslocamento lógico de B para direita.
	SRL C	CB 39	– Deslocamento lógico de C para direita.
	SRL D	CB 3A	– Deslocamento lógico de D para direita.
	SRL E	CB 3B	– Deslocamento lógico de E para direita.
	SRL H	CB 3C	– Deslocamento lógico de H para direita.
	SRL L	CB 3D	– Deslocamento lógico de L para direita.
	SRL (HL)	CB 3E	– Deslocamento lógico do conteúdo de ppqq; ppqq está em HL.
	SRL (IX + ds)	DD CB ds 3E	– Deslocamento lógico de ppqq + ds para direita; ppqq está em X.
	SRL (IY + ds)	FD CB ds 3E	– <i>Idem, idem</i> , com ppqq em Y.

Exemplos: (DESLOCAMENTOS OU SHIFTS – só Z80)

Z80

- a) – SLA C • Desloca de um bit para a esquerda, conteúdo de C, colocando bit mais à esquerda no Carry.



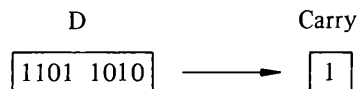
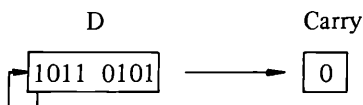
Flags: S = 1 (resultado é negativo, bit 7 = 1).
 Z = 0 (resultado é não nulo).
 P/O = 0 (número ímpar de bits um em C).

b) – SLA (HL)

- Mesma operação de a com endereço ppqq.

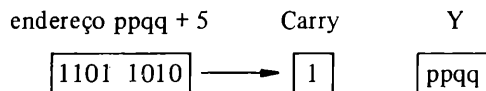
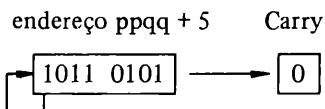
c) – SRA D

- Desloca um bit para a direita, o conteúdo de D. Bit mais à direita de D vai para Carry e Bit 7 (de sinal) de D permanece inalterado. Deslocamento aritmético não muda bit de sinal.



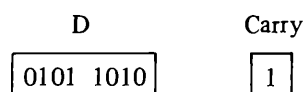
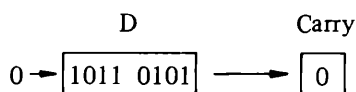
d) – SRA (IY + 5)

- Mesma operação de c, com endereço ppqq + 5, e ppqq está em Y.



e) – SRL D

- Mesma operação de c, mas o bit 7 de sinal torna-se zero, pois deslocamento lógico não mantém o bit de sinal.



IX.4 PROGRAMA-EXEMPLO USANDO DESLOCAMENTOS E GIROS:

MULTIPLICAÇÃO BINÁRIA DE DOIS VALORES DE OITO BITS CADA

Multiplicar o MULTIPLICANDO de 8 bits, que está no endereço simbólico MCANDO, com o MULTIPLICADOR de 8 bits, que está no endereço simbólico MCADOR.

O resultado PRODUTO deve ficar no endereço PROD, que deve ter 16 bits, isto é, no endereço PROD estará o byte menos significativo (LSB – Least Significant byte) e no endereço PROD + 1 estará o byte mais significativo (MSB – Most Significant byte) do resultado.

Algoritmo:

Passo 1: Endereço que vai receber resultado PROD = 0000.

Contador de bits recebe valor 8.

Repetir Passos 2, 3 e 4 oito vezes:

Passo 2: PRODUTO = PRODUTO x 2 (efetua deslocamento à esquerda de um bit na área de resultado).

MULTIPLICADOR = MULTIPLICADOR x 2 (efetua deslocamento à esquerda de um bit).

Passo 3: Se (Carry resultante do deslocamento do MULTIPLICADOR for 1) então PRODUTO = PRODUTO + MULTIPLICANDO.

Passo 4: Decrementa contador de bits.

Se contador de bits for maior que zero, então vai para Passo 2.

Passo 5: Caso contrário, terminar.

Nota: Este algoritmo efetua a multiplicação bit a bit, partindo do bit da direita para o bit da esquerda.

Exemplo:

```

MULTIPLICANDO = 03 = 0000 0011
MULTIPLICADOR = 05 = 0000 0101 x
                    0000 0011
                    00000 110  +
PRODUTO          = 0F = 0000 1111
    
```

Ilustração:

Condição inicial: Passo 1:

CARRY	MULTIPLICADOR	MULTIPLICANDO	PRODUTO	CONTADOR
0	0000 0101	0000 0011	0000 0000 0000 0000	08

Repetir Passos 2, 3 e 4 por oito vezes:

Primeira vez: Deslocamento à esquerda do PRODUTO E MULTIPLICADOR (Passos 2, 3 e 4)

0	←	0000 1010	0000 0000 0000 0000	07
---	---	-----------	---------------------	----

Segunda vez: Passos 2, 3 e 4:

0	←	0001 0100	0000 0000 0000 0000	06
---	---	-----------	---------------------	----

Permanece inalterado até que ocorra CARRY = 1:

Sexta vez: Passos 2, 3 e 4:

1	←	0100 0000	0000 0000 0000 0000	02
---	---	-----------	---------------------	----

Soma MULTIPLICANDO ao PRODUTO

+ 0000 0011

Novo PRODUTO:

0000 0000 0000 0011

Sétima vez: Passos 2, 3 e 4:

0	←	1000 0000	0000 0000 0000 0110	01
---	---	-----------	---------------------	----

Oitava vez: Passos 2, 3 e 4:

1	←	0000 0000	0000 0000 0000 1100
---	---	-----------	---------------------

Soma MULTIPLICANDO ao PRODUTO:

+ 0000 0011

Novo PRODUTO: (Resultado Final)

0000 0000 0000 1111

00

VERSÃO PARA 8080/85

Código-Objeto

Assembler

;
; MULTIPLICACAO BINARIA DE DOIS
; DADOS COM 8 BITS CADA.

0000 211D00 0003 5E 0004 1600 0006 23 0007 7E 0008 210000	EX8517: LXI MOV MVI INX MOV LXI	H,NCANDO E,H D,0 H A,H H,0	; ; ; ; ; ; ;	1) 2) 3) 4) 5) 6)
--	--	---	---------------------------------	----------------------------------

```

0008 0608      MVI      B,8      ; 7)
0000 29      DESLOC: DAD      H      ; 8)
000E 17      RAL      ; 9)
000F 021300    JNC      VAI      ;10)
0012 19      DAD      D      ;11)
0013 05      VAI:    DCR      B      ;12)
0014 C20000    JNZ      DESLOC    ;13)
0017 221F00    SHLD     PROD      ;14)
001A C31A00    PARE:    JMP      PARE ;15)
001D 03      NCAND0: DB      03H    ;16)
001E 05      NCAD0R: DB      05H    ;17)
001F 00      PROD:    DB      00H    ;18)
0020 00      DB      00H      ;19)
0021      END      EXB517

```

VERSÃO PARA Z80

Código-Objeto

Assembler

```

; MULTIPLICACAO BINARIA DE DOIS
; DADOS COM 8 BITS CADA.
;
1000      ORG      1000H
1000 211810    EX17Z80: LD      HL,NCAND0 ; 1)
1003 5E      LD      E,(HL) ; 2)
1004 1600      LD      D,0 ; 3)
1006 23      INC      HL ; 4)
1007 7E      LD      A,(HL) ; 5)
1008 210000    LD      HL,0 ; 6)
1008 0608      LD      B,8 ; 7)
1000 29      DESLOC: ADD      HL,HL ; 8)
100E 17      RLA      ; 9)
100F 3001      JR      NC,VAI ;10)
1011 19      ADD      HL,DE ;11)
1012 10F9      VAI:    DJNZ    DESLOC ;12)
1014 221A10    LD      (PROD),HL ;13)
1017 76      HALT      ;14)
1018 03      NCAND0: DEFB     03H ;15)
1019 05      NCAD0R: DEFB     05H ;16)
101A 00      PROD:    DEFB     00H ;17)
101B 00      DEFB     00H ;18)
101C      END      EX17Z80

```

Comentários (8080/85)

- 1) Endereço MCANDO do MULTIPLICANDO vai para par HL.
- 2) Valor do MULTIPLICANDO vai para registro E.
- 3) Zerar registro D.
- 4) Incrementa HL e aponta para endereço MCADOR do MULTIPLICADOR.
- 5) Valor do MULTIPLICADOR fica em A.
- 6) Zera HL, que será usado como PRODUTO.
- 7) Contador (registro B) recebe valor 8.
- 8) Soma HL com HL, isto é, multiplica PRODUTO por 2.
- 9) Desloca MULTIPLICADOR para esquerda.
- 10) Se Carry = 0 prossegue em VAI.
- 11) Se Carry = 1, soma HL com DE, isto é, obtém PRODUTO + MULTIPLICANDO.
- 12) Decrementa contador B.
- 13) Se contador maior que zero, repete tudo prosseguindo em DESLOC.
- 14) Se contador for zero, resultado final (em HL) vai para área PROD.
- 15) Pare.
- 16) Define na memória valor do MULTIPLICANDO = 03.
- 17) Define na memória valor do MULTIPLICADOR = 05.
- 18 e 19) Define na memória local de 2 bytes (endereços PROD e PROD + 1) para receber o valor de PRODUTO.
- 20) Fim do programa.

Comentários (Z80)

- 1) a 11) Idêntico aos do 8080/85.
- 12) Equivale a 12) e 13) do 8080/85, devido ao uso do comando DJNZ no lugar de DCR e JNZ.
- 13) a 18) Equivalem de 14 a 19 do 8080/85.

EXERCÍCIOS**De Recapitulação**

1. Descrever e citar exemplos de cada uma das formas possíveis das seguintes instruções:

- ARITHMETIC SHIFTS
- LOGICAL SHIFTS
- ROTATES

De Aplicação

Escrever a versão Assembler e respectivo código-objeto dos programas abaixo mencionados. Todos os programas devem ser testados manualmente, instrução por instrução, através de um exemplo numérico, mencionando os conteúdos dos registros e memórias afetadas.

1. Multiplicação binária de um valor de 16 bits por outro de 8 bits.
2. Multiplicação binária de um valor de 16 bits por outro de 16 bits.
3. **CONVERSÃO DE ALGARISMO HEXADECIMAL PARA DECIMAL** – Converter um algarismo hexadecimal (de 00 a 0F) que está no registo ou acumulador A, para o valor decimal correspondente (00 a 15), colocando os dois algarismos decimais resultantes nos endereços DEZENA e UNIDAD, respectivamente.
4. **CONVERSÃO DE VALOR BINÁRIO DE OITO BITS PARA DECIMAL** – Converter um valor binário de 8 bits (um byte da memória) em valor decimal correspondente, colocando os algarismos decimais nos endereços CENTENA, DEZENA e UNIDAD, respectivamente.

CAPÍTULO

X

INSTRUÇÕES DE DESVIO (CALL) E RETORNO (RET) DA SUB-ROTINA

X.1 INTRODUÇÃO

Neste capítulo estudamos o uso dos comandos de desvios e retornos de uma sub-rotina, definida pelo próprio programador.

A sub-rotina ou sub-programa é um recurso bastante utilizado em qualquer tipo de programação, pois permite ao programa principal efetuar operações longas repetidas vezes, sem necessidade de codificar essas operações a cada momento de seu uso. Basta programar essas operações como SUB-ROTINA ou SUB-PROGRAMA uma única vez e usá-la quantas vezes necessárias, através do comando de chamada do tipo CALL.

Em linguagens de alto nível, a sub-rotina aparece sob as seguintes formas:

<i>BASIC</i>	<i>FORTRAN</i>	<i>PASCAL</i>
<i>Desvio para sub-rotina:</i>		
GOSUB 100	CALL SUB-X	SUB-X;
<i>Sub-rotina a ser executada:</i>		
100 ...	SUBROUTINE SUB-X	PROCEDURE SUB-X;
110		BEGIN
(comandos)	(comandos)	(comandos)
...	...	...
200 RETURN	RETURN	END;

Em Assembler, o mesmo mecanismo é utilizado, através das instruções:

<i>8080/85</i>	<i>Z80</i>	
CALL endereço	CALL endereço	● Desvio incondicional para sub-rotina.

Ccc endereço CALL cc, endereço • Desvio se condição cc for verdade.

Para retornar de uma sub-rotina usam-se as instruções:

RET RET • Retorno incondicional.
Rcc RET cc • Retorno se cc verdadeiro.

O desvio ou ativação de sub-rotinas especiais, existentes em locais fixos da memória (endereço 00H, 08H, 10H, 18H, 20H, 28H, 30H e 38H), pode ser efetuado através da instrução RST n. Essas sub-rotinas, em geral, são montadas pelo próprio programa monitor do microcomputador e podem ser dependentes do hardware usado.

X.2 CALL SUBROUTINE – CHAMA OU DESVIA PARA SUB-ROTINA

8080/85		Z80		Explicação
Assembler	Código-Objeto	Assembler	Código-Objeto	
CALL endereço	CD qq pp	CALL endereço	CD qq pp	– Desvia para sub-rotina no endereço ppqq.
CNZ endereço	C4 qq pp	CALL NZ, endereço	C4 qq pp	– Desvia para sub-rotina se flag Z = 0 (NOT ZERO).
CZ endereço	CC qq pp	CALL Z, endereço	CC qq pp	– Desvia para sub-rotina se flag Z = 1 (ZERO).
CNC endereço	D4 qq pp	CALL NC, endereço	D4 qq pp	– Desvia para sub-rotina se flag C = 0 (NOT CARRY).
CC endereço	DC qq pp	CALL C, endereço	DC qq pp	– Desvia para sub-rotina se flag C = 1 (CARRY).
CPO endereço	E4 qq pp	CALL PO, endereço	E4 qq pp	– Desvia para sub-rotina se flag P/O = 0 (PARITY ODD).
CPE endereço	EC qq pp	CALL PE, endereço	EC qq pp	– Desvia para sub-rotina se flag P/O = 1 (PARITY EVEN).
CP endereço	F4 qq pp	CALL P, endereço	F4 qq pp	– Desvia para sub-rotina se flag S = 0 (PLUS).
CM endereço	FC qq pp	CALL M, endereço	FC qq pp	– Desvia para sub-rotina se flag S = 1 (MINUS).

X.3 RET – RETURN FROM SUBROUTINE – RETORNO DA SUB-ROTINA

RET C9 RET C9 – Retorno de sub-rotina.
RNZ C0 RET NZ C0 – Retorno de sub-rotina se flag Z = 0 (NOT ZERO).

RZ	C8	RET Z	C8	– Retorno de sub-rotina se flag Z = 1 (ZERO).
RNC	D0	RET NC	D0	– Retorno de sub-rotina se flag C = 0 (NOT CARRY).
RC	D8	RET C	D8	– Retorno de sub-rotina se flag C = 1 (CARRY).
RPO	E0	RET PO	E0	– Retorno de sub-rotina se flag P/0 = 0 (PARITY ODD).
RPE	E8	RET PE	E8	– Retorno de sub-rotina se flag P/0 = 1 (PARITY EVEN).
RP	F0	RET P	F0	– Retorno de sub-rotina se flag S = 0 (PLUS).
RM	F8	RET M	F8	– Retorno de sub-rotina se flag S = 1 (MINUS).

X.4 RST – RESTART SUBROUTINE – DESVIA PARA SUB-ROTINA COLOCADA EM UM DOS SEGUINTE OITO ENDEREÇOS ESPECIAIS DE UM BYTE: 00H, 08H, 10H, 18H, 20H, 28H, 30H e 38H

RST 0	C7	RST 00	C7	– Desvia para sub-rotina no endereço 00H.
RST 1	CF	RST 08H	CF	– Desvia para sub-rotina no endereço 08H.
RST 2	D7	RST 10H	D7	– Desvia para sub-rotina no endereço 10H.
RST 3	DF	RST 18H	DF	– Desvia para sub-rotina no endereço 18H.
RST 4	E7	RST 20H	E7	– Desvia para sub-rotina no endereço 20H.
RST 5	EF	RST 28H	EF	– Desvia para sub-rotina no endereço 28H.
RST 6	F7	RST 30H	F7	– Desvia para sub-rotina no endereço 30H.
RST 7	FF	RST 38H	FF	– Desvia para sub-rotina no endereço 38H.

Exemplos:

8080/85

Z80

a) CALL 0F10H

CALL 0F10H

- Desvia para sub-rotina definida a partir do endereço hexadecimal 0F10H.

Sub-rotina definida no endereço 0F10H:

...	...	• Instruções da sub-rotina.
(instruções da sub-rotina)	(instruções da sub-rotina)	
RET	RET	• Retorna para endereço seguinte ao da instrução CALL 0F10H.

- b) CZ ROTINA1 CALL Z, ROTINA1 • Desvia para sub-rotina que começa no endereço simbólico ROTINA1, se flag Z = 1.

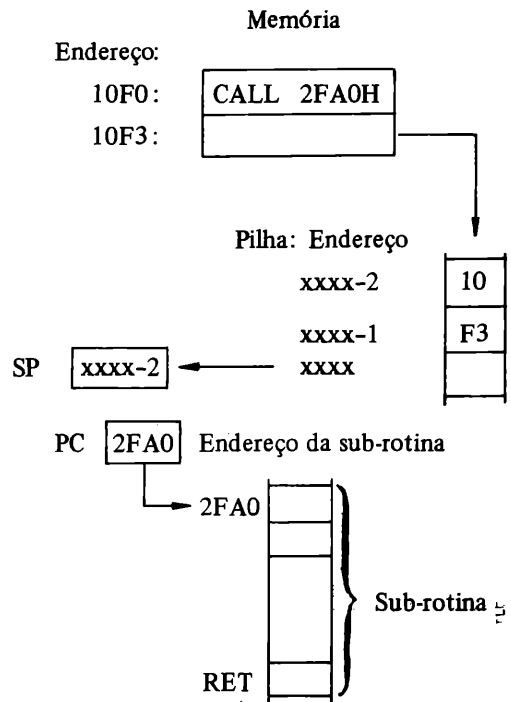
Sub-rotina definida no endereço ROTINA1:

ROTINA1:	ROTINA1: ...	• Instruções da sub-rotina.
...	...	
RC	RET C	• Retorna para endereço seguinte ao da instrução de chamada de sub-rotina ROTINA1 se flag C = 1.

X.5 MECANISMO DE DESVIO E RETORNO DE SUB-ROTINA

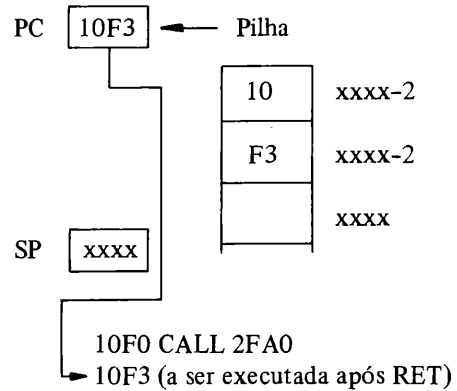
a) CALL 2FA0H

- Instrução de desvio CALL 2FA0H está no endereço 10F0H.
- Instrução de retorno após o CALL é 10F3H.
- Salva endereço 10F3H no topo da pilha e decrementa SP (Stack Pointer) de 2: xxxx-2 é a posição do novo topo da pilha.
- Endereço 2FA0H da sub-rotina é transferido para PC (Program Counter) a fim de executar o desvio.
- Desvia para endereço 2FA0 e começa a rotina.



b) RET – Retorno da sub-rotina

- Durante execução da sub-rotina, a instrução RET (ou similar) é encontrada.
- A instrução RET transfere o endereço 10F3H das duas posições do topo da pilha para PC. SP é incrementado de 2 para apontar novo topo da pilha, e conterá xxxx.
- Próxima instrução a ser executada, de acordo com o conteúdo de PC, é 10F3H, isto é, instrução imediatamente após o CALL 2FA0.



Nota: Se ocorrer uso recursivo do CALL, isto é, desvio para sub-rotina, durante a execução de outra sub-rotina, o mecanismo acima se repete automaticamente. A instrução de retorno (RET ou similar) coloca no PC sempre o último endereço do topo da pilha (dois últimos bytes), retornando assim para a última instrução CALL ocorrida.

X.6 PROGRAMA-EXEMPLO:**USO DE SUB-ROTINA**

Obter, através de uma sub-rotina, o número de caracteres de uma string ou série de caracteres de um byte cada, e que estão na memória, a partir do endereço STRIN e em notação ASCII. O último caractere do string é o caractere CARRIAGE RETURN ou CR (0D em hexadecimal) que marca o fim da série.

O número de caracteres assim obtido deve ser colocado no endereço COMPR.

A sub-rotina que calcula o número de caracteres será chamada RSTRI.

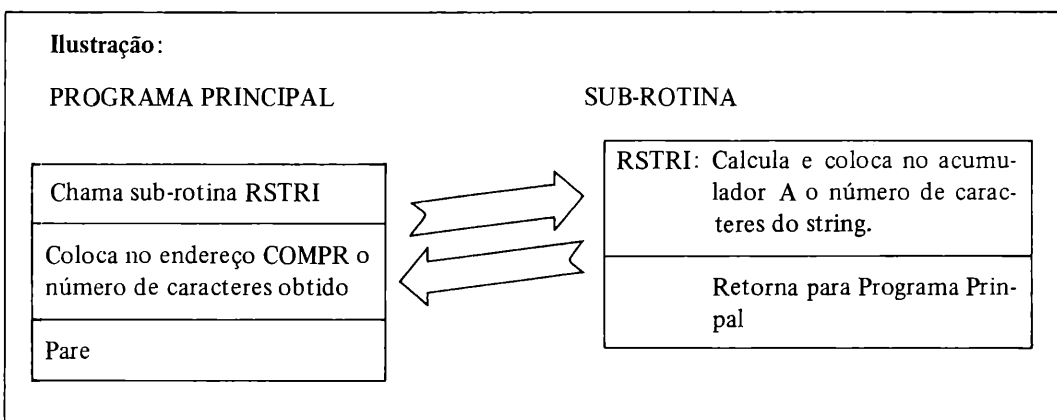
Algoritmo:**PROGRAMA PRINCIPAL:**

- Começar o programa a partir do endereço desejado, por exemplo 1000.
- Definir, na memória, uma nova pilha ou “stack” para receber o endereço de retorno, que será necessário após a execução da sub-rotina. Sem esta nova pilha, será usada a pilha definida pelo monitor do próprio microprocessador.
- Chamar a sub-rotina RSTRI para obter o número de caracteres da série.

- Colocar o número de caracteres obtido, e que está no acumulador A, no endereço COMPR.
- Terminar.

SUB-ROTINA RSTRI (conta o número de caracteres do string ou série):

- Coloca no acumulador A o caractere CR ou OD em hexadecimal.
- Repetir: compara acumulador A com um caractere do string.
Se houver igualdade (achou marcador de fim CR) vai para Retornar; caso contrário, incrementa contador B de caracteres e obtém endereço do próximo caractere.
 - Vai para Repetir.
- Retornar: Coloca valor do contador B de caracteres no registro ou acumulador A.
 - Retorno da sub-rotina.



VERSÃO PARA 8080/85

Código-Objeto

Assembler

```

;
;USO DE SUBROTINA ATRAVES DE PROGRAMA PRINCIPAL
;
;PROGRAMA PRINCIPA ESTA NO ENDEREÇO 1000H
1000      EX8521:  ORG 1000H      ;1)
1000 318000  PRINC1:  LXI SP,80H  ;2)
1003 CD5010          CALL RSTRI  ;3)
1006 32A010          STA COMPR   ;4)
1009 C30910  PARE:   JMP PARE    ;5)

;
;SUBROTINA: CONTA NUMERO DE CARACTERES
;          DE UM STRING
1050      ORG 1050H      ;6)
1050 0600  RSTRI:  MVI B,0      ;7)
    
```

```

1052 3E0D          MVI A,0DH      ;8)
1054 21A110        LXI H,STRIN    ;9)
1057 BE           REPET:  CMP H      ;10)
1058 CA6010         JZ  TERMINI    ;11)
1058 04            INR B           ;12)
105C 23            INX H           ;13)
105D C35710        JMP REPET      ;14)
1060 7B           TERMINI: MOV A,B  ;15)
1061 C9            RET            ;16)

;
;STRING COMEÇA NO ENDEREÇO 10A0H
;

10A0              ORG 10A0H        ;17)
10A0 00           COMPR: DB 0      ;18)
10A1 52           STRIN: DB 'R'    ;19)
10A2 41           DB 'A'          ;20)
10A3 54           DB 'T'          ;21)
10A4 4F           DB 'O'          ;22)
10A5 00           DB 0DH          ;23)
10A6 00           DB 0
10A7 00           DB 0
10A8              END EX8521

```

VERSÃO PARA Z80

Código-Objeto Assembler

```

;
;USO DE SUBROTINA
;
;PROGRAMA PRINCIPAL
1000 EX21Z80: ORG 1000H
1000 318000       PRINCI: LD SP,80H  ;1)
1003 CD5010       CALL RSTRI        ;2)
1006 32A010       LD (COMPR),A      ;3)
1009 76          HALT               ;4)

;
;SUBROTINA: CONTA CARACTERES
;          DE UM STRING
;
1050              ORG 1050H          ;5)
1050 0600       RSTRI: LD B,0        ;6)
1052 3E0D       LD A,0DH            ;7)
1054 21A110     LD HL,STRIN         ;8)
1057 BE        REPET: CP (HL)       ;9)
1058 2804       JR Z,TERMI          ;10)

```

```

105A 04          INC B          ;11)
105B 23          INC HL        ;12)
105C 18F9        JR REPET      ;13)
105E 78          TERMI: LD A,B  ;14)
105F C9          RET           ;15)

;
;STRING COMEÇA NO ENDEREÇO 10A0H
;

10A0          ORG 10A0H
10A0 00          COMPR: DEFB 0   ;16)
10A1 52          STRIN: DEFB 'R' ;17)
10A2 41          DEFB 'A'       ;18)
10A3 54          DEFB 'T'       ;19)
10A4 4F          DEFB 'O'       ;20)
10A5 00          DEFB 00H       ;21)
END

```

Comentários (8080/85)

PROGRAMA PRINCIPAL: começa no endereço 1000.

- 1) Define valor PC = 1000 para começar programa em 1000.
- 2) Define nova pilha no endereço 80H, fazendo Stack Pointer SP = 80H. (*)
- 3) Desvia para sub-rotina RSTRI.
- 4) Coloca número de caracteres, obtido na sub-rotina, no endereço COMPR.
- 5) Pare.

SUB-ROTINA RSTRI: começa no endereço 1050H.

- 6) Define valor PC = 1050H para começar programa em 1050H.
- 7) Contador de caracteres é registro B igual a zero.
- 8) Coloca em A o caractere CR = 0DH, para testar marcador de fim de string.
- 9) Par HL recebe endereço STRIN, apontando para primeiro caractere.
- 10) Compara A com o primeiro caractere do string.
- 11) Se comparação for igual, vai para TERMI.
- 12) Se comparação for diferente, incrementa contador B.
- 13) Incrementa HL para apontar próximo caractere.
- 14) Volta para REPET.
- 15) Terminou, coloca número de caracteres de B em A.
- 16) Retorna para Programa Principal.

STRING: começa no endereço 10A0H.

- 17) Define PC = 10A0 para começar a colocar dados a partir de 10A0.
- 18) Define na memória o endereço COMPR para receber o comprimento do string.
- 19) Define na memória a partir do endereço STRIN, o primeiro caractere do string.
- 20) Define segundo caractere.
- 21) Define terceiro caractere.
- 22) Define quarto caractere.
- 23) Define caractere CR = 0DH como marcador de fim de string.
- 24) Fim do programa.

Nota (*): Normalmente, não há necessidade de se definir nova pilha para cada sub-rotina usada, pois utiliza-se a pilha definida pelo próprio monitor do microprocessador ou uma única pilha para todos os programas envolvidos.

Comentários (Z80)

São os mesmos para a versão 8080/85, alterando-se apenas a numeração, pois certos comandos ORG na versão Z80 não entraram no comentário.

EXERCÍCIOS

De Recapitulação

1. Executar manualmente o programa-exemplo USO DA SUB-ROTINA, descrevendo o conteúdo dos registros e locais de memórias afetados, após a execução de cada instrução.

De Aplicação

1. Organizar, em forma de sub-rotina a ser utilizada através de um programa principal, os seguintes programas-exemplos já estudados:
 - ZERAGEM DE UMA ÁREA DE K BYTES.
 - SOMA DE DOIS VALORES DE 16 BITS CADA.
 - SOMA DE N VALORES DE UM BYTE CADA.
 - SOMA DE DOIS VALORES DE MÚLTIPLA PRECISÃO.
 - MULTIPLICAÇÃO BINÁRIA DE VALORES DE OITO BITS.

Escrever também o programa principal que utilize cada uma dessas sub-rotinas através de uma série de instruções do tipo CALL A SUBROUTINE.

CAPÍTULO

XI

INSTRUÇÕES DE TESTES DE BITS, INTERRUPÇÕES, PILHAS E ENTRADA/SAÍDA

XI.1 FORMATOS GERAIS DAS INSTRUÇÕES

As instruções descritas neste capítulo são:

TESTAR BIT DA POSIÇÃO n DO REGISTRO OU MEMÓRIA

8080/85 Z80

- | | |
|---------------------|--|
| BIT n , reg_i | – Testa bit da posição n do registro reg_i . |
| BIT n , (HL) | – Testa bit da posição n da memória, endereço em HL. |
| BIT n , (IX + ds) | – <i>Idem</i> , endereço em IX + ds. |
| BIT n , (IY + ds) | – <i>Idem</i> , endereço em IY + ds. |

COLOCAR BIT UM (SET) NA POSIÇÃO n DO REGISTRO ou MEMÓRIA

- | | |
|---------------------|--|
| SET n , reg_i | – Bit da posição n de reg_i fica igual a um. |
| SET n , (HL) | – <i>Idem</i> , para memória, endereço em HL. |
| SET n , (IX + ds) | – <i>Idem</i> , endereço em IX + ds. |
| SET n , (IY + ds) | – <i>Idem</i> , endereço em IY + ds. |

COLOCAR BIT ZERO (RESET) NA POSIÇÃO n DO REGISTRO ou MEMÓRIA

- | | |
|---------------------|------------------|
| RES n , reg_i | – Análogo a SET. |
| RES n , (HL) | – <i>Idem</i> . |
| RES n , (IX + ds) | |
| RES n , (IY + ds) | |

SET OU COMPLEMENT CARRY BIT; HALT e NOP

STC	SCF	– Flag de Carry recebe valor um (SET CARRY).
CMC	CCF	– Complementa Carry.

ATIVA OU DESATIVA INTERRUPTÃO (DISABLE OU INABLE INTERRUPTS).

COLOCA (PUSH) OU RETIRA (POP) UM VALOR DA PILHA.

INSTRUÇÕES DE LEITURA DE DADOS (INPUT) OU SAÍDA DE DADOS (OUTPUT).

As instruções de uso da INTERRUPTÃO e de ENTRADA/SAÍDA são quase sempre dependentes da implantação do tipo de hardware (UCP), portos e terminais do microcomputador em uso. Sendo assim, um bom conhecimento desses tipos de hardware é necessário. O manual do hardware do microprocessador ou referências mais avançadas devem ser consultadas.

O programa-exemplo LEITURA E ESCRITA ATRAVÉS DA MÁQUINA TTY OU TECLADO discutido no capítulo seguinte procura apresentar o uso dessas instruções.

XI.2 BIT –TESTA O BIT NA POSIÇÃO *n* DO REGISTRO OU MEMÓRIA (SÓ Z80)

O complemento do bit a ser testado é colocado no Flag Z, a fim de ser testado. O valor *n* ou *bbb* (valor 000 a 111) indicando a posição zero a sete do bit, da esquerda para a direita, aparece na instrução Assembler ou código-objeto.

8080/85	Z80		
–	BIT <i>n</i> , A	CB 01bb b111	– Coloca complemento do bit da posição <i>n</i> ou <i>bbb</i> de A, no Flag Z, a fim de ser testado.
	BIT <i>n</i> , B	CB 01bb b000	– <i>Idem</i> , bit do registro B.
	BIT <i>n</i> , C	CB 01bb b001	– Registro C.
	BIT <i>n</i> , D	CB 01bb b010	– Registro D.
	BIT <i>n</i> , E	CB 01bb b011	
	BIT <i>n</i> , H	CB 01bb b100	
	BIT <i>n</i> , L	CB 01bb b101	
	BIT <i>n</i> , (HL)	CB 01bb b110	– <i>Idem</i> , bit posição <i>n</i> da memória, endereço em HL.
	BIT <i>n</i> , (IX + ds)	DD CB ds 01bb b110	– <i>Idem</i> , memória com endereço em IX + ds.
	BIT <i>n</i> , (IY + ds)	FD CB ds 01bb b110	– <i>Idem</i> , endereço em IY + ds.

XI.3 SET – COLOCA VALOR UM (SET) NO BIT DA POSIÇÃO n (SÓ Z80)

–	SET n, A	CB 11bb b111	– Registro A.
	SET n, B	CB 11bb b000	– Registro B.
	SET n, C	CB 11bb b001	
	SET n, D	CB 11bb b010	
	SET n, E	CB 11bb b011	
	SET n, H	CB 11bb b100	
	SET n, L	CB 11bb b101	
	SET n, (HL)	CB 11bb b110	– Memória com endereço em HL.
	SET n, (IX + ds)	DD CB ds 11bb b110	– Memória com endereço em IX + ds.
	SET n, (IY + ds)	FD CB ds 11bb b110	– Memória com endereço em IY + ds.

XI.4 RESET – COLOCA VALOR ZERO (RESET) NO BIT DA POSIÇÃO n (SÓ Z80)

–	RES n, A	CB 10bb b111	– Registro A.
	RES n, B	CB 10bb b000	
	RES n, C	CB 10bb b001	
	RES n, D	CB 10bb b010	
	RES n, E	CB 10bb b011	
	RES n, H	CB 10bb b100	
	RES n, L	CB 10bb b101	
	RES n, (HL)	CB 10bb b110	– Memória com endereço em HL.
	RES n, (IX + ds)	DD CB ds 10bb b110	– Memória com endereço em IX + ds.
	RES n, (IY + ds)	FD CB ds 10bb b110	– Memória com endereço em IY + ds.

XI.5 SET OR COMPLEMENT CARRY FLAG

STC	37	SCF	37	– Carry C = 1
CMC	3F	CCF	3F	– Complementa Carry C.

X.6 HALT E NOP – PARADA E NENHUMA OPERAÇÃO

HLT	76	HALT	76	– Suspende execução.
NOP	00	NOP	00	– Nenhuma operação é executada.

Nota: A parada de um programa por HALT acarreta problemas com a interrupção. É recomendável usar outro tipo de parada independente ou que controle a interrupção.

XI.7 INTERRUPTS – INSTRUÇÃO PARA USAR INTERRUPÇÕES

<i>8080/85</i>	<i>Z80</i>	<i>Explicação</i>
	IM 0 ED 46	– Interrupt mode 0.
	IM 1 ED 56	– Interrupt mode 1.
	IM 2 ED 5E	– Interrupt mode 2.
DI F3	DI F3	– Desativa interrupção (Disable Interrupts).
EI FB	EI FB	– Ativa interrupção (Enable interrupts).

RIM – READ INTERRUPT MASK (só 8080/85)

RIM 20	– –	– Carrega (lê) no registro A o conteúdo da máscara de interrupção e linha de leitura serial.
--------	-----	--

SIM – SET INTERRUPT MASK (só 8080/85)

SIM 30	– –	– Coloca o conteúdo de A na máscara de interrupção e na linha SOD (Serial Output Data).
--------	-----	---

XI.8 PILHAS – POP E PUSH – OPERAÇÃO COM 16 BITS

8080/85 *Z80*

POP B	C1	POP BC	C1	– Pilha e Pilha + 1 vão para C e B, respectivamente.
POP D	D1	POP DE	D1	– Pilha e Pilha + 1 vão para E e D, respectivamente.
POP H	E1	POP HL	E1	– Pilha e Pilha + 1 vão para L e H, respectivamente.
POP PSW	F1	POP AF	F1	– Pilha e Pilha + 1 vão para PSW/F (Flags) e A, respectivamente.

POP IX DD E1 — Pilha e Pilha + 1 vão para IX.
 POP IY FD E1 — Pilha e Pilha + 1 vão para IY.

Nota: “Pilha” é o endereço do topo da pilha (valor do SP ou STACK POINTER) antes da execução da instrução. Após a execução SP terá valor pilha + 2.

PUSH B C5 PUSH BC C5 — B e C vão para Pilha – 1 e Pilha – 2, respectivamente.
 PUSH D D5 PUSH DE D5 — D e E vão para Pilha – 1 e Pilha – 2, respectivamente.
 PUSH H E5 PUSH HL E5 — H e L vão para pilha – 1 e Pilha – 2, respectivamente.
 PUSH PSW F5 PUSH AF F5 — A e F (flags) vão para Pilha – 1 e Pilha – 2, respectivamente.
 PUSH IX DD E5 — IX vai para Pilha – 1 e Pilha – 2.
 PUSH IY FD E5 — IY vai para Pilha – 1 e Pilha – 2.

Nota: Antes da execução da instrução, o endereço do topo da pilha (valor que está no SP ou STACK POINTER) é o valor “pilha”. Após a execução, esse valor será “pilha – 2”.

XI.9 INPUT – LEITURA DE DADOS DO PORTO DE ENTRADA PARA ACUMULADOR, REGISTRO OU MEMÓRIA

LEITURA DE UM BYTE DO PORT yy para ACUMULADOR

8080/85 Z80 *Explicação*

IN yy DB yy IN A, (yy) DB yy — Lê um byte do porto de entrada yy e coloca no acumulador.

LEITURA DE UM BYTE DO PORTO COM ENDEREÇO EM C, PARA O REGISTRO

— IN A, (C) ED 78 — Byte do porto com endereço em C vai para A.
 IN B, (C) ED 40 — Byte do porto com endereço em C vai para B.
 IN C, (C) ED 48 — Byte do porto com endereço em C vai para C (destrói endereço yy do porto).
 IN D, (C) ED 50 — Byte do porto com endereço em C vai para D.
 IN E, (C) ED 58 — Byte do porto com endereço em C vai para E.
 IN H, (C) ED 60 — Byte do porto com endereço em C vai para H.
 IN L, (C) ED 68 — Byte do porto com endereço em C vai para L.
 IN F, (C) ED 70 — Byte do porto com endereço em C vai para F (Flags).

LEITURA COM REPETIÇÃO DE BYTES DO PORTO COM ENDEREÇO EM C PARA A MEMÓRIA

—	IND	ED AA	— Byte do porto vai para memória com endereço em HL; decrementa HL e B.
	INDR	ED BA	— Leitura com repetição, decrementando HL e B até que contador B fique zero.
	INI	ED A2	— Byte do porto vai para memória com endereço HL; incrementa HL e decrementa B.
	INIR	ED B2	— Leitura com repetição, incrementando HL e decrementando B até este se tornar nulo.

XI.10 OUTPUT – COLOCA UM BYTE DO ACUMULADOR, REGISTRO OU MEMÓRIA NO PORTO DE SAÍDA

OUT yy D3 yy	OUT (yy), A D3 yy	— Operação inversa de IN.
	OUT (C), A	ED 79
	OUT (C), B	ED 41
	OUT (C), C	ED 49
	OUT (C), D	ED 51
	OUT (C), E	ED 59
	OUT (C), H	ED 61
	OUT (C), L	ED 69
	OUT (C), F	ED 71
	MEMÓRIA COM REPETIÇÃO	
	OUTD	ED AB — Operação inversa de IND.
	OUTDR	ED BB
	OUTI	ED A3
	OUTIR	ED 83

EXERCÍCIOS

Ver e examinar: PROGRAMA-EXEMPLO: LEITURA E ESCRITA DE DADOS PELA MÁQUINA TTY (TELETYPEWRITER) OU TERMINAL DE TV.

CAPÍTULO

XII

EXEMPLO DE ASSEMBLER: LEITURA E ESCRITA DE CARACTERES ATRAVÉS DE UMA MÁQUINA TTY (TELETYPEWRITER) (*) OU TERMINAL DE TV

PROBLEMA — Ler e escrever caracteres em código ASCII através de uma máquina de escrever serial tipo TTY, com velocidade de 10 caracteres por segundo.

XII.1 CONSIDERAÇÕES GERAIS SOBRE LEITURA E ESCRITA EM MICROPROCESSADORES

CONFIGURAÇÃO DE BITS POR CARACTERE

A transferência de dados (entrada ou saída) em uma TTY comum é efetuada do seguinte modo:

1. Escolhe-se o estado Entrada ou Saída da linha de transmissão de dados.
2. Cada caractere vem precedido por um bit zero de PARTIDA.
3. Cada caractere está codificado em código ASCII de sete bits cada, e a transmissão de bits é efetuada do bit menos significativo para o mais significativo.
4. O bit seguinte aos do caractere, isto é, o nono bit é o bit de PARIDADE que indica número par (ou um) ou ímpar (ou zero) de bits.
5. Dois bits de PARADA, iguais a um, seguem cada caractere.

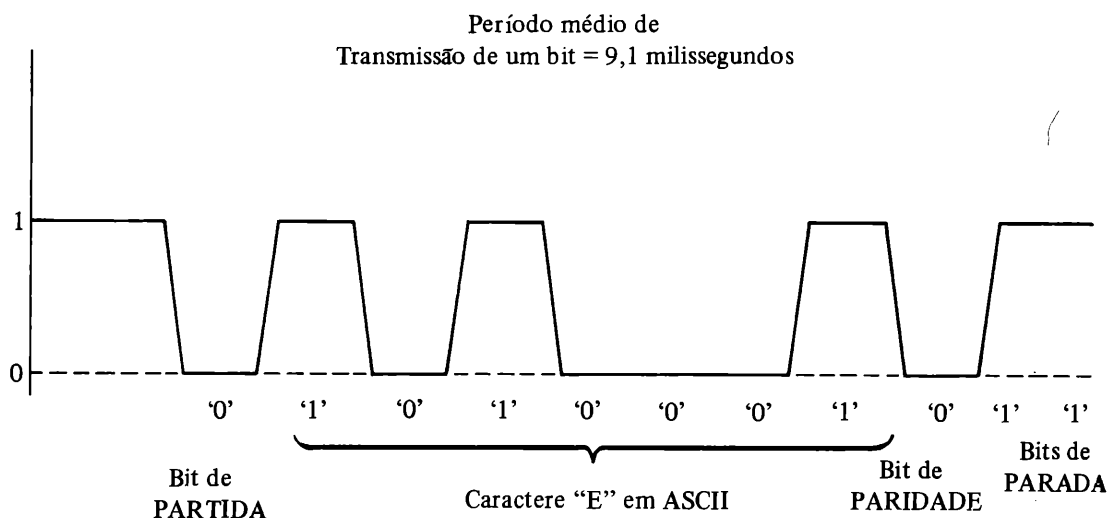
Temos portanto um total de ONZE bits transmitidos por caractere.

(*) Adaptado da Referência 3 e 4, com permissão da Editora McGraw-Hill do Brasil Ltda.

Velocidade de Transmissão de Cada Bit

Como a TTY opera à velocidade de 10 caracteres por segundo, e cada caractere é formado por 11 bits, a velocidade de transmissão de cada bit é de 10×11 ou 110 bauds (bits por segundo).

Cada bit gasta, portanto, em média, um intervalo de tempo (ou atraso) de $1/110$ segundo ou 9,1 milissegundos, que deve ser levado em consideração por ocasião da transmissão.



Algoritmo de leitura de caracteres pela TTY

Baseado nos fatos descritos, podemos elaborar o seguinte algoritmo:

- Passo 1 — Procurar o BIT DE PARTIDA de valor zero ao longo da transmissão.
- Passo 2 — Aguardar um intervalo de 4,55 ms (meio atraso) para centralizar a posição de transmissão ao longo da linha.
- Passo 3 — Receber cada bit do caractere, aguardando 9,1 ms (um atraso de transmissão) entre cada recepção.
- Passo 4 — Montar os bits recebidos em uma palavra do Acumulador, passando pelo bit de CARRY.
- Passo 5 — Calcular o valor do bit de paridade dos bits recebidos e comparar com o BIT DE PARIDADE. Se não coincidir indicar "ERRO DE PARIDADE".
- Passo 6 — Receber os dois BITS DE PARADA, se ambos não forem bit um, indicar "ERRO DE ARMAÇÃO" (Frame error).

O algoritmo de escrita de um caractere é análogo ao da leitura.

XII.2 ENTRADA/SAÍDA DE DADOS PELO MICROPROCESSADOR 8080/85

A operação de Entrada/Saída é efetuada diretamente pelas instruções IN e OUT que recebem ou levam de ou para os portos de Entrada/Saída (identificadas por um número de 00 a 256) o conteúdo do Acumulador ou Registro A.

Os códigos numéricos dos portos utilizados dependem da implantação feita pelo hardware. Usaremos os códigos simbólicos RPORT e TPORT e os valores numéricos podem ser atribuídos de modo variado pelas instruções ou diretivas EQU (EQUATE).

LEITURA DE DADOS

Ler um caractere ASCII enviado pela TTY através do BIT ZERO (bit da posição zero) do dado enviado pela "linha" do porto RPORT. Colocar o caractere lido no endereço CHRIN da memória do microprocessador.

Erros de paridade e de "frame" não serão detectados, pois requerem uma rotina adicional.

VERSÃO PARA 8080/85

Código-Objeto

Assembler

```

; LEITURA DE DADOS PELA TTY.
;
0020 =   CHROUT: EQU    20H    ; 1)
0021 =   CHRIN:  EQU    21H    ; 2)
0003 =   RPORT:  EQU     3     ; 3)
;
0000     EX8518: ORG     00H    ; 4)
0000 0803   LER:    IN      RPORT ; 5)
0002 1F      RAR          ; 6)
0003 DA0000   JC      LER      ; 7)
0006 CD3000   CALL    MATRA    ; 8)
0009 1680     MVI      D,80H    ; 9)
0008 CD3500   LEBIT: CALL    ATRASO ;10)
000E D803     IN      RPORT    ;11)
0010 1F      RAR          ;12)
0011 7A      MOV      A,D      ;13)
0012 1F      RAR          ;14)
0013 57      MOV      D,A      ;15)
0014 D20800   JNC      LEBIT    ;16)
0017 322100   STA      CHRIN    ;17)
001A C9      RET              ;18)

```



```

;
; SUBROTINA DE ATRASO E MEIO ATRASO.
;
0030      EX8519: ORG      30H      ;19)
0030 0604  MATRA: MVI      B,4      ;20)
0032 C33700      JMP      ATR1     ;21)
0035 0608  ATRASO: MVI      B,8      ;22)
0037 0E96  ATR1:  MVI      C,150     ;23)
0039 00      CICLO: DCR      C      ;24)
003A C23900      JNZ      CICLO     ;25)
003D 05      DCR      B      ;26)
003E C23700      JNZ      ATR1     ;27)
0041 C9      RET      ;28)
0042      END      EX851819
```

Comentários

- 1, 2, 3 e 4) Definição de endereços.
- 5) Dado do porto vai para A.
- 6) Bit zero lido vai para CARRY.
- 7) Espera até ler bit zero de partida.
- 8) SE chegou, aguarda 4,5 ms.
- 9) Bit de fim de contagem para D.
- 10) Aguarda 9,1 ms (um atraso).
- 11) Dado do porto vai para A.
- 12) Bit zero lido vai para CARRY.
- 13) D para A.
- 14) Bit lido (do CARRY) vai para A.
- 15) Guarda resultado em D.
- 16) Bit de contagem chegou ao CARRY?
- 17) Sim, coloca caractere em CHRIN.
- 18) Retorna.

SUB-ROTINA DE ATRASO E MEIO ATRASO – Comentários

- 19) Rotina começa em 0030H.
- 20) Contador de meio atraso (4 ciclos).
- 21) Desvia para obter meio atraso.

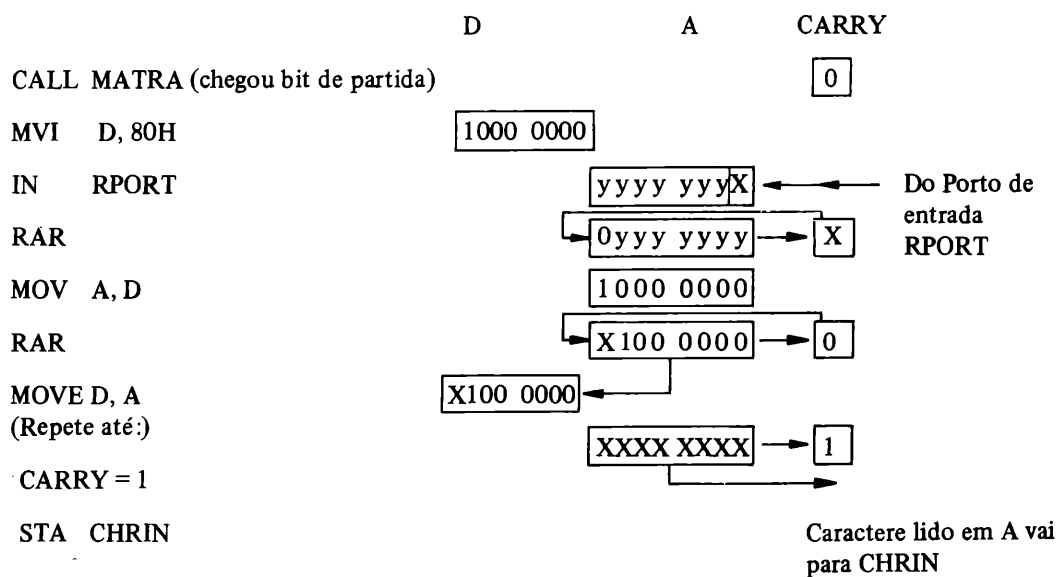
- 22) Contador de um atraso (8 ciclos).
- 23) Contador de um ciclo é 150.
- 24) Decrementa contador de ciclo.
- 25) Completou ciclo?
- 26) Sim, decrementa contador de atraso.
- 27) Completou um ou meio atraso?
- 28) Sim, retorna.

Nota: Os valores 4 ou 8 e 150 dos contadores de meio atraso, um atraso e um ciclo estão relacionados com o tempo de execução da instrução DCR C e JNZ CICLO, para completar 4,5 ms ou 9,1 ms.

ENDEREÇOS UTILIZADOS	Conteúdo:
CHROUT: EQU 20H	Caractere ASCII a ser escrito.
CHRIN: EQU 21H	Caractere ASCII lido.
RPORT: EQU 3	Código do porto usado para Entrada.
TPORT: EQU 3	Código do porto usado para Saída.

Observação: Os endereços acima são fixos e dependentes da implantação do hardware do microprocessador. Manual sobre hardware do microprocessador deve ser consultado.

Ilustração:



ESCREVER DADOS NA TTY

Escrever um caractere ASCII pela TTY, através do BIT ZERO (bit da posição zero) do porto de saída TPORT. O caractere a ser escrito está no endereço CHROUT da memória do microprocessador. A paridade dos bits transmitidos não será calculada, o que requer outra rotina.

VERSÃO PARA 8080/85

Código-Objeto

Assembler

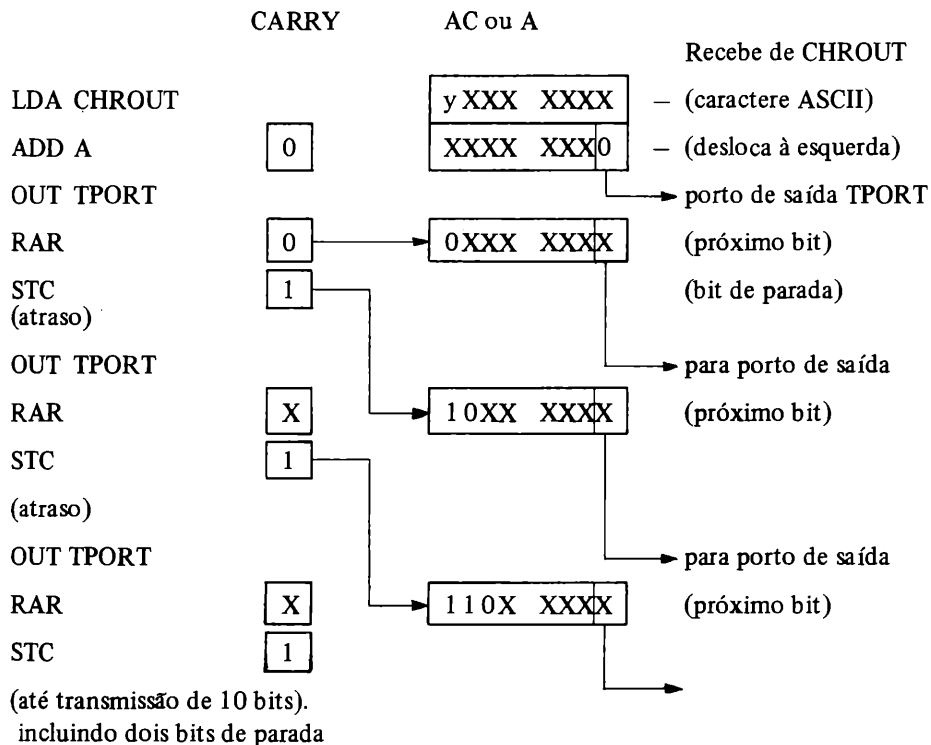
```

; ESCREVER DADOS NA TTY.
;
0020 =   CHROUT: EQU    20H    ; 1)
0021 =   CHRIN:  EQU    21H    ; 2)
0003 =   TPORT:  EQU     3     ; 3)
;
0050     EX8520: ORG     50H    ; 4)
0050 3A2000  ESCREV: LDA    CHROUT ; 5)
0053 87      ADD     A        ; 6)
0054 1608     MVI     D,11    ; 7)
0056 D303     ESBIT: OUT    TPORT ; 8)
0058 1F      RAR        ; 9)
0059 37      STC        ;10)
005A CD3500   CALL    ATRASO ;11)
005D 15      DCR     D        ;12)
005E C25600   JNZ     ESBIT  ;13)
0061 C9      RET         ;14)

;
; SUBROTINA DE ATRASO E MEIO ATRASO.
;
0030     EX8519: ORG     30H    ;15)
0030 0604     MATRA: MVI     B,4  ;16)
0032 C33700   JMP     ATR1    ;17)
0035 0608     ATRASO: MVI     B,8  ;18)
0037 0E96     ATR1:  MVI     C,150 ;19)
0039 0D      CICLO: DCR     C        ;20)
003A C23900   JNZ     CICLO    ;21)
003D 05      DCR     B        ;22)
003E C23700   JNZ     ATR1    ;23)
0041 C9      RET         ;24)
0042         END     EX852019
```

Comentários:

- 1) 2) 3) Definição de endereços.
- 4) Rotina começa em 0050H.
- 5) Caractere ASCII (7 bits) em A.
- 6) Cria bit de partida na posição zero.
- 7) Contador de bits é onze bits.
- 8) Coloca bit zero no porto de saída.
- 9) Próximo bit vai para posição zero.
- 10) CARRY = 1, indicando bits de parada.
- 11) Aguarda um atraso de 9,1 ms.
- 12) Decrementa contador de bits.
- 13) Transmitiu dez bits?
- 14) Sim, retorna.
- 15) a 24) Rotina de atraso e meio atraso, já utilizado para leitura.

Ilustração:

XII.3 ENTRADA/SAÍDA DE DADOS PELO Z80 ATRAVÉS DO CIRCUITO PIO (PARALLEL INPUT/OUTPUT)

A operação de Entrada/Saída do Z80 é efetuada pelo circuito chamado PIO, que opera com dois portos de Entrada/Saída:

PORTO A, que contém:

- Registro de Dados de 8 bits, conectado ao endereço simbólico PIODRA da memória do Z80.
- Registro de Controle de E/S, conectado ao endereço simbólico PIOCRA.

PORTO B, que contém:

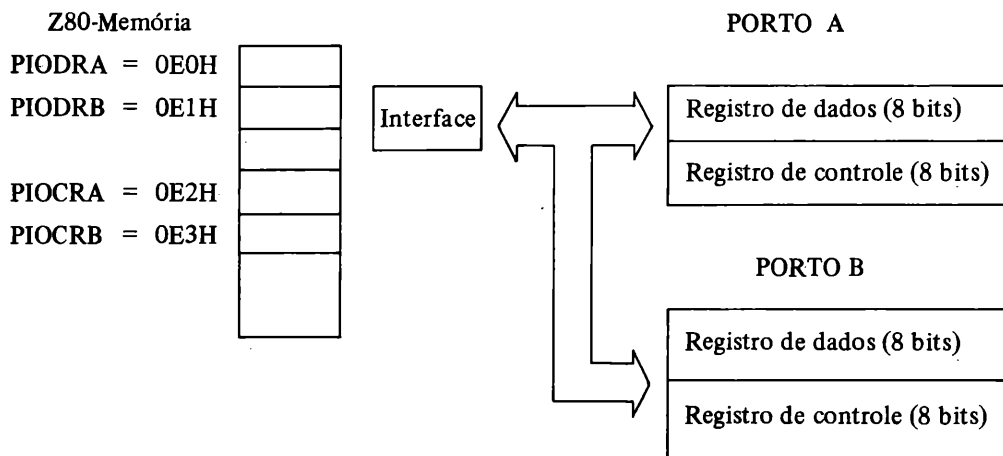
- Registro de Dados de 8 bits, conectado ao endereço simbólico PIODRB da memória do Z80.
- Registro de Controle de E/S, conectado ao endereço simbólico PIOC RB.

Cada bit do registro de dados atua como uma linha ou canal de transmissão de dados, através de um dos pinos do porto. Sendo assim, cada porto consegue efetuar, ao mesmo tempo, até oito operações de Entrada/Saída.

Cada bit do registro de controle de E/S serve para determinar se o pino de transmissão de dados é de Entrada (bit 1) ou de Saída (bit zero).

Os endereços efetivos (absolutos) dos símbolos adotados dependem da implementação efetuada no hardware, mas pode ser escolhida através das instruções diretivas EQU (EQUATE). Por exemplo, usaremos os endereços indicados no esquema simplificado:

Z80 – PARALLEL INPUT/OUTPUT (PIO) – Esquema



LEITURA DE DADOS

Ler um caractere ASCII, através do BIT SETE do porto A do PIO. Colocar o caractere lido no endereço CHRIN da memória do microprocessador. Erros de paridade e de "frame" não serão detectados, o que requer outra rotina.

VERSÃO PARA Z80

Código-Objeto

Assembler

```

; LEITURA DE DADOS PELA TTY.
;
0020 =   CHROUT: EQU   20H       ; 1)
0021 =   CHRIN:  EQU   21H       ; 2)
00E0 =   PIODRA: EQU   0E0H       ; 3)
00E2 =   PIOCRA: EQU   0E2H       ; 4)
;
1000     EX18Z80: ORG   1000H      ; 5)
1000 3E4F   LER:    LD     A,4FH    ; 6)
1002 D3E2           OUT    (PIOCRA),A ; 7)
1004 D8E0   ESPERA: IN     A,(PIODRA) ; 8)
1006 17           RLA              ; 9)
1007 38FB           JR     C,ESPERA ;10)
1009 CD3010        CALL   MATRA     ;11)
100C 1680           LD     D,80H     ;12)
100E CD3510   LEBIT: CALL   ATRASO   ;13)
1011 D8E0           IN     A,(PIODRA) ;14)
1013 17           RLA              ;15)
1014 CB1A           RR     D         ;16)
1016 30F6           JR     NC,LEBIT  ;17)
1018 7A           LD     A,D         ;18)
1019 322100        LD     (CHRIN),A ;19)
101C C9           RET              ;20)
;
; SUBROTINA DE ATRASO E MEIO ATRASO.
;
1030     EX19Z80: ORG   1030H      ;21)
1030 D5   MATRA:  PUSH   DE         ;22)
1031 1608           LD     D,8       ;23)
1033 1803           JR     ATR1     ;24)
1035 D5   ATRASO: PUSH   DE         ;25)
1036 1616           LD     D,16      ;26)

```

1038	1E8D	ATR1:	LD	E,8DH	;27)
103A	1D	CICLO:	DEC	E	;28)
103B	20FD		JR	NZ,CICLO	;29)
103D	15		DEC	D	;30)
103E	20F8		JR	NZ,ATR1	;31)
1040	D1		POP	DE	;32)
1041	C9		RET		;33)
1042			END	EX1819Z80	

Comentários:

- 1) , 2), 3), 4) e 5) Definição de endereços.
- 6) Valor 4F vai para PIOCRA.
- 7) E torna porto A, uma entrada.
- 8) Ler registro de dados do porto A.
- 9) CARRY = bit 7 do valor lido.
- 10) Espera bit de partida.
- 11) Chegou partida, aguarda 4,5 ms.
- 12) Coloca bit de fim de contagem em D.
- 13) Aguarda um atraso de 9,1 ms.
- 14) Ler dados do porto A.
- 15) Bit 7 lido vai para CARRY.
- 16) Monta bit 1 lido (Carry) em D.
- 17) Bit de contagem chegou ao CARRY?
- 18) Sim, caractere foi lido em D.
- 19) Coloca em CHRIN.
- 20) Retorna.
- 21) Rotina começa no endereço 1030H.
- 22) Salva registros D e E na pilha.
- 23) Contador de meio atraso = 8 ciclos,
- 24) e desvia para executar meio atraso.
- 25) Salva D e E na pilha.
- 26) Contador de um atraso = 16 ciclos.
- 27) Contador de um ciclo = 8DH = 141.
- 28) Decrementa contador de ciclo.
- 29) Completou um ciclo?
- 30) Sim, decrementa contador de atraso.

ESCREVER DADO PELA TTY

Escrever um caractere em código ASCII, pelo bit da posição ZERO do porto B. O caractere a ser escrito está no endereço CHROUT da memória do microprocessador. Assumir que o bit de paridade não é calculado durante a transmissão, o que requer outra rotina.

VERSÃO PARA Z80

Código-Objeto

Assembler

```

; ESCREVER DADOS NA TTY.
;
0020 = CHROUT: EQU 20H ; 1)
0021 = CHRIN: EQU 21H ; 2)
00E0 = PIODRB: EQU 0E1H ; 3)
00E2 = PIOCRL: EQU 0E3H ; 4)
;
1050 EX20Z80: ORG 1050H ; 5)
1050 3E0F ESCREV: LD A,0FH ; 6)
1052 D3E3 OUT (PIOCRB),A ; 7)
1054 3A2000 LD A,(CHROUT) ; 8)
1057 87 ADD A ; 9)
1058 0611 LD B,11 ; 10)
105A D3E1 ESBIT: OUT (PIODRB),A ; 11)
105C 1F RRA ; 12)
105D 37 SCF ; 13)
105E CD3510 CALL ATRASO ; 14)
1061 10F7 DJNZ ESBIT ; 15)
1063 C9 RET ; 16)
;
; SUBROTINA DE ATRASO E MEIO ATRASO.
;
1030 EX19Z80: ORG 1030H ; 17)
1030 05 NATRA: PUSH DE ; 18)
1031 1608 LD D,8 ; 19)
1033 1803 JR ATR1 ; 20)
1035 05 ATRASO: PUSH DE ; 21)
1036 1616 LD D,16 ; 22)
1038 1E80 ATR1: LD E,80H ; 23)
103A 1D CICLO: DEC E ; 24)
1038 20FD JR NZ,CICLO ; 25)
103D 15 DEC D ; 26)

```

```

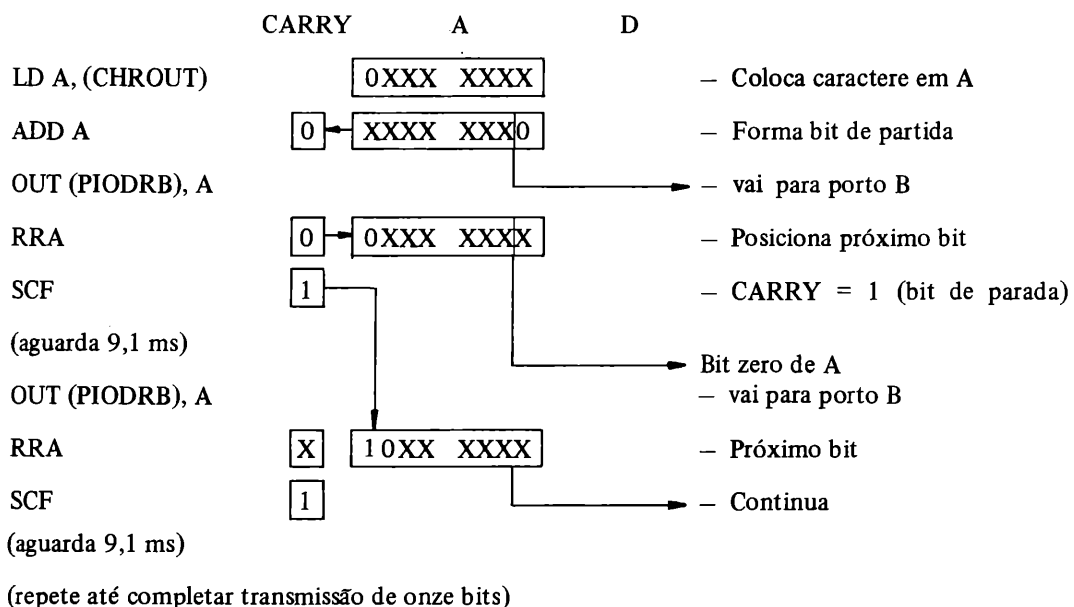
103E 20F8      JR    NZ,ATR1    ;27)
1040 D1        POP    DE        ;28)
1041 C9        RET             ;29)
1042          END    EX2019Z80

```

Comentários:

- 1) , 2), 3) e 4) Definição de endereços.
- 5) Rotina começa em 1050H.
- 6) Valor OF colocado no Registro de Controle.
- 7) Torna B, porto de saída.
- 8) Coloca caractere em A (7 bits).
- 9) Cria bit de partida na posição zero.
- 10) Contador de total de 11 bits.
- 11) Coloca bit zero no porto B.
- 12) Próximo bit vai na posição zero.
- 13) CARRY = 1 será bit de parada.
- 14) Aguarda atraso de 9,1 ms.
- 15) Decrementa B e repete se não nulo.
- 16) Se B = 0, terminou, retorna.

Nota: Rotina de atraso e meio atraso é a mesma usada para leitura

Ilustração:

EXERCÍCIO

De Recapitulação

1. Utilizando a execução manual, instrução a instrução das ROTINAS PARA LER E ESCRIVER
DADOS: — executar a leitura dos caracteres 1, 7, X e Y.
— executar a escrita dos caracteres 1, 7, X e Y.

PARTE



**LINGUAGEM ASSEMBLER DO MICROPROCESSADOR
6502**

CAPÍTULO

XIII

ELEMENTOS BÁSICOS DA ARQUITETURA DO MICROPROCESSADOR 6502

XIII.1 ORGANIZAÇÃO INTERNA DA UCP DO 6502

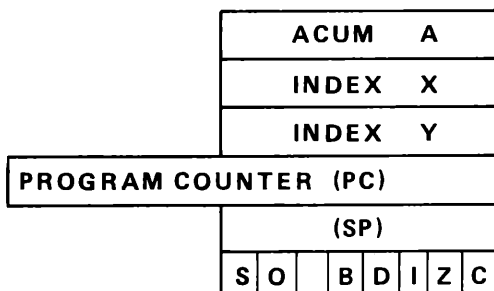
O 6502 possui os seguintes elementos básicos, de interesse na programação em Assembly ou linguagem de máquina:

- Um acumulador A ou Ac de 8 bits.
- Um registro de índice X de 8 bits.
- Um registro de índice Y de 8 bits.
- Um Contador de Programa PC de 16 bits, que endereça memória de 0000 até FFFF.
- Um Ponteiro de Pilha SP de 8 bits.
- Um Registro de Status ou Flags chamado P e que contém os seguintes Flags:

Carry ou C	– Para indicar ocorrência ou não de VAI-UM.
Zero ou Z	– Para indicar se o resultado da operação é zero.
Overflow ou V	– Para indicar ocorrência de OVERFLOW ou TRANSBORDO.
Sinal ou S	– Para indicar se o resultado da operação é negativo (bit 7 é um) ou não-negativo (bit 7 é zero).
Modo Decimal ou D	– Se D = 1, todas as instruções ADC e SBC (ADD e SUBTRACT) são executadas como se os operandos estivessem em notação BCD (Binary Coded Decimal).
Break ou B	– Se B = 1, haverá interrupção, causada pelo software, quase sempre através da instrução BREAK (PARE).
Interrupção ou I	– Inibe ou desinibe o estado de interrupção.

Nota: O flag Carry ou C, no 6502, atua de modo contrário à maioria dos microprocessadores na instrução SBC (SUBTRACT) e CMP (COMPARE). Ver exemplos dessas instruções.

6502



Endereço da pilha (STACK) definido pelo ponteiro (STACK POINTER) SP

O ponteiro da pilha SP (STACK POINTER) é um registro que contém apenas 8 bits, podendo conter valores de 00 a FF em hexadecimal.

A UCP (Unidade Central de Processamento) antepõe automaticamente o byte “01” ao valor do ponteiro SP, formando assim um endereço que varia de 0100 a 01FF, que é o endereço da memória onde é definida a pilha.

Notação dos flags de status

Os flags de status podem ou não ser afetados pelo resultado de uma instrução. A seguinte notação será adotada, quando for o caso:

- “X” — Indica que o flag é afetado pelo resultado da instrução. Ver exemplos ilustrativos de cada tipo de instrução.
- “1” — Indica que o flag sempre assume valor um, após a execução da instrução.
- “0” — Indica que o flag sempre assume valor zero, após a execução da instrução.
- “Ø” — Indica que o flag não é afetado pela instrução, mantendo o mesmo valor.

XIII.2 INSTRUÇÃO EM ASSEMBLER

Modos de endereçamento de operandos

Forma geral da instrução Assembler 6502:

Rótulo	Código Mnemônico	Operando
--------	------------------	----------

Os operandos de uma instrução 6502 podem assumir uma das seguintes formas ou modos de endereçamento:

Instruções sem operandos

São instruções em que os operandos (registros, acumulador ou memória) normalmente usados não são utilizados.

Exemplos:

- | | |
|-----|---|
| NOP | – Nenhuma operação é executada, serve para ocupar posição da memória. |
| RTS | – Retorna de sub-rotina. |

Registro ou acumulador

Neste caso, o código do registro ou acumulador aparece no código mnemônico da instrução.

Exemplos:

- | | |
|-----|---|
| INX | – Incrementa registro X. |
| DEY | – Decrementa registro Y. |
| TAX | – Transfere conteúdo do acumulador A para registro X. |

Operando imediato

São valores numéricos (decimal, hexadecimal, binário ou octal) ou alfabéticos (em código ASCII) que aparecem precedidos do sinal “#” e são diretamente utilizados pela instrução.

Exemplos:

- | | |
|-----------|---|
| LDA #\$0F | – Valor hexadecimal 0F vai para acumulador A. |
|-----------|---|

Endereço direto da memória

ENDEREÇO qq CURTO NA PÁGINA ZERO OU ENDEREÇO LONGO pp qq

Neste caso, o operando é um endereço da memória. Se o endereço absoluto mencionado na instrução ou assumido pelo Assembler por ocasião da tradução estiver no intervalo 0000 a 00FF, digamos que o endereço está na PÁGINA ZERO e o código-objeto correspondente possui apenas

dois bytes (byte do código de operação e byte do endereço qq). Nos demais casos, o endereço direto possui forma normal ppqq e a instrução possui três bytes (byte do código-objeto e dois bytes para o endereço ppqq).

Exemplos:

- LDA \$A0 A5 A0 (código-objeto) — Carrega A com conteúdo do endereço A0 da PÁGINA ZERO.
- LDA \$F043 AD 43 F0 (código-objeto) — Carrega A com conteúdo do endereço F043, que aparece com os bytes invertidos no código-objeto.
- LDA Z1 — O endereço simbólico pode ser transformado em forma curta qq ou longa ppqq, conforme localização de Z1 no programa.

Endereço indexado com registro X ou Y

O operando é um endereço da memória ppqq. O valor ppqq chamado de *endereço efetivo* é obtido pela soma do conteúdo do registro X ou Y citado na instrução, mais o endereço absoluto ou simbólico (também chamado de deslocamento, acréscimo ou offset) citado na instrução.

Exemplos:

- LDA \$A0, X B5 A0 (código-objeto) — Conteúdo do endereço A0 + X vai para A (endereço A0 está na página zero)
- LDA \$F043, X B9 43 F0 (código-objeto) — Conteúdo do endereço F043 + X vai para A.
- LDA Z1, X — Endereço efetivo ppqq é formado pela soma do endereço absoluto atribuído a Z1 mais o conteúdo de X.

Endereço indireto

No modo indireto, o endereço efetivo ou final do operando não aparece diretamente na instrução. O endereço formado pelos elementos (endereço qq e registro) que aparecem na instrução contém o endereço efetivo que é usado pela instrução.

Além disso, o endereço indireto aparece na forma indexada por X ou Y e temos as duas formas seguintes:

a) ENDEREÇO INDIRETO PRÉ-INDEXADO COM X**Exemplo:**

Forma geral: LDA (endereço qq, X) A1 qq — O endereço qq está sempre na página zero.

LDA (\$FA, X) A1 FA – Conteúdo do endereço efetivo rrss vai para A, endereço FA + X contém ss e endereço FA + X + 1 contém rr.

b) ENDEREÇO INDIRETO PÓS-INDEXADO COM Y

Exemplo:

Forma geral: LDA (endereço qq), Y B1 qq – O endereço qq está sempre na página zero.

LDA (\$FA), Y B1 FA – Conteúdo do endereço rrss + Y vai para A; endereços qq ou FA contém ss e qq + 1 ou FA + 1 contém rr.

Nota: No caso a) o endereço efetivo rrss foi obtido pela indexação prévia de qq com X, ao passo que no caso b, o endereço qq forneceu o valor rrss, que foi posteriormente indexado por Y.

Endereço relativo das instruções de desvios

Nas instruções de desvios Bcc (Branch), o código-objeto contém apenas um valor “ds” (de 8 bits), variando de -126 a $+129$ e correspondente à posição relativa de desvio para posições antes ou depois da instrução Bcc. Na instrução Assembler, o programador utiliza diretamente o endereço absoluto ou simbólico de desvio e o tradutor Assembler calcula o valor “ds” correspondente.

Exemplos:

BCC Z1 90 ds – Desvia para endereço Z1.

Ilustração:

a) Operando Imediato

LDA #\$0F Valor imediato 0F → A 0F

b) Endereço direto da memória

MEMÓRIA

LDA \$A0 Endereço 00A0 0F → A 0F

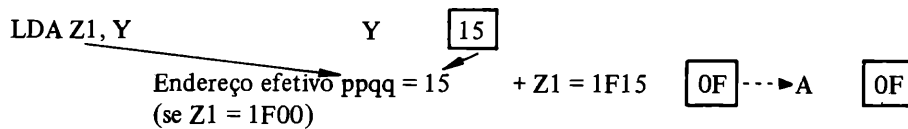
LDA Z1 Endereço Z1 0F → A 0F

c) Endereço indexado com X ou Y

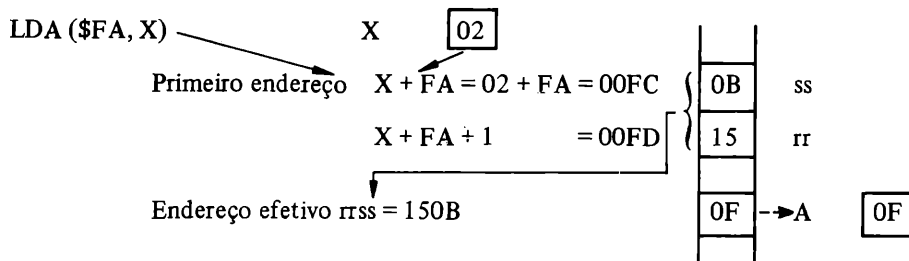
LDA \$A0, X X 13

Endereço efetivo ppqq = 13 + A0 = 00B3

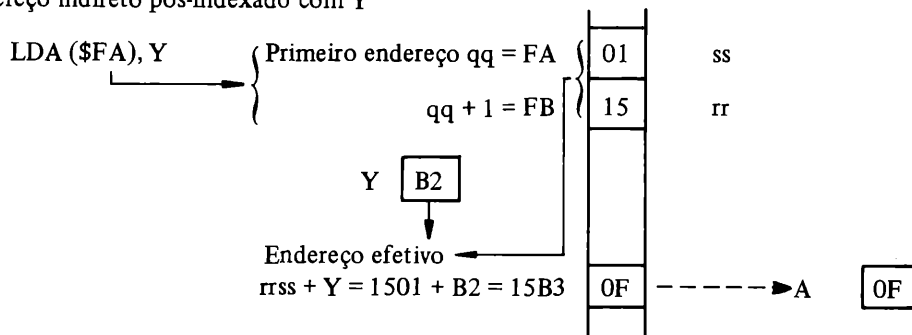
0F → A 0F



d) Endereço indireto pré-indexado com X



e) Endereço indireto pós-indexado com Y



XIII.3 PRINCIPAIS INSTRUÇÕES DIRETIVAS

A linguagem Assembler do 6502 admite, em geral, pelo menos as seguintes instruções direti-vas:

a) DEFINIR BYTES OU VALORES CONSTANTES NA MEMÓRIA

BYT ou .BYTE – Define um valor constante de 1 byte.

DBY .DBYTE – Define um valor constante de 2 bytes, colocando byte mais signi-
ficativo em primeiro lugar.

.WORD – Análogo a DBYTE, mas colocando byte menos significativo em
primeiro lugar.

b) DEFINE FIM DO PROGRAMA ASSEMBLER

END ou .END

c) DEFINE UMA SÉRIE (ou STRING) DE CONSTANTE EM NOTAÇÃO ASCII

ASC ou .TEXT

d) DEFINE ORIGEM, IGUALDADE OU UMA ÁREA DE ENDEREÇOS

- EQU ou = – Estabelece igualdade entre dois endereços (EQUATE).
 ORG ou * = \$FFF – Define origem (ORG) do programa a partir do endereço \$FFF.
 DFS ou * = * + N – Define uma área de N bytes, a partir desta instrução.

Nota: Existem várias versões do Assembler 6502. A versão mais comum usa as formas .BYTE, .DBYTE, .END, * = etc. A versão LISA (*) usa BYT, DBY, END, ORG, EQU etc. O Manual Assembler que está em uso deve ser consultado.

Exemplos:

- | | | | |
|--|----|---|--|
| INIC = \$1000 | ou | } | – Endereço simbólico INIC será colocado no endereço \$1000 (EQUATE) em hexadecimal. |
| INIC EQU \$1000 | | | |
| * = INIC | ou | } | – Origem do programa-objeto será a partir do endereço INIC = \$1000. |
| ORG INIC | | | |
| V1 .BYTE \$F | ou | } | – Endereço V1 conterá valor de um byte 0F. |
| V1 BYT \$F | | | |
| V3 .WORD \$F1F2 | | | – Endereço V3 conterá valor \$F2 (byte menos significativo em primeiro lugar) e valor \$F1 será colocado no endereço V3 + 1. |
| CODIG .TEX /FORMULA-1/ – Coloca nas nove posições seguintes, a contar do endereço CODIG, os caracteres F, O, R, M, U, L, A, –, e 1 em notação ASCII. | | | |
| AREA * = * + 9 | ou | } | – Reserva, a partir do endereço ÁREA, nove posições de bytes, na memória. |
| AREA DFS 9 | | | |
| BYT ou .BYTE 15 | | | – Define, nesta posição, sem nome ou rótulo, o valor decimal quinze. |
| BYT ou .BYTE %1111 | | | – Define, nesta posição, o valor binário 1111, equivalente ao byte 0F. |

(*) LISA: A PROFESSIONAL ASSEMBLY DEVELOPMENT SYSTEM FOR APPLE COMPUTERS, por Randy Hyde, Publicação da On-Line Systems, 1981.

EXERCÍCIOS

De Recapitulação

1. Quais são os elementos básicos da arquitetura do microprocessador 6502?

2. Dizer qual é a capacidade em bits e para que serve cada um dos seguintes elementos da arquitetura do 6502:

- Acumulador A.
- Registros X e Y.
- Contador de Programa.
- Ponteiro da Pilha.
- Flags C, Z, V, S, D, B e I.
- Página Zero.

Sugestão: consultar capítulo I sobre Noções Gerais da Arquitetura de um microprocessador.

3. Quais são os elementos que formam uma instrução Assembler do 6502? Para que serve cada um desses elementos?

Sugestão: ver Capítulo II sobre elementos básicos de uma linguagem Assembler.

4. Descrever a operação e citar exemplos de cada modo de endereçamento do operando Assembler 6502:

- Instrução sem operandos.
- Instrução com registros.
- Instrução com operando imediato.
- Instrução com endereço direto da memória.
- Instrução com endereço indireto da memória.
- Instrução com endereço indexado.
- Instrução com endereço relativo.

5. Citar exemplos de cada uma das seguintes instruções diretivas do Assembler 6502:

- Define valor de um byte na memória.
- Define fim de programa.
- Define início ou origem de um programa ou área.
- Define igualdade ou equivalência de endereços.
- Define uma área de k bytes.
- Define valor com dois bytes.

CAPÍTULO

XIV

INSTRUÇÕES DE MOVIMENTO OU TRANSFERÊNCIA DE DADOS DO 6502: LOAD E STORE

XIV.1 FORMAS GERAIS DAS INSTRUÇÕES

Neste capítulo apresentaremos as seguintes instruções:

LOAD ACCUMULATOR – Carrega Acumulador A com o valor do operando

- | | |
|-------------------|---|
| LDA #dd | – operando imediato. |
| LDA endereço | – endereço direto da memória. |
| LDA endereço, X | – endereço indexado com X. |
| LDA endereço, Y | – endereço indexado com Y. |
| LDA (endereço, X) | – endereço indireto pré-incrementado por X. |
| LDA (endereço), Y | – endereço indireto pós-incrementado por Y. |

LOAD REGISTER X OU Y – Carrega registro X ou Y com o valor do operando.

- | | |
|---------------------|--------------------|
| LDX ou LDY #dd | – imediato. |
| LDX ou LDY endereço | – endereço direto. |
| LDX endereço, Y | – indexado por Y. |
| LDY endereço, X | – indexado por X. |

STORE ACCUMULATOR – Armazena conteúdo de A no operando.

- | | |
|----------------------|---------------------------------|
| STA endereço | – endereço direto. |
| STA endereço, X ou Y | – endereço indexado por X ou Y. |
| STA (endereço, X) | – indireto. |
| STA (endereço), Y | – indireto. |

STORE REGISTER X OR Y – Armazena conteúdo de X ou Y no operando

STX ou STY endereço – endereço direto.

STX endereço, Y – indexado.

STY endereço, X – indexado.

TRANSFERÊNCIA DE DADOS ENTRE REGISTROS OU STACK POINTER

TAX, TAY, TXA, TYA, TSX e TXS

PROGRAMAS-EXEMPLOS – os seguintes programas-exemplos são estudados:

- TRANSFERÊNCIA DE VALOR USANDO ENDEREÇOS ABSOLUTOS.
- TRANSFERÊNCIA DE VALOR USANDO ENDEREÇOS SIMBÓLICOS.
- TRANSFERÊNCIA DE DADOS COM PROGRAMA MONTADO A PARTIR DO ENDEREÇO 2000.

XIV.2 LDA (LOAD ACCUMULATOR) – CARREGA ACUMULADOR COM O VALOR DO OPERANDO IMEDIATO OU CONTEÚDO DA MEMÓRIA

Assembler	Código-Objeto	
Imediato		
LDA #dd	A9 dd	– Valor imediato dd vai para acumulador A.
Direto		
LDA endereço	A5 qq	– Conteúdo do endereço 00qq (página zero) vai para A.
	AD qq pp	– Conteúdo do endereço ppqq vai para A.
Indexado		
LDA endereço, X	B5 qq	– Conteúdo do endereço 00qq + X (página zero) vai para A.
	BD qq pp	– Conteúdo do endereço ppqq + X vai para A.
LDA endereço, Y	B9 qq pp	– Conteúdo do endereço ppqq + Y vai para A.
Indireto Indexado		
Pré-incrementado com X		– Conteúdo de rrss vai para Acumulador A.
LDA (endereço, X)	A1 qq	Endereço 00qq + X contém ss e 00qq + X + 1 contém rr.
Pós-incrementado com Y		– Conteúdo do endereço rrss + Y vai para A.
LDA (endereço), Y	B1 qq	Endereço 00qq contém ss e 00qq + 1 contém rr.

LDX ou LDY (LOAD REGISTER X OR Y) – CARREGA REGISTRO X ou Y COM O VALOR DO OPERANDO IMEDIATO OU CONTEÚDO DA MEMÓRIA.

Imediato

LDX #dd	A2 dd	– Valor imediato dd vai para Registro X.
LDY #dd	A0 dd	– <i>Idem</i> , para registro Y.

Direto

LDX endereço	A6 qq	– Conteúdo do endereço 00qq (página zero) vai para X.
	AE qq pp	– Conteúdo do endereço ppqq vai para X.
LDY endereço	A4 qq	– Conteúdo do endereço 00qq (página zero) vai para Y.
	AC qq pp	– Conteúdo do endereço ppqq vai para Y.

Indexado

LDX endereço, Y	B6 qq	– Conteúdo do endereço 00qq + Y vai para X.
	BE qq pp	– Conteúdo do endereço ppqq + Y vai para X.
LDY endereço, X	B4 qq	– Conteúdo de 00qq + X vai para Y.
	BC qq pp	– Conteúdo de ppqq + X vai para Y.

Indireto

– Não existe modo indireto.

Nota: Se “endereço” estiver na PÁGINA ZERO (endereço 0000 a 00FF), o endereço “qq” de um byte é usado no código-objeto, caso contrário, é usado o endereço “ppqq” de dois bytes.

Exemplos:

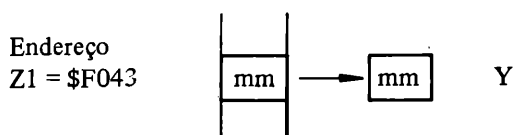
Explicação

LDA #\$0F	A9 0F	– Valor (Imediato) hexadecimal 0F vai para A.
LDA \$A0	A5 A0	– (Direto Página Zero) Conteúdo do endereço A0 vai para A.
LDA Z1	AD 43 F0	– (Direto longo) Conteúdo do endereço simbólico Z1, que recebeu endereço absoluto \$F043, vai para A. Se Z1 receber endereço qq da página zero (00 a FF), código-objeto seria A5 qq.
LDA \$A0, X	B5 A0	– (Indexado com X) Conteúdo do endereço A0 + X vai para A.
LDA Z1, Y	B9 43 F0	– (Indexado com Y) Conteúdo do endereço Z1 + Y ou F043 + Y vai para A.

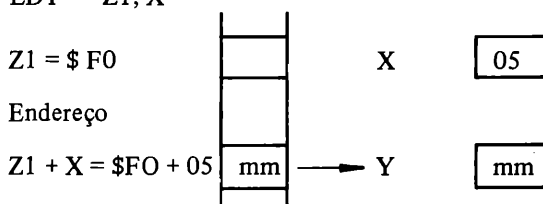
LDA (\$FA, X)	A1 FA	– (Indireto pré-incrementado com X) Conteúdo do endereço ppqq vai para A; endereço FA + X contém qq e FA + X + 1 contém pp.
LDA (Z1), Y	B1 A0	– (Indireto pós-incrementado com Y) Endereço simbólico Z1 precisa estar na página zero. Por exemplo Z1 = A0, então conteúdo de ppqq + Y vai para A; endereço Z1 = A0 contém qq e Z1 + 1 = A1 contém pp.
LDX #\$0F	A2 0F	– Análogo a LDA #\$0F.
LDY Z1	A4 A0	– Conteúdo de Z1 = A0 vai para A, se Z1 estiver no endereço A0 (página zero).
	AC 43 F0	– Conteúdo de Z1 = F043 vai para A, se Z1 = F043.
LDY \$F043, X	BC 43 F0	– (Indexado com X) Conteúdo do endereço F043 + X vai para Y.

Ilustração:

a) LDY Z1



b) LDY Z1, X

**XIV.3 STA (STORE ACCUMULATOR) – ARMAZENA O CONTEÚDO DO ACUMULADOR NA MEMÓRIA**

Assembler

Código-Objeto

Imediato

– Modo imediato não é usado.

Direto

STA endereço

85 qq

– Conteúdo do Acumulador A vai para endereço 00qq (página zero).

	8D qq pp	– Conteúdo de A vai para endereço ppqq.
Indexado		
STA endereço, X	95 qq	– Conteúdo de A vai para o endereço 00qq + X.
	9D qq pp	– Conteúdo de A vai para o endereço ppqq + X.
STA endereço, Y	99 qq pp	– Conteúdo de A vai para o endereço ppqq + Y.
Indireto Indexado		
Pré-incrementado com X		
STA (endereço, X)	81 qq	– Conteúdo de A vai para o endereço rrss, ss está no endereço 00qq + X e rr está em 00qq + X + 1.
Pós-incrementado com Y.		
STA (endereço), Y	91 qq	– Conteúdo de A vai para o endereço rrss + Y, ss está no endereço 00qq e rr no endereço 00qq + 1.

STY ou STX (STORE REGISTER X ou Y) – ARMAZENA O CONTEÚDO DO REGISTRO X ou Y NA MEMÓRIA

Imediato

– Não existe.

Direto

STX endereço	86 qq	– Conteúdo do registro X vai para endereço 00qq (página zero).
	8E qq pp	– Conteúdo de X vai para endereço ppqq.
STY endereço	84 qq	– Conteúdo de Y vai para o endereço 00qq (página zero).
	8C qq pp	– Conteúdo de Y vai para o endereço ppqq.

Indexado

STX endereço, Y	96 qq	– Conteúdo de X vai para o endereço 00qq + Y.
STY endereço, X	94 qq	– Conteúdo de Y vai para o endereço 00qq + X.

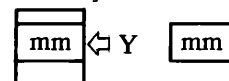
Indireto

– Não existe.

Exemplos:

STY Z1 8C 43 F0 – Conteúdo de Y vai para endereço Z1 = \$F043

Endereço Z1 = \$F043.



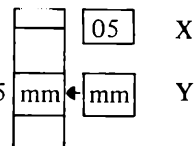
STY Z1, X

94 A0

— Conteúdo de Y vai para Z1 + X = \$A0 + X

Z1 = \$A0

Endereço Z1 + X = \$A0 + 05



XIV.4 TRANSFERÊNCIA DE DADOS (MOVE) ENTRE ACUMULADOR OU STACK POINTER E REGISTROS X OU Y

	Assembler	Código-Objeto
DE ACUMULADOR PARA REGISTRO X :	TAX	AA
PARA REGISTRO Y :	TAY	A8
DE REGISTRO X PARA ACUMULADOR :	TXA	8A
DE REGISTRO Y PARA ACUMULADOR :	TYA	98
DE STACK POINTER PARA REGISTRO X:	TSX	BA
DE REGISTRO X PARA STACK POINTER:	TXS	9A

XIV.5 PROGRAMA-EXEMPLO

TRANSFERÊNCIA DE VALOR USANDO ENDEREÇOS ABSOLUTOS

Transferir o valor 2B que está no endereço absoluto F1 para o endereço absoluto F3.

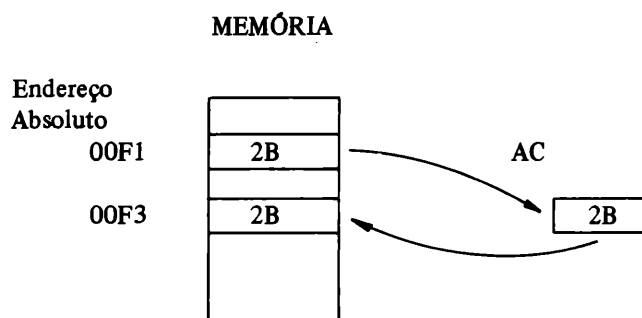
2000	1	EX6501	ORG \$2000
2000 A5 F1	2		LDA \$F1
2002 85 F3	3		STA \$F3
2004 00	4		BRK
2005	5		END

***** END OF ASSEMBLY

Comentários:

- 1) Programa começa no endereço 2000.
- 2) Conteúdo de F1 para Acumulador A.
- 3) Conteúdo de A para endereço F3.
- 4) Pare.
- 5) Fim do programa Assembler.

Observação: Os endereços F1 e F3 estão na PÁGINA ZERO (de 00 a FF) e sendo assim as instruções LDA e STA ocupam apenas dois bytes cada.



XIV.6 PROGRAMA-EXEMPLO:

TRANSFERÊNCIA DE DADOS USANDO ENDEREÇOS SIMBÓLICOS

Transferir o valor 2B que está no endereço simbólico Z1 para o endereço simbólico Z2.

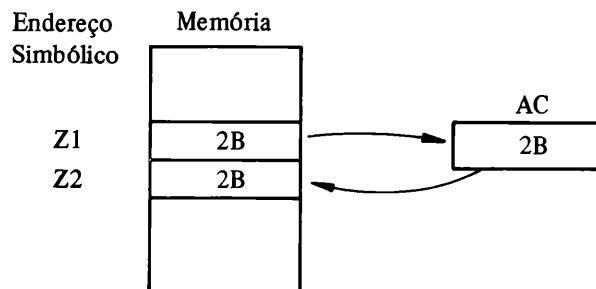
```
0800 AD 07 08    1 EX6502 LDA Z1
0803 8D 08 08    2          STA Z2
0806 00          3          BRK
0807 2B          4 Z1      BYT $2B
0808 00          5 Z2      BYT 0
0809             6          END
```

***** END OF ASSEMBLY

Comentários:

- 1) Conteúdo de Z1 vai para A.
- 2) Conteúdo de A vai para endereço Z2.
- 3) Pare.
- 4) Instrução diretiva coloca valor hexadecimal 2B no endereço Z1.
- 5) *Idem* define endereço Z2 que vai receber o valor.
- 6) Fim do programa.

Notas: Sem a instrução diretiva ORG, o programa foi montado automaticamente a partir do endereço hexadecimal 800. Os endereços 0807 e 0808 dos símbolos Z1 e Z2 aparecem, no código-objeto, com os bytes trocados.



Importante: O uso de endereços simbólicos TORNA OBRIGATÓRIO o uso de instruções diretivas (BYTE, DBYTE, WORD ou DFS) ou outro tipo de rótulo, para definir o endereço efetivo ocupado por esses endereços simbólicos.

Quando usamos endereços absolutos, supomos que os valores a serem utilizados já estão nesses endereços, dispensando a definição de seus valores.

XIV.7 PROGRAMA-EXEMPLO:

TRANSFERÊNCIA DE DADOS COM O PROGRAMA MONTADO A PARTIR DO ENDEREÇO 2000

Transferir o valor 2B que está no endereço simbólico Z1 para o endereço simbólico Z2. Usar a instrução diretiva correspondente (ORG ou equivalente) para montar o código-objeto a partir do endereço 2000.

VERSÃO PARA 6502

```

2000          1 EX6503  ORG $2000
2000 AD 07 20   2      LDA Z1
2003 8D 08 20   3      STA Z2
2006 00        4      BRK
2007 2B        5 Z1    BYT $2B
2008 00        6 Z2    BYT 0
2009          7      END

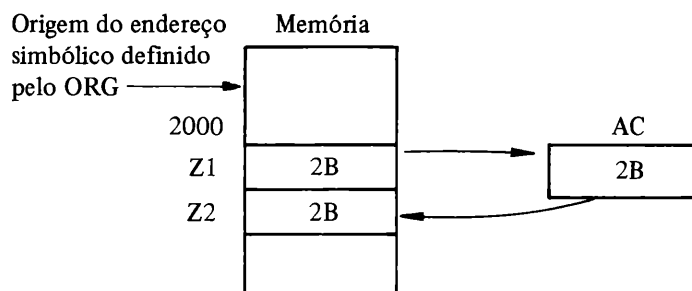
```

***** END OF ASSEMBLY

Observações:

- Programa é idêntico ao anterior, porém com o endereço inicial de montagem fixado em 2000.
- Os endereços simbólicos Z1 e Z2 não entram na PÁGINA ZERO, e assim as instruções LDA e STA são longas, ocupando três bytes.
- Os endereços 2007 e 2008 entram com os bytes invertidos nas instruções LDA e STA.

- **VERSÕES DO ASSEMBLER** – As instruções diretivas *ORG*, *BYT* e *END* são da versão Assembler denominada *LISA*. São equivalentes às diretivas **=*, *.BYTE* e *.END*. Verificar o Manual Assembler que está sendo usado, para escolher a sua versão.



EXERCÍCIOS

De Recapitulação

1. Descrever e citar exemplos de cada uma das formas possíveis das seguintes instruções:
 - *LOAD OR STORE ACCUMULATOR*.
 - *LOAD OR STORE REGISTER X OU Y*.
 - *TRANSFER REGISTERS OR STACK POINTER*.

De Aplicação

Escrever a versão Assembler e respectivo código-objeto dos programas abaixo citados. Todos os programas devem ser testados manualmente, instrução por instrução, através de um exemplo numérico ilustrado, mencionando os conteúdos dos registros e memórias afetadas.

1. Transferir um valor do endereço absoluto hexadecimal 1500 para o endereço hexadecimal 1601.
2. Transferir os bytes dos endereços hexadecimais 1500 e 1501 para 1601 e 1605, respectivamente.
3. Transferir quatro bytes dos endereços simbólicos V1, V2, V3 e V4 para os endereços simbólicos S1, S2, S3 e S4, respectivamente.
4. Modificar o programa do exercício anterior 3 de modo que:
 - Os códigos de instrução (área de programa) sejam colocados a partir do endereço hexadecimal 1500.
 - Os endereços simbólicos V1, V2, V3, V4 sejam colocados a partir do endereço hexa 1700 e os endereços S1, S2, S3 e S4, a partir do endereço hexa 1800.

Sugestão: utilizar instrução diretiva *ORG* ou *"=*".

5. **ZERAR ENDEREÇOS** Modificar os programas dos exercícios 1, 2, 3 e 4 de maneira que os valores transferidos para os endereços destinos sejam sempre ZERO.
6. **TRANSFERÊNCIA DE TEXTO.** Colocar, a partir do endereço NOME, os caracteres do seu nome e sobrenome pessoal. Transferir esses caracteres para outra área, que começa com o endereço simbólico NOVONOME.

CAPÍTULO

XV

INSTRUÇÕES DE OPERAÇÕES ARITMÉTICAS DO 6502: INCREMENT, DECREMENT, ADD E SUBTRACT

XV.1 FORMAS GERAIS DAS INSTRUÇÕES

As instruções apresentadas neste capítulo são:

INX e INY	— Incrementa registro X ou Y.
INC endereço	— Incrementa memória com endereço direto.
INC endereço, X	— Incrementa memória com endereço indexado.
DEX e DEY	— Decrementa registro X ou Y.
DEC endereço	— Decrementa memória com endereço direto.
DEC endereço, X	— Decrementa memória com endereço indexado.
ADC #dd	— Soma imediato de dd com Carry.
ADC endereço	— Soma com endereço direto e Carry.
ADC endereço, X ou Y	— Soma com endereço indexado e Carry.
ADC (endereço, X)	— Soma com endereço indireto pré-indexado e Carry.
ADC (endereço), Y	— Soma com endereço indireto pós-indexado e Carry.
SBC #dd	— Subtrai de A o valor imediato dd e o complemento de Carry.
SBC endereço	— Subtração com endereço direto.
SBC endereço, X ou Y	— Subtração com endereço indexado.
SBC (endereço, X)	— Subtração com endereço indireto pré-indexado.
SBC (endereço), Y	— Subtração com endereço indireto pós-indexado.

PROGRAMAS-EXEMPLOS — Os programas-exemplos estudados são:

- SOMA DE VALORES COM OITO BITS.
- SOMA DE DOIS VALORES COM DEZESSEIS BITS CADA.

XV.2 INX, INY E INC (INCREMENT X, Y OR MEMORY) — INCREMENTA DE VALOR UM O CONTEÚDO DO REGISTRO X, Y OU DA MEMÓRIA

Assembler	Código-Objeto	
Registros		
INX	E8	— Incrementa Registro X.
INY	C8	— Incrementa Registro Y.
Direto		
INC endereço	E6 qq	— Incrementa conteúdo do endereço 00qq (página zero).
	EE qq pp	— Incrementa conteúdo do endereço ppqq.
Indexado		
INC endereço, X	F6 qq	— Incrementa conteúdo do endereço 00qq + X.
	FE qq pp	— Incrementa conteúdo do endereço ppqq + X.

XV.3 DEX, DEY E DEC (DECREMENT X, Y OR MEMORY) — DECREMENTA DE VALOR UM O CONTEÚDO DO REGISTRO X, Y OU DA MEMÓRIA

Registro		
DEX	CA	— Decrementa Registro X.
DEY	88	— Decrementa Registro Y.
Direto		
DEC endereço	C6 qq	— Decrementa conteúdo do endereço 00qq (página zero).
	CE qq pp	— Decrementa conteúdo do endereço ppqq.
Indexado		
DEC endereço, X	D6 qq	— Decrementa conteúdo do endereço 00qq + X.
	DE qq pp	— Decrementa conteúdo do endereço ppqq + X.

Exemplos:

INC \$A0	E6 A0	— Incrementa conteúdo do endereço \$A0.
DEC Z1	CE 43 F0	— Decrementa conteúdo do endereço \$F043 = Z1.
INC Z1, X	FE 43 F0	— Incrementa conteúdo de Z1 + X = \$F043 + X.

XV.4 ADC (ADD WITH CARRY) – SOMA O CONTEÚDO DO ACUMULADOR COM O OPERANDO IMEDIATO OU CONTEÚDO DA MEMÓRIA ACRESCIDO DO CARRY. RESULTADO PERMANECE NO ACUMULADOR

Assembler	Código-Objeto	
Imediato		
ADC #dd	69 dd	– Soma valor imediato dd e CARRY com Acumulador A.
Direto		
ADC endereço	65 qq	– Soma conteúdo do endereço 00qq (página zero) com Carry e Acumulador A.
	6D qq pp	– Soma conteúdo do endereço ppqq com CARRY e A.
Indexado		
ADC endereço, X	75 qq	– Soma conteúdo de 00qq + X com Carry e Acumulador A.
	7D qq pp	– Soma conteúdo de ppqq + X com Carry e Acumulador A.
ADC endereço, Y	79 qq pp	– Soma conteúdo de ppqq + Y com Carry e Acumulador A.
Indireto		
ADC endereço, X	61 qq	– Soma conteúdo do endereço rrss com Carry e Acumulador; endereço 00qq + X contém ss e 00qq + X + 1 contém rr.
ADC endereço, Y	71 qq	– Soma conteúdo do endereço rrss + Y com Carry e Acumulador; endereço 00qq contém ss e 00qq + 1 contém rr.

Exemplos:

		Explicação
ADC #\$0F	69 0F	— Soma valor hexadecimal 0F com CARRY e conteúdo de A, e resultado fica em A.
ADC \$A0	65 A0	— Soma conteúdo do endereço A0 com CARRY e conteúdo de A. Resultado fica em A.
ADC Z1	6D 43 F0	— Soma conteúdo de Z1 = F043 com CARRY e conteúdo de A.
ADC \$A0, X	75 A0	— Soma conteúdo de A0 + X com CARRY e conteúdo de A.
ADC Z1, X	7D 43 F0	— Soma conteúdo de Z1 + X = F043 + X com CARRY e A.
ADC (\$A0), Y	71 A0	— Soma conteúdo do endereço ppqq + Y com CARRY e A; endereço A0 contém pp e A1 contém qq.
ADC (Z1, X)	61 A0	— Soma conteúdo de ppqq com CARRY e A; endereço Z1 + X = A0 + X contém qq e Z1 + X + 1 = A0 + X + 1 contém pp; Z1 precisa estar na página zero (00 a FF).

Ilustração:

a) ADC #\$0F

Registro A:	A1
Valor imediato:	+ 0F
CARRY:	<u>1</u>
SOMA EM A:	B1

b) ADC Z1

Endereço Z1 = \$A0 :	+ 20
	<u>1</u>
	C2

c) ADC Z1, X

Endereço Z1 + X = F043 + X :	20
	<u>1</u>
SOMA EM A:	C2

RESULTADO

EM BINÁRIO: 1011 0001 = B1

1100 0010 = C2

Status Flags:

Carry	C = 0 (não houve Carry)	C = 0 (<i>idem</i>)
Sinal	S = 1 (resultado é negativo)	S = 1 (<i>idem</i>)
Zero	Z = 0 (resultado não é zero)	Z = 0 (<i>idem</i>)
Transbordo	V = 0 (não houve overflow)	V = 0 (<i>idem</i>)

O Flag V de transbordo é o resultante da operação EXCLUSIVE OR dos bits VAI-UM ou VAI-ZERO ocorrido na soma dos bits das posições 6 e 7, isto é, nos dois bits mais à esquerda do resultado. Nos dois exemplos acima ocorreram: VAI-ZERO entre soma dos bits da posição seis e VAI-ZERO entre soma dos bits da posição sete, resultando em bit V = 0. Para maior detalhe consultar o manual de Hardware do Microprocessador 6502.

XV.5 SBC (SUBTRACT WITH BORROW) – SUBTRAI DO ACUMULADOR, O VALOR DO OPERANDO IMEDIATO OU O CONTEÚDO DA MEMÓRIA ACRESCIDO DO COMPLEMENTO DO CARRY. O RESULTADO PERMANECE NO ACUMULADOR.

Assembler	Código-Objeto	
Imediato		
SBC #dd	E9 dd	— Subtrai de A o valor imediato dd e o complemento do Carry ($\overline{C}$).
Direto		
SBC endereço	E5 qq	— Subtrai de A o conteúdo do endereço 00qq (página zero) e o complemento do Carry.
	ED qq pp	— <i>Idem</i> , o conteúdo de ppqq e o complemento do Carry.
Indexado		
SBC endereço, X	F5 qq	— <i>Idem</i> , o conteúdo do endereço 00qq + X e o complemento do Carry.
	FD qq pp	— <i>Idem</i> , o conteúdo do endereço ppqq + X e o complemento do Carry.
SBC endereço, Y	F9 qq pp	— <i>Idem</i> , o conteúdo do endereço ppqq + Y e o complemento do Carry.
Indireto indexado		
SBC (endereço, X)	E1 qq	— Subtrai de A, o conteúdo do endereço rrss e $\overline{C}$; endereço 00qq + X contém ss e 00qq + X + 1 contém rr.
SBC (endereço), Y	F1 qq	— Subtrai de A o conteúdo do endereço rrss + Y e $\overline{C}$; endereço 00qq contém ss e 00qq + 1 contém rr.

Exemplos:

		Explicação
SBC #\$0F	E9 0F	– Conteúdo de A menos ($\$0F + \overline{\text{CARRY}}$) fica em A (Imediato).
SBC \$A0	E5 A0	– Conteúdo de A menos (conteúdo de \$A0 mais $\overline{\text{CARRY}}$) fica em A (direto com página zero).
SBC Z1	ED 43 F0	– Conteúdo de A menos (conteúdo de Z1 = F043 mais $\overline{\text{CARRY}}$) fica em A (direto longo).
SBC Z1, Y	F9 43 F0	– Conteúdo de A menos (conteúdo de \$F043 + Y mais $\overline{\text{CARRY}}$) fica em A (Indexado com Y).
SBC (\$A0, X)	E1 A0	– Conteúdo de A menos (conteúdo do endereço ppqq mais $\overline{\text{CARRY}}$) fica em A. Endereço A0 + X contém qq e A0 + X + 1 contém pp. (Indireto pré-incrementado com X.)
SBC (\$A0), Y	F1 A0	– Conteúdo de A menos (conteúdo de ppqq + Y mais $\overline{\text{CARRY}}$) fica em A. Endereço A0 contém qq e A0 + 1 contém pp. (Indireto pós-incrementado com Y.)
SBC (Z1, X)	E1 A0	– Análogo a SBC (\$A0, X) com Z1 = \$A0 na página zero.

Nota: $\overline{\text{CARRY}} = 1 - \text{CARRY}$ = Complemento do CARRY.

Ilustração:**a) SBC #\$34**

Registro A:	14
Operando Imediato:	34
Complemento Carry:	1
Resultado em A:	$14 - (34 + 1)$

b) SBC \$A0, X

Registro X:	02
Registro A:	14
Endereço \$A0 + 02:	34
Complemento Carry:	1
Resultado em A:	$14 - (34 + 1)$

c) SBC (\$A0, X)

Registro X:	02
Endereço \$A0 + 02:	qq = 21
Endereço \$A1 + 02:	pp = 1F
Registro A:	14
Em ppqq = 1F21:	34
Complemento Carry:	1
Resultado em A:	$14 - (34 + 1)$

RESULTADO EM BINÁRIO:

$$\begin{array}{r}
 14 = 0001 \ 0100 \\
 - (34 + 1) = -35 = 1100 \ 1011 \quad (\text{complemento de dois de } 35) \\
 \hline
 \end{array}$$

Resultado em A: $-21 = 1101 \ 1111$

Flags de Status:

Carry $C = 0$ (não houve VAI-UM no resultado ou houve EMPRESTA-UM ou BORROW no resultado)

Sinal $S = 1$ (resultado é negativo)

Zero $Z = 0$ (resultado não é zero)

Overflow $V = 0$ (não houve transbordo no resultado)

Nota: A ocorrência de EMPRESTA-UM no resultado da subtração equivale a não haver VAI-UM na adição com complemento de dois no 6502. No 8085 e Z80, EMPRESTA-UM é indicado por $C = 1$.

O flag de overflow depende de VAI-UM ou VAI-ZERO nas somas dos bits da posição seis e sete do resultado.

XV.6 PROGRAMA-EXEMPLO**SOMA DE 8 BITS**

Somar o conteúdo do endereço absoluto 00F1 (ou endereço simbólico Z1) com o conteúdo do endereço absoluto 1A00 (ou endereço simbólico Z2) e colocar o resultado no endereço absoluto 1B01 (ou endereço simbólico RESULT).

Algoritmo

Endereço 00F1 (ou Z1) contém: 21

Endereço 1A00 (ou Z2) contém: A3

Soma usando Acumulador AC: $21 + A3 = C4$

Endereço 1B01 (ou RESULT) conterà: C4

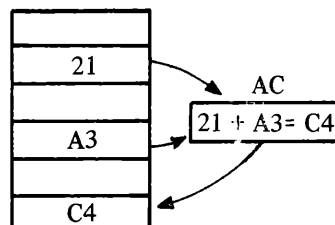
Endereços

00F1 (ou Z1)

1A00 (ou Z2)

1B01 (ou RESULT)

Memória



Versão para 6502 (usando endereços absolutos 00F1, 1A00 e 1B01)

Código-Objeto

Assembler

2000	1	EX6504	ORG \$2000
2000 18	2		CLC
2001 A5 F1	3		LDA \$F1
2003 6D 00 1A	4		ADC \$1A00

```
2006 8D 01 1B    5      STA $1B01
2009 00          6      BRK
200A           7      END
```

******* END OF ASSEMBLY**

Versão para 6502 (usando endereços simbólicos Z1, Z2 e RESULT)

Código-Objeto		Assembler
2000	1	EX6505 ORG \$2000
2000 1B	2	CLC
2001 AD 0B 20	3	LDA Z1
2004 6D 0C 20	4	ADC Z2
2007 8D 0D 20	5	STA RESULT
200A 00	6	BRK
200B 21	7	Z1 BYT \$21
200C A3	8	Z2 BYT \$A3
200D 00	9	RESULT BYT 0
200E	10	END

******* END OF ASSEMBLY**

Comentários:

- 1) Instrução diretiva fixa endereço inicial do programa em 2000.
 - 2) Carry igualada a zero, pois ADC sempre inclui Carry.
 - 3) Carrega acumulador com conteúdo de 00F1 (ou Z1).
 - 4) Soma acumulador com conteúdo de 1A00 (ou Z2).
 - 5) Armazena resultado em 1B01 (ou RESULT).
 - 6) Pare.
- (para versão com endereços simbólicos).
- 7) Define valor hexadecimal 21 no endereço Z1.
 - 8) Define valor hexadecimal A3 no endereço Z2.
 - 9) Define valor inicial zero no endereço RESULT.

Observação: O endereço absoluto 00F1 está na PÁGINA ZERO e sendo assim, sua instrução LDA \$F1 só ocupa dois bytes de código-objeto.
Na versão com endereços absolutos 00F1 e 1A00, supõe-se que os valores iniciais \$21 e \$A3 das parcelas já existam na memória, uma vez que não é necessário usar a instrução diretiva BYT ou .BYTE.

XV.7 PROGRAMA-EXEMPLO:**SOMA DE DOIS VALORES COM 16 BITS**

Somar dois valores de 16 bits cada: o primeiro valor está colocado nos endereços consecutivos F1 e F2 (ou simbólicos Z1 e Z1 + 1); o segundo valor está nos endereços F3 e F4 (ou Z2 e Z2 + 1); e o resultado será colocado nos endereços F5 e F6 (ou RESULT e RESULT + 1).

Algoritmo-Exemplo

Soma: 2315 (hexa) + 25F1 (hexa) = 4906 (hexadecimal)

Primeiro valor 2315 (hexa) foi colocado em:

00F1 ou Z1 = 15 (byte menos significativo)

00F2 ou Z1 + 1 = 23 (byte mais significativo)

Segundo valor 25F1 (hexa) foi colocado em:

00F3 ou Z2 = F1 (byte menos significativo)

00F4 ou Z2 + 1 = 25 (byte mais significativo)

Resultado da soma será colocado em:

00F5 ou RESULT = 06 (byte menos significativo)

00F6 ou RESULT + 1 = 49 (byte mais significativo)

Guardar e incluir o CARRY na soma de um byte para outro.

Memória		
		Carry: 1 0
Z1 ou 00F1	15	} Primeiro valor: 23 15
00F2	23	
Z2 ou F3	F1	} Segundo Valor: 25 F1 +
F4	25	
RESULT ou F5	06	} Soma: 49 06
F6	49	

Versão para 6502 (usando endereços absolutos F1, F2, . . . , F6)

2000	1	EX6506	ORG \$2000
2000 18	2		CLC
2001 A5 F1	3		LDA \$F1
2003 65 F3	4		ADC \$F3
2005 85 F5	5		STA \$F5
2007 A5 F2	6		LDA \$F2
2009 65 F4	7		ADC \$F4
200B 85 F6	8		STA \$F6
200D 00	9		BRK
200E	10		END

***** END OF ASSEMBLY

Versão para 6502 (usando endereços simbólicos Z1, Z2 e RESULT)

2000	1	EX6507	ORG \$2000
2000 18	2		CLC
2001 AD 14 20	3		LDA Z1
2004 5D 16 20	4		ADC Z2
2007 8D 18 20	5		STA RESULT
200A AD 15 20	6		LDA Z1+1
200D 6D 17 20	7		ADC Z2+1
2010 8D 19 20	8		STA RESULT+1
2013 00	9		BRK
2014 15 23	10	Z1	DBY \$1523
2016 F1 25	11	Z2	DBY \$F125
201B 00 00	12	RESULT	DBY 0
201A	13		END

***** END OF ASSEMBLY

Comentários:

- 1) Inicia programa-objeto no endereço \$2000.
- 2) CARRY = 0.
- 3) Carrega AC com byte menos significativo do primeiro valor.
- 4) Soma AC com byte menos significativo do segundo valor, CARRY = 1.
- 5) Armazena resultado de AC no byte menos significativo do resultado.
- 6) Carrega AC com byte mais significativo do primeiro valor.
- 7) Soma AC com byte mais significativo do segundo valor, junto com CARRY.

- 8) Armazena resultado de AC no byte mais significativo do resultado.
- 9) Pare.
- 10) Primeiro valor definido a partir de Z1 (versão com endereço simbólico).
- 11) Segundo valor definido a partir de Z2 (versão com endereço simbólico).
- 12) Resultado colocado a partir de RESULT, após a execução do programa.
- 13) Fim.

Nota: Na versão com endereço absoluto, supõe-se que os valores das parcelas já estão colocados na memória.

Observação: A colocação dos bytes da palavra em posições invertidas

- a) Cada valor de 16 bits foi colocado na memória com os bytes invertidos, isto é, o byte menos significativo vem em primeiro lugar, a fim de utilizar endereços crescentes de F1 a F6 (ou de Z1, Z1 + 1 até RESULT + 1).

COLOCAÇÃO DOS ENDEREÇOS SIMBÓLICOS Z1, Z2 e RESULT A PARTIR DO ENDEREÇO 00F1

Podemos colocar os endereços simbólicos Z1, Z2 e RESULT a partir do endereço que desejamos montar a instrução. Sendo assim, para colocar os valores e resultados a partir do endereço 00F1, temos:

Para 6502

Assembler		Código-Objeto
1)	CLC	0000 (os mesmos da versão com endereços absolutos
	...	F1, F2, F3, F4, F5 e F6)
8)	BRK	
	ORG \$F1	
9) Z1	DBY \$1523	00F1 15 23
10) Z2	DBY \$F125	00F3 F1 25
11) RESULT	DBY 0	00F5 00 00
12)	.END	

EXERCÍCIOS

De Recapitulação

1. Descrever e citar exemplos de cada uma das formas possíveis das instruções:

- INCREMENT e DECREMENT.
- ADD e SUBTRACT.

De Aplicação

Escrever a versão Assembler e respectivo código-objeto dos programas abaixo citados. Todos os programas devem ser testados manualmente, instrução por instrução, através de um exemplo numérico ilustrado, mencionando os conteúdos dos registros e memórias afetadas.

1. Somar o valor que está no endereço hexadecimal 1500 com o valor do endereço hexadecimal 1601. Colocar o resultado no endereço hexadecimal 1602.
2. Somar os valores que estão nos endereços simbólicos V1 e V2 e colocar o resultado no endereço simbólico V3.
3. SOMA DE 32 BITS — Somar o valor de 32 bits que está nos endereços consecutivos V1, V2, V3 e V4 com o valor de 32 bits que está nos endereços simbólicos S1, S2, S3 e S4. Colocar o resultado da soma nos endereços T1, T2, T3 e T4.
4. Modificar os programas dos exercícios 2 e 3 de modo que os códigos de instrução (área de programa) sejam colocados a partir do endereço hexadecimal 1500 e os valores utilizados (área de dados) sejam colocados a partir do endereço hexadecimal 1800. Sugestão: utilizar a instrução diretiva ORG ou “ = * ”.
5. INCREMENTAR OU DECREMENTAR VALORES — Modificar os programas dos exercícios 1 e 2 de modo que o resultado das somas seja de incremento (ou decremento) simples de valor 1 em cada posição da memória.
6. SUBTRAÇÃO — Modificar os programas dos exercícios 1 e 2 de modo que a operação executada seja de subtração.
7. SUBTRAÇÃO DE 32 BITS — Subtrair, de um valor de 32 bits, colocado nos endereços V1, V2, V3 e V4, outro valor de 32 bits, que está nos endereços S1, S2, S3 e S4. Colocar o resultado da subtração nos endereços T1, T2, T3 e T4.

CAPÍTULO

XVI

INSTRUÇÕES DE COMPARAÇÃO E DESVIOS DO 6502: COMPARE, BRANCH E JUMP

XVI.1 FORMAS GERAIS DAS INSTRUÇÕES

As instruções mencionadas neste capítulo são:

CMP #dd	— Compara A com valor imediato dd.
CMP endereço	— Compara A com valor no endereço direto.
CMP endereço, X	— Comparação com valor no endereço indexado.
CMP endereço, Y	— Comparação com valor no endereço indexado.
CMP (endereço, X)	— Comparação com endereço indireto.
CMP (endereço), Y	— Comparação com endereço indireto.
CPX ou CPY #dd	— Comparação de X ou Y com valor imediato dd.
CPX ou CPY endereço	— Comparação de X ou Y com valor no endereço direto.
Bcc endereço	— Desvia para o endereço se condição cc for verdade.
JMP endereço	— Desvia incondicional para endereço direto.
JMP (endereço)	— Desvia incondicional para endereço indireto.

PROGRAMA-EXEMPLO – O programa-exemplo discutido é:

- **COMPARAÇÃO E DESVIO: ACHAR O MAIOR DE DOIS VALORES DA MEMÓRIA.**

XVI.2 CMP (COMPARE MEMORY WITH ACC)

Assembler	Código-Objeto	Explicação
Imediato		
CMP #dd	C9 dd	— Compara Acumulador A com valor dd.
Direto		
CMP endereço	C5 qq	— Compara A com conteúdo de qq (página zero).
	CD qq pp	— Compara A com conteúdo de ppqq.
Indexado com X		
CMP endereço, X	D5 qq	— Compara A com conteúdo de qq + X (página zero).
	DD qq pp	— <i>Idem</i> , com conteúdo de ppqq + X.
Indexado com Y		
CMP endereço, Y	D9 qq pp	— <i>Idem</i> , com conteúdo de ppqq + Y.
Indireto		
Pré-incrementado com X		
CMP (endereço, X)	C1 qq	— Compara A com conteúdo de endereço rrss: 00qq + X contém ss e qq + X + 1 contém rr.
Pós-incrementado com Y		
CMP (endereço), Y	D1 qq	— Compara A com conteúdo de rrss + Y; endereço 00qq contém ss e qq + 1 contém rr.

XVI.3 CPX (COMPARE INDEX REGISTER X WITH IMMEDIATE OR MEMORY)

Imediato		
CPX #dd	E0 dd	— Compara X com valor imediato dd.
Direto		
CPX endereço	E4 qq	— Compara X com conteúdo de 00qq (página zero).
	EC qq pp	— <i>Idem</i> , com conteúdo de ppqq.

XVI.4 CPY (COMPARE INDEX REGISTER Y WITH IMMEDIATE OR MEMORY)**Imediato**

CPY #dd C0 dd – Compara Y com valor imediato dd.

Direto

CPY endereço C4 qq – Compara Y com conteúdo de 00qq (página zero).

CC qq pp – *Idem*, com conteúdo de ppqq.

Exemplos:

- a) CMP #\$17 C9 17 – Compara conteúdo de A com valor imediato \$17:
 Operação: Resultado de (A – \$17) afeta flags S, Z, C.
 Registro A: F5 = 1111 0101
 (–) Operando Imediato: (–) 17 = 1110 1001
 (A – \$17): 1101 1110
 Flags afetados: Z = 0 Resultado diferente de zero.
 C = 1 Houve “vai-um”.
 S = 1 Bit 7 de sinal é um.
- b) CMP \$A0, X D5 A0 – Compara A com valor no endereço \$A0 + X; resultado de
 (A menos conteúdo de \$A0 + X) afeta flags S, Z, C.
 Registro X: 02
 Registro A: F5
 (–) Conteúdo de \$A0 + 02: (–) 17
 Resultado: (\$F5 – \$17)
 Flags afetados: Z = 0, C = 1 e S = 1 (análogo ao caso a).
- c) CPY Z1 C4 A0 – Compara A com conteúdo de Z1 = \$A0 (página zero).
 CC 43 F0 – Compara A com conteúdo de Z1 = \$F043.

Nota: conteúdos de A e operando não são alterados.

XVI.5 BCC – BRANCH -- DESVIA PARA O ENDEREÇO INDICADO SE A CONDIÇÃO “CC” INDICADA OCORRER.

O endereço relativo (deslocamento ou acréscimo positivo ou negativo) “ds” que aparece no código-objeto é calculado do seguinte modo:

$$ds = \text{endereço} - PC - 2$$

O valor calculado (pelo Assembler) “ds” deve estar dentro dos limites de +129 a -126 bytes relativos à posição corrente da instrução BCC (valor atual ou corrente do PC). Valor de “ds” fora desses limites é detetado pelo programa Assembler.

Nenhum flag é afetado.

O código de condição “CC” pode assumir uma das seguintes condições:

Assembler	Código-Objeto	
BCC endereço	90 ds	– Desvia se Carry Clear (C = 0).
BCS endereço	B0 ds	– Desvia se Carry Set (C = 1).
BEQ endereço	F0 ds	– Desvia se Equal to Zero (Flag Z = 1).
BMI endereço	30 ds	– Desvia se menos (Flag S = 1).
BNE endereço	D0 ds	– Desvia se não for igual a zero (Z = 0).
BPL endereço	10 ds	– Desvia se positivo (S = 0).
BVC endereço	50 ds	– Desvia se overflow clear (V = 0).
BVS endereço	70 ds	– Desvia se overflow set (V = 1).

Exemplos

Explicação

BCC Z1	90 ds	– Desvia para endereço \$A0 (Z1 = A0) se CARRY = 0.
BEQ Z1	F0 ds	– Desvia para endereço Z1 = \$43F0 se flag Z = 1.
BVS \$A0	70 ds	– Desvia para endereço \$A0 se flag V = 1.

Nota: Tanto os endereços simbólicos Z1 = \$A0 e Z1 = \$43F0 como o endereço absoluto \$A0 devem estar dentro do limite permitido do deslocamento ds. O valor “ds” depende, portanto, da posição real da instrução dentro da memória.

XVI.6 JMP (JUMP) DESVIA INCONDICIONALMENTE (SEMPRE) PARA O ENDEREÇO DIRETO OU INDIRETO ESPECIFICADO. NÃO AFETA FLAGS.

Direto

JMP endereço 4C qq pp – Desvia para o endereço ppqq.

Indireto

JMP (endereço) 6C qq pp – Desvia para o endereço rrss; ppqq contém ss e ppqq + 1 contém rr.


```
2006 A5 52      5      LDA $52
2008 85 53      6 COLOC STA $53
200A 00        7      BRK
200B          8      END
```

***** END OF ASSEMBLY

Versão para 6502 (com endereços simbólicos Z1, Z2 e Z3)

```
2000          1 EX6509 ORG $2000
2000 AD 0F 20   2      LDA Z1
2003 CD 10 20   3      CMP Z2
2006 B0 03      4      BCS COLOC
2008 AD 10 20   5      LDA Z2
200B BD 11 20   6 COLOC STA Z3
200E 00        7      BRK
200F 5F        8 Z1    BYT $5F
2010 BA        9 Z2    BYT $BA
2011 00       10 Z3    BYT 0
2012          11      END
```

***** END OF ASSEMBLY

Comentários:

- 1) Início do programa no endereço 2000.
- 2) Conteúdo do endereço 51 (ou Z1) vai para acumulador AC.
- 3) Comparar AC com conteúdo de 52 (ou Z2).
- 4) Se resultado $AC - (52 \text{ ou } Z2) > 0$, desvia para COLOC (se CARRY SET).
- 5) Caso contrário, carregue AC com valor em 52 (ou Z2).
- 6) Coloque o valor em AC (valor maior) em 53 (ou Z3).
- 7) Pare.
- 8) Endereço Z1 contém valor 5F (versão com endereços simbólicos).
- 9) Endereço Z2 contém valor BA.
- 10) Endereço Z3 conterá após a execução o valor maior BA.
- 11) FIM.

Nota: Na versão com endereços absolutos, os valores nos endereços 51, 52 devem ser colocados através de outro programa, pois não foram usadas as diretivas BYT.

XVI.8

Observação: Uso do bit CARRY na comparação.

Para comparar dois valores absolutos sem o sinal (*unsigned numbers*) usou-se o seguinte esquema:

Se $Z1 < Z2$ então $CARRY = 0$ no 6502, uma vez que $Z1 - Z2 < 0$, e haverá necessidade de “empréstimo ou borrow” indicado no CARRY.

Se $Z1 \geq Z2$ então $CARRY = 1$ no 6502, pois não há necessidade do “empréstimo ou borrow”.

Sendo assim, usou-se:

BCS COLOC (BRANCH IF CARRY SET) para 6502.

Se considerarmos os valores com o respectivo sinal no último bit à esquerda (e variando no intervalo -126 a $+127$), podemos verificar esse bit através do bit N ou S de SINAL NEGATIVO do flag de status:

BPL COLOC (BRANCH IF PLUS) para 6502.

Nota: No 6502, o flag de CARRY possui valor inverso ao dos demais microprocessadores.

EXERCÍCIOS

De Recapitulação

1. Descrever e citar exemplos de cada uma das formas possíveis das seguintes instruções:

- COMPARE MEMORY.
- COMPARE REGISTER.
- BRANCH OU JUMP.

De Aplicação

Escrever a versão Assembler e respectivo código-objeto dos programas abaixo mencionados. Todos os programas devem ser testados manualmente, instrução por instrução, através de um exemplo numérico, mencionando os conteúdos dos registros e memórias afetadas.

1. Comparar os valores que estão nos endereços hexadecimais 1500 e 1600. Colocar o menor desses valores no endereço 1601 e o maior no endereço 1602.
2. Comparar os valores que estão nos endereços simbólicos V1 e V2. Colocar o menor desses valores no endereço simbólico S1 e o maior no endereço simbólico S2.

3. Comparar, um por um, os bytes que estão nos endereços V1, V2, V3 e V4 com os bytes que estão nos endereços simbólicos S1, S2, S3 e S4. Colocar todos os bytes de maior valor a partir do endereço simbólico MAIOR e os bytes de menor valor, a partir do endereço MENOR.
4. ORDENAÇÃO DE TRÊS BYTES – Comparar os bytes que estão nos endereços simbólicos V1, V2 e V3 e colocá-los em ordem crescente de seus valores.

Sugestão: Utilizar o seguinte algoritmo:

- a) Comparar V1 e V2; se $V1 > V2$ trocar conteúdos de V1 e V2; caso contrário nada se faz.
- b) Comparar V2 e V3: se $V2 > V3$ trocar conteúdos de V3 e V2; caso contrário nada se faz.
- c) Repetir as operações a e b e terminar.

CAPÍTULO

XVII

INSTRUÇÕES DE OPERAÇÕES LÓGICAS DO 6502: AND, BIT TEST, OR E EXCLUSIVE OR

XVII.1 FORMATO DAS INSTRUÇÕES

As instruções estudadas neste capítulo são:

AND #dd	— Operação lógica AND do conteúdo de A com valor imediato dd.
AND endereço	— Operação AND de A com endereço direto da memória.
AND endereço, X ou Y	— Operação AND de A com endereço indexado da memória.
AND (endereço, X)	— Operação AND de A com endereço indireto.
AND (endereço), Y	— Operação AND de A com endereço indireto.
BIT endereço	— Operação AND com conteúdo da memória, sem modificar A.
ORA #dd	— Operação lógica OR de A com valor imediato dd.
ORA endereço	— Operação OR com endereço direto.
ORA endereço, X ou Y	— Operação OR com endereço indexado.
ORA (endereço, X)	— Operação OR com endereço indireto.
ORA (endereço), Y	— Operação OR com endereço indireto.
EOR #dd	— Operação lógica OU EXCLUSIVO de A com valor imediato dd.
EOR endereço	— Operação EOR com endereço direto.
EOR endereço, X ou Y	— Operação EOR com endereço indexado.
EOR (endereço, X)	— Operação EOR com endereço indireto.
EOR (endereço), Y	— Operação EOR com endereço indireto.

PROGRAMA-EXEMPLO – O programa-exemplo estudado é:

- USO DA OPERAÇÃO “AND” PARA SEPARAR POR MASCARAGEM OS QUATRO BITS MENOS SIGNIFICATIVOS DE UM BYTE.

XVII.2 AND (LOGICAL AND)

Executa operação lógica “AND” ou “E” do operando imediato ou conteúdo da memória com o conteúdo do Acumulador A, ficando o resultado em A.

Afeta os flags S e Z.

Operação lógica “AND”: $0 + 0 = 0$ $0 + 1 = 1$ $1 + 0 = 0$ e $1 + 1 = 1$.

Assembler	Código-Objeto	Explicação
Imediato		
AND #dd	29 dd	– AND do valor dd com A.
Direto		
AND endereço	25 qq	– AND do conteúdo de 00qq (página zero) com A.
	2D qq pp	– AND do conteúdo de ppqq com A.
Indexado		
AND endereço, X	35 qq	– AND do conteúdo de 00qq + X com A.
	3D qq pp	– AND do conteúdo de ppqq + X com A.
AND endereço, Y	39 qq pp	– AND do conteúdo de ppqq + Y com A.
Indireto		
Pré-incrementado com X		
AND (endereço, X)	21 qq	– AND do conteúdo de rrss com A; qq + X contém ss e qq + X + 1 contém rr.
Pós-incrementado com Y		
AND (endereço), Y	31 qq	– AND do conteúdo de rrss + Y com A; qq contém ss e qq + 1 contém rr.

Exemplos

AND #\$1C	29 1C	– Operação AND do valor \$1C com conteúdo de A.
-----------	-------	---

Acumulador A: 13 = 0001 0011

Valor imediato: 1C = 0001 1100

Resultado em A: 10 = 0001 0000

Flags: Z = 0 (resultado não é zero)

S = 0 (resultado é positivo)

AND \$A0, X	35 A0	— Operação AND do valor em \$A0 + X com A.
		Registro X: 02
		Acumulador A: 13
		Conteúdo de \$A0 + 02: 1C
		Resultado em A: 10
		Flags: Z = 0 e S = 0. Não afeta demais flags.

XVII.3 BIT (BIT TEST)

Executa operação AND entre o conteúdo do acumulador e memória, afetando os flags S e Z. Entretanto, deixa inalterado o conteúdo do acumulador e da memória, ao contrário da instrução lógica AND, que coloca resultado no Acumulador A.

Direto

BIT endereço	24 qq	— AND de conteúdo de A com memória afeta flags, mas sem colocar resultado no acumulador.
	2C qq pp	

XVII.4 ORA (LOGICAL OR)

Executa operação lógica “OR” ou “OU” do operando imediato ou conteúdo da memória com o conteúdo do Acumulador A, ficando o resultado em A.

Afeta flags S e Z.

Operação lógica OR : $1 + 1 = 1 + 0 = 0 + 1 = 1$ e $0 + 0 = 0$

Assembler	Código-Objeto	Explicação
Imediato		
ORA #dd	09 dd	— OR do valor imediato dd com A.

Direto

ORA endereço	05 qq	— OR do valor do endereço qq (página zero) com A.
	0D qq pp	— OR do conteúdo de ppqq com A.

Indexado

ORA endereço, X	15 qq	— OR do conteúdo de qq + X com A.
	1D qq pp	— OR do conteúdo de ppqq + X com A.
ORA endereço, Y	19 qq pp	— OR do conteúdo de ppqq + Y com A.

Indireto**Pré-incrementado com X**

ORA (endereço, X)	01 qq	— OR do conteúdo de rrss com A; qq + X contém ss e qq + X + 1 contém rr.
-------------------	-------	--

Pós-incrementado com Y

ORA (endereço), Y	11 qq	— OR do conteúdo de rrss + Y com A; qq contém ss e qq + 1 contém rr.
-------------------	-------	--

Exemplos

ORA #\$1C	09 1C	— Operação OR do valor imediato \$1C com conteúdo de A.
-----------	-------	---

Acumulador A: 13 = 0001 0011

Valor imediato: 1C = 0001 1100

Resultado em A: 1F = 0001 1111

Flags: Z = 0 (resultado não é zero)

S = 0 (resultado é positivo). Não afeta demais flags.

ORA \$A0FD, X	1D FD A0	— Operação OR do conteúdo de \$A0FD + X com A.
---------------	----------	--

Registro X: 02

Acumulador A: 13

Conteúdo de \$A0FD + 02: 1C

Resultado em A: 1F

XVII.5 EOR (EXCLUSIVE OR)

Executa operação lógica “OU EXCLUSIVO” do operando imediato ou conteúdo da memória com o conteúdo do acumulador, ficando o resultado em A.

Afeta flags S e Z.

Operação “EXCLUSIVE OR”: $0 + 1 = 1 + 0 = 1$ e $0 + 0 = 1 + 1 = 0$

Assembler	Código-Objeto	Explicação
Imediato		
EOR #dd	49 dd	– EOR do valor imediato dd com A.
Direto		
EOR endereço	45 qq	– EOR do conteúdo de qq (página zero) com A.
	4D qq pp	– EOR do conteúdo de ppqq com A.
Indexado		
EOR endereço, X	55 qq	– EOR do conteúdo de qq + X com A.
	5D qq pp	– EOR do conteúdo de ppqq + X com A.
EOR endereço, Y	59 qq pp	– EOR do conteúdo de ppqq + Y com A.
Indireto		
Pré-incrementado com X		
EOR (endereço, X)	41 qq	– EOR do conteúdo de rrss com A; qq + X contém ss e qq + X + 1 contém rr.
Pós-incrementado com Y		
EOR (endereço), Y	51 qq	– EOR do conteúdo de rrss + Y com A; qq contém ss e qq + 1 contém r.

Exemplos

EOR #\$1C	49 dd	– Operação EOR do valor imediato \$1C com A. Acumulador A: 13 = 0001 0011 Valor imediato: 1C = 0001 1100 Resultado em A: 0F = 0000 1111 Flags: análogos à instrução ORA.
EOR \$A0, X	55 A0	– Operação EOR do conteúdo de \$A0 + X com A.

Registro X:	<u>02</u>
Acumulador A:	<u>13</u>
Valor em \$A0 + 02:	<u>1C</u>
Resultado em A:	0F

XVII.6 PROGRAMA-EXEMPLO:

USO DA OPERAÇÃO LÓGICA “AND”: SEPARA (POR MASCARAGEM) OS QUATRO BITS MENOS SIGNIFICATIVOS DE UM BYTE

Separar os quatro bits menos significativos de um byte que está no endereço absoluto A1 (ou simbólico Z1) e colocar esses bits separados no endereço absoluto A2 (ou simbólico Z2).

Algoritmo ou operação:

- Apagar (por mascaram com operação AND) os quatro bits mais significativos de A1 ou Z1.
- Colocar os bits que restaram, no endereço A2 ou Z2.

Operação AND . $0 + 0 = 0$ e $0 + 1 = 1 + 0 = 0$

$1 + 1 = 1$

O bit 0 apaga o outro bit e bit 1 mantém o outro bit.

VERSÃO PARA 6502 (USANDO ENDEREÇOS ABSOLUTOS)

Código-Objeto	Assembler
2000	1 EX6510 ORG \$2000
2000 A5 A1	2 LDA \$A1
2002 29 0F	3 AND #\$0F
2004 B5 A2	4 STA \$A2
2006 00	5 BRK
2007	6 END

***** END OF ASSEMBLY

VERSÃO PARA 6502 (USANDO ENDEREÇOS SIMBÓLICOS)

```
2000          1 EX6511  ORG $2000
2000 AD 09 20   2          LDA Z1
2003 29 0F      3          AND #$0F
2005 8D 0A 20   4          STA Z2
2008 00         5          BRK
2009 3C         6 Z1      BYT $3C
200A 00         7 Z2      BYT 0
200B          8          END
```

**** END OF ASSEMBLY

Comentários:

- 1) Início do programa vai ser em 2000.
- 2) Conteúdo do endereço A1 ou Z1 vai para AC.
- 3) Operação AND do conteúdo de AC com valor imediato OF:
 $00111100 + 00001111 = 00001100 = 0C$.
- 4) Bits separados em AC vão para endereço A2 ou Z2.
- 5) Pare.
- 6) Endereço Z1 contém valor 3C.
- 7) Endereço Z2 conterá valor 0C, após a execução do programa.
- 8) Fim do programa.

Nota: Na versão com endereço absoluto, o programa separa os bits do valor qualquer existente no endereço 00A1.

EXERCÍCIOS

De Recapitulação

1. Descrever e citar exemplos de cada uma das formas possíveis das instruções seguintes:
 - LOGICAL AND.
 - LOGICAL OR.
 - LOGICAL EXCLUSIVE OR.

De Aplicação

Escrever a versão Assembler e respectivo código-objeto dos programas abaixo mencionados. Todos os programas devem ser testados manualmente, instrução por instrução, através de um exemplo numérico, mencionando os conteúdos dos registros e memórias afetadas, após cada instrução.

1. Apagar o bit de sinal (isto é, colocar zero no bit 7) de um byte, através da operação AND. Manter demais bits inalterados.
2. Colocar bit 1 nos quatro bits mais significativos de um byte, através de uma máscara de operação LOGICAL OR. Manter demais bits inalterados.
3. Colocar bit 1 (SET) nos bits das posições 0, 3 e 5 de um byte de memória, utilizando uma máscara apropriada. Manter demais bits inalterados.
4. Colocar bit 1 (SET) nos bits de posição 0 e 3 e bit zero (RESET) nos bits de posição 5 e 6 de um byte da memória, utilizando operações lógicas e máscaras apropriadas. Manter demais bits inalterados.

CAPÍTULO

XVIII

APLICAÇÕES: PROGRAMAS COM CONTADOR DE REPETIÇÕES (LOOPS)

XVIII.1 INTRODUÇÃO

Neste capítulo examinaremos vários exemplos de programas que utilizam praticamente todas as instruções já examinadas: LOAD, STORE, ADD, COMPARE, BRANCH e AND.

Todos os programas utilizam também a técnica de repetição de um mesmo trecho do programa, quantas vezes necessárias. Esta técnica denominada “LOOPS” ou REPETIÇÃO DE COMANDOS é bastante utilizada em qualquer tipo de programação, usando linguagem de alto nível ou Assembler.

Nas linguagens de alto nível, esta técnica é usada nos seguintes tipos de comandos:

BASIC

FOR I = 1 TO N STEP 2

...

NEXT I

FORTRAN

DO 25 I = 1, N, 2

...

25 CONTINUE

PASCAL

FOR I := 1 TO N DO BEGIN ... END;

REPEAT BEGIN ... END

UNTIL I = N;

WHILE I <= N DO BEGIN ... END;

Os programas-exemplos analisados neste capítulo são:

- SOMA DE N VALORES DE UM BYTE CADA.
- ACHAR O MAIOR VALOR DE UMA SÉRIE DE K NÚMEROS.
- CONTAR O NÚMERO DE CARACTERES DE UMA SEQUÊNCIA OU “STRING” DE CARACTERES.
- SOMA DE VALORES COM MÚLTIPLA PRECISÃO.

XVIII.2 PROGRAMA-EXEMPLO:**OPERAÇÃO DE REPETIÇÃO (LOOPS): SOMA DE N VALORES DE UM BYTE CADA**

Calcular a soma de N valores de um byte cada.

O valor N está no endereço absoluto A1 (ou endereço simbólico TAM) e a série de valores começa no endereço absoluto A2 (ou endereço simbólico SÉRIE).

Colocar o valor da soma no endereço simbólico A0 (ou simbólico SOMA).

Algoritmo:

Início: Colocar valor N em um contador, zerar acumulador.

Repetir a operação REPETE até o valor do contador ficar nulo.

Operação REPETE; Acumulador = Acumulador + um valor da série.

Contador = Contador - 1;

Se Contador = 0 então terminar.

Terminar: Colocar valor total do acumulador no endereço A0 ou SOMA.

6502 (Versão usando endereços simbólicos)

Código-objeto		Assembler
2000	1 EX6512	ORG \$2000
2000 A9 00	2	LDA #0
2002 AA	3	TAX
2003 18	4 REPET	CLC
2004 7D 13 20	5	ADC SERIE,X
2007 EB	6	INX
2008 EC 12 20	7	CPX TAM
200B D0 F6	8	BNE REPET
200D 8D 11 20	9	STA SOMA
2010 00	10	BRK
2011 00	11 SOMA	BYT 0
2012 03	12 TAM	BYT 3
2013 15	13 SERIE	BYT \$15
2014 05	14	BYT \$05
2015 C1	15	BYT \$C1
2016	16	END

***** END OF ASSEMBLY

Ilustração:

Condição Inicial:

Registro A ou Acumulador	00	
Registro usado como contador	00 ou N = 3	
Valores da memória:		
Endereço SOMA	00	– Local onde será colocado o valor da soma
TAM	03	– N = 3
SÉRIE	15	} Série de 3 valores
	05	
	C1	

Situação após a execução do programa:

Registro ou Acumulador A	DB	
Registro usado como contador	03 ou 00	
Valores da memória:		
Endereço SOMA	DB	} \$15 + 05 + \$C1 = \$DB
TAM	03	
SÉRIE	15	
	05	
	C1	

Nota: O contador pode ser incrementado ou decrementado.**Comentários:**

- 1) Início do programa vai ser no endereço 2000.
- 2) Acumulador igual a zero.
- 3) Contador = registro X = 0.
- 4) Flag de Carry = 0, pois 6502 só possui instrução ADD WITH CARRY.
- 5) Acumulador AC = AC + conteúdo do endereço (SÉRIE + X); soma um valor da série.
- 6) Incrementa contador X.
- 7) Compara contador X com valor N em TAM.
- 8) Se for diferente (menor que N) vai para REPET e repete a soma.
- 9) Se for igual, coloca total somado no endereço SOMA.
- 10) Pare.

- 11) Endereço SOMA irá conter o valor $\$15 + \$05 + \$C1 = \DB .
- 12) TAM contém o valor $N = 3$.
- 13) Primeiro valor da série.
- 14) Segundo valor da série.
- 15) Terceiro valor da série etc. até no máximo 256 valores podem ser somados.
- 16) Fim do programa.

XVIII.3 PROGRAMA-EXEMPLO:

MAIOR VALOR DE UMA SÉRIE DE K NÚMEROS

Achar o maior valor de uma série de K valores, de um byte cada. Esses valores estão colocados, na memória, a partir do endereço SÉRIE. O valor K que indica o tamanho da série está no endereço COMPR.

O maior valor encontrado deve ser colocado no endereço simbólico chamado MAXIM.

Algoritmo:

Início: Contador = valor K.

Acumulador = zero.

Repetir a operação REPETE até que o valor do contador seja igual a zero.

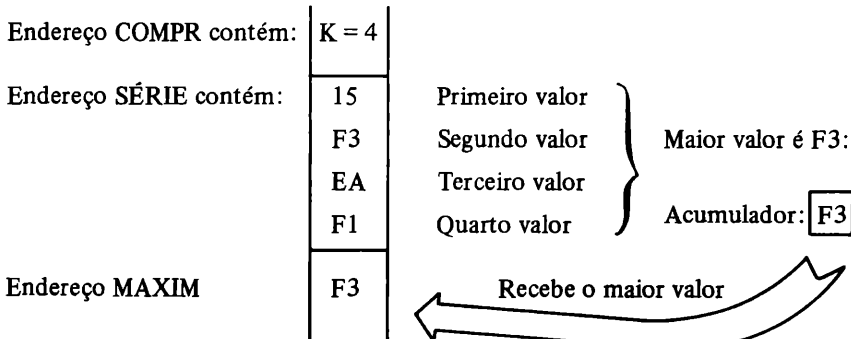
REPETE: Se acumulador for maior ou igual a um valor da SÉRIE, então nada é feito; caso contrário, acumulador = esse valor da SÉRIE.

Contador = Contador - 1.

Terminar: Coloca maior valor, que está no acumulador, no endereço MAXIM.

Nota: O contador pode decrescer de valor K até zero, ou crescer do valor 0 até K, dependendo da maior conveniência de programação.

Ilustração:



VERSÃO PARA 6502

Código-Objeto		Assembler
2000	1	EX6513 ORG \$2000
2000 AE 15 20	2	LDX COMPR
2003 A9 00	3	LDA #0
2005 DD 15 20	4	REPET CMP SERIE-1,X
2008 B0 03	5	BCS DIMIN
200A BD 15 20	6	LDA SERIE-1,X
200D CA	7	DIMIN DEX
200E D0 F5	8	BNE REPET
2010 BD 14 20	9	STA MAXIM
2013 00	10	BRK
2014 00	11	MAXIM BYT 0
2015 04	12	COMPR BYT 4
2016 15	13	SERIE BYT \$15
2017 F3	14	BYT \$F3
2018 EA	15	BYT \$EA
2019 F1	16	BYT \$F1
201A	17	END

***** END OF ASSEMBLY

Comentários (6502):

- 1) Endereço inicial do programa vai ser 2000.
- 2) Registro X usado como contador recebe valor K que está no endereço COMPR.
- 3) Acumulador = zero.
- 4) Comparar acumulador com o último byte da SÉRIE; endereço (SÉRIE - 1 + X) aponta para o último byte da SÉRIE.
- 5) Se acumulador for maior, vai para endereço DIMIN.
- 6) Se acumulador for menor, isto é, temos um novo valor maior que o anterior, então coloca esse valor maior no acumulador.
- 7) Decrementa contador X.
- 8) Se contador for maior que zero, repete operação prosseguindo em REPET.
- 9) Se contador for zero, então coloca maior valor que está no acumulador no endereço MAXIM.
- 10) Pare.
- 11) Define na memória endereço MAXIM que receberá o maior valor encontrado.
- 12) Define na memória endereço COMPR que contém o valor K = 04.

- 13) Define na memória, a partir do endereço SÉRIE, o primeiro valor da série.
- 14) Define segundo valor.
- 15) Define terceiro valor.
- 16) Define quarto valor etc. até no máximo 256 valores.
- 17) Fim do programa.

XVIII.4 PROGRAMA-EXEMPLO:

CONTA NÚMERO DE CARACTERES ASCII EM UMA SEQUÊNCIA (STRING) DE CARACTERES

- Determinar o tamanho (número de caracteres) de um string de caracteres. O string ou sequência de caracteres está colocado, na memória, a partir do endereço STRIN e termina com um caractere CR ou CARRIAGE RETURN (corresponde ao valor hexadecimal 0D em ASCII), que marca o fim da sequência.

Colocar o valor do número de caracteres obtido (sem incluir o caractere CR) no endereço COMPR.

Algoritmo:

Início: Contador de caracteres recebe valor zero.

Acumulador recebe o caractere Carriage Return = 'CR' = 0D (em hexadecimal).

Repetir: Compara próximo caractere do string com Acumulador: se for "CR", terminar, caso contrário, incrementa contador e vai para Repetir.

Terminar: Coloca valor do contador no endereço COMPR.

VERSÃO PARA 6502

Código-Objeto		Assembler
2000	1 EX6514	ORG %2000
2000 A2 00	2	LDX #0
2002 A9 0D	3	LDA #\$0D
2004 DD 12 20	4 REPET	CMP STRIN,X
2007 F0 04	5	BEQ TERMI
2009 EB	6	INX
200A 4C 04 20	7	JMP REPET
200D BE 11 20	8 TERMI	STX COMPR
2010 00	9	BRK
2011 00	10 COMPR	BYT 0

```

2012 52      11 STRIN  BYT 'R'
2013 41      12      BYT 'A'
2014 54      13      BYT 'T'
2015 4F      14      BYT 'O'
2016 00      15      BYT $0D
2017         16      END

```

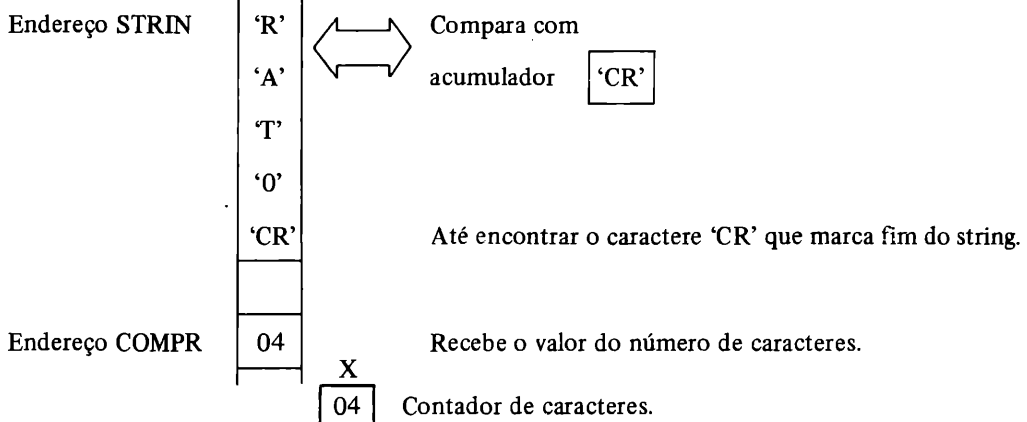
***** END OF ASSEMBLY

Ilustração:

Acumulador: contém caractere 'CR' marcador de fim de string.

Contador (Registro X): começa com 00 e termina com 04.

String de caracteres na memória:



Comentários (6502):

- 1) Programa começa em 2000.
- 2) Registro X usado como contador recebe valor zero.
- 3) Acumulador recebe caractere 'CR' = \$0D, marcador de fim de string.
- 4) Compara acumulador com primeiro caractere do string.
- 5) Se for igual a \$0D, vai para TERMI.
- 6) Se for diferente, incrementa contador X.
- 7) Vai para REPET.
- 8) Terminou, coloca valor do contador X no endereço COMPR.

- 9) Pare.
- 10) Define na memória o endereço COMPR que vai receber o número de caracteres.
- 11) Início do string: primeiro caractere.
- 12) Segundo caractere.
- 13) Terceiro caractere.
- 14) Quarto caractere.
- 15) Caractere CR = \$0D que marca o fim do string.
- 16) Fim do programa.

XVIII.5 PROGRAMA-EXEMPLO:

SOMA DE VALORES COM MÚLTIPLA PRECISÃO

Somar dois valores binários formados por N bytes cada. Por exemplo, para N = 5:

Valor um: 3A 56 F1 45 1A

Valor dois: A3 23 01 F0 E1

Soma: DD 79 F3 35 FB

Algoritmo:

Condição Inicial:

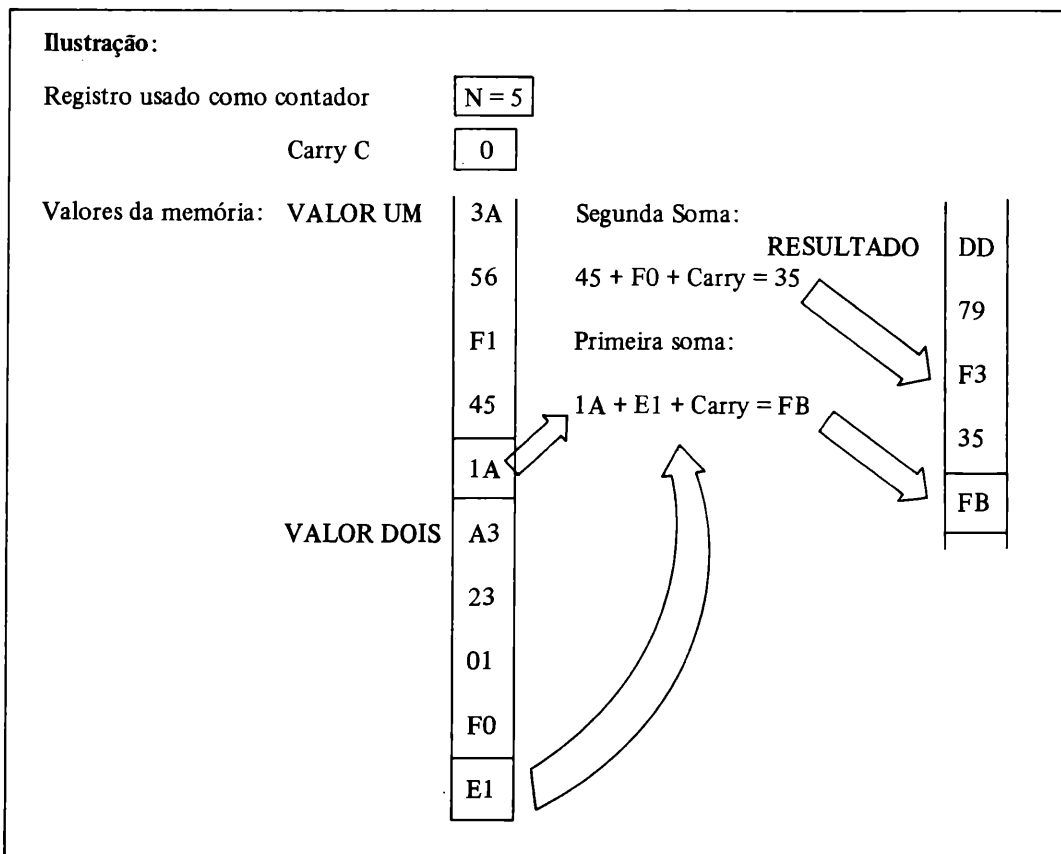
- Colocar no registro usado como contador o valor N = 5. Zerar Carry C = 0.

Repetir:

- Carregar acumulador A com o byte menos significativo do VALOR UM.
- Somar valor do acumulador com o byte menos significativo do VALOR DOIS, incluindo Carry.
- Colocar o resultado da soma na posição mais à direita (menos significativo) da área RESULT que receberá o resultado da soma.
- Decrementa o contador de valor.

Comparar:

- Verifica se contador chegou a valor zero; caso negativo, voltar para Repetir.
- Termina se contador chegou a zero.



VERSÃO PARA 6502

Código-Objeto	Assembler
2000	1 EX6515 ORG \$2000
2000 AE 20 20	2 LDX N
2003 18	3 CLC
2004 BD 10 20	4 REPET LDA VALUM-1,X
2007 7D 15 20	5 ADC VD0IS-1,X
200A 9D 1A 20	6 STA RESUL-1,X
200D CA	7 DEX
200E D0 F4	8 BNE REPET
2010 00	9 BRK
2011 3A 56	10 VALUM DBY \$3A56
2013 F1 45	11 DBY \$F145
2015 1A	12 BYT \$1A
2016 A3 23	13 VD0IS DBY \$A323

2018 01 F0	14	DBY \$01F0
201A E1	15	BYT \$E1
201B 00 00	16	RESUL DBY 0
201D 00 00	17	DBY 0
201F 00	18	BYT 0
2020 05	19	N BYT 5
2021	20	END

***** END OF ASSEMBLY

Comentários:

- 1) Início do programa em 2000.
- 2) Carrega contador (registro X) com valor N = 5.
- 3) Carry = 0.
- 4) Carrega em A, byte menos significativo do VALOR-UM:
endereço VALUM – 1 + X contém o byte \$1A.
- 5) Soma A com byte menos significativo de VALOR-DOIS:
endereço VDOIS – 1 + X contém o byte \$E1.
- 6) Resultado da soma 1A + E1 = FB vai para endereço RESUL – 1 + X.
- 7) Decrementa registro X.
- 8) Se contador X não for nulo, repete a operação com bytes seguintes.
- 9) Até contador X ser zero, quando termina.
- 10 a 12) Define na memória VALOR UM de cinco bytes.
- 13 a 15) Define na memória VALOR DOIS de cinco bytes.
- 16 a 18) Define local de cinco bytes para receber RESULTADO da soma.
- 19) Define na memória o valor N = 05.
- 20) Fim do programa.

EXERCÍCIOS

De Recapitulação

1. Através da execução manual, instrução por instrução, de um dos programas-exemplos deste capítulo, efetuar as operações abaixo mencionadas. Após a execução de cada instrução, citar o conteúdo do registro ou memória afetada pela instrução.
 - Achar a soma dos bytes: 03, 51, E1, 33, 42, 23 e A1.
 - Achar o maior valor dos bytes: 03, 51, E1, 33, 42, 23 e A1.

- Efetuar a contagem do número de caracteres do seu nome completo.
- Somar o valor de 6 bytes 03 51 E1 33 52 F1 com o valor de 6 bytes 33 21 FE DA 03 54.

De Aplicação

Escrever a versão Assembler e respectivo código-objeto dos programas abaixo mencionados. Todos os programas devem ser testados manualmente, instrução por instrução, através de um exemplo numérico, mencionando os conteúdos dos registros e memórias afetadas.

1. Subtrair sucessivamente os valores de N bytes de um único valor inicial colocado no endereço simbólico INÍCIO. Se o resultado final em INÍCIO for positivo, colocar 00 no endereço simbólico SINAL, caso contrário, colocar o valor FF.

Sugestão: modificar o programa para soma de N bytes.

2. Comparar uma série de N bytes com o valor hexadecimal 0F. Colocar os bytes acima desse valor, a partir do endereço MAIOR, e os bytes abaixo desse valor, a partir do endereço MENOR.

Sugestão: Modificar o programa para achar maior valor de uma série de N bytes. É preciso utilizar três contadores de endereços: um para ler a série de N bytes, outro para controlar os endereços dos valores maiores e um terceiro contador, para os endereços dos valores menores.

3. Contar o número de caracteres vogais (A, E, I, O, U) e caracteres consoantes do seu nome e sobrenome. Colocar os dois resultados nos endereços VOGAIS e CONSAN, respectivamente.
4. ZERAR ÁREA DE N BYTES — Zerar uma área de N bytes, a partir do endereço simbólico ÁREA.
5. Colocar o valor hexadecimal FF em cada um dos N bytes, a partir do endereço simbólico NAREA.
6. Converter em instrução Assembler e respectivo código-objeto os seguintes trechos de programas em BASIC:

a) 05 LET A = 0.0

10 FOR I = 1 TO 10 STEP 2

20 LET A = A + I

30 NEXT I

b) 10 DIM L (10)

20 ...

50 FOR I = 1 TO 10

60 IF L (I) < 0.0 THEN LET L (I) = 0.0

70 NEXT I

```
c) 10 DIM A (10, 10)
    20 ...
    50 LET SOMA = 0.0
    60 FOR I = 1 TO 10
    70 FOR J = 1 TO 10
    75 IF I <> J THEN LET SOMA = SOMA + A (I, J)
    80 NEXT J
    90 NEXT I
```

CAPÍTULO

XIX

INSTRUÇÕES DE DESLOCAMENTOS E GIROS DO 6502: SHIFT E ROTATE

XIX.1 FORMAS GERAIS DAS INSTRUÇÕES

As instruções estudadas neste capítulo são:

ASL A	— Deslocamento aritmético de um bit para a esquerda, do Acumulador A.
ASL endereço	— Deslocamento aritmético para a esquerda, do endereço direto.
ASL endereço, X	— <i>Idem</i> , do endereço indexado.
LSR A	— Deslocamento lógico de um bit para a direita do conteúdo de A.
LSR endereço	— <i>Idem</i> , do endereço direto.
LSR endereço, X	— <i>Idem</i> , do endereço indexado.
ROL A	— Gira um bit para a esquerda, o conteúdo de A, incluindo Carry.
ROL endereço	— <i>Idem</i> , o endereço direto.
ROL endereço, X	— <i>Idem</i> , o endereço indexado.
ROR A	— Gira um bit para a direita, o conteúdo de A, sem incluir Carry.
ROR endereço	— <i>Idem</i> , o endereço direto.
ROR endereço, X	— <i>Idem</i> , o endereço indexado.

PROGRAMA-EXEMPLO — O programa-exemplo analisado é:

- MULTIPLICAÇÃO BINÁRIA DE DOIS VALORES DE OITO BITS CADA.

XIX.2 ASL (ARITHMETIC SHIFT LEFT)

Efetua deslocamento de um bit, para a esquerda, do conteúdo do Acumulador A ou posição da memória. O resultado afeta o bit de sinal e o flag S. O valor do bit mais à esquerda passa para o CARRY, e o bit mais à direita torna-se zero.

Afeta flags S, Z e C.

Assembler	Código-Objeto	Explicação
Acumulador A		
ASL A	0A	— Desloca um bit para esquerda conteúdo de A.
Direto		
ASL endereço	06 qq	— Desloca um bit para esquerda o conteúdo de qq (página zero).
	0E qq pp	— Desloca um bit para esquerda o conteúdo de ppqq.
Indexado		
ASL endereço, X	16 qq	— Desloca para esquerda conteúdo de qq + X.
	1E qq pp	— Desloca para esquerda conteúdo de ppqq + X.

Exemplos

ASL	0A	— Desloca para esquerda conteúdo de A.
Carry Acumulador A Antes: 1 ← 0010 1111 ← 0 Depois: 0 0101 1110 Outros Flags: S = 0 (resultado é positivo) Z = 0 (resultado não é zero)		
ASL Z1	0E 43 F0	— Desloca para esquerda conteúdo de endereço Z1 = \$F043.
ASL \$A0, X	16 A0	— Desloca para esquerda conteúdo do endereço \$A0 + X.

XIX.3 LSR (LOGICAL SHIFT RIGHT)

Efetua deslocamento de um bit, para a direita, do conteúdo do Acumulador A ou posição da memória. O bit mais à esquerda recebe valor zero, e o bit à direita passa para o CARRY. O flag S sempre se torna igual a 0. Afeta, além disso, os flags Z e C.

Assembler	Código-Objeto	Explicação
Acumulador A		
LSR A	4A	– Desloca para a direita conteúdo de A.
Direto		
LSR endereço	46 qq	– Desloca para a direita conteúdo de qq (página zero).
	4E qq pp	– Desloca para a direita conteúdo de ppqq.
Indexado		
LSR endereço, X	56 qq	– Desloca para a direita conteúdo de qq + X.
	5E qq pp	– Desloca para a direita conteúdo de ppqq + X.

Exemplos

LSR A	4A	– Desloca para a direita conteúdo de A.						
Acumulador A Carry Antes: 0 → <table><tr><td>1101</td><td>1110</td></tr><tr><td>0110</td><td>1111</td></tr></table> → <table><tr><td>1</td></tr><tr><td>0</td></tr></table> Depois: Demais Flags: S = 0 (bit 7 de sinal torna-se zero) Z = 0 (resultado não é zero)			1101	1110	0110	1111	1	0
1101	1110							
0110	1111							
1								
0								
LSR \$F043	4E 43 F0	– Desloca para a direita conteúdo de \$F043.						
LSR Z1, X	5E 43 F0	– Desloca para a direita conteúdo de Z1 + X = \$F043 + X.						

Nota: Se existisse a instrução de deslocamento aritmético para a direita ou ARITHMETIC SHIFT RIGHT, deslocaria todos os bits para a direita, porém MANTERIA O BIT 7 COM O MESMO VALOR, não alterando, dessa forma, o sinal do valor aritmético deslocado. O deslocamento lógico para a direita (LOGICAL SHIFT RIGHT) apenas desloca o valor para a direita, sem se importar com o sinal.

XIX.4 ROL (ROTATE LEFT)

Executa giro de um bit para a esquerda de todos os bits do Acumulador A ou da posição de memória, incluindo o CARRY. O bit mais à esquerda vai para o CARRY, e o bit mais à direita recebe o bit do CARRY.

Afeta flags S, Z e C.

Assembler	Código-Objeto	Explicação
Acumulador A		
ROL A	2A	– Gira esquerda conteúdo de A.
Direto		
ROL endereço	26 qq	– Gira esquerda conteúdo de qq (página zero).
	2E qq pp	– Gira esquerda conteúdo de ppqq.
Indexado		
ROL endereço, X	36 qq	– Gira esquerda conteúdo de qq + X.
	3E qq pp	– Gira esquerda conteúdo de ppqq + X.

Exemplos

ROL A	2A	– Gira conteúdo de A para a esquerda.
Carry Acumulador A Antes: 1 ← 0101 1110 Depois: 0 1011 1101 Demais Flags: S = 1 (resultado é negativo) Z = 0 (resultado não é zero).		
ROL \$A0	26 A0	– Gira esquerda conteúdo de \$A0.
ROL Z1, X	3E 43 F0	– Gira esquerda conteúdo de Z1 + X = \$F043 + X.

XIX.5 ROR (ROTATE RIGHT)

Executa giro de um bit para a direita no conteúdo do Acumulador A ou posição da memória, passando pelo CARRY, isto é, o bit mais à esquerda recebe o valor de CARRY e CARRY recebe o valor do bit mais à direita.

Afeta flags S, Z e C.

Assembler	Código-Objeto	Explicação
Acumulador A		
ROR A	6A	– Gira direita conteúdo de A.

Direto

ROR endereço	66 qq	– Gira direita conteúdo de qq (página zero).
	6E qq pp	– Gira direita conteúdo de ppqq.

Indexado

ROR endereço, X	76 qq	– Gira direita conteúdo de qq + X.
	7E qq pp	– Gira direita conteúdo de ppqq + X.

Exemplos

ROR \$A0, X	76 A0	– Gira direita conteúdo de \$A0 + X, e CARRY.
-------------	-------	---

Registro X: 02

Carry Conteúdo de \$A0 + 02

Antes: 1 0101 1110

Depois: 0 1010 1111

Demais Flags: S = 1 (Bit 7 de sinal é um)
Z = 0 (resultado não é zero)

ROR A	6A	– Gira conteúdo de A e CARRY.
ROR Z1	6E 43 F0	– Gira conteúdo de Z1 = \$F043 e CARRY.

XIX.6 PROGRAMA-EXEMPLO:**MULTIPLICAÇÃO BINÁRIA DE DOIS VALORES DE 8 BITS CADA**

Multiplique o **MULTIPLICANDO** de 8 bits, que está no endereço simbólico **MCANDO**, com o **MULTIPLICADOR** de 8 bits, que está no endereço simbólico **MCADOR**.

O resultado **PRODUTO** deve ficar no endereço **PROD**, que deve ter 16 bits, isto é, no endereço **PROD** estará o byte menos significativo (**LSB** – Least Significant byte), e no endereço **PROD + 1** estará o byte mais significativo (**MSB** – Most Significant BYTE) do resultado.

Algoritmo:

Passo 1: Endereço que vai receber resultado **PROD** = 0000.
Contador de bits recebe valor 8.

Passo 2. $\text{PRODUTO} = \text{PRODUTO} \times 2$ (efetua deslocamento à esquerda de um bit na área de resultado).

$\text{MULTIPLICADOR} = \text{MULTIPLICADOR} \times 2$ (efetua deslocamento à esquerda de um bit).

Passo 3: Se (Carry resultante do deslocamento do MULTIPLICADOR for 1), então
 $\text{PRODUTO} = \text{PRODUTO} + \text{MULTIPLICANDO}$.

Passo 4: Decrementa contador de bits.

Se contador de bits for maior que zero, então vai para Passo 2.

Passo 5: Caso contrário, terminar.

Nota: Este algoritmo efetua a multiplicação bit a bit, partindo do bit da direita para o bit da esquerda.

Exemplo:

```

MULTIPLICANDO = 03 = 0000 0011
MULTIPLICADOR = 05 = 0000 0101 x
                    0000 0011
                    00000 110 +
PRODUTO        = 0F = 0000 1111
  
```

Ilustração:

Condição inicial: Passo 1:

CARRY	MULTIPLICADOR	MULTIPLICANDO	PRODUTO	CONTADOR
0	0000 0101	0000 0011	0000 0000 0000 0000	08

Repetir passos 2, 3 e 4 por oito vezes:

Primeira vez: Deslocamento à esquerda do PRODUTO E MULTIPLICADOR (Passos 2, 3 e 4)

0	← 0000 1010		0000 0000 0000 0000	07
---	-------------	--	---------------------	----

Segunda vez: Passos 2, 3 e 4:

0	← 0001 0100		0000 0000 0000 0000	06
---	-------------	--	---------------------	----

Permanece inalterado até que ocorra CARRY = 1:

Sexta vez: Passos 2, 3 e 4:			
1	←	0100 0000	
		0000 0000 0000 0000	02
Soma MULTIPLICANDO ao PRODUTO:		+ 0000 0011	
Novo PRODUTO:		0000 0000 0000 0011	
Sétima vez: Passos 2, 3 e 4:			
0	←	1000 0000	
		0000 0000 0000 0110	01
Oitava vez: Passos 2, 3 e 4:			
1	←	0000 0000	
		0000 0000 0000 1100	
Soma MULTIPLICANDO ao PRODUTO:		+ 0000 0011	
Novo PRODUTO: (Resultado Final)		0000 0000 0000 1111	00

VERSÃO PARA 6502

Código-Objeto	Assembler		
2000	1	EX6516	ORG \$2000
2000 A9 00	2		LDA #0
2002 8D 23 20	3		STA PROD+1
2005 A2 08	4		LDX #8
2007 0A	5	DESLO	ASL
2009 2E 23 20	6		ROL PROD+1
200B 0E 21 20	7		ASL MCADOR
200E 90 09	8		BCC VAI
2010 18	9		CLC
2011 6D 26 20	10		ADC MCANDO
2014 90 03	11		BCC VAI
2016 EE 23 20	12		INC PROD+1
2019 0A	13	VAI	DEX
201A D0 EB	14		BNE DESLO
201C 8D 22 20	15		STA PROD
201F 00	16		BRK
2020 03	17	MCANDO	BYT \$03
2021 05	18	MCADOR	BYT \$05
2022 00	19	PROD	BYT 0
2023 00	20		BYT 0
2024	21		END

***** END OF ASSEMBLY

Comentários (6502):

- 1) Início do programa será em 2000.
- 2) Acumulador zerado vai ser usado como LSB do PRODUTO.
- 3) Também coloca zero no MSB do PRODUTO (endereço PROD + 1).
- 4) Registro X usado como contador recebe valor 08.
- 5) Desloca acumulador (LSB do PRODUTO) para esquerda.
- 6) MSB do PRODUTO (PROD + 1) também entra no deslocamento, junto com Carry resultante de 5.
- 7) Desloca MULTIPLICADOR (endereço MCADOR) um bit para a esquerda.
- 8) Se Carry = 0, vai para endereço VAI nada executando.
- 9) Se Carry = 1, apaga esse Carry para não ser usado em 10 e 11.
- 10) Soma: PRODUTO (LSB) + MULTIPLICANDO (endereço MCAND).
- 11) Se Carry = 0, nada faz.
- 12) Se Carry = 1, acrescenta ao MSB de PRODUTO (endereço PROD + 1).
- 13) Decrementa contador X.
- 14) Se contador maior que zero, repete operação e vai para endereço DESLO.
- 15) Se contador for igual a zero, então LSB do PRODUTO que está em A vai para endereço PROD.
- 16) Pare.
- 17) Define na memória valor do MULTIPLICANDO = 03.
- 18) Define na memória valor do MULTIPLICADOR = 05.
- 19) Define na memória local de 2 bytes (endereços PROD e PROD + 1) para receber PRODUTO.
- 20) Define posição correspondente a PROD + 1.
- 21) Fim de programa.

EXERCÍCIOS**De Recapitulação**

1. Descrever e citar exemplos de cada uma das formas possíveis das seguintes instruções:
 - ARITHMETIC SHIFTS.
 - LOGICAL SHIFTS.
 - ROTATES.

De Aplicação

Escrever a versão Assembler e respectivo código-objeto dos programas abaixo mencionados. Todos os programas devem ser testados manualmente, instrução por instrução, através de um exemplo numérico, mencionando os conteúdos dos registros e memórias afetadas.

1. Multiplicação binária de um valor de 16 bits por outro de 8 bits.
2. Multiplicação binária de um valor de 16 bits por outro de 16 bits.
3. **CONVERSÃO DE ALGARISMO HEXADECIMAL PARA DECIMAL** – Converter um algarismo hexadecimal (de 00 a 0F) que está no registro ou acumulador A para o valor decimal correspondente (00 a 15), colocando os dois algarismos decimais resultantes nos endereços DEZENA e UNIDAD, respectivamente.
4. **CONVERSÃO DE VALOR BINÁRIO DE 8 BITS PARA DECIMAL** – Converter um valor binário de 8 bits (um byte da memória) em valor decimal correspondente, colocando os algarismos decimais nos endereços CENTENA, DEZENA e UNIDAD, respectivamente.

CAPÍTULO

XX

INSTRUÇÕES DE DESVIOS (JSR) PARA SUB-ROTINA E RETORNO DE SUB-ROTINA (RTS) PARA 6502

XX.1 INTRODUÇÃO

Neste capítulo estudamos o uso dos comandos de desvios e retornos de uma sub-rotina definida pelo próprio programador.

A *sub-rotina* ou subprograma é um recurso bastante utilizado em qualquer tipo de programação, pois permite ao programa principal, efetuar operações longas repetidas vezes, sem necessidade de codificar essas operações em cada momento de seu uso.

Basta programar essas operações como SUB-ROTINA uma única vez e usá-las quantas vezes forem necessárias, através do comando de chamada do tipo JSR (JUMP TO SUBROUTINE).

Em linguagem de alto nível, a sub-rotina aparece sob as seguintes formas:

<i>BASIC</i>	<i>FORTRAN</i>	<i>PASCAL</i>
Desvio para sub-rotina:		
GOSUB 100	CALL SUB-X	SUB-X;
Sub-rotina a ser executada:		
100. . . (comandos)	SUBROUTINE SUB-X (comandos)	PROCEDURE SUB-X; BEGIN (comandos)
200 RETURN	RETURN	END;

No Assembler 6502, o mesmo mecanismo é utilizado através das instruções:

JSR — JUMP TO SUBROUTINE

RTS — RETURN FROM A SUBROUTINE

XX.2. JUMP TO SUBROUTINE E RETURN

JSR (Jump to Subroutine) – Desvia para o endereço de uma sub-rotina salvando, antes do desvio, nas duas posições do topo da pilha, o valor corrente do PC acrescido de 2, ou seja, apontando para o terceiro byte da própria instrução JSR.

O endereço salvo na pilha (valor da PC mais dois) será usado como endereço de retorno após a execução da sub-rotina. O ponteiro da pilha fica ajustado automaticamente à próxima posição vaga na pilha.

RTS (Return from Subroutine) – Ao encontrar esta instrução, a UCP retira das duas posições do topo da pilha o endereço de retorno que foi salvo por uma instrução JSR.

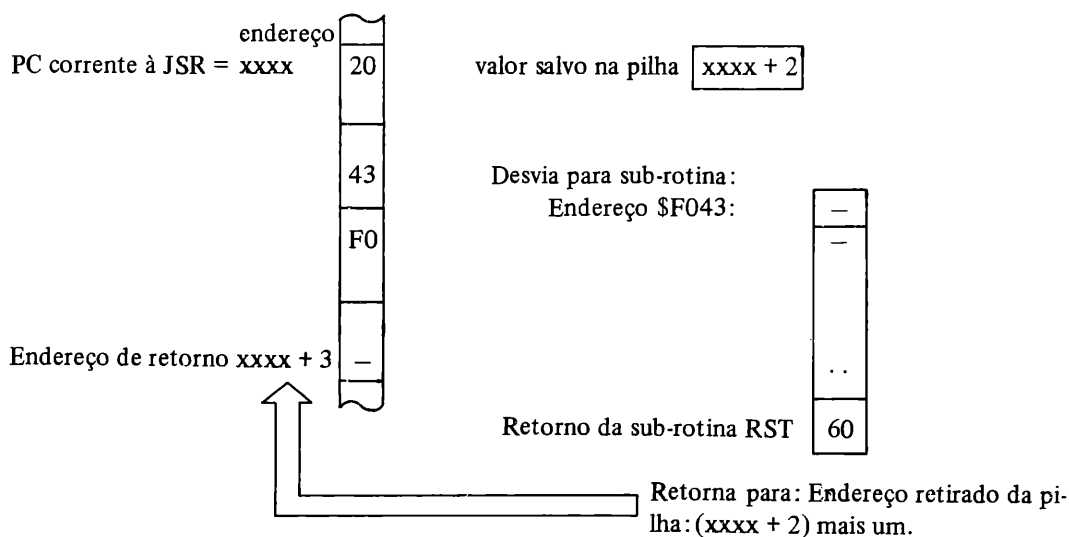
Esse endereço será acrescido do valor 1 (um) para apontar a posição seguinte ao da instrução JSR. Com isso, o retorno da sub-rotina é efetuado automaticamente. O ponteiro da pilha é ajustado automaticamente à nova posição vaga.

JSR endereço 20 qq pp – Desvia para sub-rotina que está no endereço ppqq.

RTS 60 – Retorno da sub-rotina, prosseguindo na instrução seguinte à última instrução JSR ocorrida.

Exemplos:

JSR \$F043 20 43 F0 – Desvia para executar a sub-rotina que começa no endereço \$F043.



XX.3 PROGRAMA-EXEMPLO:**USO DE SUB-ROTINA**

Obter, através de uma sub-rotina, o número de caracteres de um string ou série de caracteres de um byte cada, e que estão na memória a partir do endereço STRIN e em notação ASCII. O último caractere do string é o caractere CARRIAGE RETURN ou CR (0D em hexadecimal) que marca o fim da série.

O número de caracteres assim obtido deve ser colocado no endereço COMPR.

A sub-rotina que calcula o número de caracteres será chamada RSTRI.

Algoritmo:**PROGRAMA PRINCIPAL:**

- Começar o programa a partir do endereço desejado, por exemplo 2000.
- Definir, na memória, uma nova pilha ou “stack” para receber o endereço de retorno, que será necessário, após a execução da sub-rotina. Sem esta nova pilha, será usada a pilha definida pelo monitor do próprio microprocessador.
- Chamar a sub-rotina RSTRI para obter o número de caracteres da série.
- Colocar o número de caracteres obtido, e que está no acumulador, no endereço COMPR.
- Terminar.

SUB-ROTINA RSTRI (conta o número de caracteres do string ou série):

- Coloca no acumulador o caractere CR ou 0D em hexadecimal.
- Repetir: Compara acumulador com um caractere do string.
Se houver igualdade (achou marcador de fim CR) vai para Retornar.
Caso contrário, incrementa contador de caracteres e obtém endereço do próximo caractere.
– Vai para Repetir.
- Retornar: Coloca valor do contador de caracteres no acumulador.
– Retorno da sub-rotina.

Ilustração:**PROGRAMA PRINCIPAL**

Chama sub-rotina RSTRI.

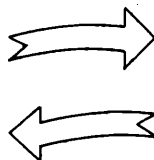
Coloca no endereço COMPR o número de caracteres obtido.

Pare.

SUB-ROTINA

RSTRI: Calcula e coloca no acumulador o número de caracteres do string.

Retorna para Programa Principal.



VERSÃO PARA 6502

```

0800      1  ; USO DE SUBROTINA
0800      2  ;
2000      3  EX6520  ORG $2000
2000 A2 FF  4  PRINCI  LDX #$FF
2002 9A     5          TXS
2003 20 50 20 6      JSR RSTRI
2006 8D A0 20 7      STA COMPR
2009 00     8          BRK
200A      9  ;
200A     10 ;SUBROTINA:CONTA CARACTERES
200A     11 ;      DE UM STRING
2050     12      ORG $2050
2050 A2 FF  13  RSTRI  LDX #$FF
2052 A9 0D  14          LDA #$0D
2054 E8     15  REPET  INX
2055 DD A1 20 16          CMP STRIN,X
2058 D0 FA  17          BNE REPET
205A 8A     18          TXA
205B 60     19          RTS
205C     20 ;
205C     21 ;STRING COMEÇA NO ENDEREÇO
205C     22 ;      $20A0
20A0     23      ORG $20A0
20A0 00     24  COMPR  BYT 0
20A1 52     25  STRIN  BYT 'R'
20A2 41     26          BYT 'A'
20A3 54     27          BYT 'T'
20A4 4F     28          BYT 'O'
20A5 0D     29          BYT $0D
20A6     30          END

```

***** END OF ASSEMBLY

Comentários (6502)

PROGRAMA PRINCIPAL: Começa no endereço 2000.

- 3) Define PC = 2000 para programa começar em 2000.
- 4) Define nova pilha a partir do endereço 00FF (*).
- 5) Colocando valor 00FF no Stack Pointer SP.
- 6) Desvia para sub-rotina RSTRI.
- 7) Após execução da sub-rotina, coloca tamanho do string no endereço COMPR.

8) Pare.

SUB-ROTINA RSTRI: Começa no endereço \$2050.

- 12) Define PC = 2050 para programa começar em \$2050.
- 13) Contador de caracteres registro X começa com valor FF = -1.
- 14) Coloca no acumulador o caractere \$0D = CR, marcador de fim do string.
- 15) Incrementa contador X.
- 16) Compara acumulador com um caractere do string; endereço STRIN + X aponta primeiro caractere.
- 17) Se comparação for diferente, vai para REPET.
- 18) Se comparação for igual (chegou ao último caractere), coloca valor contado que está em X no acumulador.
- 19) Retorna para Programa Principal.

STRING: Começa no endereço \$20A0.

- 23) Define PC = \$20A0 para colocar dados a partir do endereço \$20A0.
- 24) Define na memória o endereço COMPR para receber o comprimento do string.
- 25) Define na memória, a partir do endereço STRIN, o primeiro caractere.
- 26) Define segundo caractere.
- 27) Define terceiro caractere.
- 28) Define quarto caractere.
- 29) Define caractere CR = \$0D como marcador de fim de string.
- 30) Fim do programa.

Nota (*) Caso não fosse definida uma nova pilha do usuário, o programa usaria uma pilha definida pelo programa monitor do 6502.

EXERCÍCIOS

De Recapitulação

1. Executar manualmente o programa-exemplo USO DA SUB-ROTINA, descrevendo o conteúdo dos registros e locais de memórias afetadas, após a execução de cada instrução.

De Aplicação

1. Organizar em forma de sub-rotina, a ser utilizada através de um programa-principal, os seguintes programas-exemplos já vistos:

- ZERAGEM DE UMA ÁREA DE K BYTES.
- SOMA DE DOIS VALORES DE 16 BITS CADA.
- SOMA DE N VALORES DE UM BYTE CADA.
- SOMA DE DOIS VALORES DE MÚLTIPLA PRECISÃO.
- MULTIPLICAÇÃO BINÁRIA DE VALORES DE 8 BITS.

Escrever também o programa principal que utilize cada uma dessas sub-rotinas, através de uma série de instruções do tipo JSR sub-rotina.

CAPÍTULO

XXI

INSTRUÇÕES COM FLAGS, PILHAS E INTERRUPÇÕES DO 6502

XXI.1 INTRODUÇÃO

As instruções examinadas neste capítulo são:

- CLEAR STATUS FLAGS** — Limpa os flags colocando valor zero.
SET STATUS FLAGS — Ativa os flags colocando valor um.

Operação com **PILHAS**

OPERAÇÕES COM INTERRUPÇÕES

- RTI** — Retorno de interrupção.

PARADA E NENHUMA OPERAÇÃO

BRK e NOP

OPERAÇÕES DE ENTRADA/SAÍDA

- São executadas através de endereços fixos da memória e dependem do hardware.

As instruções acima, principalmente as que manipulam **INTERRUPÇÕES** e **ENTRADA/SAÍDA** de dados, dependem sempre da versão implementada do hardware (UCP, sinais, portos, terminais). Sendo assim, para usar as instruções deste capítulo, uma noção mais detalhada do hardware do microprocessador 6502 é necessária.

O programa exemplo denominado:

- **LEITURA E ESCRITA DE UM DADO ATRAVÉS DA TTY (TELETYPEWRITER) OU
TERMINAL DE TV.**

ilustra o uso dessas instruções.

XXI.2 OPERAÇÃO COM FLAGS DE STATUS

CLEAR STATUS FLAG

CLC	18	– Clear Carry (Flag C = 0).
CLD	D8	– Clear Decimal Mode (Flag D = 0).
CLI	58	– Clear interrupt mask or enable interrupt (Flag I = 0).
CLV	B8	– Clear overflow (Flag V = 0).

SET STATUS FLAG

SEC	38	– Set Carry (Flag C = 1).
SED	F8	– Set Decimal Mode (Flag D = 0).
SEI	78	– Set interrupt mask or disable interrupts (Flag I = 1).

XXI.3 OPERAÇÃO COM PILHA (STACK)

PHA	48	– Push AC onto stack – Coloque AC no topo da pilha.
PHP	08	– Push Status Register P onto stack – Coloque registro P no topo da pilha.
PLA	68	– Pull AC from stack – Carregue AC do topo da pilha.
PLP	28	– Pull Status Register P from stack – Carregue registro P do topo da pilha.

XXI.4 RETORNO DA INTERRUPÇÃO

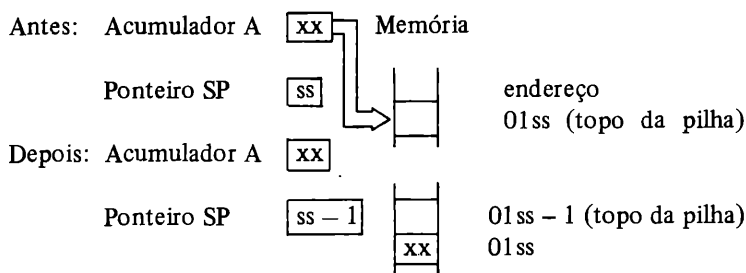
RTI	40	– Retorna de um estado de interrupção, apanhando a instrução a ser executada a seguir, da pilha.
BRK	00	– Force Break – interrompe operação.
NOP	EA	– No operation – nada executa.

XXI.5 ENTRADAS E SAÍDAS DE DADOS

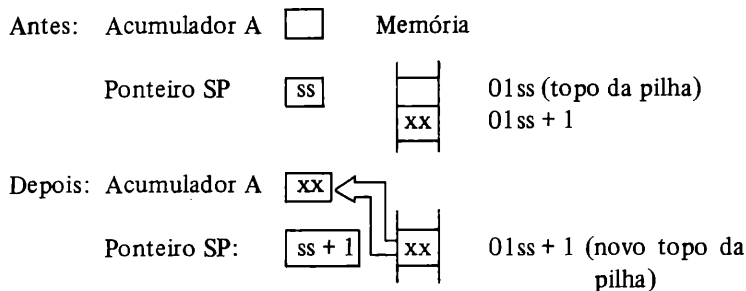
São executadas por endereços da memória diretamente conectados aos portos de saída ou entrada do microprocessador 6502 – Consultar o Manual de Hardware 6502 e o programa-exemplo do capítulo seguinte.

Exemplos**Explicação**

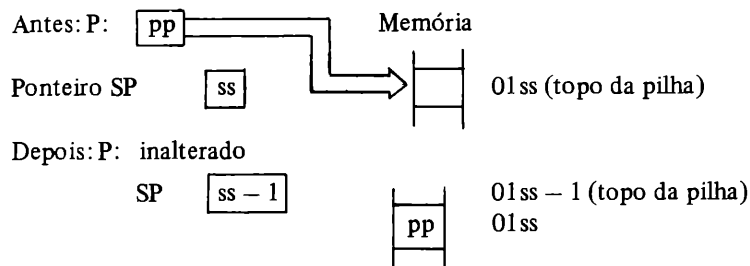
- PHA** 48 — Conteúdo do Acumulador A é colocado no topo da pilha, após o que o ponteiro da pilha SP é decrementado de um. Não afeta outros flags.



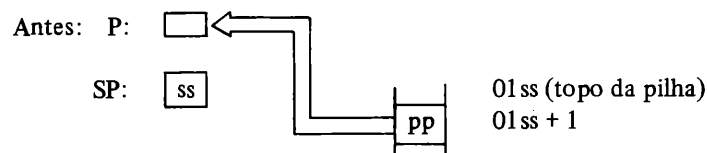
- PLA** 68 — Incrementa ponteiro da pilha SP de um e traz o conteúdo do topo da pilha para A.



- PHP** 08 — Conteúdo do Registro de Status P (que contém os flags S, V, B, D, I, Z e C) é colocado no topo da pilha, após o que o ponteiro da pilha SP é decrementado de um. Não afeta outros flags.



- PLP** 28 — Operação inversa de PHP.



Depois P: pp 01ss

SP: ss + 1 01ss + 1 (topo da pilha)

Nota: Endereço 01ss da Pilha ou Stack: O ponteiro da pilha SP (STACK POINTER) contém um endereço que varia de 00 a FF. A UCP antepõe automaticamente a esse valor o byte "01", formando o endereço efetivo da pilha na posição 01ss da memória.

EXERCÍCIOS

Ver programa-exemplo do próximo capítulo:

- LEITURA E ESCRITA DE DADOS PELA MÁQUINA TTY (TELETYPEWRITER) OU TERMINAL DE TV.

CAPÍTULO

XXII

EXEMPLO DE ASSEMBLER: LEITURA E ESCRITA DE CARACTERES ATRAVÉS DE UMA MÁQUINA TTY (TELETYPEWRITER) (*) OU TERMINAL DE TV

PROBLEMA – Ler e escrever caracteres em código ASCII através de uma máquina de escrever serial tipo TTY com velocidade de 10 caracteres por segundo.

XXII.1 CONSIDERAÇÕES GERAIS SOBRE OPERAÇÃO DE ENTRADA/SAÍDA

Configuração de bits por caractere

A transferência de dados (entrada ou saída) em uma TTY comum é efetuada do seguinte modo:

1. Escolhe-se o estado Entrada ou Saída da linha de transmissão de dados.
2. Cada caractere vem precedido por um bit zero de PARTIDA.
3. Cada caractere está codificado em código ASCII de sete bits cada, e a transmissão de bits é efetuada do bit menos significativo para o mais significativo.
4. O bit seguinte ao do caractere, isto é, o nono bit, é o bit de PARIDADE, que indica número par (ou um) ou ímpar (ou zero) de bits.
5. Dois bits de PARADA, iguais a um, seguem cada caractere.

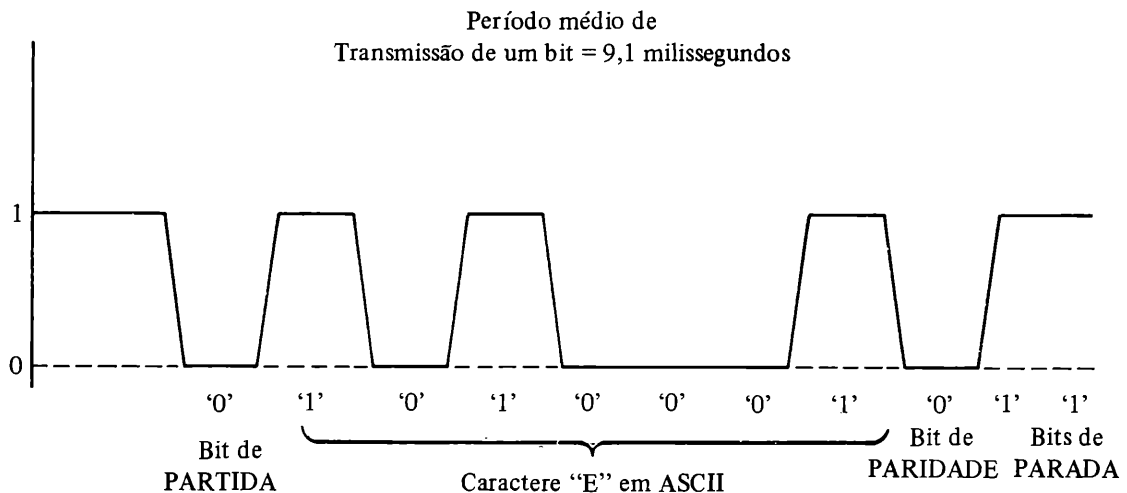
Temos, portanto, um total de 11 bits transmitidos por caractere.

Velocidade de transmissão de cada bit

Como a TTY opera à velocidade de 10 caracteres por segundo, e cada caractere é formado por 11 bits, a velocidade de transmissão de cada bit é de 10×11 ou 110 bauds (bits por segundo).

(*) Este programa foi adaptado da referência 2 com permissão da Editora MCGRAW-HILL DO BRASIL LTDA.

Cada bit gasta, portanto, em média, um intervalo de tempo (ou atraso) de 1/110 segundos ou 9,1 milissegundos, que deve ser levado em consideração por ocasião da transmissão.



Algoritmo de leitura de caracteres pela TTY

Com base nos fatos descritos, podemos elaborar o seguinte algoritmo:

- Passo 1: Procurar o BIT DE PARTIDA de valor zero ao longo da transmissão.
- Passo 2: Aguardar um intervalo de 4,55 ms (meio atraso) para centralizar a posição de transmissão ao longo da linha.
- Passo 3: Receber cada bit do caractere, aguardando 9,1 ms (um atraso de transmissão) entre cada recepção.
- Passo 4: Montar os bits recebidos em uma palavra do acumulador, passando pelo bit de CARRY.
- Passo 5: Calcular o valor do bit de paridade dos bits recebidos e comparar com o BIT DE PARIDADE. Se não coincidir, indicar "ERRO DE PARIDADE".
- Passo 6: Receber os dois BITS DE PARADA, e se ambos não forem bit um, indicar "ERRO DE ARMAÇÃO" (Frame error).

O algoritmo de escrita de um caractere é análogo ao de leitura.

ENTRADA/SAÍDA DE DADOS PELA 6502 ATRAVÉS DA 6522 – VIA (VERSATILE INTERFACE ADAPTER)

O microprocessador 6502 não possui "linhas" especiais de Entrada/Saída.

As operações de Entrada/Saída são efetuadas através das “linhas” disponíveis em um dos seguintes dispositivos:

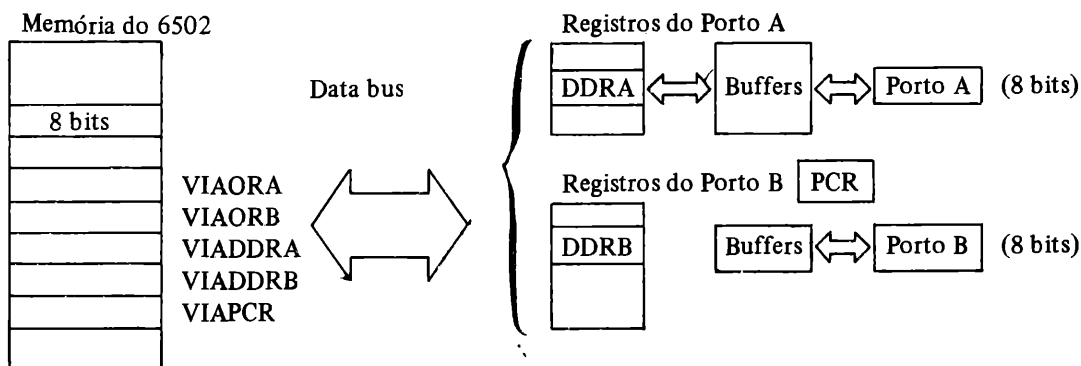
- 6502 PERIPHERAL INTERFACE ADAPTER ou 6502 – PIA.
- 6522 VERSATILE INTERFACE ADAPTER ou 6522 – VIA.
- 6532 PERIPHERAL INTERFACE/MEMORY.

Os elementos da 6522 – VIA usados na nossa operação de Entrada/Saída são:

- Dois PORTOS de Entrada/Saída de 8 bits cada, chamados porto A e porto B e usados tanto para entrada como saída de dados. Os valores desses portos serão colocados ou lidos (pelo hardware) nos endereços VIAORA e VIAORB da memória do 6502.
- Dois REGISTROS DE DIREÇÃO DE DADOS: O DATA DIRECTION REGISTER A ou DDRA e o DATA DIRECTION REGISTER B ou DDRB, que indicam se o sinal do porto A e B são de saída (bit 1) ou de entrada (bit zero) de dados.
- Registro de CONTROLE PERIFÉRICO: o PERIPHERAL CONTROL REGISTER ou PCR, que determina o estado (de entrada ou de saída) de cada linha de controle da VIA.

Em resumo, utilizaremos os seguintes elementos:

6522 – VIA	MEMÓRIA do 6502
Porto A	que está ligado ao endereço simbólico VIAORA da memória
Porto B	que está ligado ao endereço simbólico VIAORB da memória
Registro DDRA	que está ligado ao endereço simbólico VIADDRA da memória
Registro DDRB	que está ligado ao endereço simbólico VIADDRB da memória
Registro PCR	que está ligado ao endereço simbólico VIAPCR da memória



Os endereços efetivos (absolutos) de VIAORA, VIAORB etc. dependem do hardware implementado, mas podem ser alterados através do comando ou instrução diretiva do tipo EQUATE (= em Assembler do 6502).

LEITURA DE DADO PELO 6502

Ler um caractere ASCII enviado pela TTY através do BIT SETE do PORTO A da 6522 – VIA e colocar o caractere lido no endereço CHRIN da memória.

Erro de paridade e de “frame” (bits de parada) não serão detectados, pois necessitam de rotina à parte.

ROTINA DE LEITURA DE DADOS PARA 6502

```

2000          1 EX6517  ORG $2000
2000 A9 00     2 LER      LDA #0
2002 80 00 A0   3          STA VIAPCR
2005 80 01 A0   4          STA VIADDR
2008 AD 03 A0   5 ESPERA  LDA VIAORA
200B 30 FB     6          BMI ESPERA
200D 20 30 20   7          JSR MATRA
2010 A9 80     8          LDA #$80
2012 20 34 20   9 LEBIT   JSR ATRASO
2015 2E 03 A0  10          ROL VIAORA
2018 6A        11          ROR
2019 90 F7     12          BCC LEBIT
201B 8D 21 00  13          STA CHRIN
201E 60        14          RTS
0021          15 CHRIN   EQU $21
A000          16 VIAPCR  EQU $A000
A001          17 VIADDR  EQU $A001
A003          18 VIAORA  EQU $A003
2030          19 MATRA   EQU $2030
2034          20 ATRASO  EQU $2034
201F          21          END

```

***** END OF ASSEMBLY

ROTINA DE ATRASO E MEIO ATRASO PARA 6502

```

2030          1 EX6518  ORG $2030
2030 A0 05     2 MATRA   LDY #5
2032 D0 02     3          BNE ATR1
2034 A0 0A     4 ATRASO  LDY #10

```

2036 A2 B4	5	ATR1	LDX #5B4
2038 CA	6	CICLO	DEX
2039 D0 FD	7		BNE CICLO
203B 88	8		DEY
203C D0 F8	9		BNE ATR1
203E 60	10		RTS
203F	11		END

***** END OF ASSEMBLY

Comentários — Leitura de dados

- 1) Início do programa em 2000.
- 2) Coloca zero no acumulador para deixar.
- 3) Bits do PCR no estado INPUT.
- 4) Registro DDRA = 0, Porto A é INPUT.
- 5) Espera bit de partida no porto A.
- 6) Verificando valor negativo (bit 7).
- 7) Achou partida, aguarda 4,5 ms, referente a meio atraso.
- 8) Bit de preparação em AC = 80.
- 9) Espera um atraso de 9,1 ms.
- 10) Bit lido (X) (7 de VIAORA) para CARRY.
- 11) Bit lido (X) para esquerda de AC.
- 12) Bit contagem chegou a CARRY?
- 13) Sim, caractere lido vai para CHRIN.
- 14) Retorna.
- 15 a 20) Definição de endereços.

Comentários — Rotina de atraso e meio atraso

- 1) Começa no endereço \$2030.
- 2) Contador de meio atraso.
- 3) Vai para contagem do atraso em ATR1.
- 4) Contador de um atraso.
- 5) Contador de ciclo.
- 6) Decrementa contador de ciclo.
- 7) Até completar um ciclo.

- 8) Decrementa contador de atraso.
- 9) Até ficar zero.
- 10) Retorna.

Nota: Os valores 5, 10 e \$B4 dos contadores de atraso, meio atraso e do ciclo estão relacionados ao tempo de execução da instrução DEX e BNE para completar 4,5 ms ou 9,1 ms.

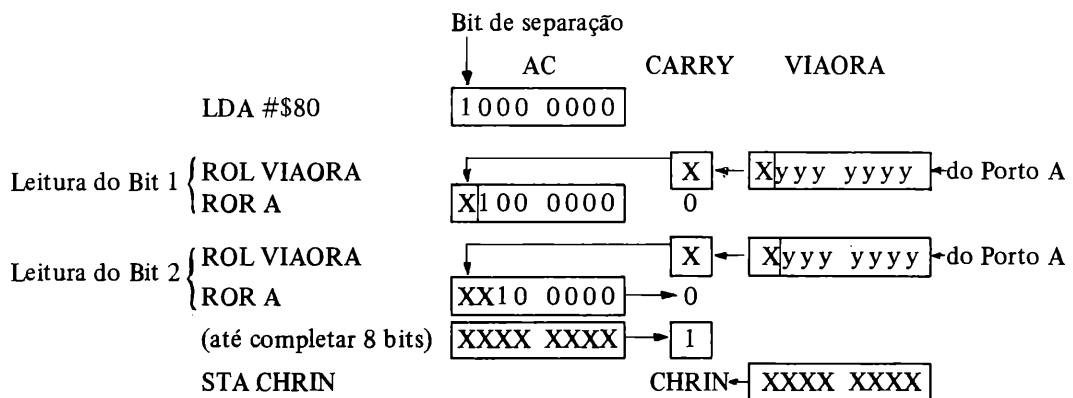
ENDEREÇOS UTILIZADOS

Conteúdo:

CHROUT = \$20	; Caractere ASCII a ser escrito.
CHRIN = \$21	; Caractere ASCII lido.
VIAPCR = \$A000	; Valor do PCR.
VIADDRA = \$A001	; Valor do DDRA.
VIADDRB = \$A002	; Valor do DDRB.
VIAORA = \$A003	; Valor do dado no porto A.
VIAORB = \$A004	; Valor do dado no porto B.

Nota: Os endereços acima são apenas para exemplificação, pois dependem da implantação que foi efetuada no Hardware. Consultar o manual do seu microcomputador.

Ilustração:



ESCREVER DADO NA TTY

Escrever pela TTY um caractere ASCII enviado através do BIT ZERO (BIT da posição zero) do PORTO B da 6522 – VIA. Cada caractere a ser escrito está colocado no endereço CHROUT da memória do 6502. A paridade dos bits transmitidos não será calculada durante a transmissão, o que requer uma outra rotina.

PROGRAMA PARA ESCREVER UM DADO EM 6502

2040	1	EX6519	ORG \$2040
A000	2	VIAPCR	EQU \$A000
A004	3	VIAORB	EQU \$A004
A002	4	VIADDRB	EQU \$A002
0020	5	CHROUT	EQU \$20
2034	6	ATRASO	EQU \$2034
2040 A9 00	7	ESCREV	LDA #0
2042 BD 00 A0	8		STA VIAPCR
2045 BD 04 A0	9		STA VIAORB
2048 A9 FF	10		LDA #\$FF
204A BD 02 A0	11		STA VIADDRB
204D AD 20 00	12		LDA CHROUT
2050 A2 08	13		LDX #11
2052 20 34 20	14	ESBIT	JSR ATRASO
2055 38	15		SEC
2056 6A	16		ROR
2057 2E 04 A0	17		ROL VIAORB
205A CA	18		DEX
205B D0 F5	19		BNE ESBIT
205D 60	20		RTS
205E	21		END

***** END OF ASSEMBLY

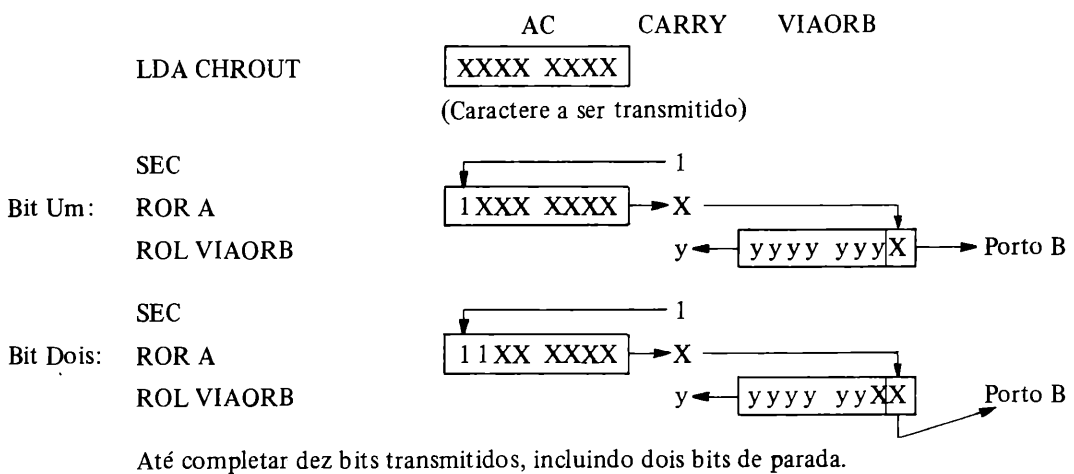
Comentários – Escrever um dado

- 1) Rotina começa no endereço 2040.
- 2 até 6) Definição de endereços, Comuns à leitura, escrita atraso e meio atraso.
- 7) Zerar acumulador para.
- 8) PCR ficar no estado de OUTPUT.
- 9) Também bit de partida zero é colocado no porto B.
- 10) Coloca \$FF em DDRB,
- 11) tornando porto B no estado OUTPUT.
- 12) Recebe caractere a ser escrito no acumulador.
- 13) Contador de bits a transmitir é onze.
- 14) Aguarda atraso de 9,1 ms, rotina ATRASO.
- 15) Carry = 1 serve de bit de parada ou fim de caractere para acumulador.
- 16) Gira um bit e Carry está com um bit do caractere a ser escrito.

- 17) Esse bit vai para o bit da posição zero do porto B.
- 18) Decrementa contador de bits.
- 19) Se contador não nulo, repete para próximo bit.
- 20) Se completou 10 bits então terminou.
- 21) Retorno da sub-rotina.

Nota: Na posição mais à esquerda do caractere está o BIT DE PARIDADE que deve ser confrontado com a contagem da paridade efetuada durante a transmissão.

Ilustração:



EXERCÍCIOS

De Recapitulação

1. Executar, manualmente, instrução por instrução, os programas-exemplos de LEITURA E ESCRITA DE DADOS através dos seguintes dados:
 - LER PELA MÁQUINA DE ESCREVER OS CARACTERES 1, 7, A e F.
 - ESCREVER PELA MÁQUINA DE ESCREVER OS CARACTERES 1, A, 7 e F.

PARTE

IV

APÊNDICES

APÊNDICE

A

O SISTEMA BINÁRIO DE REPRESENTAÇÃO DE DADOS

- SISTEMA BINÁRIO E ELEMENTOS BIESTÁVEIS.
- BIT ZERO E BIT UM.
- REPRESENTAÇÃO BINÁRIA: SISTEMA OCTAL, HEXADECIMAL.
- OPERAÇÃO COM NÚMEROS BINÁRIOS.
- REPRESENTAÇÃO DE NÚMEROS EM PONTO FLUTUANTE.
- REPRESENTAÇÃO DE DADOS NÃO-NUMÉRICOS: NOTAÇÃO ASCII, EBCDIC E BCD.
- OPERAÇÕES DE CONVERSÃO.
- ADIÇÃO E SUBTRAÇÃO EM HEXADECIMAL.

1. SISTEMA BINÁRIO E ELEMENTOS BIESTÁVEIS

A representação de dados (para leitura, gravação, cálculo aritmético, transmissão etc.) através de equipamentos quase sempre está baseada em códigos ou sinais binários, isto é, combinação de sinais que assumem apenas dois valores possíveis. Isto porque muitos equipamentos e meios físicos (como eletricidade, magnetismo) estão mais bem capacitados para representar valores de dois estados e são chamados *biestáveis*.

Exemplos de fenômenos de dois estados são:

- existe ou não existe marca ou perfuração em certa posição de um cartão;
- carga positiva ou negativa de eletricidade;
- luz acesa ou apagada;
- chave ligada ou desligada;
- porta (ou janela) aberta ou fechada;
- fita magnetizada ou não-magnetizada.

2. BIT ZERO E BIT UM

Cada fenômeno biestável ou binário é representado, por convenção, pelos valores ou bits: *um* (para presença de sinal, estado ligado) e *zero* (para ausência de sinal, estado desligado). BIT corresponde à abreviação BINARY DIGIT.

2.1 Portas lógicas (GATES, AND, OR, NOT)

A base do *hardware* de computador e qualquer tipo de circuitos eletrônicos de representação binária (também chamados circuitos digitais) são os componentes denominados *portas lógicas* ou *gates*.

As portas lógicas são fabricadas de pequenos transistores.

Toda porta recebe ou não recebe, em cada fio de entrada, corrente elétrica de baixa tensão (*bit* 1 ou 0). Fornece no fio de saída corrente elétrica (*bit* 1), dependendo das correntes dos fios de entrada e do tipo da porta.

Exemplos:

A – Porta “E” ou “AND”

Fornece corrente elétrica na sua saída (e dizemos que a saída $Z = 1$) somente se chegar corrente elétrica nas suas duas entradas (isto é, $A = 1$ e $B = 1$).

B – Porta “OU” ou “OR”

Fornece corrente na saída ($Z = 1$) se uma ou ambas as entradas receberem corrente.

C – Porta “OU EXCLUSIVO” ou “XOR” ou “EOR”

Fornece corrente na saída ($Z = 1$) se receber corrente em apenas uma das entradas.

D – Porta inversora ou “NOT”

A saída Z é sempre contrária à entrada ocorrida em A , isto é, $Z = 1$ se $A = 0$, e $Z = 0$ se $A = 1$.

2.2 Álgebra de Boole

Circuitos eletrônicos formados por portas lógicas manipulam valores *zero* e *um* e são denominados *circuitos binários* ou *circuitos lógicos digitais*.

Os circuitos lógicos digitais podem ser estudados matematicamente por uma teoria chamada Álgebra de Boole, que manipula somente valores *zero* e *um*.

2.3 Circuitos integrados (CI)

Circuitos lógicos que contêm portas AND, OR, NOT etc. tornam-se mais complexos à medida que se sofisticam sua operação. Por exemplo, é necessário construir circuitos lógicos para:

- a Unidade Central de Processamento do computador;
- a memória;
- as unidades de Entrada e Saída;
- os relógios digitais etc.

Dessa forma, circuitos lógicos que contêm grande número de portas lógicas não são interligados por fios, mas são integrados em uma única peça ou pastilha (*chip*) feita de material especial que forma o circuito integrado...

3. REPRESENTAÇÃO BINÁRIA: SISTEMA OCTAL, HEXADECIMAL

3.1 Representação binária por um bit e dois bits

São os casos mais simples de representação binária. Como já vimos, cada *bit* assume o valor *zero* ou *um* e pode ser representado, fisicamente, por uma lâmpada, um pulso eletrônico, uma perfuração de cartão, posição magnetizável da memória, fita, disco etc.

Representação binária por dois *bits* fornecem quatro (2^2) configurações possíveis.

3.2 Representação binária por três bits: sistema octal

Neste caso, já é possível formar um sistema de numeração real, pois os três *bits* representam oito (2^3) configurações diferentes que podem ser numeradas de 0 a 7.

Temos então o sistema octal que é bastante utilizado em computadores.

O sistema octal possui somente oito algarismos possíveis (0 a 7) e qualquer outro valor numérico deve ser representado por combinações desses algarismos ou por combinações de grupos de três *bits*.

No sistema octal, portanto, *não existem* os algarismos 8 e 9.

3.3 Sistema decimal versus octal

O sistema de numeração decimal utiliza dez algarismos (0 a 9) distintos e qualquer valor numérico deve ser representado por combinações desses dez algarismos. No sistema octal, após o valor 7, é preciso usar *dois* algarismos para representar os valores decimais 8, 9, 10, 11 etc.

Exemplo:**TABELA DE VALORES DECIMAIS VERSUS OCTAL**

Sistema decimal	Sistema octal	
	Em bits	Em algarismo octal
0	000	0
1	001	1
2	010	2
3	011	3
4	100	4
5	101	5
6	110	6
7	111	7
8	001 000	10
9	001 001	11
10	001 010	12
11	001 011	13
12	001 100	14
13	001 101	15
14	001 110	16
15	001 111	17
16	010 000	20
17	010 001	21
18	010 010	22
19	010 011	23
20	010 100	24
(etc.)	(etc.)	(etc.)

3.4 Representação binária por quatro bits: sistema hexadecimal

Seguindo sempre o mesmo raciocínio e utilizando-se *quatro bits*, é possível obter 16 ($2^4 = 16$) representações ou configurações distintas que correspondem aos 16 algarismos do *sistema hexadecimal*: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 ou letra A, 11 ou B, 12 ou C, 13 ou D, 14 ou E e 15 ou F.

Devemos manter a *convenção* de que cada algarismo de um sistema de numeração deve ser representado por um único símbolo. Então, no sistema hexadecimal:

- o valor decimal 10 é representado pela letra A (convenção);
- o valor decimal 11 é representado pela letra B;
- o valor decimal 12 é representado pela letra C;
- o valor decimal 13 é representado pela letra D;

- o valor decimal 14 é representado pela letra E;
- o valor decimal 15 é representado pela letra F.

O sistema hexadecimal é o sistema mais utilizado em computadores.

3.5 Sistema decimal versus hexadecimal

Novamente não existe correspondência exata entre os algarismos do sistema decimal e hexadecimal.

No sistema octal tivemos *dois algarismos a menos* (8 e 9) que o sistema decimal.

No sistema hexadecimal temos *seis algarismos a mais* (A, B, C, D, E e F). Veja exemplo na página seguinte.

4. OPERAÇÃO COM NÚMEROS BINÁRIOS

4.1 Soma binária

É efetuada obedecendo-se às seguintes regras básicas:

$$0 + 0 = 0 \quad 1 + 0 = 1$$

$$0 + 1 = 1 \quad 1 + 1 = 10 \text{ (isto é, coloca 0 na posição e "vai-um" para a posição seguinte)}$$

Exemplos:

Em binário	Em decimal
1011	11
+ 0110	+ 6
10001	17

4.2 Subtração binária

Basta aplicar o processo inverso da soma, isto é, somar o valor negativo do segundo elemento. Entretanto, o processo mais conhecido é o chamado “somar o *complemento de dois* ao subtraendo”.

O *complemento de dois* de um número binário é obtido da seguinte forma:

- trocam-se todos os bits 1 por bit 0 e todos os bits 0 por 1;
- adiciona-se bit 1 ao resultado.

Exemplo:

Número binário	001110	(14 em decimal)
Troca de bits	110001	
Soma 1	+ 1	
Complemento de dois	110010	(- 14 em decimal)

A subtração é efetuada considerando-se sempre o mesmo número de *bits* tanto para as parcelas como para o resultado.

Exemplo:

Em binário	100001	Em decimal	33
	+ 110010		+ (- 14)
	<u>010011</u>		<u>19</u>

EXEMPLOS DE VALORES DECIMAIS VERSUS HEXADECIMAIS

Decimal	Binário	Hexa-decimal	Decimal	Binário	Hexa-decimal	Decimal	Binário	Hexa-decimal
0	0000	0	8	1000	8	16	0001 0000	10
1	0001	1	9	1001	9	17	0001 0001	11
2	0010	2	10	1010	A	18	0001 0010	12
3	0011	3	11	1011	B	19	0001 0011	13
4	0100	4	12	1100	C	20	0001 0100	14
5	0101	5	13	1101	D	21	0001 0101	15
6	0110	6	14	1110	E	22	0001 0110	16
7	0111	7	15	1111	F	23	0001 0111	17
						24	0001 1000	18
						25	0001 1001	19
						26	0001 1010	1A
						27	0001 1011	1B
						etc.		etc.

4.3 Multiplicação binária

Usa as seguintes operações básicas:

$$0 \times 0 = 0 \quad 1 \times 0 = 0$$

$$0 \times 1 = 0 \quad 1 \times 1 = 1$$

Exemplo:

<pre> 110101 x 1101 ----- 110101 000000 110101 110101 + ----- 1010110001 </pre>	Multiplica-se cada bit do multiplicador por cada bit do multiplicando, somando-se os resultados.
---	---

4.4 Divisão binária

É efetuada combinando-se as três operações binárias anteriores.

5. REPRESENTAÇÃO DE NÚMEROS EM PONTO FLUTUANTE

Normalmente, conhecemos os seguintes tipos de representação de números, independente de serem decimais ou binários:

INTEIRA (sem ponto decimal) Exemplos: 100 – 115 9 etc.

DECIMAL (com ponto decimal) Exemplos: 15.356 100.000 – 0.431 etc.

Essas representações são conhecidas em computação como notações em *ponto fixo*, pois a posição do ponto decimal não varia com a representação.

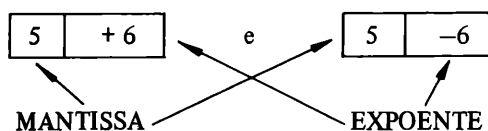
Entretanto, essas formas de representação são inconvenientes, pois ocupariam muito espaço na memória, cada vez que o valor do número crescesse ou decrescesse bastante. Por exemplo, os valores 5000000 e .000005 ocupariam sete posições.

Podemos usar uma relação do tipo abaixo, chamada *ponto flutuante*:

$$5000000 = 5 \times 10^6$$

$$.000005 = 5 \times 10^{-6}$$

Como usamos sempre a mesma base da potência (no caso, 10), para representar esses valores, basta guardar o valor 5 (chamado *mantissa* do número) e o valor do expoente em cada caso. Teremos, então, uma representação que ocuparia 3 posições de memória para representar os dois valores:



Desta forma, em um número fixo de posições podemos representar valores numéricos bastante variados. Por exemplo, em 10 posições ou dígitos podemos ter os seguintes valores: (o sinal ocupa uma posição)

MANTISSA	EXPOENTE	VALOR NUMÉRICO
– 3 5 0 3 5 2	– 4 3	– 350352 x 10 ⁻⁴³
+ 4 2 3 5 5 6	+ 8 2	423556 x 10 ⁸²
– 3 5 0 0 0 0	+ 0 1	– 350000 x 10
+ 0 0 0 0 2 5	+ 0 0	25 x 10 ⁰ = 25

Na memória do computador, tanto a mantissa como o expoente de um número em Ponto Flutuante estão em forma binária (octal, hexadecimal ou outro sistema).

O primeiro *bit* à esquerda da mantissa ou do expoente é o *bit do sinal*, sendo 0 se positivo e 1 se negativo.

Exemplo:

Mantissa (16 bits)				Expoente (8 bits)		
0001	1100	0101	0110	0011	1100	Binário
+ 1	C	5	6	+ 3	C	Hexadecimal

6. REPRESENTAÇÃO DE DADOS NÃO-NUMÉRICOS: NOTAÇÃO ASCII, EBCDIC E BCD

Além dos valores numéricos representados em sistema Hexadecimal ou Octal, o computador também armazena em sua memória os dados não numéricos formados por letras, sinais ou algarismos que entram na formação de:

NOME DE PESSOA, NOME E NÚMERO DE RUA, DATAS, CÓDIGOS DE MERCADORIAS etc.

Tais dados são armazenados no computador usando alguma notação alfabético-numérica especial. Os sistemas de notação mais usados são:

- ASCII (American Standard Code for Information Interchange)
- EBCDIC (Extended Binary Coded Decimal Interchange Code)
- BCD (Binary Coded Decimal)

Os dados não numéricos da memória e que estão representados em ASCII ou EBCDIC não podem ser usados para operações aritméticas, mas apenas para serem transferidos de um local para

outro da memória para o dispositivo de Entrada/Saída ou vice-versa, ou usados em operação de comparação e edição.

Os dados numéricos, representados por notação Hexadecimal ou Octal, por sua vez, para serem colocados em algum dispositivo de saída, devem ser previamente convertidos em notação ASCII ou EBCDIC ou BCD.

TABELA DE REPRESENTAÇÃO EM ASCII, EBCDIC E BCD

<i>Caractere</i>	<i>Código BCD</i>	<i>Código EBCDIC</i>	<i>Código ASCII</i>
espaço	110 000	0100 0000	0010 0000
.	011 011	0100 1011	0010 1110
(	111 100	0100 1101	0010 1000
+	010 000	0100 1110	0010 1011
\$	101 011	0101 1011	0010 0100
*	101 100	0101 1100	0010 1010
)	011 100	0101 1101	0010 1001
—	100 000	0110 0000	0010 1101
/	110 001	0110 0001	0010 1111
'	111 011	0110 1011	0010 1100
,	001 100	0111 1101	0010 0111
=	001 011	0111 1110	0011 1101
A	010 001	1100 0001	0100 0001
B	010 010	1100 0010	0100 0010
C	010 011	1100 0011	0100 0011
D	010 100	1100 0100	0100 0100
E	010 101	1100 0101	0100 0101
F	010 110	1100 0110	0100 0110
G	010 111	1100 0111	0100 0111
H	011 000	1100 1000	0100 1000
I	011 001	1100 1001	0100 1001
J	100 001	1101 0001	0100 1010
K	100 010	1101 0010	0100 1011
L	100 011	1101 0011	0100 1100
M	100 100	1101 0100	0100 1101
N	100 101	1101 0101	0100 1110
O	100 110	1101 0110	0100 1111
P	100 111	1101 0111	0101 0000
Q	101 000	1101 1000	0101 0001
R	101 001	1101 1001	0101 0010
S	110 010	1110 0010	0101 0011
T	110 011	1110 0011	0101 0100
U	110 100	1110 0100	0101 0101
V	110 101	1110 0101	0101 0110

W	110 110	1110 0110	0101 0111
X	110 111	1110 0111	0101 1000
Y	111 000	1110 1000	0101 1001
Z	111 001	1110 1001	0101 1010
0	000 000	1111 0000	0011 0000
1	000 001	1111 0001	0011 0001
2	000 010	1111 0010	0011 0010
3	000 011	1111 0011	0011 0011
4	000 100	1111 0100	0011 0100
5	000 101	1111 0101	0011 0101
6	000 110	1111 0110	0011 0110
7	000 111	1111 0111	0011 0111
8	001 000	1111 1000	0011 1000
9	001 001	1111 1001	0011 1001

7. OPERAÇÕES DE CONVERSÃO

7.1 Conversão de valor binário para decimal

Basta multiplicar cada **bit** pelo valor 2^{k-1} , onde **k** indica a posição do **bit** dentro do valor binário, a contar da direita para a esquerda. Somando-se todos os resultados, obtém-se o valor decimal correspondente.

Este raciocínio é análogo à representação decimal dos valores, onde cada algarismo decimal é multiplicado pelo valor $10^k - 1$ após o que, somando-se os resultados, obtém-se o valor decimal.

Exemplo:

O valor binário 1 0 1 1 equivale a $1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 8 + 0 + 2 + 1 = 11$ em decimal.

7.2 Conversão decimal para binária

Efetua-se a divisão do valor decimal por 2, guardando-se o resto da divisão (que deve ser 0 ou 1). O resultado da divisão é novamente dividido por 2 e guarda-se o resto. Assim sucessivamente até que o resultado da divisão seja zero.

A representação binária é fornecida pela SEQUÊNCIA DE RESTOS 0 ou 1, tomada na ORDEM INVERSA das divisões efetuadas.

Exemplo:

Valor decimal: 53 dividido por 2 = 26 e resto 1
 26 dividido por 2 = 13 e resto 0
 13 dividido por 2 = 6 e resto 1
 6 dividido por 2 = 3 e resto 0
 3 dividido por 2 = 1 e resto 1
 1 dividido por 2 = 0 e resto 1

Valor binário: 1 1 0 1 0 1 (sequência inversa dos restos)

Verificação: aplicando-se a regra de conversão binária para decimal, obtém-se novamente o valor decimal 53, pois:

$$110101 \text{ equivale a } (1 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0) = \\ = (32 + 16 + 0 + 4 + 0 + 1) = 53$$

7.3 Conversão de valor hexadecimal para decimal (base = 16)

Aplicando a regra de conversão análoga ao caso binário para decimal (cada algarismo hexadecimal é multiplicado por 16^{k-1}), temos:

Exemplo:

Valor hexadecimal 5 F 3 equivale a $(5 \times 16^2 + F \times 16^1 + 3 \times 16^0)$ e
 lembrando que $F = 15$ $(5 \times 16^2 + 15 \times 16^1 + 3 \times 16^0) =$
 em decimal, $= (1280 + 240 + 3) = 1523$
 (em decimal)

7.4 Conversão decimal para hexadecimal

O procedimento é novamente análogo ao caso da conversão decimal para binário, isto é, divide-se o valor decimal sucessivamente por 16 (valor base) e guarda-se o resto da divisão, que pode variar de 0, 1, 2 a 15, ou em hexadecimal, de 0, 1, 2, ..., 9, A, B, C, D, E, F.

Exemplo:

Decimal 1523 dividido por 16 = 95 e resto 3
 95 dividido por 16 = 5 e resto 15 ou **F** em hexadecimal
 5 dividido por 16 = 0 e resto 5

Valor hexadecimal correspondente: **5 F 3**.

Tabela de conversão entre decimal e hexadecimal.

	POSIÇÃO DO ALGARISMO HEXADECIMAL			
	k = 4	k = 3	k = 2	k = 1
Potências	16^3	16^2	16^1	16^0
Algarismos Hexadecimais	Valores Decimais			
0	0	0	0	0
1	4 096	256	16	1
2	8 192	512	32	2
3	12 288	768	48	3
4	16 384	1 024	64	4
5	20 480	1 280	80	5
6	24 576	1 536	96	6
7	28 672	1 792	112	7
8	32 768	2 048	128	8
9	36 864	2 304	144	9
A	40 960	2 560	160	10
B	45 056	2 816	176	11
C	49 152	3 072	192	12
D	53 248	3 328	208	13
E	57 344	3 584	224	14
F	61 440	3 840	240	15

Exemplo:

Hexadecimal **5 F 3** corresponde à:

Linha 5 na coluna k = 3 : 1280

Linha F na coluna k = 2 : 240

Linha 3 na coluna k = 1 : 3

Soma 1523 (valor decimal)

No processo inverso, o valor decimal 1523 corresponde a:

Maior valor da tabela contido em 1523 é 1280, referente à linha 5 da coluna $k = 3$

Maior valor contido em $(1523 - 1280) = 243$ é 240, referente à linha F da coluna $k = 2$

Maior valor contido em $(243 - 240) = 3$ é 3, referente à linha 3 da coluna $k = 1$

E, assim, obtemos novamente: 5 F 3.

8. ADIÇÃO E SUBTRAÇÃO EM HEXADECIMAL

8.1 Adição de valores hexadecimais

Na adição entre dois algarismos hexadecimais ocorrem casos como se seguem:

$1 + 1 = 2$ em hexadecimal (e 2 em decimal)

$3 + 4 = 7$ em hexadecimal (e 7 também em decimal)

$5 + 5 = A$ em hexadecimal (e 10 em decimal)

$9 + 2 = B$ em hexadecimal (e 11 em decimal)

$9 + 6 = F$ em hexadecimal (e 15 em decimal)

$F + 1 = 10$ em hexadecimal (e 16 em decimal)

$F + B = 1A$ em hexadecimal (e 26 em decimal)

Podemos formar portanto a seguinte tabela:

TABELA DE ADIÇÃO DE ALGARISMOS HEXADECIMAIS

+	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10
2	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11
3	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12
4	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13
5	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14
6	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15
7	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16
8	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17
9	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18
A	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19
B	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A
C	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B
D	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C
E	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D
F	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E

Exemplo:

$$\begin{array}{rcl}
 6 \text{ A} & (A + 1) = B \text{ na Tabela} & \\
 + \quad 3 \text{ 1} & (6 + 3) = 9 \text{ na Tabela} & \\
 \hline
 9 \text{ B} & \text{Resultado} & \\
 \\
 F \text{ C} & (C + 4) = 10 \text{ na Tabela; portanto, 0 e "vai 1"} & \\
 + \quad 4 & (F + \text{Vai 1}) = 10 \text{ na Tabela} & \\
 \hline
 10 \text{ 0} & \text{Resultado em hexadecimal} & \\
 \\
 F \text{ F} & (F + F) = 1E, \text{ portanto E e "vai 1"} & \\
 F \text{ F} & (F + F + 1 = 1E + 1 = 1F) & \\
 \hline
 1F \text{ E} & \text{Resultado} &
 \end{array}$$

8.2 Subtração em hexadecimal

A – Pela regra de “Emprestar 1” do próximo algarismo

O empréstimo de 1 do algarismo seguinte corresponde a emprestar valor 16 em decimal.

Exemplos:

$$\begin{array}{rcl}
 7 \text{ D A} & & \\
 - \quad 2 \text{ 2} & & \\
 \hline
 7 \text{ B 8} & (\text{não houve uso do "empresta 1"}) &
 \end{array}$$

Emprestando 1 e adicionando ao algarismo 4, temos:

$$\begin{array}{rcl}
 7 \text{ 5 4} & (14 - 5 = F \text{ ou } 20 - 5 = 15 \text{ em decimal}) & \\
 - \quad 2 \text{ 5} & (\text{nas demais casas, a subtração foi normal}) & \\
 \hline
 7 \text{ 2 F} & &
 \end{array}$$

B – Pela soma do “Complemento de dois” do subtraendo

Exemplo:

$(7 \text{ D A} - 22)$

O valor 22 ou 022 do subtraendo é em binário (0000 0010 0010); e “Complemento de dois” é $(1111 \ 1101 \ 1101 + 1) = (1111 \ 1101 \ 1110)$ ou *F D E* em hexadecimal. Somando-se os dois valores temos:

$$\begin{array}{rcl}
 \text{Somando} & 7 \text{ D A} & (E + A = 18, \text{ isto é, 8 com vai 1}): (D + D + 1 = 1B) \text{ e} \\
 + & F \text{ D E} & \\
 \hline
 & 17 \text{ B 8} & (7 + F + 1 = 17, \text{ isto é, 7 com vai 1})
 \end{array}$$

O último “vai-um” é perdido, pois a operação com “complemento de dois” deve ser feita entre valores com mesmo número de algarismos, e assim teremos o resultado correto: $7B8$.

EXERCÍCIOS DE RECAPITULAÇÃO

1. O que é Sistema Octal? Dar exemplos.
2. O que é Sistema Hexadecimal? Dar exemplos.
3. O que é Representação em Ponto Flutuante? Dar exemplos.
4. O que é notação ASCII? EBCDIC? BCD? Dar exemplos.

EXERCÍCIOS DE APLICAÇÃO

1. Efetuar as seguintes operações binárias:
 - a) $(110\ 111) + (001\ 111)$
 - b) $(111\ 000) - (111\ 101)$
 - c) $34 + 72$ (valores em decimal: converter para notação binária, efetuar a operação e dar resultado novamente em decimal)
 - d) 33×4 (valores em decimal: *idem*)
 - e) $A9 + 5B$ (valores em hexadecimal: converter para binário, efetuar a operação e dar resposta em hexadecimal)
 - f) $FF - 13$ (*idem*)
 - g) 33×4 (hexadecimal)
 - h) Resolver e e f usando a Tabela de Adição Hexadecimal.
2. Escrever cada caractere da mensagem seguinte em notação BCD, EBCDIC e ASCII (nota: o espaço também é um caractere válido): “VALOR-TOTAL IGUAL A CR\$ 3.500.00”
3. Transcreva a representação em notação ASCII da mensagem do exercício anterior em representação hexadecimal. (Cada grupo de 4 bits corresponde a um algarismo hexadecimal e, portanto, cada caractere em ASCII corresponde a dois algarismos hexadecimais.)
4. Converter para notação hexadecimal, usando a tabela de conversão:
Valores decimais: 341, 1789, 1111, 44.349 e 2222.
5. Converter para notação decimal os valores hexadecimais:
5346, 1AA, 10 10F, FFFF, FE01, 10A2 e AAAA
6. Efetuar a soma e subtração de valores hexadecimais:
 $(89 + 1)$, $(7A - 3)$, $(10A + ABC)$, $(45 - 12)$, $(46 + A6)$.

7. Programar e testar (em BASIC, FORTRAN ou ASSEMBLER) cada uma das operações de conversão:

- binária para decimal;
- decimal para binária;
- hexadecimal para decimal;
- decimal para hexadecimal.

APÊNDICE

B

INSTRUÇÕES DO MICROPROCESSADOR 8080/85 E Z80 EM ORDEM ALFABÉTICA

B.1 — INSTRUÇÕES DE USO COMUM

Forma Geral: 8080/85

Z80

Assembler		Código-Objeto	Assembler		Código-Objeto
ADIÇÃO: (A) + operando → A Flags: 8080/85: Ac = P = S = C = Z = "X" Z80: S = Z = Ac = PO = C = "X" e N = 0			ADIÇÃO COM CARRY: (A) + operando + Carry → A Flags: análogos à ADD		
8080/85	Z80		8080/85	Z80	
ADI dd	C6 dd	ADD A, dd	C6 dd	ACI dd	CE dd
ADD A	87	ADD A, A	87	ADC A	8F
ADD B	80	ADD A, B	80	ADC B	88
ADD C	81	ADD A, C	81	ADC C	89
ADD D	82	ADD A, D	82	ADC D	8A
ADD E	83	ADD A, E	83	ADC E	8B
ADD H	84	ADD A, H	84	ADC H	8C
ADD L	85	ADD A, L	85	ADC L	8D
ADD M	86	ADD A, (HL)	86	ADC M	8E
		ADD A, (IX + ds)	DD 86 ds		ADC A, (IX + ds)
		ADD A, (IY + ds)	FD 86 ds		ADC A, (IY + ds)
					DD 8E ds
					FD 8E ds
ADIÇÃO DE 16 BITS: Soma pares de registros ou ponteiro da pilha SP. Flags: 8080/85: C = "X" Z80: ADD: Ac = C = "X"; N = 0. ADC: S = Z = Ac = PO = "X" e N = 0.					

8080/85	Z80			
DAD B 09	ADD HL, BC	09	– (HL) + (BC) → HL	
DAD D 19	ADD HL, DE	19	– (HL) + (DE) → HL	
DAD H 29	ADD HL, HL	29	– (HL) + (HL) → HL	
DAD SP 39	ADD HL, SP	39	– (HL) + (SP) → HL	
	ADC HL, BC	ED 4A	– (HL) + (BC) + C → HL	
	ADC HL, DE	ED 5A	– (HL) + (DE) + C → HL	
	ADC HL, HL	ED 6A	– (HL) + (HL) + C → HL	
	ADC HL, SP	ED 7A	– (HL) + (SP) + C → HL	
	ADD IX, BC	DD 09	– (IX) + (BC) → IX	
	ADD IX, DE	DD 19	– (IX) + (DE) → IX	
	ADD IX, IX	DD 29	– (IX) + (IX) → IX	
	ADD IX, SP	DD 39	– (IX) + (SP) → IX	
	ADD IY, BC	FD 09	– (IY) + (BC) → IY	
	ADD IY, DE	FD 19	– (IY) + (DE) → IY	
	ADD IY, IY	FD 29	– (IY) + (IY) → IY	
	ADD IY, SP	FD 39	– (IY) + (SP) → IY	
AND: (A) and operando → A Flags: 8080/85: Ac=P=S=Z = “X” e C = 0 Z80: S = Z = PO = “X”; Ac = 1 e N = C = 0				
8080/85	Z80			
ANI dd E6 dd	AND dd (*)	E6 dd		
ANA A A7	AND A	A7		
ANA B A0	AND B	A0		
ANA C A1	AND C	A1		
ANA D A2	AND D	A2		
ANA E A3	AND E	A3		
ANA H A4	AND H	A4		
ANA L A5	AND L	A5		
ANA M A6	AND (HL)	A6		
	AND (IX + ds)	DD A6 ds		
	AND (IY + ds)	FD A6 ds		
CALL SUBROUTINE Flags: não são afetados.				
BIT TEST: Complemento do bit da posição n ou bbb (em binário) é colocado no flag Z. Flags: Z80: Z = complemento do bit n; N = 0 e Ac = 1				
	Z80			
	BIT n, A	CB 01bb b111		
	BIT n, B	CB 01bb b000		
	BIT n, C	CB 01bb b001		
	BIT n, D	CB 01bb b010		
	BIT n, E	CB 01bb b011		
	BIT n, H	CB 01bb b100		
	BIT n, L	CB 01bb b101		
	BIT n, (HL)	CB 01bb b110		
	BIT n, (IX + ds)	DD CB ds 01bb b110		
	BIT n, (IY + ds)	FD CB ds 01bb b110		

8080/85		Z80		
CALL ender.	CD qq pp	CALL ender.	CD qq pp	— Desvia para sub-rotina no endereço ppqq.
CNZ ender.	C4 qq pp	CALL NZ, ender.	C4 qq pp	— <i>Idem</i> se flag Z = 0 (NOT ZERO).
CZ ender.	CC qq pp	CALL Z, ender.	CC qq pp	— <i>Idem</i> se flag Z = 1. (ZERO).
CNC ender.	D4 qq pp	CALL NC, ender.	D4 qq pp	— <i>Idem</i> se flag C = 0. (NOT CARRY).
CC ender.	DC qq pp	CALL C, ender.	DC qq pp	— <i>Idem</i> se flag C = 1. (CARRY).
CPO ender.	E4 qq pp	CALL PO, ender.	E4 qq pp	— <i>Idem</i> se flag P ou PO = 0. (PARITY ODD).
CPE ender.	EC qq pp	CALL PE, ender.	EC qq pp	— <i>Idem</i> se flag P ou PO = 1. (PARITY EVEN).
CP ender.	F4 qq pp	CALL P, ender.	F4 qq pp	— <i>Idem</i> se flag S = 0. (PLUS).
CM ender.	FC qq pp	CALL M, ender.	FC qq pp	— <i>Idem</i> se flag S = 1. (MINUS).

COMPARE: (A) comparado com operando. Flags: 8080/85: Ac = P = S = C = Z = "X" Z80: S = Z = Ac = PO = C = "X" e N = 1				COMPARE COM REPETIÇÃO: Flags: Z80: S = Z = Ac = PO = "X": N = 1
8080/85		Z80		Z80
CPI dd	FE dd	CP dd (*)	FE dd	CPD ED 49 — A com ppqq em HL.
CMP A	BF	CP A	BF	CPDR ED B9 — A com ppqq em HL,
CMP B	B8	CP B	B8	com repetição, decre-
CMP C	B9	CP C	B9	mentando HL e BC.
CMP D	BA	CP D	BA	CPI ED A1 — A com ppqq em HL.
CMP E	BB	CP E	BB	CPIR ED B1 — A com ppqq em HL,
CMP H	BC	CP H	BC	com repetição, incre-
CMP L	BD	CP L	BD	mentando HL e de-
CMP M	BE	CP (HL)	BE	crementando BC.
		CP (IX + ds)	DD BE ds	
		CP (IY + ds)	FD BE ds	

COMPLEMENTA A Flags: Z80 : Ac = N = 1				DECIMAL AJUSTE NO EM A Flags: 8080/85: Ac = P = S = C = Z = "X" Z80: S = Z = Ac = PO = C = "X" e N = 0			
8080/85	Z80			8080/85	Z80		
CMA	2F	CPL	2F	DAA	27	DAA	27
DECREMENTA: Flags: 8080/85: Ac = P = S = Z = "X" Z80: S = Z = Ac = PO = C = "X"; N = 1				DECREMENTA PAR DE REGISTROS OU SP Não afeta flags.			
8080/85	Z80			8080/85	Z80		
DCR A	3D	DEC A	3D	DCX B	0B	DEC BC	0B
DCR B	05	DEC B	05	DCX D	1B	DEC DE	1B
DCR C	0D	DEC C	0D	DCX H	2B	DEC HL	2B
DCR D	15	DEC D	15	DCX SP	3B	DEC SP	3B
DCR E	1D	DEC E	1D			DEC IX	DD 2B
DCR H	25	DEC H	25			DEC IY	FD 2B
DCR L	2D	DEC L	2D				
DCR M	35	DEC (HL)	35				
		DEC (IX + ds)	DD 35 ds				
		DEC (IY + ds)	FD 35 ds				
EXCHANGE REGISTER PAIR OU STACK POINTER				Não afeta flags.			
8080/85	Z80						
XCHG	EB	EX DE, HL	EB	– Troca conteúdo de DE e HL.			
XTHL	E3	EX (SP), HL	E3	– Troca conteúdo de topo da pilha com HL.			
		EX (SP), IX	DD E3	– <i>Idem</i> , com IX ou registro X.			
		EX (SP), IY	FD E3	– <i>Idem</i> , com IY ou registro Y.			
		EX AF, AF'	08	– Troca conteúdo de AF com par alternativo AF'			
		EXX	D9	– Troca BC, DE, HL com par alternativo B' C', D' E', H' L'.			
HALT OU PARE:							
8080/85	Z80						
HLT	76	HALT	76				
INCREMENTA REGISTRO OU MEMÓRIA Flags: 8080/85: Ac = P = S = Z = "X" e C = "ψ" Z80: S = Z = Ac = PO = "X" e N = 0				INCREMENTA PAR DE REGISTRO OU SP Não afeta flags.			

8080/85	Z80	8080/85	Z80
INR A 3C	INC A 3C	INX B 03	INC BC 03
INR B 04	INC B 04	INX D 13	INC DE 13
INR C 0C	INC C 0C	INX H 23	INC HL 23
INR D 14	INC D 14	INX SP 33	INC SP 33
INR E 1C	INC E 1C		INC IX DD 23
INR H 24	INC H 24		INC IY FD 23
INR L 2C	INC L 2C		
INR M 34	INC (HL) 34		
	INC (IX + ds) DD 34 ds		
	INC (IY + ds) FD 34 ds		

JUMP TO ADDRESS:
 Não afeta flags.

8080/85	Z80	
JMP ender. C3 qq pp	JP ender. C3 qq pp	— Desvia para endereço ppqq.
JNZ ender. C2 qq pp	JP NZ, ender. C2 qq pp	— Desvia para ppqq se flag Z = 0. (NOT ZERO)
JZ ender. CA qq pp	JP Z, ender. CA qq pp	— <i>Idem</i> , se Z = 1 (ZERO).
JNC ender. D2 qq pp	JP NC, ender. D2 qq pp	— <i>Idem</i> , se C = 0 (NOT CARRY).
JC ender. DA qq pp	JP C, ender. DA qq pp	— <i>Idem</i> , se C = 1 (CARRY).
JPO ender. E2 qq pp	JP P0, ender. E2 qq pp	— <i>Idem</i> , se P ou P0 = 0 (PARITY ODD).
JPE ender. EA qq pp	JP PE, ender. EA qq pp	— <i>Idem</i> , se P ou P0 = 1 (PARITY EVEN).
JP ender. F2 qq pp	JP P, ender. F2 qq pp	— <i>Idem</i> , se S = 0 (PLUS).
JM ender. FA qq pp	JP M, ender. FA qq pp	— <i>Idem</i> , se S = 1 (MINUS).
PCHL E9	JP (HL) E9	— Desvia para endereço em HL.
	JP (IX) DD E9	— Desvia para endereço em X.
	JP (IY) FD E9	— Desvia para endereço em Y.

JUMP RELATIVE TO ADDRESS:

JR ender. 18 ds	— Desvio relativo para endereço.
JR NC, ender. 30 ds	— Desvio relativo se C = 0 (NOT CARRY).
JR C, ender. 38 ds	— Desvio relativo se C = 1 (CARRY)
JR NZ, ender. 20 ds	— Desvio relativo se Z = 0 (NOT ZERO).

JR Z, ender.	28 ds	— Desvio relativo se Z = 1 (ZERO).
DJNZ ender.	10 ds	— Decrementa B e desvio relativo se B não for zero.

MOVE OU LOAD IMMEDIATE:

Valor imediato dd vai para registro ou memória. Não afeta flags.

8080/85		Z80	
MVI A, dd	3E dd	LD A, dd	3E dd
MVI B, dd	06 dd	LD B, dd	06 dd
MVI C, dd	0E dd	LD C, dd	0E dd
MVI D, dd	16 dd	LD D, dd	16 dd
MVI E, dd	1E dd	LD E, dd	1E dd
MVI H, dd	26 dd	LD H, dd	26 dd
MVI L, dd	2E dd	LD L, dd	2E dd
MVI M, dd	36 dd	LD (HL), dd	36 dd
		LD (IX + ds), dd	DD 36 ds dd
		LD (IY + ds), dd	FD 36 ds dd

MOVE OU LOAD REGISTRO A REGISTRO:

MOV ou LD reg_i, reg_j : Conteúdo de Reg_j para Reg_i. Não afeta flags.

8080/85		Z80		8080/85		Z80	
MOV A, A	7F	LD A, A	7F	MOV D, C	51	LD D, C	51
MOV A, B	78	LD A, B	78	MOV D, D	52	LD D, D	52
MOV A, C	79	LD A, C	79	MOV D, E	53	LD D, E	53
MOV A, D	7A	LD A, D	7A	MOV D, H	54	LD D, H	54
MOV A, E	7B	LD A, E	7B	MOV D, L	55	LD D, L	55
MOV A, H	7C	LD A, H	7C	MOV E, A	5F	LD E, A	5F
MOV A, L	7D	LD A, L	7D	MOV E, B	58	LD E, B	58
MOV B, A	47	LD B, A	47	MOV E, C	59	LD E, C	59
MOV B, B	40	LD B, B	40	MOV E, D	5A	LD E, D	5A
MOV B, C	41	LD B, C	41	MOV E, E	5B	LD E, E	5B
MOV B, D	42	LD B, D	42	MOV E, H	5C	LD E, H	5C
MOV B, E	43	LD B, E	43	MOV E, L	5D	LD E, L	5D
MOV B, H	44	LD B, H	44	MOV H, A	67	LD H, A	67
MOV B, L	45	LD B, L	45	MOV H, B	60	LD H, B	60
MOV C, A	4F	LD C, A	4F	MOV H, C	61	LD H, C	61
MOV C, B	48	LD C, B	48	MOV H, D	62	LD H, D	62
MOV C, C	49	LD C, C	49	MOV H, E	63	LD H, E	63
MOV C, D	4A	LD C, D	4A	MOV H, H	64	LD H, H	64

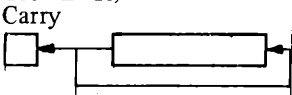
MOV C, E 4B	LD C, E 4B	MOV H, L 65	LD H, L 65
MOV C, H 4C	LD C, H 4C	MOV L, A 6F	LD L, A 6F
MOV C, L 4D	LD C, L 4D	MOV L, B 68	LD L, B 68
MOV D, A 57	LD D, A 57	MOV L, C 69	LD L, C 69
MOV D, B 50	LD D, B 50	MOV L, D 6A	LD L, D 6A
		MOV L, E 6B	LD L, E 6B
		MOV L, H 6C	LD L, H 6C
		MOV L, L 6D	LD L, L 6D
MOVE OU LOAD MEMÓRIA PARA REGISTRO Não afeta flags.		MOVE OU STORE REGISTRO PARA MEMÓRIA Não afeta flags.	
8080/85	Z80	8080/85	Z80
MOV A, M 7E	LD A, (HL) 7E LD A, (IX + ds) DD 7E ds LD A, (IY + ds) FD 7E ds	MOV M, A 77	LD (HL), A 77 LD (IX + ds), A DD 77 ds LD (IY + ds), A FD 77 ds
MOV B, M 46	LD B, (HL) 46 LD B, (IX + ds) DD 46 ds LD B, (IY + ds) FD 46 ds	MOV M, B 70	LD (HL), B 70 LD (IX + ds), B DD 70 ds LD (IY + ds), B FD 70 ds
MOV C, M 4E	LD C, (HL) 4E LD C, (IX + ds) DD 4E ds LD C, (IY + ds) FD 4E ds	MOV M, C 71	LD (HL), C 71 LD (IX + ds), C DD 71 ds LD (IY + ds), C FD 71 ds
MOV D, M 56	LD D, (HL) 56 LD D, (IX + ds) DD 56 ds LD D, (IY + ds) FD 56 ds	MOV M, D 72	LD (HL), D 72 LD (IX + ds), D DD 72 ds LD (IY + ds), D FD 72 ds
MOV E, M 5E	LD E, (HL) 5E LD E, (IX + ds) DD 5E ds LD E, (IY + ds) FD 5E ds	MOV M, E 73	LD (HL), E 73 LD (IX + ds), E DD 73 ds LD (IY + ds), E FD 73 ds
MOV H, M 66	LD H, (HL) 66 LD H, (IX + ds) DD 66 ds LD H, (IY + ds) FD 66 ds	MOV M, H 74	LD (HL), H 74 LD (IX + ds), H DD 74 ds LD (IY + ds), H FD 74 ds
MOV L, M 6E	LD L, (HL) 6E LD L, (IX + ds) DD 6E ds LD L, (IY + ds) FD 6E ds	MOV M, L 75	LD (HL), L 75 LD (IX + ds), L DD 75 ds LD (IY + ds), L FD 75 ds
LDAX B 0A	LD A, (BC) 0A	STAX B 02	LD (BC), A 02

NEGATIVO DE A Flags: S=Z=Ac=P0=C="X" e N=1 Z80 NEG ED 44	NOP – NENHUMA OPERAÇÃO 8080/85 Z80 NOP 00 NOP 00
OR: (A) or operando → A Flags: 8080/85: P=S=Z="X" e C=Ac=0 Z80: S=Z=P0="X"; Ac=1; N=C=0 8080/85 Z80 ORI dd F6 dd OR dd (*) F6 dd ORA A B7 OR A B7 ORA B B0 OR B B0 ORA C B1 OR C B1 ORA D B2 OR D B2 ORA E B3 OR E B3 ORA H B4 OR H B4 ORA L B5 OR L B5 ORA M B6 OR (HL) B6 OR (IX+ds) DD B6 ds OR (IY+ds) FD B6 ds	POP OU PUSH STACK: Conteúdo de par de registros vai para as duas posições do topo da pilha e vice-versa. Não afeta Flags. 8080/85 Z80 POP B C1 POP BC C1 – Pilha para BC. POP D D1 POP DE D1 – Pilha para DE. POP H E1 POP HL E1 – Pilha para HL. POP PSW F1 POP AF F1 – Pilha para AF ou PSW. POP IX DD E1 – Pilha para X. POP IY FD E1 – Pilha para Y. PUSH B C5 PUSH BC C5 – BC para pilha. PUSH D D5 PUSH DE D5 – DE para pilha. PUSH H E5 PUSH HL E5 – HL para pilha. PUSH PSW F5 PUSH AF F5 – AF ou PSW para pilha. PUSH IX DD E5 – X para pilha. PUSH IY FD E5 – Y para pilha.
RESET: Coloca valor zero (RESET) no bit da posição n ou bbb (em binário) do registro ou memória. Não afeta flags. Z80 RES n, A CB 10bb b111 RES n, B CB 10bb b000 RES n, C CB 10bb b001 RES n, D CB 10bb b010	

RES n, E CB 10bb b011
 RES n, H CB 10bb b100
 RES n, L CB 10bb b101

RES n, (HL) CB 10bb b110
 RES n, (IX + ds) DD CB ds 10bb b110
 RES n, (IY + ds) FD CB ds 10bb b110

ROTATE LEFT CIRCULAR;

Flags: Vide  observação

8080/85 Z80

RLC 07 RLCA 07
 RLC A CB 07
 RLC B CB 00
 RLC C CB 01
 RLC D CB 02
 RLC E CB 03
 RLC H CB 04
 RLC L CB 05

 RLC (HL) CB 06
 RLC (IX + ds) DD CB ds 06
 RLC (IY + ds) FD CB ds 06

ROTATE LEFT:

Flags: Ver  observação

8080/85 Z80

RAL 17 RLA 17
 RL A CB 17
 RL B CB 10
 RL C CB 11
 RL D CB 12
 RL E CB 13
 RL H CB 14
 RL L CB 15

 RL (HL) CB 16
 RL (IX + ds) DD CB ds 16
 RL (IY + ds) FD CB ds 16

ROTATE RIGHT CIRCULAR:

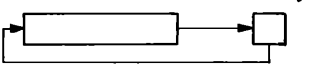
Flags: Ver  observação.

8080/85 Z80

RRC 0F RRCA 0F
 RRC A CB 0F
 RRC B CB 08
 RRC C CB 09
 RRC D CB 0A
 RRC E CB 0B
 RRC H CB 0C
 RRC L CB 0D

 RRC (HL) CB 0E
 RRC (IX + ds) DD CB ds 0E
 RRC (IY + ds) FD CB ds 0E

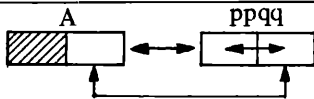
ROTATE RIGHT:

Flags: Ver  observação.

8080/85 Z80

RAR 1F RRA 1F
 RR A CB 1F
 RR B CB 18
 RR C CB 19
 RR D CB 1A
 RR E CB 1B
 RR H CB 1C
 RR L CB 1D

 RR (HL) CB 1E
 RR (IX + ds) DD CB ds 1E
 RR (IY + ds) FD CB ds 1E

ROTATE DIGIT BCD:			Flags: Z80: S=Z=P=0=X; Ac=N=0
Z80			
RLD	ED 6F	– Gira esquerda um dígito (4 bits) de ppqq e bits menos significativos de A; ppqq em HL.	
RRD	ED 67	– Gira direita um dígito.	

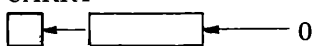
RETURN FROM SUBROUTINE				RESTART SUBROUTINE:			
Não afeta flags.				Desvia para sub-rotina do endereço n. Não afeta flags.			
8080/85		Z80		8080/85		Z80	
RET C9	RET C9	– Return.		RST 0 C7	RST 00 C7		
RNZ C0	RET NZ C0	– Idem, NOT ZERO.		RST 1 CF	RST 08H CF		
RZ C8	RET Z C8	– Idem, ZERO.		RST 2 D7	RST 10H D7		
RNC D0	RET NC D0	– Idem, NOT CARRY.		RST 3 DF	RST 18H DF		
RC D8	RET C D8	– Idem, CARRY.		RST 4 E7	RST 20H E7		
RPO E0	RET P0 E0	– Idem, PARITY ODD.		RST 5 EF	RST 28H EF		
RPE E8	RET PE E8	– Idem, PARITY EVEN.		RST 6 F7	RST 30H F7		
RP F0	RET P F0	– Idem, PLUS.		RST 7 FF	RST 38H FF		
RM F8	RET M F8	– Idem, MINUS.					

SET CARRY E COMPLEMENTA CARRY:			
8080/85		Z80	
STC 37	SCF 37	– Carry = 1.	
CMC 3F	CCF 3F	– Carry = complemento do Carry.	

SET:	
Coloca bit um (SET) na posição n ou bbb (em binário) do registro ou memória.	
Não afeta flags.	
Z80	
SET n, A	CB 11bb b111
SET n, B	CB 11bb b000
SET n, C	CB 11bb b001
SET n, D	CB 11bb b010
SET n, E	CB 11bb b011
SET n, H	CB 11bb b100
SET n, L	CB 11bb b101

SET n, (HL) CB 11bb b110
 SET n, (IX + ds) DD CB ds 11bb b110
 SET n, (IY + ds) FD CB ds 11bb b110

SHIFT LEFT ARITHMETIC: CARRY



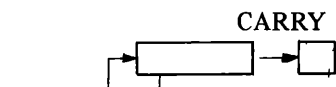
Flags: S = Z = P0 = "X"
 C = Bit 7

Z80

SLA A CB 27
 SLA B CB 20
 SLA C CB 21
 SLA D CB 22
 SLA E CB 23
 SLA H CB 24
 SLA L CB 25

SLA (HL) CB 26
 SLA (IX + ds) DD CB ds 26
 SLA (IY + ds) FD CB ds 26

SHIFT RIGHT ARITHMETIC:



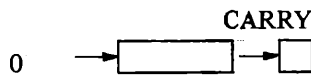
Flags: Z = P0 = "X"
 C = Bit 0; S = mesmo

Z80

SRA A CB 2F
 SRA B CB 28
 SRA C CB 29
 SRA D CB 2A
 SRA E CB 2B
 SRA H CB 2C
 SRA L CB 2D

SRA (HL) CB 2E
 SRA (IX + ds) DD CB ds 2E
 SRA (IY + ds) FD CB ds 2E

SHIFT RIGHT LOGICAL:



Flags: Z = P0 = S = "X"
 C = Bit 0.

Z80

SRL A CB 3F
 SRL B CB 38
 SRL C CB 39
 SRL D CB 3A
 SRL E CB 3B
 SRL H CB 3C
 SRL L CB 3D

SRL (HL) CB 3E
 SRL (IX + ds) DD CB ds 3E
 SRL (IY + ds) FD CB ds 3E

SUBTRAÇÃO:

(A) menos operando → A

Flags: 8080/85: Ac = P = S = C = Z = "X"
 Z80: S = Z = Ac = P0 = C = "X"
 e N = 1

8080/85

Z80

SUI dd	D6 dd	SUB dd (*)	D6 dd
SUB A	97	SUB A	97
SUB B	90	SUB B	90
SUB C	91	SUB C	91
SUB D	92	SUB D	92
SUB E	93	SUB E	93
SUB H	94	SUB H	94
SUB L	95	SUB L	95

SUB M	96	SUB (HL)	96
		SUB (IX + ds)	DD 96 ds
		SUB (IY + ds)	FD 96 ds

SUBTRAÇÃO COM CARRY:

(A) menos (operando + carry) → A

Flags: idênticos a SUB.

8080/85

Z80

SBI dd	DE dd	SBC A, dd	DE dd
SBB A	9F	SBC A, A	9F
SBB B	98	SBC A, B	98
SBB C	99	SBC A, C	99
SBB D	9A	SBC A, D	9A
SBB E	9B	SBC A, E	9B
SBB H	9C	SBC A, H	9C
SBB L	9D	SBC A, L	9D
SBB M	9E	SBC A, (HL)	9E
		SBC A, (IX + ds)	DD 9E ds
		SBC A, (IY + ds)	FD 9E ds

SUBTRAÇÃO DE 16 BITS COM PARES DE REGISTROS OU STACK POINTER SP

Flags: idênticos a SUB.

Z80

SBC HL, BC	ED 42	– (HL) – (BC + Carry) → HL
SBC HL, DE	ED 52	– (HL) – (DE + Carry) → HL
SBC HL, HL	ED 62	– (HL) – (HL + Carry) → HL
SBC HL, SP	ED 72	– (HL) – (SP + Carry) → HL

EXCLUSIVE OR:

(A) xor operando → A

Flags: 8080/85: Ac = C = 0 e P = S = Z = "X"

Z80: S = Z = P0 = "X"; Ac = 1 e N = C = 0

8080/85**Z80**

XRI dd	EE dd	XOR dd (*)	EE dd
XRA A	AF	XOR A	AF
XRA B	A8	XOR B	A8
XRA C	A9	XOR C	A9
XRA D	AA	XOR D	AA
XRA E	AB	XOR E	AB
XRA H	AC	XOR H	AC
XRA L	AD	XOR L	AD
XRA M	AE	XOR (HL)	AE
		XOR (IX + ds)	DD AE ds
		XOR (IY + ds)	FD AE ds

NOTAÇÕESOperandos 8080/85 Z80
de oito bits:

dd	dd	– Imediato.
A	A	– Registro A.
B	B	– Registro B.
C	C	– Registro C.
D	D	– Registro D.
E	E	– Registro E.
H	H	– Registro H.
L	L	– Registro L.
M	(HL)	– Memória com endereço ppqq em HL.
	(IX + ds)	– Memória com endereço indexado por X e ds.
	(IY + ds)	– Memória com endereço indexado por Y e ds.

(*) – Z80 permite a forma dd ou A, dd. Exemplo: AND dd ou AND A, dd.

Operandos
de 16 bits:

B	BC	— Par de registro B e C.
D	DE	— <i>Idem</i> , D e E.
H	HL	— <i>Idem</i> , H e L.
SP	SP	— Ponteiro da pilha ou Stack Pointer SP.
	IX	— Registro de Índice IX ou X.
	IY	— Registro de Índice IY ou Y.

Flags (Status Flags): “X” — O flag é afetado pela operação, isto é, assume valor UM ou ZERO de acordo com o resultado.

1 — O flag sempre assume valor UM nessa operação.

0 — O flag assume sempre valor ZERO.

Observação

FLAGS DE “ROTATES”

No 8080/85 — RLC, RAL, RRC e RAR: apenas o carry é afetado pela operação. Demais flags permanecem inalterados.

No Z80 — RLCA, RLA, RRCA E RRA (equivalente a RLC, RAL, RRC e RAR do 8080/85):
Ac = N = 0 e carry é afetado pela operação.

Nas demais formas:

RL, RLC, RR e RRC os flags são S = Z = $\tilde{P}0$ = C = X; Ac = N = 0.

B.2 — INSTRUÇÕES DE INTERRUPTÃO, LEITURA E SAÍDA DE DADOS

INTERRUPTÕES

8080/85		Z80		
DI	F3	DI	F3	— Desativa interrupção.
EI	FB	EI	FB	— Ativa interrupção.
		IM 0	ED 46	— Interrupção de modo 0.
		IM 1	ED 56	— Interrupção de modo 1.
		IM 2	ED 5E	— Interrupção de modo 2.
		LD A, I	ED 57	— Vetor de interrupção I vai para A.
		LD A, R	ED 5F	— Registro de resfriamento R vai para A.
RIM	20			— Máscara de interrupção vai para A.
SIM	30			— Conteúdo de A vai para máscara de interrupção.

INPUT: LEITURA DE DADOS

IN yy	DB yy	IN A, (yy)	DB yy	— Byte do porto yy para registro A.
		IN A, (C)	ED 78	— Byte do porto com endereço em C, vai para A.

IN B, (C)	ED 40	— <i>Idem</i> , para B.
IN C, (C)	ED 48	— <i>Idem</i> , para C.
IN D, (C)	ED 50	— <i>Idem</i> , para D.
IN E, (C)	ED 58	— <i>Idem</i> , para E.
IN H, (C)	ED 60	— <i>Idem</i> , para H.
IN L, (C)	ED 68	— <i>Idem</i> , para L.
IN F, (C)	ED 70	— <i>Idem</i> , para flags F.
IND	ED AA	— Byte do porto com endereço em C vai para memória ppqq, decrementa HL e B.
INDR	ED BA	— Leitura com repetição, decrementando HL e B.
INI	ED A2	— Byte do porto com endereço em C vai para memória ppqq, incrementa HL e decrementa B.
INIR	ED B2	— Leitura com repetição, incrementando HL e decrementando B.

OUTPUT – SAÍDA DE DADOS

OUT yy	D3 yy	OUT (yy), A	D3 yy	— Operação inversa de IN correspondente.
		OUT (C), A	ED 79	
		OUT (C), B	ED 41	
		OUT (C), C	ED 49	
		OUT (C), D	ED 51	
		OUT (C), E	ED 59	
		OUT (C), H	ED 61	
		OUT (C), L	ED 69	
		OUT (C), F	ED 71	
		OUTD	ED AB	— Operação inversa de IND.
		OUTDR	ED BB	
		OUTI	ED A3	
		OUTIR	ED 83	

B.3 — PRINCIPAIS INSTRUÇÕES DIRETIVAS (PSEUDO-INSTRUÇÕES)

8080/85	Z80	
DB	DEFB	— Define valor de um byte na memória.
DW	DEFW	— Define valor de dois bytes ou uma palavra (16 bits) na memória.
DS	DEFS	— Define área (storage) de um determinado tamanho na memória.

END	END	– Define fim do programa Assembler.
ORG	ORG	– Define origem ou reinício de endereço do programa ou área de dados.
EQU	EQU	– Define equivalência ou igualdade de endereço entre dois símbolos ou rótulos.

NOTAÇÃO DOS OPERANDOS OU ENDEREÇOS

RÓTULO OU OPERANDO SIMBÓLICO (LABELS): Formado por até 6 (seis) (Z80) e 5 (cinco) (8080/85) letras ou algarismos. Alguns caracteres como ? ou – (sub-linha) podem ser usados. O primeiro caractere deve ser letra.

STRING OU SEQUÊNCIA DE CARACTERES ASCII – Deve começar e terminar com um sinal de apóstrofo ('). Exemplo: 'ANTONIO'

OPERANDOS COM VALOR DECIMAL: Exemplos: 1234, -0010.

OPERANDOS COM VALOR HEXADECIMAL: O conjunto de algarismos hexadecimais deve estar seguido pela letra "H".

Exemplos: +237H, 0FABH, 100H.

É recomendável colocar o algarismo zero antes de valores hexadecimais que começam com A, B, C, D, E, ou F.

OPERANDOS COM VALOR BINÁRIO: A sequência de bits zero ou um deve estar imediatamente seguida pela letra "B".

Exemplo: 1010111B.

APÊNDICE

C

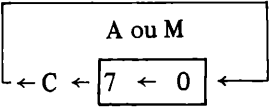
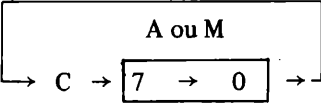
INSTRUÇÕES DO MICROPROCESSADOR 6502 EM ORDEM ALFABÉTICA

C.1 – INSTRUÇÕES

ADC – ADD WITH CARRY $\{dd \text{ ou } M\} + C + A \rightarrow A$ Flags S, Z, V, C afetados. ADC #dd 69 dd ADC endereço 65 qq 6D qq pp ADC endereço, X 75 qq 7D qq pp ADC endereço, Y 79 qq pp ADC (endereço, X) 61 qq ADC (endereço), Y 71 qq	Bcc – BRANCH IF CONDITION cc TRUE Desvia para endereço se condição for verdade. Não altera flags. BCC endereço 90 ds – Carry Clear (C = 0) BCS endereço B0 ds – Carry Set (C = 1) BEQ endereço F0 ds – Equal to zero (Z = 1) BMI endereço 30 ds – Minus (S = 1) BNE endereço D0 ds – Not Equal (Z = 0) BPL endereço 10 ds – Positive (S = 0) BVC endereço 50 ds – Overflow Clear (V = 0) BVS endereço 70 ds – Overflow Set (V = 1) “ds” : deslocamento relativo
AND – LOGICAL AND $\{dd \text{ ou } M\} \text{ “and” } A \rightarrow A$ Flags S, Z afetados AND #dd 29 dd AND endereço 25 qq 2D qq pp AND endereço, X 35 qq 3D qq pp AND endereço, Y 39 qq pp AND (endereço, X) 21 qq AND (endereço), Y 31 qq	BIT – BIT TEST A “and” M afeta flags S, V, Z. Z = 1 se A “and” M = 0 Z = 0 se A “and” M $\neq$ 0 S = bit 7 de M V = bit 6 de M BIT endereço 24 qq 2C qq pp
	BRK – BREAK OR SOFTWARE INTERRUPT Força parada, afetando interrupção e pilha. BRK 00

ASL – ARITHMETIC SHIFT LEFT A ou M C ← 7 ← 0 ← 0 Flags afetados: S, Z, C ASL A 0A ASL endereço 06 qq 0E qq pp ASL endereço, X 16 qq 1E qq pp	CLC – CLEAR STATUS FLAG c CLC 18 – Clear Carry (C = 0) CLD D8 – Clear Decimal mode (D = 0) CLI 58 – Clear Interrupt (I = 0) CLV B8 – Clear Overflow (V = 0)
CMP – COMPARE MEMORY OR IMMEDIATE WITH ACCUMULATOR A – {dd ou M} afeta flags S, Z, C CMP #dd C9 dd CMP endereço C5 qq CD qq pp CMP endereço, X D5 qq DD qq pp CMP endereço, Y D9 qq pp CMP (endereço, X) C1 qq CMP (endereço, Y) D1 qq	EOR – EXCLUSIVE OR {dd ou M} “ou excl” A → A Flags S, Z afetados. EOR #dd 49 dd EOR endereço 45 qq 4D qq pp EOR endereço, X 55 qq 5D qq pp EOR endereço, Y 59 qq pp EOR (endereço, X) 41 qq EOR (endereço, Y) 51 qq
CPX – COMPARE REGISTER X WITH MEMORY OR IMMEDIATE X – {dd ou M} afeta flags S, Z, C CPX #dd E0 dd CPX endereço E4 qq EC qq pp	DEC – DECREMENT MEMORY M – 1 → M Flags S, Z afetados. DEC endereço C6 qq CE qq pp DEC endereço, X D6 qq DE qq pp
CPY – COMPARE REGISTER Y WITH MEMORY OR IMMEDIATE Y – {dd ou M} afeta flags S, Z, C CPY #dd C0 dd CPY endereço C4 qq CC qq pp	DEX e DEY – DECREMENT REGISTER X OR Y {X ou Y} – 1 → X ou Y Flags S, Z afetados DEX CA DEY 88

INC – INCREMENT MEMORY $M + 1 \rightarrow M$ Flags S, Z afetados. INC endereço E6 qq EE qq pp INC endereço, X F6 qq FE qq pp	LDX ou LDY – LOAD REGISTER X OU Y WITH MEMORY OR IMMEDIATE $\{dd \text{ ou } M\} \rightarrow X \text{ ou } Y$ Flags S, Z afetados. LDX #dd A2 dd LDY #dd A0 dd LDX endereço A6 qq AE qq pp LDY endereço A4 qq AC qq pp LDX endereço, Y B6 qq BE qq pp LDY endereço, X B4 qq BC qq pp
INX e INY – INCREMENT REGISTER X OR Y $\{X \text{ ou } Y\} + 1 \rightarrow X \text{ ou } Y$ Flags S, Z afetados. INX E8 INY C8	LSR – LOGICAL SHIFT RIGHT A ou M 0 → 7 → 0 → C Flag S = 0 e Z, C são afetados. LSR A 4A LSR endereço 46 qq 4E qq pp LSR endereço, X 56 qq 5E qq pp
JMP – JUMP TO MEMORY Desvia para endereço. Não afeta flags. JMP endereço 4C qq pp JMP (endereço) 6C qq pp	ORA – LOGICAL OR $\{dd \text{ ou } M\} \text{ "or" } A \rightarrow A$ Flags S, Z afetados. ORA #dd 09 dd ORA endereço 05 qq 0D qq pp ORA endereço, X 15 qq 1D qq pp ORA endereço, Y 19 qq pp ORA (endereço, X) 01 qq ORA (endereço, Y) 11 qq
LDA – LOAD ACCUMULATOR WITH MEMORY OR IMMEDIATE $\{dd \text{ ou } M\} \rightarrow A$ Flags S, Z afetados. LDA #dd A9 dd LDA endereço A5 qq AD qq pp LDA endereço, X B5 qq BD qq pp LDA endereço, Y B9 qq pp LDA (endereço, X) A1 qq LDA (endereço, Y) B1 qq	NOP – NO OPERATION NOP EA

PUSH OR POP STACK PHA 48 — Ac A → Pilha. PHP 08 — Reg. Status P → Pilha. PLA 68 — Topo pilha → Ac A; afeta S, Z. PLP 28 — Top pilha → Reg. Status P.	SBC — SUBTRACT WITH BORROW $A - \{dd \text{ ou } M\} - \bar{C} \rightarrow A$ Flags S, Z, V, C afetados. SBC #dd E9 dd SBC endereço E5 qq ED qq pp SBC endereço, X F5 qq FD qq pp SBC endereço, Y F9 qq pp SBC (endereço, X) E1 qq SBC (endereço), Y F1 qq
ROL — ROTATE LEFT  Flags S, Z, C afetados. ROL A 2A ROL endereço 26 qq 2E qq pp ROL endereço, X 36 qq 3E qq pp	SET STATUS FLAG SEC 38 — Set Carry (C = 1) SED F8 — Set Decimal Mode (D = 1) SEI 78 — Disable Interrupts (I = 1)
ROR — ROTATE RIGHT  Flags S, Z, C afetados. ROR A 6A ROR endereço 66 qq 6E qq pp ROR endereço, X 76 qq 7E qq pp	STA — STORE ACCUMULATOR IN MEMORY $A \rightarrow M$ Não afeta flags. STA endereço 85 qq 8D qq pp STA endereço, X 95 qq 9D qq pp STA endereço, Y 99 qq pp STA (endereço, X) 81 qq STA (endereço), Y 91 qq
RTI — RETURN FROM INTERRUPT Retorna do estado de interrupção, executando instrução que está na pilha. Afeta todos os flags. RTI 40	STX e STY — STORE REGISTER X OR Y IN MEMORY $X \text{ ou } Y \rightarrow M$ Não afeta flags. STX endereço 86 qq 8E qq pp STY endereço 84 qq 8C qq pp STX endereço, Y 96 qq STY endereço, X 94 qq
RTS — RETURN FROM SUBROUTINE Retorno da sub-rotina, apanhando endereço de retorno, na pilha. Não afeta flags. RTS 60	

Txy – TRANSFER REGISTERS A, X, Y OR SP

TAX	AA	– A → X
TAY	A8	– A → Y
TXA	8A	– X → A
TYA	98	– Y → A
TSX	BA	– SP → X
TXS	9A	– X → SP

Notação

M : endereço de memória, pode ser: Direto (00qq (página zero) ou ppqq); indexado por X (endereço + X = 00qq + X ou ppqq + X); Indexado por Y (endereço + Y = ppqq + Y); Indireto pré-incrementado por X (endereço de M está em 00qq + X) e Indireto pós-incrementado por Y (endereço de M é rrss + Y com rrss estando em 00qq e 00qq + 1).

#dd : operando imediato.

endereço : endereço direto simbólico ou absoluto; pode estar na página zero (00qq) ou não (ppqq).

endereço, X : endereço indexado por X.

endereço, Y : endereço indexado por Y.

(endereço, X) : endereço indireto pré-indexado por X.

(endereço), Y : endereço indireto pós-indexado por Y.

C.2 – PRINCIPAIS INSTRUÇÕES DIRETIVAS (PSEUDO-INSTRUÇÕES)

6502

Versão padrão Versão LISA

.BYTE	BYT	– Define valor de um byte na memória.
.DBYTE	DBY	– Define valor de dois bytes (16 bits) na memória.
.DWORD		– Define valor de dois bytes (16 bits) colocando byte menos significativo em primeiro lugar.
* = * + N	DFS N	– Define uma área de N bytes na memória.
.END	END	– Define fim do programa Assembler.
* = XX	ORG XX	– Define início do programa ou área de dados a partir do endereço XX.
=	EQU	– Define equivalência ou igualdade de endereço entre dois rótulos ou símbolos.

NOTAÇÃO DOS OPERANDOS

OPERANDO SIMBÓLICO OU RÓTULO (LABEL) — é formado por até seis caracteres (letras, dígitos ou alguns caracteres). O primeiro caractere deve ser sempre letra.

OPERANDO COM VALOR DECIMAL — formado por algarismos de 0 a 9.

Exemplos: 100, -125.

OPERANDO COM VALOR HEXADECIMAL — formado por seqüência de algarismos hexadecimais precedida pelo caractere \$ (cifrão).

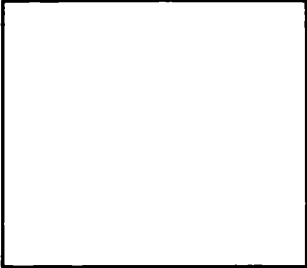
Exemplos: \$100, \$1FFA.

OPERANDO COM VALOR BINÁRIO — formado pela seqüência de valores um ou zero, precedida pelo caractere % (porcentagem).

Exemplos: %1100, %100.

OPERANDO É SEQUÊNCIA OU STRINGS DE CARACTERES ASCII — A seqüência de caracteres deve estar precedida pelo sinal de apóstrofo (').

Exemplo: 'ANTONIO', 'F'.



REFERÊNCIAS

ADAM OSBORNE • *Introdução aos Microprocessadores*
• McGraw-Hill, 1984.

LANCE A. LEVENTHAL • *6502 Assembly Language Programming*
• OSBORNE/McGRAW-HILL, 1979.

LANCE A. LEVENTHAL • *Z80 Assembly Language Programming*
• OSBORNE/McGRAW-HILL, 1979.

LANCE A. LEVENTHAL • *8080A/85 Assembly Language Programming*
• OSBORNE/McGRAW-HILL, 1979.

RANDY HYDE • *Lisa: A Professional Assembly Development System for Apple Computers*
• On-Line Systems, Inc., 1981.

OUTROS LIVROS NA ÁREA

- | | |
|-----------------|--|
| Carvalho | — Assembler para o TK 90X |
| Ciarcia | — Construa seu próprio Microcomputador Z80 |
| Cramer | — 68000 — Microprocessador |
| Hoffman | — MSX — Guia do Usuário |
| Malvino | — Microcomputadores e Microprocessadores |
| Mottola | — Linguagem de Programação Assembly para Apple II — 6502 |
| Poole | — Apple II — Guia do Usuário |
| Taub | — Circuitos Digitais e Microprocessadores |
| Tokheim | — Introdução aos Microprocessadores |