
Universidade Federal de Goiás

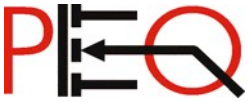

Escola de Engenharia Elétrica e de Computação

# Notas de Aula

# Microprocessador 8085

# Microprocessador 8088

**Prof. José Wilson Lima Nerys**




Núcleo de Estudo e Pesquisa em
Processamento da Energia e Qualidade

**Goiânia, 2007**

# 1. INTRODUÇÃO AOS COMPUTADORES E MICROPROCESSADORES

## 1.1 Histórico sobre Computadores

**4000 A.C - ÁBACO –** Invenção do *ábaco* pelos babilônios. Instrumento usado para realizar operações aritméticas, onde cada coluna representa uma casa decimal. Era o principal instrumento de cálculo do século XVII e é usado até hoje.

Ábaco

Octograma chinês Yin Yang

Data da mesma época do ábaco o octograma chinês *Yin Yang*, o qual é tido como a primeira representação binária dos números de 0 a 8. Foi criado pelo imperador chinês *Fou-Hi* para representar a interação entre as duas energias que juntas são o fundamento da "totalidade".

**1614 – LOGARITMO –** O cientista escocês **JOHN NAPIER** criou os *logaritmos*. Através das tabelas criadas, as operações de multiplicação e divisão tornaram-se mais simples, pois eram substituídas por operações de adição e subtração, reduzindo o tempo de processamento.

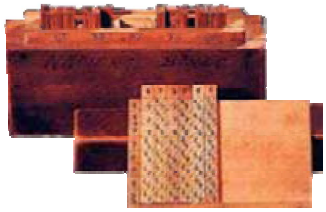
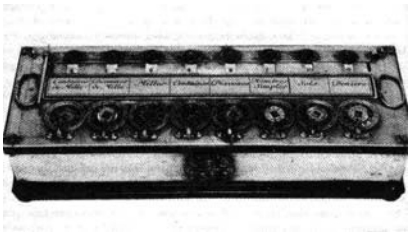


Tabela de logaritmos



Pascaline

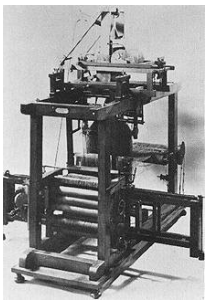
**1623 – RELÓGIO DE CALCULAR – WILHELM SHICKARD**, professor de matemática da Universidade de Tübingen, Alemanha, inventou um relógio de calcular que é considerado a primeira máquina mecânica de calcular da história. Fazia multiplicação e divisão, mas requeria várias intervenções do operador. Usava o princípio desenvolvido por Napier ("Napier's bones"). Essa calculadora foi desenvolvida para auxiliar o matemático e astrônomo Johannes Kepler.

**1642 – PASCALINE –** O cientista francês **BLAISE PASCAL** criou uma calculadora capaz de realizar operações de adição e subtração. A máquina implementada utilizava rodas e engrenagens, com as quais era possível representar números decimais de 0 a 9. Pascal desenvolveu essa máquina para ajudar seu pai na coleta de impostos. A máquina teve mais de 50 versões diferentes em uma década.

**1671 –** O matemático alemão francês **GOTTFRIED LEIBNIZ** criou uma calculadora de 4 funções, capaz de realizar operações de adição, subtração, multiplicação e divisão. É a antecessora das calculadoras atuais. O problema comum às calculadoras até esta época era a necessidade de entrar com todos os resultados intermediários.

Calculadora de 4 funções de Leibniz

**1738 – ANDROIDES PROGRAMÁVEIS** – O cientista francês **JACQUES VAUCANSON** criou (robôs imitando a aparência humana). Eram capazes de tocar flautas. Sua criação mais famosa foi "O Pato". Esse pato mecânico era capaz de imitar todos os movimentos de um pato real (bater asas, movimentar a cabeça, fazer barulho equivalente, comer e evacuar. ( www.automates-anciens.com). Em **1749** ele construiu o primeiro **TEAR AUTOMÁTICO**, que aceitava comandos através de um cilindro de metal perfurado.

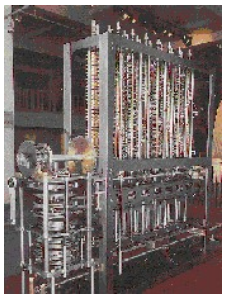


Tear de Vaucanson

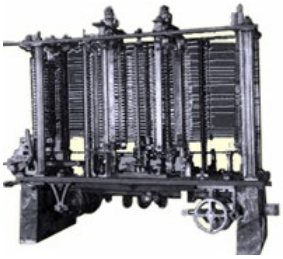
Tear de Jacquard

**1801 – CARTÃO PERFURADO –** O Tecelão francês **JOSEPH MARIE JACQUARD** aperfeiçoou o tear construído por **Vaucanson**. Ele construiu uma máquina de tear que memorizava em cartões perfurados os padrões de desenho dos tecidos e depois os reproduzia com fidelidade, lendo comandos na presença ou ausência de orifícios. A versão seguinte do Tear, em **1804**, era totalmente automatizada e podia fazer desenhos muito complicados. Esse é considerado o primeiro registro de programação semelhante à de computadores modernos.

**1822 – MÁQUINA DE DIFERENÇA e MÁQUINA ANALÍTICA –** Aborrecido pelos inúmeros e freqüentes erros que encontrava nas tabelas de *logaritmos*, o professor de matemática **CHARLES BABBAGE** (inglês) decidiu construir uma máquina que eliminasse o trabalho repetitivo de fazer esses cálculos, a "***Máquina de Diferença***". O modelo apresentado em **1822** encantou o Governo Britânico que decidiu financiá-lo na construção de uma máquina de diferença completa, movida a vapor e completamente automática, comandada por um programa de instrução fixo capaz de imprimir as tabelas. Baseada em operações de adição e subtração e na técnica de diferenças finitas, era capaz de resolver funções polinomiais e trigonométricas (cálculo de tabelas de navegação).



Máquina de diferenças



máquina analítica

O projeto da sua nova máquina levou 10 anos e foi abandonada em **1833**, quando decidiu criar a "***Máquina Analítica***", um computador mecânico-automático totalmente programável, função que designou para sua esposa, a condessa ***Ada Lovelace*** (filha de **Lord Byron**). O novo computador decimal paralelo a vapor operaria números de 50 dígitos e proveria de uma memória de 1000 números, usando cartões perfurados e condicionais (IF), além de instruções de desvio. Apesar de ter uma estrutura correta, a metalurgia da época não permitia a simetria e resistência das peças, razão ao qual a máquina nunca funcionou. Seria capaz de fazer uma adição em 1 segundo e uma multiplicação em 1 minuto.

**1885 - O CARTÃO DE HOLLERITH –** ***Herman Hollerith*** , funcionário do Departamento de Estatística dos Estados Unidos, construiu uma máquina de cartão perfurado para fazer o recenseamento da população americana. Antes da máquina o recenseamento durava 7 anos e ocupava 500 empregados. Com a máquina o recenseamento de 1890 durou 1 ano e ocupou 43 empregados. A máquina foi aproveitada nas mais diversas aplicações em repartições públicas, comércio e indústria, e aperfeiçoada para realizar operações aritméticas elementares. Em 1896 *Hollerith* fundou a TMC (***T**abulation **M**achine **C**ompany*).

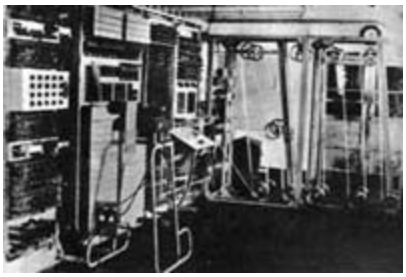
Para ampliar seus negócios, a **TMC** se uniu com duas pequenas empresas para formar a **CTRC** (***C**omputing **T**abulation **R**ecording **C**ompany*), em 1914. Em 1924, a **CTRC** se tornou uma empresa internacional e mudou seu nome para **IBM** (***I**nternacional **B**usiness **M**achine*).

Máquina de Herman Hollerith

Z4

**1936 – COMPUTADORES Z1, Z3 e Z4 –** O cientista alemão KONRAD **ZUSE** criou o computador - Z1, baseado em relé eletro-mecânico. Criou também o computador Z3, que foi o primeiro computador de propósito geral controlado por programa. Criou ainda o Z4, computador projetado para o desenvolvimento de mísseis. Ele foi destruído por bomba na 2ª. guerra mundial.

**1943 – COLOSSO –** Na Inglaterra, em 1943, ***Alan Turing***, do Serviço de Inteligência Britânico, construiu o ***Colosso***, de dimensões gigantescas. A máquina, abrigada em ***Bletchley Park***, tinha 2000 válvulas e lia símbolos perfurados numa argola de fita de papel, inserida na máquina de leitura fotoelétrica, comparando a mensagem codificada com sequências conhecidas até encontrar uma coincidência. Processava cerca de 5 mil caracteres por segundo e foi usada para descodificar as mensagens dos alemães, tendo sido decisiva no resultado final da guerra.



Colosso

Mark I

**1944 – MARK I –** Na Universidade de Harvard em 1937, o professor ***Howard Aiken***, financiado pela IBM, começou a construir o ***Mark I***, concluído em 1944. Baseado em um sistema decimal, manipulava números de até 23 dígitos e tinha medidas grotescas: 15 m de comprimento e 2,5 m de altura; 760.000 peças envoltas em vidro e aço inoxidável brilhante; 800 km de fios e 420 interruptores para controle.trabalhava sob o controle de um programa perfurado em uma fita de papel. Adição e subtração em 0,3 s, multiplicação em 3 s e divisão em 12 s,

**1946 – ENIAC –** (*Electronic Numerical Integrator and Computer*) - 1º Computador de propósito geral a válvula: 18.000 válvulas, 30 toneladas, 15.000 pés quadrados, 140 kW, representação e aritmética com números decimais, 5.000 adições/seg. Projetado pela *Ballistics Research Labs*. Foi aproveitado no desenvolvimento da Bomba "H".

**1946 – VON NEWMANN MACHINE –** A Máquina de Von Newman, ou "Máquina de Touring" introduziu o conceito de programa armazenado (Stored Programa Concept) no qual a memória conteria, além de dados, programas. Os computadores modernos são baseados na máquina de Von Newman.

**1950 – UNIVAC** – (Universal Automatic Computer) – Lançado pela SPERRY, foi o 1º Computador de aplicação científica e comercial. Seguiram **UNIVAC II** e **UNIVAC 1100** *series* .

**1953 – IBM 701 –** Computador desenvolvido para aplicações científicas.

**1947 – TRANSISTOR –** Invenção do transistor pelos cientistas John Bardeen, William Shockley e Walter Brattain. Passou a ser usado em escala comercial somente em 1952 pela *Bell Laboratories*.

**1958 – CIRCUITO INTEGRADO –** O engenheiro Jack Kilby, da Texas Instruments, criou o Circuito Integrado.

**1960 – IBM 7090, 7094 –** Computador transistorizado**.** Utilização de linguagens de programação de alto nível, tais como FORTRAN, COBOL e PASCAL.

**1964 – IBM 360 –** Primeira família planejada de computadores.

**DEC PDP 8 –** Introduziu o conceito de Minicomputador. Criou a estrutura de barramento, ou seja, unidadade de Entrada e Saída, Memória e CPU interligados por um conjunto de condutores.

**1971 – 4004 (INTEL)** - 1º microprocessador a ser lançado, de 4 bits, com aplicação voltada para calculadoras (manipulação de números em BCD) - 45 instruções - 640 Bytes de memória - clock de 108 KHz - 60.000 instruções/ seg. (OBS: desempenho superior ao ENIAC) - 2.300 transistores.

**1972 – 8008 (INTEL)** - 1º microprocessador de 8 bits, com aplicação voltada para terminais (que trabalham com caracteres - codificação ASCII) - 48 instruções - 16KB de memória - clock de 200 KHz - 300.000 instruções/ seg. 3500 transistores.

**1974 – 8080 (INTEL)** - Processador de 8 bits, de propósito geral - 72 instruções - opera com 12V - clock de 2 MHz - 640.000 instruções/s. 64KB de memória. 6.000 transistores.

**1975** – **Z80 (ZILOG), 6502 (MOS)** – Utilizado pelo 1º APPLE (APPLE 1) em 1976 por Steve Wozniak e Steve Jobs (data da fundação da APPLE).

**1976 – 8085 (INTEL)** – "8080" operando com 5V - 2 instruções a + que o 8080 - melhor performance. 5 MHz - 370.000 instruções/s. 6500 transistores.

**1978 – 8086 (INTEL)** - Processador 16 bits (barram. externo de 16 bits e registradores de 16 bits). 5 MHz - 0.33 MIPS, 8 MHz - 0.66 MIPS e 10 MHz - 0.75 MIPS. 29.000 transistores.

**1979 – 8088 (INTEL)** - Processador 16 bits (barram. externo de 8 bits e registradores de 16 bits) - 133 instruções - chip utilizado no primeiro PC em 1981. O PC/XT seria lançado em 1983 com HD de 10 MB e 128 Kbyte RAM. 29.000 transistores. Lançado o 68.000 (MOTOROLA) que foi utilizado no Machintosh em 1984

**1980 –** Coprocessador **8087** (processador matemático).

**8051 (INTEL –** Lançado o microcontrolador 8051: microprocessador + periféricos (RAM, ROM, Serial, Timer, Controlador de Interrupção, etc.) num único chip, voltado para aplicações de controle

**1982 – 80186/188 - 80286 - 80287 (INTEL) –** PC/AT – 16 bits, modo protegido, 24 linhas endereços.

**1985 – 80386 (INTEL) –** Processador de 32 bits - bus ext. de dados de 32bits - 275.000 transistores. 16MHz - 2.5 MIPS, 20 MHz - 2.5 MIPS, 25 MHz - 2.7 MIPS, 33 MHz - 2.9 MIPS.

**1989 – 80486 (INTEL)** - Processador de 32 bits: "386" que incorpora o 387 (coprocessador), cache interna (L1) de 8KB e maior performance - 235 instruções - 1,2 milhões de transistores. 25 MHz - 20 MIPS, 33 MHz - 27 MIPS, 50 MHz - 41 MIPS.

**1991 – WEB – Tim Berners-Lee** desenvolve a Rede Mundial de Computadores (World Wide Web). O primeiro servidor Web é lançado. O conceito de conexão de vários usuários a um único computador por via remota nasceu no MIT no final da década de 50 e início da década de 60. As idéias básicas da Internet foram desenvolvidas em 1973 por **Bob Kahn** e **Vint Cerf**.

**1993 – Pentium 60 MHz e 66 MHz** - Processador de 32 bits – bus ext. de 64 bits - 5V - 3 milhões de transistores. Primeiro processador de 5ª geração.

**1994 – Pentium 90 MHz e 100 MHz** - Alimentação de 3,3V (maior confiabilidade). 3.2 milhões de transistores.

**1996 – Pentium Pro 200** - Incorpora cache L2 de 256kB, utilizando tecnologia MCM (*Multi-Chip Module*) - 5 milhões de transistores - idealizado para programas de 32 bits. Usa memória de 64 bits.

**1997 – Pentium 200MMX** (*Pentium MultiMidia eXtensions*): contém 57 novas instruções dedicadas para programas de Multimídia. 4.5 milhões de transistores. 200 MHz e 166 MHz. Barramento de 64 bits. Cada instrução MMX equivale a várias instruções comuns.

**1997 – Pentium II 233, 266, 300MHz** – utiliza o slot I. 7,5 milhões de transistores (tecnologia 0.35 micron), cache L2 com 512kB - 242 pinos - 64GB de memória endereçável. Poder de processamento de 32 bits do Pentium Pro e maior eficiência no processamento de 16 bits. Instruções MMX.

**1998 – Pentium II 450 MHz** - Cache L2 de 512 kB, 7.5 milhões de transistores, tecnologia 0.25 micron, barramento de 64 bits. 64 GB de memória endereçável.

**1999 – Pentium III 450 e 500 MHz** (até 1,2 GHz) – Barramento de sistema de 100 MHz ou 133 MHz, cache L2 de 512 kB, processador de 32 bits, 9,5 milhões de transistores, tecnologia 0.25 micron, 64 GB de memória endereçável. 70 novas instruções voltadas para multimídia e processamento 3D.

**2000 – Pentium IV** – até 2 GHz, barramento de sistema de 400 MHz, Cachê L1 de 32 kB e L2 de 256 kB, 42 milhões de transistores.

### Alguns Exemplos de Aplicação de Microprocessadores e Microcontroladores

Microcomputadores, Calculadoras, Relógios Digitais, Controle de Fornos Micro-ondas, Lavadora de Roupas, Video Games e outros brinquedos, Controle de Motores, Controle de Tráfego, Alarmes e Sistemas de Segurança, Telefone Celular.

## 1.2 Número de Transistores em um Microprocessador

O número de transistores num único microprocessador é um dos indicadores da evolução acentuada nessa área nos últimos anos. Dr. Gordon E. Moore, co-fundador da Intel, afirmou em 1965 que o número de transistores em um microprocessador dobraria a cada dois anos. O gráfico e a tabela a seguir confirmam a previsão de Moore nos últimos 30 anos.

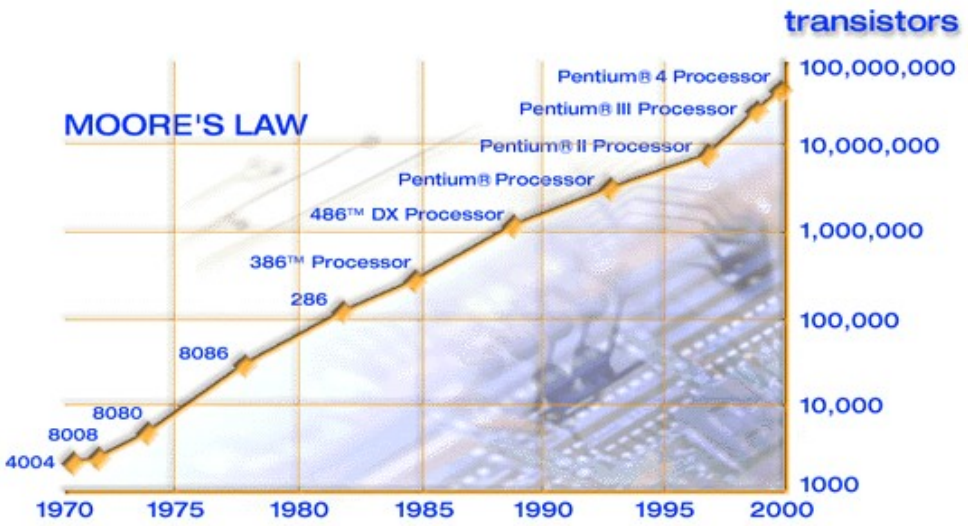



Fig. 1.1: Evolução do número de transistores no microprocessador

Tabela: Evolução do número de transistores no microprocessador

| Processador | Ano de Introdução | Número de transistores | Barramento de dados (bits) | Capacidade de endereçamento |
|---|---|---|---|---|
| 4004 | 1971 | 2.250 | 4 | 1 kB |
| 8008 | 1972 | 2.500 | 8 | 16 kB |
| 8080 | 1974 | 5.000 | 8 | 64 kB |
| 8085 | 1976 | 6.500 | 8 | 64 kB |
| 8086 | 1978 | 29.000 | 16 | 1 MB |
| 8088 | 1979 | 29.000 | 16 | 1 MB |
| 286 | 1982 | 120.000 | 16 | 16 MB |
| 386™ | 1985 | 275.000 | 32 | 4 GB |
| 486™ DX | 1989 | 1.180.000 | 32 | 4 GB |
| Pentium® | 1993 | 3.100.000 | 32 | 4 GB |
| Pentium II | 1997 | 7.500.000 | 32 | 4 GB |
| Pentium III | 1999 | 24.000.000 | 32 | 4 GB |
| Pentium 4 | 2000 | 42.000.000 | 32 | 4 GB |

A quantidade cada vez maior de transistores numa única pastilha foi acompanhada da redução do tamanho físico dos transistores. Essa redução é mostrada na Fig. 1.2.

A redução do tamanho do transistor resulta no aumento da velocidade de operação e também na redução das conexões internas, além de permitir a inserção de um número cada vez maior de transistores numa única pastilha. O aumento da capacidade de integração de transistores resulta ainda na redução do consumo de energia elétrica e do custo dos microprocessadores. Há um postulado que diz que o gate de um transistor não pode ser menor do a largura correspondente a 10 átomos. A previsão de pesquisadores da Intel é a dimensão do gate dos transistores alcançarão esse valor por volta do ano 2017 (http://www.intel.com/update/archive/issue2/focus.htm).

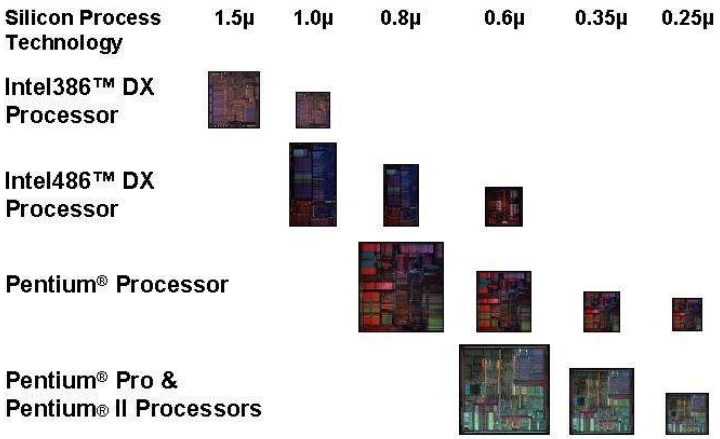



Fig. 1.2: Redução do tamanho físico dos processadores

## 1.3 Definições e Classificações Básicas

### ➢ COMPUTADOR (definições):

- Máquina destinada a reduzir cômputos (cálculos) e tomadas de decisões;
- Máquina que executa uma seqüência de operações (descritas por um programa) sobre um determinado dado e produz uma saída;
- Dispositivo com a habilidade de ser programado para operar (armazenar, relembrar e processar) dados sem a intervenção humana.

- **COMPUTADOR (classificação quanto à velocidade de processamento, número de registradores da CPU, complexidade do sistema operacional etc ):**

**Microcomputador**:
- Computador que tem a CPU implementada em um único chip: **o microprocessador**

**Minicomputador**:
- Multi-usuário;
- grande capacidade de armazenamento;
- operação com matrizes e ponto flutuante melhorada;

**Mainframes ( computadores de grande porte)** :
- suporta grandes bases de dados (organizações governamentais, grandes empresas);
- alta performance;
- processamento distribuído;

**Supercomputador**:
- Computador idealizado para resolver problemas matemáticos de processos reais, tais como: aerodinâmica, meteorologia, física, etc
- altíssima performance (GFLOPs) para repetidas operações aritméticas (iteração);
- operações com matrizes e números em ponto flutuante;
- mercado limitado.

- **COMPUTADOR (classificação quanto à grandeza manipulada):**

**Computadores Analógicos**:
- operam diretamente com grandeza física (variáveis contínuas);

**Computadores Digitais**:
- operam com variáveis discretas (números).

- **COMPUTADOR (funções básicas):**

- processamento de dados (*ex.: execução de uma adição ou de uma função lógica*);
- armazenamento de dados (*ex.: armazenamento temporário na memória RAM, Disco, DAT, etc.*);
- movimentação de dados (*comunicação com mundo exterior: teclado, monitor, impressora*);
- controle (*controle das funções anteriores*).

## 1.4 Estrutura Básica de um Computador

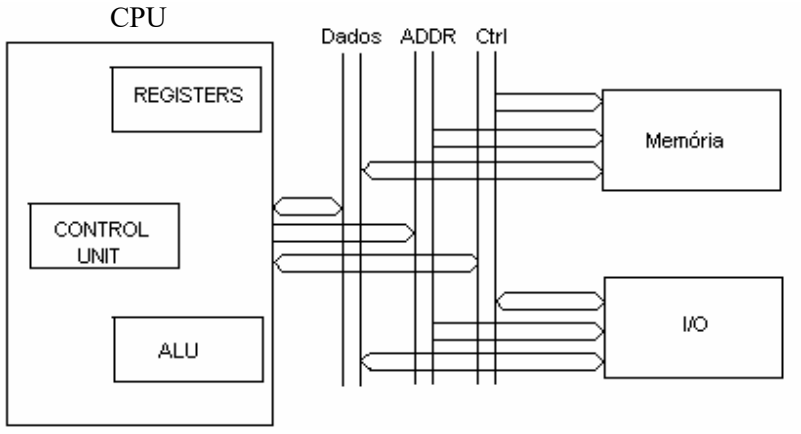



Fig. 1.3: Estrutura básica de um computador

### 1.4.1 Unidade Central de Processamento (CPU)

A Unidade Central de Processamento é a responsável por buscar e executar instruções, que ficam armazenadas na memória. A CPU possui internamente os seguintes blocos, dentre outros:

**Control Unit -** Unidade de Controle (UC) - tem por função básica o controle das demais unidades da CPU de uma forma lógica e sincronizada.

**ALU (Aritmetic and Logic Unit) -** Unidade Lógica e Aritmética (ULA) - realiza funções básicas de processamento de dados (adição, subtração, funções lógicas, etc.).

**Registers -** Registradores - São usados para o armazenamento interno da CPU. Existem diversos registradores na CPU e o principal deles é chamado de Acumulador.

**CPU interconnection -** É o barramento interno da CPU; ele permite a comunicação entre a Unidade de Controle, a Unidade Lógica e Aritmética e os Registradores.

### 1.4.2 Memória

É o local de armazenamento de dados e programas. Possui palavras de tamanho fixo, sendo cada palavra vinculada a um endereço único. Possui ainda linhas de controle, sendo as principais: READ (leitura) / WRITE (escrita).

Existem dois tipos básicos de memória: Memória somente para leitura (**ROM**), onde ficam armazenados permanentemente informações fundamentais para o funcionamento do computador e cujos dados não são perdidos na falta de energia; e a memória **RAM**, que permite gravar e apagar dados de acordo com os interesses do usuário, e cujo conteúdo é perdido quando o computador é desligado. A memória RAM divide-se ainda em Memória Dinâmica (**DRAM**) e memória estática (**SRAM**), que serão melhor detalhadas posteriormente. A principal característica a ser destacada neste ponto é a baixa velocidade de acesso da memória RAM.

Com o passar dos anos os processadores tornaram-se cada vez mais rápidos, o mesmo não acontecendo com as pastilhas de memória, que evoluíram de forma bem menos acentuada (em particular, a memória dinâmica, que possui velocidade de acesso bem menor que a estática, mas é bem mais barata). Para evitar com que a baixa velocidade de acesso da memória comprometesse o desempenho dos processadores mais modernos, um tipo especial de memória RAM foi criado: a **memória CACHE.**

A memória CACHE consiste numa pequena quantidade de memória RAM estática (SRAM) usada para acelerar o acesso à RAM dinâmica. Quando há necessidade de ler dados da memória dinâmica, estes são antes transferidos para a memória cache. Enquanto o processador lê dados da memória cache, mais dados são antecipadamente transferidos da memória dinâmica para a memória cache, de forma que o processamento torna-se mais rápido.

### 1.4.3 Unidade de Entrada e Saída (I/O)

É a unidade através da qual o usuário se comunica com o sistema. Ela abriga componentes responsáveis pelo interfaceamento do sistema com periféricos tais como teclado, LCD, mouse, impressora e monitor. É também através da unidade de entrada e saída que são enviados sinais de interrupção para a CPU.

### 1.4.4 Barramento

Barramento é o meio físico usado para o transporte de um conjunto de sinais digitais usados para comunicação entre o processador, a memória e o meio externo. O barramento específico para a comunicação entre o processador e a memória é chamado de barramento de sistema. Para a comunicação com os periféricos os três tipos mais comuns de barramento hoje são: barramento ISA, usado para interfaces seriais, paralelas, interface para drivers e alto falante; barramento PCI, usado para interfaces IDE e USB; e barramento AGP, usado para placas de vídeo 3D de alto desempenho.

Um barramento é constituído de um barramento de dados, um barramento de endereços e um barramento de controle. O barramento de dados nos computadores mais modernos possui até 64 linhas (bits) e permite o fluxo bidirecional de dados. O microprocessador 8085, objeto de estudo na primeira parte do presente curso, possui 8 bits de dados e, por esta razão, é denominado de processador de 8 bits.

A quantidade de posições de memória que um computador pode acessar é ditada pela quantidade de bits do barramento de endereços. Um barramento com 32 bits pode acessar até 4.294.967.296 ($2^{32}$) posições de memória, o que corresponde a 4 GB de memória (4.294.967.296 = 4 × 1024 × 1024 × 1024 = 4 GB). Todos os processadores da classe Pentium possuem barramento de endereço com 32 bits. Os processadores Pentium II, Pentium III e Celeron possuem barramento de endereço de 36 bits, podendo então acessar até 64 GB de memória.

O barramento de controle de um computador comporta uma série de sinais com finalidades diversas. Alguns exemplos são: sinal RW que indica se a operação é uma leitura ou uma escrita, sinal MIO, que indica se a operação envolve a memória ou a unidade de entrada e saída, sinal de RESET, entradas das interrupções, sinal de CLOCK, etc.

**Barramento ISA** - O barramento **ISA** (**I**ndustry **S**tandard **A**rchitecture) é formado por slots de 8 e 16 bits existentes nas placas de CPU e foi originado no IBM PC, na versão de 8 bits, e aperfeiçoado no IBM PC AT, quando foi criada a versão de 16 bits. Permite transferência de dados em grupos de 8 ou 16 bits a um clock de 8 MHz. Embora possua velocidade de transferência pequena para os padrões atuais, o barramento ISA ainda é muito utilizado para placas tais como fax/modem, placas de som e placas de rede, cujos desempenhos não ficam comprometidos com a baixa velocidade de transferência do barramento.

**Barramento PCI** - O barramento **PCI** (**P**eripheral **C**omponent **I**nterconnect) foi desenvolvimento pela Intel, quando do desenvolvimento do processador Pentium. Ele opera com 32 ou 64 bits, apresenta taxa de transferência de até 132 MB/s, com 32 bits e possui suporte para o padrão **PnP** (**P**lug a**n**d **P**lay). Seu clock é geralmente de 33 MHz, para valores de clock interno acima de 150 MHz.

**Barramento AGP** - O barramento **AGP** (**A**ccelerated **G**raphics **P**ort) foi devolvido pela Intel com o intuito de aumentar a taxa de transferência entre a CPU e a placa de vídeo, melhorando o desempenho de operação com gráficos. Esse barramento foi incorporado à CPU de processadores Pentium II mais modernos. A principal vantagem do AGP é o uso de maior quantidade de memória para armazenamento de texturas para objetos tridimensionais, além de alta velocidade no acesso a essas texturas para aplicação na tela.

## 1.5 Índice de Desempenho de Processadores

O aumento de desempenho (velocidade de processamento) de processadores gira em torno de pontos chaves:

- Aumento de clock
- Aumento do número interno de bits
- Aumento do número externo de bits
- Redução do número de ciclos para executar cada instrução
- Aumento da capacidade e velocidade da memória cache
- Execução de instruções em paralelo

## 1.6 Microprocessador × Microcontrolador

**Microprocessador -** É a CPU de um computador construído num único Circuito Integrado. Contém essencialmente a unidade de controle, a unidade lógica e aritmética e registradores. Precisa de periféricos tais como memória e unidade de entrada e saída, para a formação de um sistema mínimo.

**Microcontrolador -** É um computador completo construído num único Circuito Integrado. Os microcontroladores são normalmente utilizados para aplicações específicas. Eles contêm normalmente facilidades tais como portas seriais, portas de entrada e saída paralelas, timers, contadores, controles de interrupção, conversor analógico para digital, memórias RAM e ROM.

## 1.7 Outros conceitos básicos:

**MIPS** - *Millions of Instructions Per Seconds* (Milhões de Instruções Por Segundo): É uma unidade de desempenho do microprocessador.

**FLOPS** - *FLOating point instructions Per Seconds* (Instruções com Ponto Flutuante Por Segundo). É também uma unidade de desempenho do microprocessador. Indica a capacidade de trabalhar com números decimais.

**Representação em Ponto Fixo** - Sistema numérico no qual o ponto está implicitamente fixo (à direita do digito mais a direita);

**Representação em Ponto Flutuante** - Sistema numérico no qual um número real é representado por um par distinto de numerais: uma mantissa (ou significante) e um expoente. Possibilita representação de números fracionários.

**Bit -** Abreviatura para 'Binary Digit', ou, Dígito Binário. Pode assumir valor 0, que corresponde a tensão O V, ou 1, que representa normalmente uma tensão de 5 V ou 3.3 V.

**Byte -** Conjunto (cordão) de 8 bits. É a unidade básica de dados nos computadores, que também utilizam alguns múltiplos de 8, tais como 16 bits (Word) e 32 bits (Dword).

BYTE

| nibble superior | | | | nibble inferior | | | |
|---|---|---|---|---|---|---|---|
| Bit 7 | Bit 6 | Bit 5 | Bit 5 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |

**Nibble, Word, Dword** – Palavra digital composta de 4, 16, 32 bits, respectivamente.

**Set de instruções** - Conjunto de Instruções. Conjunto de Mnemônicos (siglas que fazem lembrar uma ação) que representam todas as instruções do processador. Cada processador possui o seu set de instruções particular.

**CISC** - *Complex Instruction Set Computer*: Tecnologia atribuída às CPUs de um modo geral, que contém set de instruções complexo. O barramento de comunicação entre as unidades que compõem a CPU é comum a todas as unidades, ou seja, não há comunicação direta entre unidades, através de um barramento exclusivo.

**RISC** - *Reduced Instruction Set Computer*: Computador com set de instruções reduzido. Principais características:

- Conjunto de instruções limitado e simples;
- Grande número de registradores de propósito geral;
- Pipeline otimizado. Em outras palavras, há comunicação direta entre algumas unidades, através de barramento exclusivo, possibilitando, assim, o processamento paralelo de intruções.

**BIOS** - Basic Input/Output System – É o conjunto mínimo de instruções necessárias para a inicialização do computador. Também gerencia o fluxo de dados entre o sistema operacional do computador e os dispositivos periféricos conectados.

**Memória EPROM** - (Erasable Programmable Read Only Memory) - São memórias somente de leitura usadas para a gravação de programas. A gravação é feita através de uma gravadora específica e os dados gravados podem ser apagados através de raios ultravioletas. Pode-se repetir esse processo gravar/apagar por várias vezes.

**Memória EEPROM ou $E^2$PROM** - (Electrically Erasable Programmable Read Only Memory) - São memórias que podem ser usadas tanto para leitura quanto para escrita porque a gravação pode ser através de gravadora específica ou pelo sistema. São apagadas eletricamente. o é feita através de uma gravadora específica e os dados gravados podem ser apagados através de raios ultravioletas. Pode-se repetir esse processo gravar/apagar por várias vezes.

## 1.8 Sistemas de Numeração

### 1.8.1 Sistema Decimal

O sistema decimal utiliza 10 dígitos, que vão de 0 a 9. **Exemplo: $346_{10}$**

1º dígito: Armazena as unidades (ou $10^0 = 1$). No ex.: seis unidades (ou $6 \times 10^0$);
2º dígito: Armazena as dezenas (ou $10^1 = 10$). No ex.: quatro dezenas (ou $4 \times 10^1$);
3º dígito: Armazena as centenas (ou $10^2 = 100$). No ex.: três centenas (ou $3 \times 10^2$);

A soma destas parcelas eqüivale a: $6 + 40 + 300 = 346_{10}$

A ponderação é dada pelo número 10 elevado à potência representada pela coluna, sendo que a 1ª coluna da direita é 0.

### 1.8.2 Sistema Binário

O sistema binário é o sistema de numeração que o computador entende. Utiliza 2 dígitos, 0 e 1 ou (OFF e ON) ou (0V e 5V), ou (0V e 3,3V). **Exemplo: $11001011_2$**

1º dígito: Armazena o equivalente a $2^0$ (1). No ex.: $1 \times 2^0$
2º dígito: Armazena o equivalente a $2^1$ (2). No ex.: $1 \times 2^1$
3º dígito: Armazena o equivalente a $2^2$ (4). No ex.: $0 \times 2^2$
…
8º dígito: Armazena o equivalente a $2^7$ : No ex.: $1 \times 2^7$

A soma destas parcelas resulta no seguinte equivalente decimal:

$1 + 2 + 0 + 8 + 0 + 0 + 64 + 128 = 203_{10}$

A ponderação é dada pelo número 2 elevado à potência representada pela coluna, sendo que a 1ª coluna é 0, a segunda coluna é 1 e assim sucessivamente.

1 kbyte $= 2^{10} = 1.024$ bytes;
1 Mbyte $= 2^{10} \times 2^{10} = 1.048.576$ bytes = 1.024 kbytes;
1 Gbyte $= 2^{10} \times 2^{10} \times 2^{10} = 1.073.741.824$ bytes = 1.024 Mbytes

### 1.8.3 Sistema BCD (Binary-Coded Decimal)

O Sistema BCD é o sistema em que se combina o sistema binário e o sistema decimal. É utilizado como formato de saída de instrumentos. Utiliza 2 dígitos: 0 e 1 que são dispostos em grupos de 4 dígitos, utilizados para representar um dígito decimal (número 0 até 9). A representação de um número maior que 9 deve ser feita por outro grupo de 4 bits, com a ponderação dada pelo sistema decimal. **Exemplo: $973_{10}$ = 1001 0111 0011.** Note a diferença entre este valor e o valor do número binário $100101110011_2 = 2419_{10}$

### 1.8.4 Sistema Octal

O Sistema Octal é baseado nos mesmos princípios do decimal e do binário, apenas utilizando base 8. Utiliza 8 dígitos: 0 a 7. **Exemplo: $3207_8$**

1º dígito: Armazena o equivalente a $8^0$ (1). No ex.: $7 \times 8^0$
2º dígito: Armazena o equivalente a $8^1$ (8). No ex.: $0 \times 8^1$

3º dígito: Armazena o equivalente a $8^2$ (64). No ex.: $2 \times 8^2$
4º dígito: Armazena o equivalente a $8^3$ (512). No ex.: $3 \times 8^3$

O equivalente decimal é: $7 + 0 + 128 + 1536 = 1671_{10}$

### 1.8.5 Sistema Hexadecimal

O Sistema Hexadecimal é baseado nos mesmos princípios do decimal, apenas utiliza base 16. Utiliza 16 dígitos: 0 a 9, A, B, C, D, E e F. Exemplo: **Ex.: 20DH ou 20Dh ou $20D_{16}$**

1º dígito: Armazena o equivalente a $16^0$ (1). No ex.: $13 \times 16^0$
2º dígito: Armazena o equivalente a $16^1$ (16). No ex.: $0 \times 16^1$
3º dígito: Armazena o equivalente a $16^2$ (256). No ex.: $2 \times 16^2$

O equivalente decimal é: $13 + 0 + 512 = 525_{10}$

O sistema hexadecimal é mais fácil de trabalhar que o sistema binário e é geralmente utilizado para escrever endereços. Cada dígito hexadecimal é convertido em 4 dígitos binários equivalentes. Cada número binário é convertido em hexadecimal convertendo-se grupos de 4 bits em seus dígitos hexadecimais equivalentes.

**Ex.: 7 D 3 $F_{16}$ =** $0111\ 1101\ 0011\ 1111_2$

**Ex.: $1010000110111000_2$ =** $1010\ 0001\ 1011\ 1000_2 = A\ 1\ B\ 8_{16}$

## 1.9 Exercícios Propostos

1 - Quais as funções básicas de um computador?

2. Em que consiste a CPU de um computador?

3. Como é feita a conexão da CPU com as demais unidades de um computador?

4. Diferencie memória DRAM de memória SRAM

5. Qual a finalidade da memória CACHE?

6. Quantas posições de memória podem ser endereçadas com um barramento de endereços de 16 bits? E de 20 bits?

7. Como os indicadores abaixo contribuem para o aumento do desempenho de um processador?
   (a) aumento da freqüência de clock
   (b) redução do número de ciclos para executar uma instrução
   (c) processamento paralelo de instruções

8. Qual a diferença entre microprocessador e microcontrolador?

9. Diferencie bit, nibble e byte.

10. Diferencie EPROM e EEPROM.

11. Converta os números abaixo (que estão na base indicada) para hexadecimal

(a) $345_{10}$ = _____ h
(b) $10100111_2$ = _____ h
(c) $10101010_8$ = _____ h
(d) $255_{10}$ = _____ h
(e) $128_{10}$ = _____ h
(f) $101001110111_2$ = _____ h

## 1.10 Referências Bibliográficas

[1] Ziller, Roberto M., "Microprocessadores – Conceitos Importantes," Edição do Autor, Florianópolis, SC, 2000.

[2] http://www.intel.com/research/

[3] www.maebee.com.br

[4] http://www.intel.com/

[5] http://www.computerhistory.org/timeline/

# 2. ARQUITETURA E PRINCÍPIO DE FUNCIONAMENTO DO 8085

## 2.1 Diagrama de Blocos do Microprocessador 8085

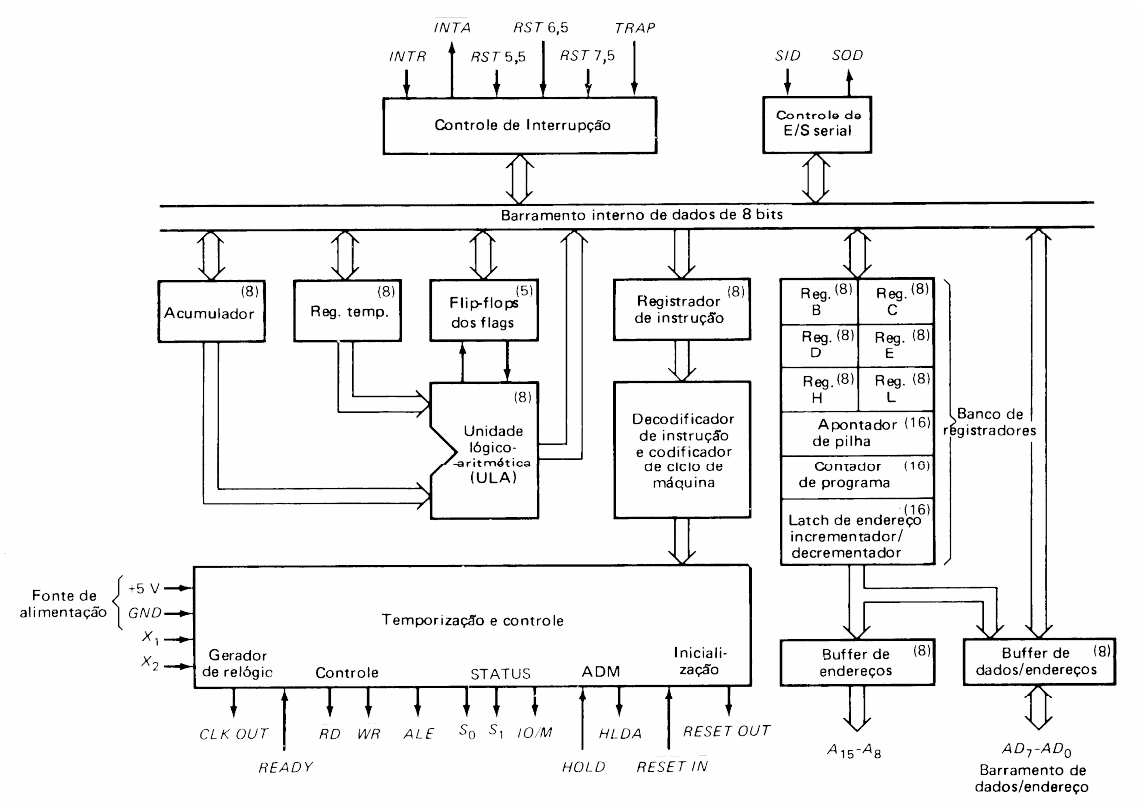



Fig. 2.1: Diagrama de blocos do microprocessador 8085

## Principais Características:

- microprocessador de 8 bits de propósito geral (com 6.200 transistores);
- opera com +5V e GND. O 8080, seu antecessor, opera com +12V, +5V e -5V;
- 100% compatível em software com o 8080A;
- conjunto de instruções com 74 instruções. Estas 74 instruções resultam num total de 246 opcodes distintos;
- instruções do 8085 possuem 1, 2 ou 3 bytes;
- Há 2 registradores temporários de 8 bits (**W e Z**) não aparentes ao programador (não endereçáveis);
- 8 linhas de dados: barramento bidirecional e com 3S (*three state*);
- 16 linhas de endereço; permite endereçamento de até 64 kbytes = 65.536 posições de memória;
- barramento de dados multiplexado com parte baixa do barramento de endereço (o hardware deve conter um latch (ex.: 74373) para armazenar os endereços baixos: $A_0$ a $A_7$);
- possui pino de seleção de Entrada (I) e Saída (O) - (IO/M)
- possui gerador de clock interno (é necessário apenas um cristal externo, juntamente com dois capacitores).
- Reset do 8085: PC em 0000h; Flip-Flop IE (*Interrupt Enable*) em 0 (indicando interrupções desabilitadas); HLDA em 0; demais registradores com valores indeterminados; Interrupções RST 5.5, RST 6.5, RST 7.5 mascaradas; SOD em 0.

## 2.2 Unidades Internas e Registradores do 8085

**Unidade "Controle de Interrupção"**

Unidade responsável pelo tratamento das 5 interrupções externas do 8085. Essas interrupções são vetoradas, o que significa que há um endereço fixo, pré-definido, para cada uma (RST n salta para a posição de memória 8 vezes n: RST 5.5 = $44_{10}$ = 2Ch; RST 6.5 = 34h; RST 7.5 = 3Ch.). As interrupções RST 5.5, RST 6.5 e RST 7.5 podem ser mascaradas, ou seja, elas podem ser bloqueadas via "software". Já a interrupção TRAP não pode ser bloqueada e é a interrupção de maior prioridade do 8085. Ela normalmente é ativada quando há problemas de falta de energia, para um desligamento seguro do microprocessador. A interrupção INTR, na verdade, é um canal para expansão da capacidade de interrupção. Através desse canal um CI especial (Exemplo: CI 8259) é conectado ao 8085, de modo a permitir um número maior de interrupções. O sinal INTA\ faz parte da comunicação entre o 8085 e o CI usado para expansão da capacidade de interrupção.

**Unidade "Controle de Entrada/Saída Serial"**

É através dessa unidade que o microprocessador recebe e envia dados de forma serial, ou seja, bit a bit, ao invés de um byte por vez. O pino SID (Serial Imput Data) é usado para a entrada de dados de forma serial e o pino SOD (Serial Output Data) é usado para a saída de dados de forma serial.

**Unidade "Temporização e Controle"**

Esta unidade é responsável por gerar todos os sinais de controle do 8085, tais como os sinais de leitura (RD\) e escrita (WR\) de memória, os sinais de liberação de barramento para um periférico (HLDA) e o sinal de habilitação de endereço para um periférico (ALE). Todas as unidades internas do 8085 são controladas por esta unidade, que contém, dentre outros, um contador em anel para sincronização da operação de todas as unidades do 8085. Os sinais de controle para outras unidades são enviados após a decodificação das instruções vindas do Registrador de Instruções (IR). Recebe ainda sinais do registrador de Flags e da unidade de interrupções.

**Unidade "Unidade Lógico-Aritmética (ULA ou ALU)"**

É responsável por todo o processamento realizado na CPU (execução de instruções aritméticas e lógicas). É controlada por sinais internos emitidos pela Unidade de Controle. Tem como entrada os registradores A (Acumulador) e TEMP (Temporário). É responsável pela sinalização de status das operações (FLAGS). É um registrador de 8 bits.

**Registrador "Acumulador"**

É o principal registrador da CPU. É utilizado como Buffer temporário de entrada da Unidade Lógica e Aritmética (ALU ou ULA). Freqüentemente é o registrador de entrada ou saída da CPU. É utilizado implicitamente na maioria das instruções. É um registrador de 8 bits, o que permite trabalhar com números sem sinal de 0 a 255 e números com sinal de -128 a +127. O resultado das operações resultantes da ULA é enviado para o Acumulador.

**Registrador "TEMP"**

É um registrador auxiliar usado para a entrada de dados da Unidade Lógico-Aritmética. Os dados desse registrador são enviados para a ULA juntamente com os dados do Acumulador.

**Registrador "Flip-flops dos flags"**

É também conhecido como registrador F (de Flags), ou registrador PSW (***P**rogram **S**tatus **W**ord*) É um registrador de 8 bits (mas somente 5 bits são utilizados) que armazena o estado da última operação realizada na ULA. São 5 as flags do 8085, conforme mostrado a seguir:

Registrador F

| bit 7 | bit 6 | bit 5 | bit 4 | bit 3 | bit 2 | bit 1 | bit 0 |
|---|---|---|---|---|---|---|---|
| **S** | **Z** | × | **AC** | × | **P** | × | **CY** |

**S = Flag de Sinal** - assume o valor 1 quando o resultado de uma operação é negativo

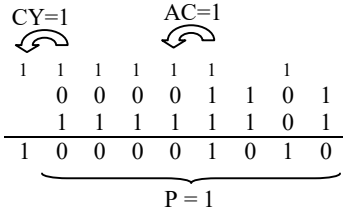
**Z = Flag de Zero** - assume o valor 1 quando o resultado de uma operação é zero

**AC = Auxiliar de Carry** = flag usada como auxiliar de transporte. Assume valor 1 quando há transporte do bit 3 para o bit 4. É usada em operações BCD

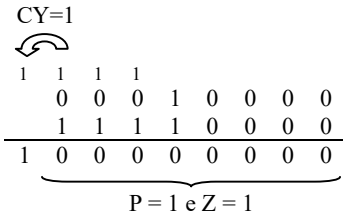
**P = Flag de Paridade** = assume valor 1 quando há um número de par de algarismos 1 no acumulador

**CY = Flag de Carry (transporte)** = assume valor 1 quando há transporte do bit 7.

Exemplos de operação e os Flags resultantes:




- Houve transporte do bit 3 para o bit 4, daí, AC = 1
- Houve transporte do bit 7 para o bit 8, daí, CY = 1
- Há quantitade par de bits "1" no resultado (o resultado está nos 8 primeiros bits porque todos os registradores possuem apenas 8 bits). Daí, P = 1.
- O resultado da operação é diferente de zero, daí, Z=0.
- O bit 7 do resultado da operação é zero. Daí, S = 0.




- Não houve transporte do bit 3 para o bit 4, daí, AC = 0
- Houve transporte do bit 7 para o bit 8, daí, CY = 1
- Há quantitade par de bits "1" no resultado (o resultado está nos 8 primeiros bits porque todos os registradores possuem apenas 8 bits). Daí, P = 1.
- O resultado da operação é igual a zero, daí, Z = 1.
- O bit 7 do resultado da operação é zero. Daí, S = 0.

**"Registrador de Instrução" (IR - Instruction Register)**

É um registrador de 8 bits que armazena o primeiro byte da instrução (OPCODE), ou seja, o conteúdo da memória apontado (endereçado) pelo registrado PC.

**Registrador "Decodificador de Instrução e Codificador de Ciclo de Máquina"**

É o registrador responsável pela decodificação de cada instrução e de definição dos ciclos de máquina que serão controlados pela unidade de controle.

**Registradores B, C, D, E, H e L**

São registradores de propósito geral de 8 bits e que podem ser combinados aos pares para formar registradores par (**rp**: *register pair*) para armazenar endereços (16 bits). Os pares formados são: **BC**, **DE** e **HL**. O primeiro registrador de cada par armazena o byte mais significativo, isto é, B, D e H.

**Registrador par HL**

Registrador usado como apontador de dados na memória RAM, à semalhança do registrador PC, que aponta instruções e dados na memória ROM (como será visto numa seção posterior). O registrador HL é usado implicitamente em várias instruções e é referenciado nessas instruções "M", de *Memory*.

**Registrador "Apontador de Pilha"**

O registrador apontador de pilha **SP** (Stack Pointer) é um registrador de 16 bits usado como apontador de dados numa região especial da memória RAM, denominada de **Pilha** (**Stack**). Esse espaço de memória é especialmente destinado a guardar temporariamente informações de registradores que serão usados em outra tarefa. A ordenação de elementos na pilha é tal que somente um dado pode ser acessado num determinado instante e a última palavra digital que entra é a primeira que sai (Lista **LIFO** - ***L**ast **I**n **F**irst*

***Out***). O apontador de pilha (registrador SP) aponta sempre para o topo desta pilha (*top of stack*), ou seja, para o último dado que foi armazenado. Os dados normalmente armazenados são endereços de chamadas/retornos de subrotina e endereços de retorno de interrupções, que automaticamente armazenados pelo 8085 e ainda outros dados que podem ser armazenados pelo programador usando a instrução **PUSH**. Posteriormente esses dados são retirados da pilha usando a instrução **POP**.

**Registrador "Contador de Programa"**

O registrador "Contador de Programa" **PC** (**P**rogram **C**ounter) é o registrador que armazena o endereço da próxima instrução a ser executada. É incrementado pela unidade de controle após a execução de uma instrução. (OBS: as instruções estão localizadas na memória e precisam ser transferidas para dentro da CPU). Sendo um registrador de 16 bits o registrador PC pode indicar até 65536 diferentes endereços (0 a 65535 ou 0000h a FFFFh).

## 2.3 Frequência de Clock

A freqüência de clock do **8085A** deve ficar na faixa de **500 kHz** a **3,125 MHz**. Já a freqüência de clock do **8085A-2** deve ficar entre 500 kHz e 5 MHz. A freqüência de clock é metade da freqüência do cristal oscilador, como mostrado na expressão que segue. Isso ocorre porque o flip-flop que gera o clock divide o sinal de entrada por 2.

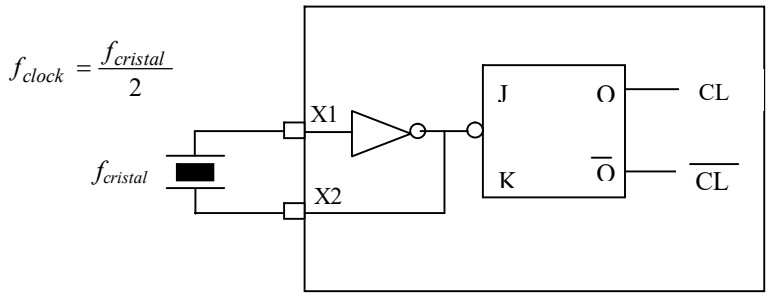



Fig. 2.2: Conexão do cristal oscilador para a geração da frequência de clock

## 2.4 Pinagem do 8085

A Fig. 2.3 mostra uma pastilha do microprocessador 8085 com a pinagem completa e a função de cada pino. A tabela a seguir mostra, através dos pinos IO/M\, S1 e S0, o estado em que se a encontra a CPU durante a execução de uma instrução.

**Estado do ciclo de máquina:**

| IO/M | S1 | S0 | ESTADO |
|---|---|---|---|
| 0 | 0 | 1 | escrita em memória |
| 0 | 1 | 0 | leitura de memória |
| 0 | 1 | 1 | busca de opcode |
| 1 | 0 | 1 | escrita em porta (instrução OUT porta) |
| 1 | 1 | 0 | leitura de porta (instrução IN porta) |
| 1 | 1 | 1 | reconhecimento de interrupção (INTA) |
| 3S | 0 | 0 | HLT (parada: sai com INT, HOLD ou RESET |
| 3S | x | x | Hold |
| 3S | x | x | Reset |

| | | | | | |
|---|---|---|---|---|---|
| Pinos de conexão do cristal. $X_1$ pode ser uma onda quadrada. Nesse caso, $X_2$ pode ficar aberto. | $X_1$ | 1 | 40 | $V_{CC}$ | Pino de alimentação. $V_{cc}$ = +5 V |
| | $X_2$ | 2 | 39 | HOLD | Requerimento de barramento. Ativo alto. |
| Indicas aos periféricos que a CPU está sendo ressetada | RESET OUT | 3 | 38 | HLDA | Reconhecimento de HOLD. Indica que pedido foi aceito. |
| Pino de saída de dado serial. | SOD | 4 | 37 | CLK (OUT) | Saída de clock para os demais chips do sistema |
| Pino de entrada de dado serial. | SID | 5 | 36 | $\overline{RESET\ IN}$ | Reset do sistema. PC = 0000 h. Interrupções com máscara. |
| Pino da interrupção de maior prioridade. Não mascarável. | TRAP | 6 | 35 | READY | 1 → memória ou perif. prontos para acesso. 0 → wait |
| Pino da interrupção que desvia para o endereço 3C h. | RST 7.5 | 7 | 34 | $IO/\overline{M}$ | Indica se a operação é de entrada/saída ou com memória |
| Pino da interrupção que desvia para o endereço 34 h. | RST 6.5 | 8 | 33 | $S_1$ | Com $S_0$ e IO/M\ indicam estados do ciclo de máquina |
| Pino da interrupção que desvia para o endereço 2C h. | RST 5.5 | 9 | 32 | $\overline{RD}$ | Sinal que habilita periférico ou memória para leitura |
| Pino usado para expandir a capacidade de interrupção. | INTR | 10 | 31 | $\overline{WR}$ | Sinal que habilita periférico ou memória para escrita |
| Reconhecimento de pedido de interrupção. | $\overline{INTA}$ | 11 | 30 | ALE | Sinal para carga da parte baixa do endereço (no latch 74373) |
| Byte menos significativo do endereço e barramento de dados. | $AD_0$ | 12 | 29 | $S_0$ | Com $S_1$ e IO/M\ indicam estados do ciclo de máquina |
| | $AD_1$ | 13 | 28 | $A_{15}$ | Byte mais significativo do endereço |
| | $AD_2$ | 14 | 27 | $A_{14}$ | |
| | $AD_3$ | 15 | 26 | $A_{13}$ | |
| | $AD_4$ | 16 | 25 | $A_{12}$ | |
| | $AD_5$ | 17 | 24 | $A_{11}$ | |
| | $AD_6$ | 18 | 23 | $A_{10}$ | |
| | $AD_7$ | 19 | 22 | $A_9$ | |
| Pino de terra (GND). | $V_{SS}$ | 20 | 21 | $A_8$ | |

Fig. 2.3: Pinagem do microprocessador 8085

## 2.5 Sistema Básico de Temporização e Princípio de Operação

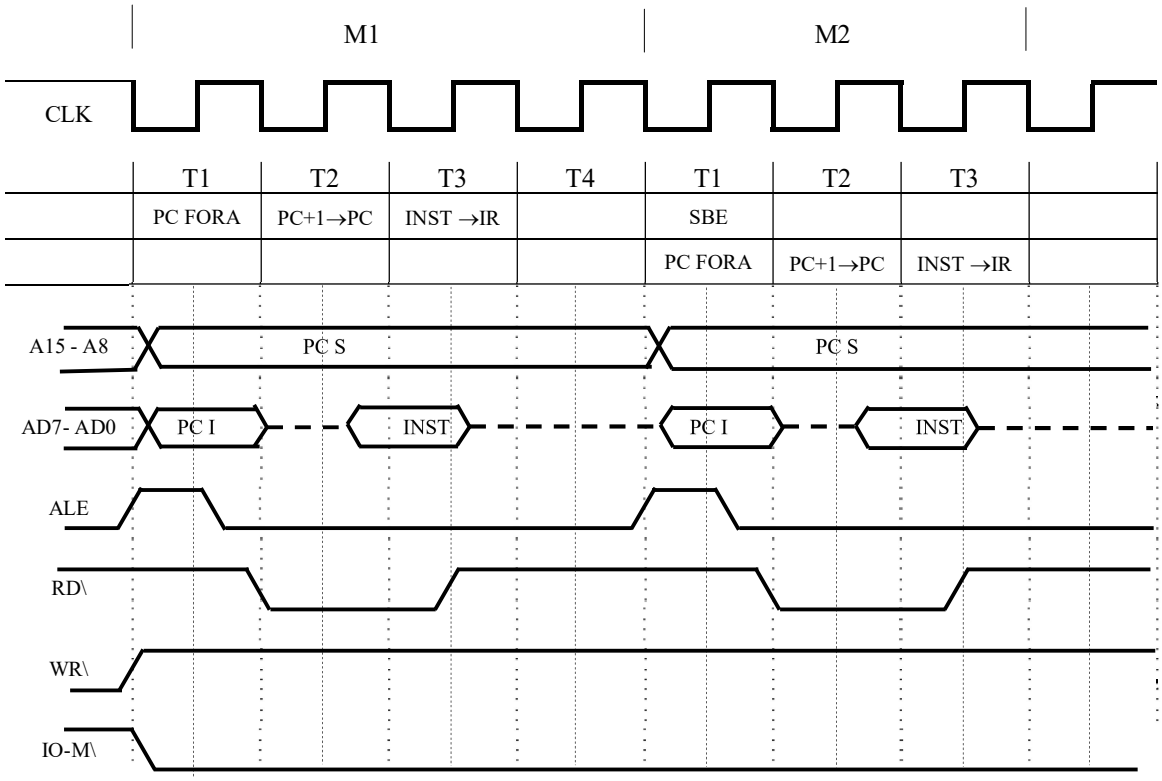



Fig. 2.4: Diagrama básico de temporização do 8085

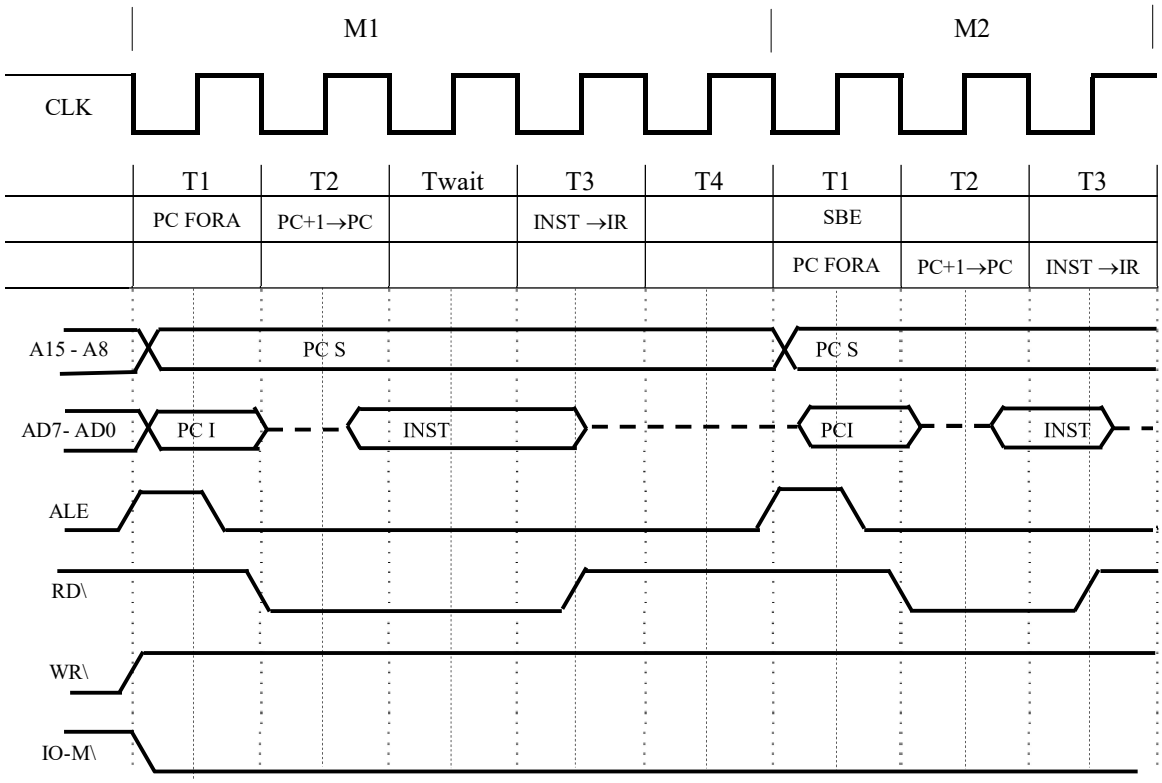



Fig. 2.5: Diagrama básico de temporização com período de espera Twait

O diagrama de temporização mostra os sinais essenciais no processo de busca e execução de cada instrução. Observa-se, por exemplo, que a cada novo ciclo de máquina, no estado **T1**, um novo valor de endereço é colocado no barramento pelo contador de programa **PC**. Esse novo endereço pode ser o endereço da próxima instrução ou os bytes restantes (ou o byte restante) da instrução em andamento. O sinal **ALE** é o sinal de habilitação do Latch do endereço, ou seja, o endereço é transferido para a memória **ROM**, de onde será lida a instrução.

O sinal **RD** vai a zero após o estado **T1** indicando que haverá uma operação de leitura. Como o sinal **IO-M** permanece zero durante todo o tempo, figura mostrada, significa que trata-se de leitura de memória, e não de qualquer dispositivo de entrada e saída. Assim, a memória **ROM** é lida nesse intervalo. O sinal **WR** permanece alto durante todo o intervalo mostrado, indicando que não há operação de escrita, seja na memória, seja em dispositivo de entrada e saída.

No estado **T3** a instrução lida da **ROM** é transferida para o Registrador de Instrução (**IR**), encerrando aí o ciclo de busca. A instrução é decodificada e sinais de controle são emitidos da Unidade de Temporização e Controle de forma que o ciclo de execução já tenha início a partir do estado **T4**.

Muitas das instruções do 8085, depois do ciclo de busca, precisam apenas de mais um ciclo de clock para o ciclo de execução. O ciclo de execução pode ocorrer no estado **T4**, ou no próximo estado **T2**, durante o cilco de busca da próxima instrução. No último caso diz-se que houve sobreposição dos ciclos de busca e execução (**SBE**).

No diagrama da Fig. 2.4 aparece um estágio a mais, denominado de Período de Espera ($T_{wait}$). Ele é gerado quando há necessidade de retardar o processo de busca da instrução, em função de atraso no processamento de informações por um periférico. Esse intervalo de espera aparece sempre após o estado **T2**.

Para exemplificar o princípio de operação do 8085 é mostrado a seguir o processo de busca e execução das instruções **ADD B** (adição do conteúdo do registrador **B** ao conteúdo do registrador A. O resultado é guardado em A) e **MOV B,A** ( move - copia - o conteúdo do registrador A no registrador **B** ).

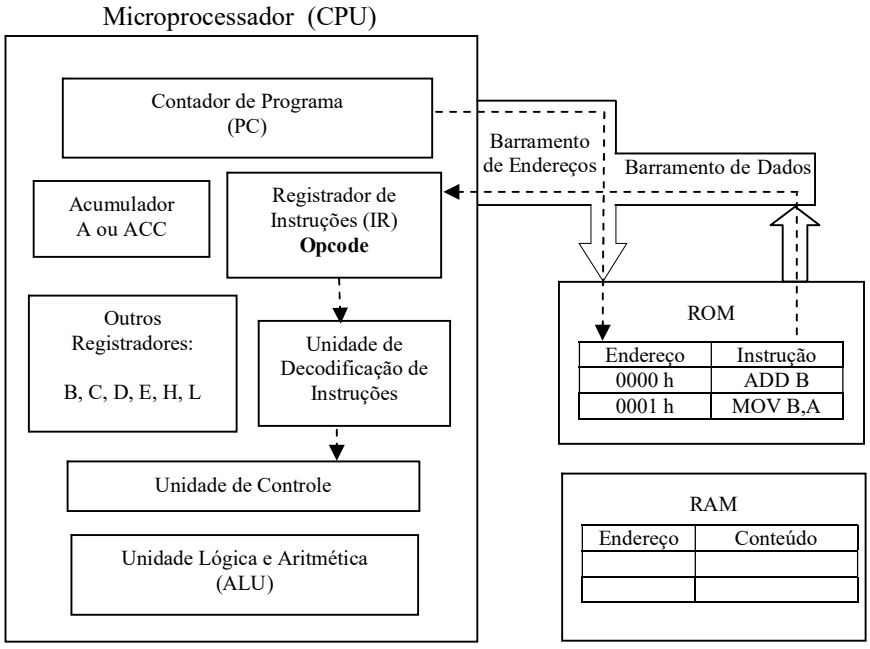



Fig. 2.6: Diagrama de blocos simplificado de um computador

O barramento de endereços é de 16 bits, sendo que os 8 bits menos significativos são usados também para a transferência de dados, enquanto que os 8 bits mais significativos são exclusivos para endereços. Assim, a chave de três estados desempenha um papel essencial na operação do microprocessador. Durante a comunicação entre dois blocos os outros blocos encontram-se no estado de alta impedância, não havendo, portanto, transferência de dados desses blocos para o barramento ou do barramento para estes blocos. O diagrama de temporização para as duas instruções é mostrado na Fig. 2.6.

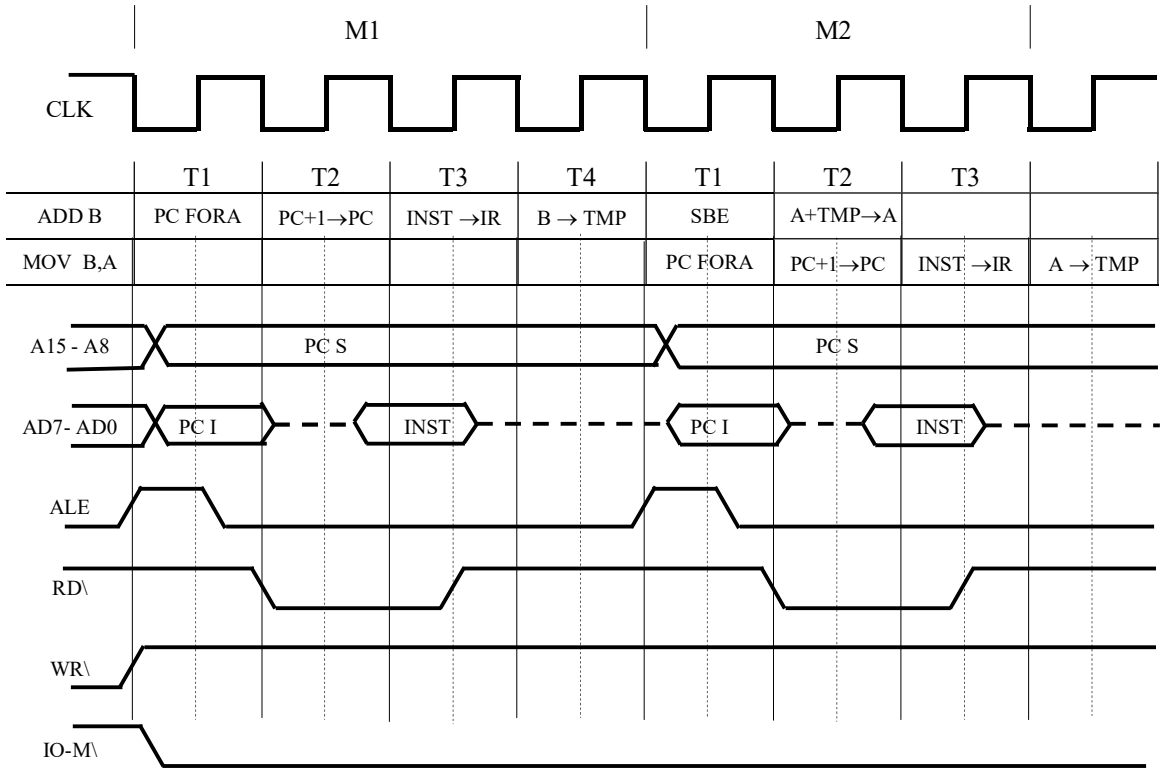



Fig. 2.7: Diagrama de temporização das instruções de adição e transferência

Supondo-se que o acumulador contenha o valor **10h** e que o registrador **B** (que faz parte do bloco denominado "outros registradores") contenha o valor **5h**, a adição do conteúdo de **B** ao conteúdo de **A** é obtido com a instrução **ADD B**. O resultado é guardado no próprio acumulador. Posteriormente, com a instrução **MOV B,A** o conteúdo de A é copiado em **B**. Os passos para execução das duas instruções em seqüência são dados a seguir:

**CICLO DE BUSCA:**

1 Na descida do sinal de clock em **T1** o Contador de Programa (**PC**) é ativado, colocando o endereço atual no barramento de 16 bits. O sinal **ALE** é o trigger para a transferência dos endereços de **PC** para o barramento.

2 Na subida do sinal de clock em **T1** o endereço é transferido do barramento para a memória **ROM**.

3 Na descida do sinal de clock (estado **T2**) o **PC** é incrementado em 1, ficando pronto para apontar a próxima instrução. O barramento está disponível para outras operações. Nesse instante, também, O sinal **RD** torna-se baixo, habilitando uma operação de leitura. Como o sinal **IO/M** permanece baixo, trata-se de leitura de memória.

4 Na subida do sinal de clock (ainda estado **T2**) nenhum bloco está ativo. O barramento continua disponível.

5 Na descida do sinal de clock (estado **T3**) o código da instrução **ADD B**, lida da memória **ROM** (endereço transferido do barramento para a memória ROM no passo 2) é transferido para o barramento.

6 Na subida do sinal de clock o bloco **IR** (Registrador de Instrução) carrega a instrução vinda da **ROM** e que está presente no barramento. **Termina o ciclo de busca** da instrução **ADD B**.

**CICLO DE EXECUÇÃO DE <u>ADD B</u> E BUSCA DE <u>MOV B,A</u>:**

7 Na descida do sinal de clock, no início do estado **T4**, o conteúdo de B é transferido para o barramento.

8 Na subida do clock no estado **T4** o conteúdo de **B**, presente no barramento, é transferido para um registrador temporário (**TMP**), para, depois, ser transferido para a unidade lógica e aritmética.

9 Durante o estado **T1** do ciclo de máquina **M2** não há nenhuma operação na instrução **ADD B**, daí uma outra instrução pode utilizar o barramento para iniciar o ciclo de busca. No caso mostrado dá-se início ao ciclo de

busca da instrução **MOV B,A**. Portanto, há "sobreposição busca-execução" (**SBE**). Busca da instrução **MOV B,A** e execução da instrução **ADD B**.

10 No estado **T2** do ciclo de máquina **M2** a instrução iniciada (**MOV B,A**) não usa o barramento de dados/endereços. Nessa fase a instrução **ADD B** é encerrada. Na descida do sinal de clock em **T2** os conteúdos de **A (10 h)** e do registrador temporário **TMP (B = 5 h)** são simultaneamente transferidos para a Unidade Lógica e Aritmética (**ULA**), de onde o resultado da adição sai direto para o barramento. Na subida do sinal de clock em **T2** o resultado da operação é carregado no acumulador (**A = 15 h**). **Encerra a execução da instrução ADD B**.

11 No estado **T3** do ciclo de máquina **M2** a instrução **MOV B,A** lida da **ROM** é transferida para o Registrador de Instruções e decodificada, encerrando o ciclo de busca dessa instrução.

12 No estado **T4** do ciclo **M2** o conteúdo de **A** é transferido para um registrador temporário para, posteriormente, ser transferido para o registrador **B**.

## 2.6 Formato das Instruções

As instruções de adição e de transferência de dados, mostradas na seção anterior, são instruções de apenas 1 (um) byte, que é chamado de **OPCODE** (Operation Code). Nesse caso toda a informação necessária para a execução das duas instruções está contida no byte único. Algumas instruções, no entanto, precisam de informações adicionais para sua execução. Assim, além das instruções de 1 byte, o 8085 também tem instruções de 2 bytes e de 3 bytes. Os bytes adicionais são chamados de **OPERANDOS**. No caso da instrução de **2 bytes** tem-se o **Opcode** e **1 operando** e no caso da instrução de **3 bytes** tem-se o **Opcode** e **2 operandos**. É mostrado a seguir o formato dos três tipos de instrução. Antes, porém, é importante mostrar o formato dos dados no 8085.

| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|
| MSB | | | | | | | LSB |

MSB = Most Significant Bit (Bit mais significativo)
LSB = Least Significant Bit (Bit menos significativo)

a) Instruções de 1 byte:

| Opcode (byte 1) | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|

Exemplo: **ADD B** - Adiciona o conteúdo de B ao acumulador
(A) ← (A) + (B)

| Opcode (byte 1) | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | = 80 h |
|---|---|---|---|---|---|---|---|---|---|

b) Instruções de 2 bytes:

| | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|
| Opcode (byte 1) | | | | | | | | |
| Operando (byte 2) | | | | | | | | |

Exemplo: **MVI A, 32h** - Move imediatamente o conteúdo 32h para A
(A) ← (byte 2)

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Opcode (byte 1) | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | = 3E h |
| Operando (byte 2) | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | = 32 h |

c) Instruções de 3 bytes:

| | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|
| Opcode (byte 1) | | | | | | | | |
| Operando 1 (byte 2=LSB) | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| Operando 2 (byte 3 = MSB) | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |

Os bytes 2 e 3 contêm um dado ou um endereço de 16 bits. O byte 2 armazena o byte menos significativo do endereço (low-order addr) ou o byte menos significativo do dado de 16 bits (low-order data). O byte 3

armazena o byte mais significativo do endereço (high-order addr) ) ou o byte mais significativo do dado (high-order data)

Exemplo: **STA 1234h** - guarda o conteúdo do acumulador na posição de memória indicada pelo endereço **addr**
((byte 3)(byte 2) ← (A)

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Opcode (byte 1) | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | = 32 h |
| Operando 1 (byte 2) | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | = 34 h |
| Operando 2 (byte 3) | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | = 12 h |

No ciclo de busca (primeiro ciclo de máquina) o microprocessador 8085 transfere o primeiro byte da instrução (**opcode** = código de operação) para o **Registrador de Instrução** (**IR**). Nos ciclos de máquina que se seguem os outros bytes são buscados na memória. Primeiro o byte 2 é transferido para um registrador temporário (**Z**), depois é o byte 3 que é transferido para um registrador temporário (**W**).

Cada uma das instruções pode ser visualizada com um diagrama de temporização que mostra cada passo da instrução.

O primeiro ciclo de máquina do 8085 é flexível podendo ter de 4 a 6 ciclos de clock; os demias ciclos de máquina contém 3 ciclos de clock. O ciclo de instrução pode ter de 1 a 5 ciclos de máquina, como mostrado a seguir, onde **M** é o ciclo de máquina e T é o estado dentro do ciclo de máquina:

| **M1** | | | | | | **M2** | | | **M3** | | | **M4** | | | **M5** | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| T1 | T2 | T3 | T4 | T5 | T6 | T1 | T2 | T3 | T1 | T2 | T3 | T1 | T2 | T3 | T1 | T2 | T3 |

## 2.7 Exercícios Propostos

1. Enumerar todos os registradores (endereçáveis ou não) do microprocessador 8085. Qual a capacidade (em bits) de cada um deles?

2. Quais os possíveis registradores pares do 8085?

3. Com que finalidade é usado o registrador-par HL?

4. Qual a função dos registradores PC e SP?

5. Qual a função dos registradores W e Z?

6. Qual a função do registrador TEMP?

7. Descreva a estrutura interna da CPU?

8. Qual a função de cada uma das FLAGS do 8085?

9. Dizer, em poucas palavras, a função de cada um dos seguintes pinos do 8085: **HOLD**, **HLDA**, **INTR**, **INTA**, **RST5.5**, **TRAP**, **READY**, **ALE**, **IO/M**, **SID** e **SOD.**

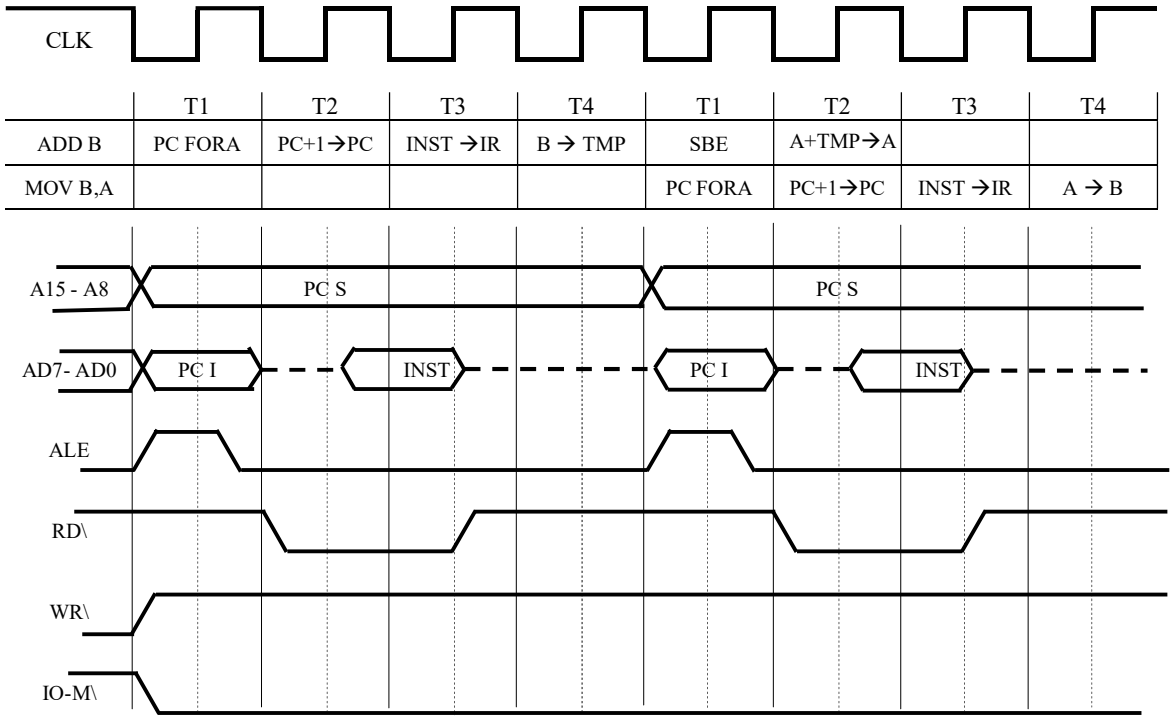
10. Apresentar um circuito combinacional para decodificar os sinais dos pinos **RD**, **WR** e **IO/M** em **MEM_RD**, **MEM_WR**, **IO_RD**, **IO_WR**. OBS: Utilizar apenas gates.

11. Codifique as instruções da tabela a seguir, coloque o endereço de cada instrução, escreva o significado de cada instrução e responda às questões relativas à tabela.

| **Operação** | **End.** | **Mnemônico** | **Comentário** |
|---|---|---|---|
| (SP) ← 2090 h | 2000 | | |
| (H,L) ← 2050 h | | | |
| (A) ← 53 h | | | |
| (B) ← 0F h | | | |
| (A) ← (A) + (B) | | | |
| ((H,L)) ← (A) | | | |

(a) Qual o valor final do registrador A?
(b) Quais os valores das flags de carry, de sinal, de zero, auxiliar de carry e paridade, após a última instrução da tabela?

12. O diagrama de temporização a seguir refere-se às instruções ADD B e MOV B,A do microprocessador 8085.




Use o diagrama dado para responder às questões que seguem.

(c) Marque, no diagrama, os estados T que correspondem ao período de execução da instrução MOV B,A;
(d) Marque, no diagrama, com simbologia diferente da usada no item (a), os estados T que correspondem ao período de busca da instrução ADD B;
(e) Explique o significado dos sinais RD\, WR\ e IO-M\ no diagrama dado.

## 2.8 Referências Bibliográficas

[1] Ziller, Roberto M., "Microprocessadores - Conceitos Básicos,"2a. Ed., Editora do Autor, Florianópolis, SC, 2000.

[2] Kleitz, William, "Digital and Microprocessor Fundamentals - Theory and Applications," Prentice Hall, Englewood Cliffs, New Jersey, 1990

[3] Malvino, Albert P., “Microcomputadores e Microprocessadores,” Tradução: Anatólio Laschuk, revisão técnica: Rodrigo Araes Caldas Farias, McGraw-Hill, São Paulo, 1985.

# 3. CONJUNTO DE INSTRUÇÕES DO MICROPROCESSADOR 8085

## 3.1 Simbologia das Instruções

- addr = address (endereço) = quantidade de 16 bits.
- dado8 = dado de 8 bits
- dado16 = dado de 16 bits
- byte 2 = segundo byte da instrução
- byte 3 = terceiro byte da instrução
- r, r1, r2 = Um dos registradores: A, B, C, D, E, H, L
- $\wedge$ = operador lógico AND
- $\vee$ = operador lógico OR

**Exemplo 1:**

Instrução : **MOV r1, r2**

Indicação simbólica : (r1) $\leftarrow$ (r2)

Significado: O conteúdo do registrador r2 é transferido (copiado) para o registrador r1. O valor de SSS identifica o registrador r2 (origem) e o valor de DDD identifica o registrador r1 (destino).

Para transferir o conteúdo do registrador B para o registrador A, o mnemônico da instrução é **MOV A,B**. A cada mnemônico corresponde um código de operação (opcode) em hexadecimal. No caso dessa instrução é 78 h.

Para transferir o conteúdo do registrador E para o registrador D, o mnemônico da instrução é **MOV D,E**. O opcode é 53 h.

**Exemplo 2:**

Instrução : **LDA addr**

Indicação simbólica : (A) $\leftarrow$ ((byte 3)(byte 2))

Significado: O conteúdo da memória, cujo endereço é especificado nos bytes inferior (byte 2) e superior (byte 3), é transferido (copiado) para o registrador A. É uma instrução de 3 bytes (opcode + byte 3 + byte 2)

Para transferir o conteúdo do endereço **234B h** da memória, a instrução e o código da instrução seriam:
**LDA 234B h** Opcode: 3A 4B 23 (O conteúdo do byte menos significativo é digitado primeiro)

## 3.2 Modos de Endereçamento

As **instruções do 8085** fazem referência aos dados de forma explícita ou implícita. Há **4 maneiras** distintas de se fazer esta referência:

**IMEDIATO**: A instrução contém o dado no byte ou bytes seguintes ao Opcode.
Exemplos: **MVI r, dado8** = move o dado especificado para o registrador **r**
( r ) $\leftarrow$ (byte 2)

**ADI dado8** = adiciona o dado especificado ao acumulador
(A) $\leftarrow$ (A) + (byte 2)

**DIRETO**: O 2º e o 3º bytes da instrução contém o endereço da posição de memória onde se encontra o dado.

Exemplo: **LDA addr** = carrega o acumulador com o dado do endereço indicado

(A) ← ((byte 3)(byte 2))

**REGISTRO**: A instrução especifica o registrador ou o par de registradores onde o dado está armazenado.

Exemplo: **MOV r1, r2** = move conteúdo do registrador r2 para o registrador r1

(r1) ← (r2)

**ADD r** = adiciona o conteúdo do registrador **r** ao acumulador

(A) ← (A) + ( r )

**INDIRETO POR REGISTRO**: A instrução especifica o registrador par (rp) que contém o endereço da posição de memória onde o dado está armazenado.

Exemplo: **MOV r, M** = move para o registrador r o conteúdo da memória localizado na posição indicada pelo para HL

( r ) ← ((H)(L))

## 3.3 Grupos de Instruções

As instruções do 8085 são distribuídas em 5 grupos, cujas características são dadas a seguir:

1. **Grupo de Transferência de Dados** - Move dados entre registradores ou posições de memória e registradores. Inclui movimentos, cargas, armazenamentos a troca de dados.

   Exemplo: **MVI M, dado8** = move o dado especificado para a posição de memória indicada pelo registrador par HL.

   ((H)(L)) ← (byte 2)

2. **Grupo Aritmético** - Adições, subtrações, incrementos, ou decrementos de dados em registradores ou memória.

   Exemplo: **SUB r** = o conteúdo do registrador **r** é subtraído do acumulador. O resultado é guardado de volta no acumulador

   (A) ← (A) - ( r )

3. **Grupo Lógico** - ANDs, ORs, XORs, comparações, rotações, ou complementos de dados em registradores ou entre memória e um registrador.

   Exemplo: **ANA r** = os conteúdos do acumulador e do registrador **r** são submetidos ao operador lógico AND. O resultado é guardado de volta no acumulador.

   (A) ← (A) ∧ (r)

4. **Grupo de Desvio** - Inicia desvios condicionais ou incodicionais, chamadas de subrotina, retornos e reinícios.

   Exemplo: **JMP addr** = desvia incodicionalmente para o endereço indicado

   (PC) ← (byte 3) (byte 2)

5. **Grupo de Controle, Pilha, Entrada e Saída** - Inclui instruções para manutenção da pilha, leitura de portas, escritas para portas, setar e ler máscaras de interrupção e setar e limpar flags.

   Exemplo: **IN porta** = O dado de 8 bits presente na porta de entrada indicada é carregado no acumulador

   (A) ← (data)

## 3.4 Instruções de Transferência de Dados

| Mnemônico Genérico | Simbologia | Nº de Ciclos | Nº de Estados | Modo de Endereçamento | Flags Afetadas | Comentário |
|---|---|---|---|---|---|---|
| **MVI r, dado8** | (r) ← (byte 2) | 2 | 7 | Imediato | nenhuma | move o dado para o registrador **r** indicado |
| **MOV r1, r2** | (r1) ← (r2) | 1 | 4 | Registrador | nenhuma | move o conteúdo do registrador **r2** para o registrador **r1** |
| **MOV r, M** | (r) ← ((H)(L)) | 2 | 7 | Indireto por registrador | nenhuma | move para o registrador **r** o dado presente no endereço de memória especificado pelo registrador par H-L |
| **MOV M, r** | ((H)(L)) ← (r) | 2 | 7 | Indireto por registrador | nenhuma | move o conteúdo do registrador **r** para a posição de memória especificada pelo registrador par H-L. |
| **MVI M, dado8** | ((H)(L)) ← (byte 2) | 3 | 10 | Indireto por registrador e imediato | nenhuma | Carrega o dado na posição de memória especificada pelo registrador par H-L. |
| **LXI rp, dado16** | (rh) ← (byte 3)<br>(rl) ← (byte 2) | 3 | 10 | Imediato | nenhuma | Carrega o dado de 16 bits no registrador par indicado em **rp**. O byte 2 da instrução é colocado no registrador de menor ordem, ou byte menos significativo, **rl**. O byte 3 da instrução é colocado no registrador de maior ordem, ou byte mais significativo, **rh** |
| **LDA addr** | (A)← ((byte 3)(byte 2)) | 4 | 13 | direto | nenhuma | Carrega acumulador com o dado armazenado na posição de memória indicada pelo endereço **addr**. O byte 2 armazena o byte inferior do endereço. O byte 3 da instrução armazena o byte superior do endereço. |
| **STA addr** | ((byte 3)(byte 2)) ← (A) | 4 | 13 | direto | nenhuma | Move o conteúdo do acumulador para a posição de memória indicada pelo endereço **addr**. O byte 2 armazena o byte inferior do endereço. O byte 3 da instrução armazena o byte superior do endereço. |
| **LHLD addr** | (L)←((byte 3)(byte 2))<br>(H)←((byte 3)(byte 2) + 1) | 5 | 16 | direto | nenhuma | Carrega o conteúdo da posição de memória dada por **addr** ((byte 3)(byte 2)) no registrador L. Carrega o conteúdo da posição subsequente ((byte 3)(byte 2) + 1) no registrador H. |
| **SHLD addr** | ((byte 3)(byte 2)) ← (L)<br>((byte 3)(byte 2) + 1)← (H) | 5 | 16 | direto | nenhuma | Move o conteúdo do registrador L para a posição de memória dada por **addr** ((byte 3)(byte 2)). Move o conteúdo do registrador H para a posição subsequente de memória ((byte 3)(byte 2) + 1). |
| **LDAX rp** | (A) ← ((rp)) | 2 | 7 | Indireto por registrador | nenhuma | Carrega acumulador com o conteúdo da posição de memória indicada pelo registrador par **rp**. **rp** pode ser B (do registrador para BC) ou D (do registrador par DE). |
| **STAX rp** | ((rp)) ← (A) | 2 | 7 | Indireto por registrador | nenhuma | Move o conteúdo do acumulador para a posição de memória indicada pelo registrador par **rp**. **rp** pode ser B (do registrador para BC) ou D (do registrador par DE). |
| **XCHG** | (H) ↔ (D)<br>(L) ↔ (E) | 1 | 4 | registrador | nenhuma | O conteúdo do registrador H é trocado com o conteúdo do registrador D. O conteúdo de L é trocado com o conteúdo de E. |

As instruções apresentadas na tabela anterior estão na forma genérica. Cada uma dessas instruções é representada por diferentes códigos de operação (OPCODES) para diferentes registradores. Na tabela a seguir as instruções de transferência de dados são desmembradas em seus diferentes opcodes.

| MNEMÔNICO | OPCODE | MNEMÔNICO | OPCODE | MNEMÔNICO | OPCODE | MNEMÔNICO | OPCODE |
|---|---|---|---|---|---|---|---|
| LDA adr | 3A | MOV B,H | 44 | MOV E,C | 59 | MOV L,M | 6E |
| LDAX B | 0A | MOV B,L | 45 | MOV E,D | 5A | MOV M,A | 77 |
| LDAX D | 1A | MOV B,M | 46 | MOV E,E | 5B | MOV M,B | 70 |
| LHLD addr | 2A | MOV C,A | 4F | MOV E,H | 5C | MOV M,C | 71 |
| LXI B, Dado16 | 01 | MOV C,B | 48 | MOV E,L | 5D | MOV M,D | 72 |
| LXI D, Dado16 | 11 | MOV C,C | 49 | MOV E,M | 5E | MOV M,E | 73 |
| LXI H, Dado16 | 21 | MOV C,D | 4A | MOV H,A | 67 | MOV M,H | 74 |
| LXI SP, Dado16 | 31 | MOV C,E | 4B | MOV H,B | 60 | MOV M,L | 75 |
| MOV A,B | 78 | MOV C,H | 4C | MOV H,C | 61 | MVI A, Dado8 | 3E |
| MOV A,C | 79 | MOV C,L | 4D | MOV H,D | 62 | MVI B, Dado8 | 06 |
| MOV A,D | 7A | MOV C,M | 4E | MOV H,E | 63 | MVI C, Dado8 | 0E |
| MOV A,E | 7B | MOV D,A | 57 | MOV H,H | 64 | MVI D, Dado8 | 16 |
| MOV A,H | 7C | MOV D,B | 50 | MOV H,L | 65 | MVI E, Dado8 | 1E |
| MOV A,L | 7D | MOV D,C | 51 | MOV H,M | 66 | MVI L, Dado8 | 2E |
| MOV A,M | 7E | MOV D,D | 52 | MOV L,A | 6F | MVI M, Dado8 | 36 |
| MOV B,A | 47 | MOV D,E | 53 | MOV L,B | 68 | SHLD addr | 22 |
| MOV B,B | 40 | MOV D,H | 54 | MOV L,C | 69 | STA addr | 32 |
| MOV B,C | 41 | MOV D,L | 55 | MOV L,D | 6A | STAX B | 02 |
| MOV B,D | 42 | MOV D,M | 56 | MOV L,E | 6B | STAX D | 12 |
| MOV B,D | 42 | MOV E,A | 5F | MOV L,H | 6C | XCHG | EB |
| MOV B,E | 43 | MOV E,B | 58 | MOV L,L | 6D | | |

Exemplos de uso de instruções de transferência de dados:

| Mnemônico | Código | Comentário |
|---|---|---|
| MVI H,10h | 26 10 | Carrega acumulador H com valor 10h |
| MVI L,00h | 2E 00 | Carrega registrador L com valor 00h |
| MVI A,0Ah | 3E 0A | Carrega acumulador com valor 0Ah |
| MOV M,A | 77 | Move conteúdo de A para posição 1000h de memória |
| MOV C,M | 4E | Move conteúdo da posição 1000h para registrador C. C = 0Ah |
| MVI M,2Bh | 36 2B | Coloca valor 2Bh na posição 1000h de memória |

| Mnemônico | Código | Comentário |
|---|---|---|
| LXI B,1000h | 01 00 10 | Carrega registrador duplo BC com valor 1000h |
| LXI D,2000h | 11 00 20 | Carrega registrador duplo DE com valor 2000h |
| LXI H,3000h | 21 00 30 | Carrega registrador duplo HL com valor 3000h |
| LXI SP,4000h | 31 00 40 | Carrega registrador duplo SP (apontador de pilha) com valor 4000h |
| LDA 1000h | 3A 00 10 | Carrega acumulador com valor armazenado na posição de memória 1000h |
| STA 2000h | 32 00 20 | Move conteúdo do acumulador para a posição de memória 2000h |

| Mnemônico | Código | Comentário |
|---|---|---|
| LHLD 1000h | 2A 00 10 | Carrega conteúdo da posição 1000h no registrador L. Carrega conteúdo da posição 1001h no registrador H |
| SHLD 2000h | 22 00 20 | Move o conteúdo do registrador L para a posição 2000h. Move o conteúdo do registrador H para a posição de memória 2001h. |
| LXI H,3000h | 21 00 30 | Carrega registrador duplo HL com valor 3000h |
| LXI B,4000h | 01 00 40 | Carrega registrador duplo BC com valor 4000h |
| LXI D,5000h | 11 00 50 | Carrega registrador duplo DE com valor 5000h |
| LDAX B | 0A | Carrega acumulador com o conteúdo da posição de memória indicada pelo registrador duplo BC, ou seja, posição 4000h |
| STAX D | 12 | Move conteúdo do acumulador para a posição de memória indicada pelo registrador duplo DE, ou seja, posição 5000h |
| XCHG | EB | O conteúdo de H (30h) é trocado com o conteúdo de D (50h). O conteúdo de L (00h) é trocado com o conteúdo de E (00h).<br>Depois da instrução, temos: HL = 5000h e DE = 3000h |

## 3.5 Instruções Aritméticas

Obs.: A menos que seja indicado, todas as instruções desse grupo afetam todas as Flags: Zero, Sinal, Paridade, Transporte (Carry) e Auxiliar de Transporte (Auxiliar de Carry)

| Mnemônico Genérico | Simbologia | Nº de Ciclos | Nº de Estados | Modo de Endereçamento | Flags Afetadas | Comentário |
|---|---|---|---|---|---|---|
| **ADD r** | (A) ← (A) + ( r ) | 1 | 4 | registrador | todas | O conteúdo do registrador **r** é adicionado ao conteúdo do acumulador. |
| **ADD M** | (A) ← (A) + ( (H) + (L)) | 2 | 7 | registrador | todas | O conteúdo da posição de memória indicado pelo par HL é adicionado ao conteúdo do acumulador. |
| **ADI dado8** | (A) ← (A) + (byte 2) | 2 | 7 | imediato | todas | O valor dado em data é adicionado ao conteúdo do acumulador. |
| **ADC r** | (A) ← (A) + ( r ) + (CY) | 1 | 4 | registrador | todas | O conteúdo do registrador **r** é adicionado com carry ao conteúdo do acumulador. |
| **ADC M** | (A) ← (A) + ( (H)(L)) + (CY) | 2 | 7 | imediato por registrador | todas | O conteúdo da posição indicada pelo par HL é adicionado com carry ao conteúdo do acumulador. |
| **ACI dado8** | (A) ← (A) + (byte 2) + (CY) | 2 | 7 | imediato | todas | O valor dado em data é adicionado com carry ao conteúdo do acumulador. |
| **SUB r** | (A) ← (A) - ( r ) | 1 | 4 | registrador | todas | O conteúdo do registrador **r** é subtraído do conteúdo do acumulador. |
| **SUB M** | (A) ← (A) - ( (H)(L)) | 2 | 7 | indireto por registrador | todas | O conteúdo da posição de memória indicado pelo par HL é subtraído do conteúdo do acumulador. |
| **SUI dado8** | (A) ← (A) - (byte 2) | 2 | 7 | imediato | todas | O valor dado em data é subtraído do conteúdo do acumulador. |
| **SBB r** | (A) ← (A) - ( r ) - (CY) | 1 | 4 | registrador | todas | O conteúdo do registrador **r** é subtraído com carry do conteúdo do acumulador. |
| **SBB M** | (A) ← (A) - ( (H)(L)) - (CY) | 2 | 7 | indireto por registrador | todas | O conteúdo da posição indicada pelo par HL é subtraído com carry do conteúdo do acumulador. |
| **SBI dado8** | (A) ← (A) - (byte 2) - (CY) | 2 | 7 | imediato | todas | O valor dado em data é subtraído com carry do conteúdo do acumulador. |
| **INR r** | (r) ← (r) + 1 | 1 | 4 | registrador | Z, S, P e AC | O conteúdo do registrador **r** é adicionado de 1. Todas as **Flags** são afetadas, exceto CY. |
| **INR M** | ((H)(L)) ← ((H)(L)) + 1 | 3 | 10 | indiretor por registrador | Z, S, P e AC | O conteúdo da posição apontada pelo par HL é incrementado de 1. |
| **DCR r** | (r) ← (r) - 1 | 1 | 4 | registrador | Z, S, P e AC | O conteúdo do registrador **r** é decrementado. Todas as **Flags** são afetadas, exceto CY. |
| **DCR M** | ((H)(L)) ← ((H)(L)) - 1 | 3 | 10 | indireto por registrador | Z, S, P e AC | O conteúdo da posição apontada pelo par HL é decrementada de 1. |
| **INX rp** | (rh)(rl) ← (rh)(rl) + 1 | 1 | 6 | registrador | nenhuma | O conteúdo do registrador par **rp** é adicionado de 1. Nenhuma **Flag** é afetada. |

## Instruções Aritméticas - Continuação

| Mnemônico Genérico | Simbologia | Nº de Ciclos | Nº de Estados | Modo de Endereçamento | Flags Afetadas | Comentário |
|---|---|---|---|---|---|---|
| **DCX rp** | (rh)(rl) ← (rh)(rl) - 1 | 1 | 6 | registrador | nenhuma | O conteúdo do registrador par **rp** é decrementado de 1. |
| **DAD rp** | ((H)(L))←((H)(L))+ (rh)(rl) | 3 | 10 | registrador | CY | O conteúdo do registrador par **rp** é adicionado ao conteúdo do registrador par HL. Somente a Flag de Carry (CY) é afetada. |
| **DAA** | | 1 | 4 | registrador | todas | Faz o ajuste decimal do número no acumulador. O número de 8 bits do acumulador é ajustado para formar dois números de 4 bits em BCD.<br>A regra seguida é a seguinte:<br>1. Se o número representado pelo nibble inferior for maior do que 9, ou se a Flag AC estiver setada, o número 6 é adicionado ao conteúdo do acumulador.<br>2. Se o número representado pelo nibble superior for maior do que 9, ou se a Flag CY estiver setada, o número 6 é adicionado ao nibble superior. |

Na tabela a seguir as instruções aritméticas de dados são desmembradas em seus diferentes opcodes.

| MNEMÔNICO | OPCODE | MNEMÔNICO | OPCODE | MNEMÔNICO | OPCODE | MNEMÔNICO | OPCODE | MNEMÔNICO | OPCODE | MNEMÔNICO | OPCODE |
|---|---|---|---|---|---|---|---|---|---|---|---|
| ACI Dado8 | CE | ADD B | 80 | DAD H | 29 | DCX D | 1B | INX B | 03 | SBB M | 9E |
| ADC A | 8F | ADD C | 81 | DAD SP | 39 | DCX H | 2B | INX D | 13 | SBI Dado8 | DE |
| ADC B | 88 | ADD D | 82 | DCR A | 3D | DCX SP | 3B | INX H | 23 | SUB A | 97 |
| ADC C | 89 | ADD E | 83 | DCR B | 05 | INR A | 3C | INX SP | 33 | SUB B | 90 |
| ADC D | 8A | ADD H | 84 | DCR C | 0D | INR B | 04 | SBB A | 9F | SUB C | 91 |
| ADC E | 8B | ADD L | 85 | DCR D | 15 | INR C | 0C | SBB B | 98 | SUB D | 92 |
| ADC H | 8C | ADD M | 86 | DCR E | 1D | INR D | 14 | SBB C | 99 | SUB E | 93 |
| ADC L | 8D | ADI Dado8 | C6 | DCR H | 25 | INR E | 1C | SBB D | 9A | SUB H | 94 |
| ADC M | 8E | DAA | 27 | DCR L | 2D | INR H | 24 | SBB E | 9B | SUB L | 95 |
| ADD A | 87 | DAD B | 09 | DCR M | 35 | INR L | 2C | SBB H | 9C | SUB M | 96 |
| | | DAD D | 19 | DCX B | 0B | INR M | 34 | SBB L | 9D | SUI Dado8 | D6 |

Exemplos de uso de intruções aritméticas:

| Mnemônico | Código | Comentário |
|---|---|---|
| MVI A,05h | 3E 05 | Carrega acumulador com valor 05h |
| MVI C,02h | 0E 02 | Carrega registrador C com valor 02h |
| ADD C | 81 | Adiciona conteúdo de C ao conteúdo de A. A = 05 + 02 = 07h |
| ADI 10h | C6 10 | Adiciona 10h ao conteúdo de A. A = 07h + 10h = 17h |
| ADC A | 8F | Adiciona o conteúdo de A ao próprio conteúdo de A, incluindo o valor de carry. A = 17h + 17h + 0 = 2Eh. O carry aqui é Zero. |
| ACI 03h | CE 03 | Adiciona 03h ao conteúdo do acumulador. A = 2Eh+03h = 31h |

| Mnemônico | Código | Comentário |
|---|---|---|
| MVI A,05h | 3E 05 | Carrega acumulador com valor 05 h |
| MVI C,02h | 0E 02 | Carrega registrador C com valor 02 h |
| LXI H,2050h | 21 50 20 | Carrega registrador duplo HL com valor 2050 h |
| MVI M,08h | 36 08 | Move valor 08h para posição 2050 h (apontada por HL) |
| SUB C | 91 | Subtrai o conteúdo de C do conteúdo de A. A = 05 - 02 = 03h |
| SUI 02h | D6 02 | Adiciona 10h ao conteúdo de A. A = 03h - 02h = 01h |
| SBB C | 99 | Subtrai o conteúdo de C do conteúdo de A, incluindo o valor de carry. A = 01h - 02h - 0 = FFh. O carry antes é Zero. Depois passa para 1. |
| SBI 03h | DE 03 | Subtrai 03h do conteúdo do acumulador, incluindo carry. A = FFh - 03h – 1h = FBh. A Flag CY passa para Zero. CY = 0. |
| SBB M | 9E | Subtrai com carry o conteúdo da posição de memória 4100h do conteúdo do acumulador. A = FBh – 08h – 0 = F3h |

Obs.: Não se esquecer de que a subtração no 8085 não é feita diretamente. A subtração é feita através de uma adição com o complementar de 2.

Ex.: Subtração direta: A = 03 – 02 = 01h
Subtração com complementar de 2: A = 03 + (FD + 1) = 01h

Na operação com complementar de 2 houve um transporte (carry), mas na subtraçao no 8085 a **Flag CY** é o complementar do carry, ou seja, CY = 0, como deveria ser o resultado de uma subtração direta.

**Outros exemplos para verificação da Flag de Transporte CY:**

**Exemplo 1: Operação de adição**

```
  1 1 1 1 0 0 0 1
+ 0 0 0 0 1 0 0 0
-----------------
  1 1 1 1 1 0 0 0
```
Carry = 0 → CY = 0

```
  1 1 1 1 1 1 1 1
+ 0 0 0 0 0 0 0 1
-----------------
1 0 0 0 0 0 0 0 0
```
Carry = 1 → CY = 1

**Exemplo 2: Operação de subtração**

Obs.: As operações de subtração são executadas pelo 8085 usando o complementar de 2

subtração normal

```
  0 0 0 0 1 1 1 1
- 0 0 0 0 0 0 0 1
-----------------
  0 0 0 0 1 1 1 0
```
Carry = 0 → CY = 0

subtração usando complementar de 2

```
  0 0 0 0 1 1 1 1
+ 1 1 1 1 1 1 1 1
-----------------
1 0 0 0 0 1 1 1 0
```
Carry = 1 → CY = 0

**Exemplo 3: Operação de subtração**

subtração normal

```
  0 0 0 0 1 1 0 0
- 0 0 0 1 0 0 1 0
-----------------
1 1 1 1 1 1 0 1 0
```

Carry = 1 → CY = 1

subtração usando complementar de 2

```
  0 0 0 0 1 1 0 0
+ 1 1 1 0 1 1 1 0
-----------------
  1 1 1 1 1 0 1 0
```

Carry = 0 → CY = 1

Exemplo: Suponhamos que o acumulador contém o valor **1Bh** e a instrução **DAA** é usada. O resultado depois da instrução DAA é **21** h.

Exemplo:

| Mnemônico | Código | Comentário |
|---|---|---|
| MVI A,09h | 3E 09 | Carrega acumulador com valor 09h |
| MVI B,03h | 06 03 | Carrega registrador B com valor 03h |
| MVI C,10h | 0E 10 | Carrega registrador C com valor 10h |
| LXI D,1234h | 11 34 12 | Carrega registrador duplo DE com valor 1234h |
| LXI H,0123h | 21 23 01 | Carrega registrador duplo HL com valor 0123h |
| INR A | 3C | Incrementa 1 ao acumulador. A = 09h + 01h = 0Ah |
| DCR C | 0D | Decrementa 1 do conteúdo de C. C = 10h - 01h = 0Fh |
| INX D | 13 | Incrementa 1 ao registrador par DE. DE = 1234h + 1h = 1235h |
| DCX B | 0B | Decrementa 1 do conteúdo do registrador par BC. BC = 030Fh - 1h = 030Eh |
| DAD B | 09 | Adiciona ao registrador par HL o conteúdo do registrador par BC. HL = 0123h + 0310h = 0433h |
| DAA | 27 | Corrige para decimal os nibbles inferior e superior do acumulador. Antes: A = 0Ah. Depois: A = 10h |

**Exemplo especial:** Programa em assembly do 8085 para fazer a adição de dois números com mais de 8 bits (maiores que 255). Os números são: 452 e 926. 452 decimal = 1C4 h. 926 decimal = 39E h

| Mnemônico | Código | Comentário |
|---|---|---|
| MVI A,C4h | | Carrega registradores com bytes inferior e superior das parcelas |
| MVI B,01h | | |
| MVI C,9Eh | | |
| MVI D,03h | | |
| ADD C | | Adiciona parcelas dos bytes inferiores ( A = A + C) |
| MOV L,A | | Guarda resultado dos bytes inferiores em L |
| MOV A,B | | Carrega acumulador com uma das parcelas do byte superior |
| ADC D | | Adiciona parcelas dos bytes superiores, incluindo o bit de Carry |
| MOV H,A | | Guarda resultado dos bytes superiores em H |

Obs.: Após a primeira adição (ADD C), a flag auxiliar de carry AC assume o valor 1, porque há transporte do bit 3 para o bit 4. Também há transporte do bit 7 para o bit 8 (o qual está fora da capacidade do acumulador. A ilustração da adição acima é mostrada abaixo em decimal, hexadecimal e binário.

| Decimal | Hexa | Binário | |
|---|---|---|---|
| | | Byte Superior | Byte Inferior |
| 1 | 1 1 | 1 1 1 | 1 1 1 |
| 4 5 2 | 1 C 4 | 1 | 1 1 0 0 0 1 0 0 |
| 9 2 6 | 3 9 E | 1 1 | 1 0 0 1 1 1 1 0 |
| 1 3 7 8 | 5 6 2 | 1 0 1 | 0 1 1 0 0 0 1 0 |

## 3.6 Instruções Lógicas

Obs.: Com exceção dos casos indicados, todas as intruções lógicas afetam as Flags

| Mnemônico Genérico | Simbologia | N$^{O}$ de Ciclos | N$^{o}$ de Estados | Modo de Endereçamento | Flags Afetadas | Comentário |
|---|---|---|---|---|---|---|
| **ANA r** | (A) ← (A) ∧ (r) | 1 | 4 | registrador | todas | O conteúdo do acumulador passa por uma operação lógica **AND** com o conteúdo do registrador indicado em **r**. **A flag CY é zerada e a flag AC é setada**. |
| **ANA M** | (A) ← (A) ∧ ((H)(L)) | 2 | 7 | indireto por registrador | todas | O conteúdo do acumulador passa por uma operação lógica **AND** com o conteúdo da posição de memória apontada pelo registrador par HL. **A flag CY é zerada e a flag AC é setada**. |
| **ANI dado8** | (A) ← (A) ∧ (byte 2) | 2 | 7 | imediato | todas | o conteúdo do acumulador passa por uma operação lógica AND com o dado fornecido no byte 2 da instrução. **A flag CY é zerada e a flag AC é setada.** |
| **XRA r** | (A) ← (A) ∀ (r) | 1 | 4 | registrador | todas | O conteúdo do acumulador passa por uma operação lógica **XOR** com o conteúdo do registrador r indicado. **As flags CY e AC são zeradas**. |
| **XRA M** | (A) ← (A) ∀ ((H)(L)) | 2 | 7 | indireto por registrador | todas | O conteúdo do acumulador passa por uma operação lógica **XOR** com o conteúdo do endereço de memória apontado pelo registrador par HL. As flags CY e AC são zeradas. |
| **XRI dado8** | (A) ← (A) ∀ (byte 2) | 2 | 7 | imediato | todas | o conteúdo do acumulador passa por uma operação lógica **XOR** com o dado fornecido no byte 2 da instrução. As flags CY e AC são zeradas. |
| **ORA r** | (A) ← (A) ∨ (r) | 1 | 4 | registrador | todas | O conteúdo do acumulador passa por uma operação lógica **OR** com o conteúdo do registrador indicado em **r.** As flags CY e AC são zeradas. |
| **ORA M** | (A) ← (A) ∨ ((H)(L)) | 2 | 7 | indireto por registrador | todas | O conteúdo do acumulador passa por uma operação lógica **OR** com o conteúdo da posição de memória apontada pelo registrador par HL. As flags CY e AC são zeradas. |
| **ORI dado8** | (A) ← (A) ∨ (byte 2) | 2 | 7 | imediato | todas | O conteúdo do acumulador passa por uma operação lógica **OR** com o dado presente no byte 2 da instrução. As flags CY e AC são zeradas. |
| **CMP r** | (A) - (r) | 1 | 4 | registrador | todas | Compara o conteúdo do registrador **r** com o conteúdo do acumulador. Na operação o conteúdo de **r** é subtraído do conteúdo de **A,** sem que o resultado seja guardado em A. Se o resultado da subtração for zero, ou seja (A) = (r), a flag de zero Z = 1. Caso contrário, Z = 0. Se (A) < (r), a flag de carry CY = 1. Caso contrário, CY = 0. |
| **CMP M** | (A) ← (A) - ((H)(L)) | 2 | 7 | indireto por registrador | todas | Compara o conteúdo da posição apontada pelo par HL com o conteúdo do acumulador. Na operação o conteúdo da posição apontada por HL é subtraído do conteúdo de **A,** sem que o resultado seja guardado em A. Se o resultado da subtração for zero, ou seja (A) = ((H)(L)), a flag de zero Z |

| | | | | | | |
|---|---|---|---|---|---|---|
| | | | | | | = 1. Caso contrário, Z = 0. Se (A) < ((H)(L)), a flag de carry CY = 1. Caso contrário, CY = 0. |
| **CPI dado8** | (A) ← (A) - (byte 2) | 2 | 7 | imediato | todas | Compara o conteúdo do byte 2 da instrução com o conteúdo do acumulador. Na operação o conteúdo do byte 2 é subtraído do conteúdo de **A,** sem que o resultado seja guardado em A. Se o resultado da subtração for zero, ou seja (A) = (byte 2), a flag de zero Z = 1. Caso contrário, Z = 0. Se (A) < (byte 2), a flag de carry CY = 1. Caso contrário, CY = 0. |
| **RLC** | $(A_{n+1}) \leftarrow (A_n)$<br>$(A_0) \leftarrow (A_7)$<br>$(CY) \leftarrow (A_7)$ | 1 | 4 | | CY | (Rotate Left) = (Rotacionar à Esquerda) = O conteúdo do acumulador é rotacionado uma posição à esquerda. O conteúdo do último bit (bit 7) é transferido tanto para a posição do bit 0, quanto para o Carry. O conteúdo do bit 'i' é transferido para o bit 'i + 1'**. Somente a flag de carry é afetada**. |
| **RRC** | $(A_n) \leftarrow (A_{n+1})$<br>$(A_7) \leftarrow (A_0)$<br>$(CY) \leftarrow (A_0)$ | 1 | 4 | | CY | (Rotate Right) = (Rotacionar à Direita) = O conteúdo do acumulador é rotacionado uma posição à direita. O conteúdo do bit menos significativo (bit 0) é transferido tanto para a posição do bit 7, quanto para o Carry. O conteúdo do bit 'i' é transferido para o bit 'i - 1'**. Somente a flag de carry CY é afetada**. |
| **RAL** | $(A_{n+1}) \leftarrow (A_n)$<br>$(CY) \leftarrow (A_7)$<br>$(A_0) \leftarrow (CY)$ | 1 | 4 | | CY | (Rotate Left through Carry) = (Rotacionar à Esquerda através do Carry) = O conteúdo do acumulador é rotacionado uma posição à esquerda, incluindo o bit de carry como o oitavo bit. Assim, o conteúdo do bit 7 é transferido para o bit de Carry (CY) e o conteúdo do bit de Carry é transferido para o bit menos significativo (bit 0). **Somente a flag de carry é afetada**. |
| **RAR** | $(A_n) \leftarrow (A_{n+1})$<br>$(CY) \leftarrow (A_0)$<br>$(A_7) \leftarrow (CY)$ | 1 | 4 | | CY | (Rotate Right through Carry) = (Rotacionar à Esquerda) = O conteúdo do acumulador é rotacionado uma posição à direita, incluindo o carry como oitavo bit. Assim, o conteúdo do último bit de Carry (CY) é transferido para a posição do bit 7 e conteúdo do bit 0 é transferido para o bit de Carry. **Somente a flag de carry é afetada**. |
| **CMA** | (A) ← (A\) | 1 | 4 | | nenhuma | (Complement Accumulator) = (Complementa Acumulador) = O conteúdo do acumulador é complementado, bit a bit. Os bits zero tornam-se 1 e os bits 1 tornam-se zero. **Nenhuma flag é afetada**. |
| **CMC** | (CY) ← (CY\) | 1 | 4 | | CY | (Complement Carry) = (Complementa Carry) = O conteúdo da flag de Carry complementado. **Nenhuma outra flag é afetada**. |
| **STC** | (CY) ← 1 | 1 | 4 | | CY | (Set Carry) = (Seta o bit de Carry) = A flag de Carry (CY) é feita igual a 1. **Nenhuma outra flag é afetada**. |

Resumo das condições da instrução CMP r

| Condição | Z | CY |
|---|---|---|
| (A) = (r) | 1 | 0 |
| (A) > (r) | 0 | 0 |
| (A) < (r) | 0 | 1 |

Opcode e Mnemônico das Instruções Lógicas:

| **MNEMÔNICO** | **OPCODE** | **MNEMÔNICO** | **OPCODE** | **MNEMÔNICO** | **OPCODE** |
|---|---|---|---|---|---|
| ANA A | A7 | ORA B | B0 | RLC | 07 |
| ANA B | A0 | ORA C | B1 | RNC | D0 |
| ANA C | A1 | ORA D | B2 | RRC | 0F |
| ANA D | A2 | ORA E | B3 | XRA A | AF |
| ANA E | A3 | ORA H | B4 | XRA B | A8 |
| ANA H | A4 | ORA L | B5 | XRA C | A9 |
| ANA L | A5 | ORA M | B6 | XRA D | AA |
| ANA M | A6 | ORI Dado8 | F6 | XRA E | AB |
| ANI Dado8 | E6 | RAL | 17 | XRA H | AC |
| CMC | 3F | RAR | 1F | XRA L | AD |
| ORA A | B7 | RC | D8 | XRA M | AE |

Exemplo de programa usando instruções lógicas:

| Mnemônico | Código | Comentário |
|---|---|---|
| MVI A,0Fh | 3E 0F | Carrega acumulador com valor 0Fh |
| MVI C,52h | 0E 52 | Carrega registrador C com valor 52h |
| MVI B,46h | 06 46 | Carrega registrador B com valor 46h |
| ANA C | A1 | Faz operação A **AND** C, ou, 0F **AND** 52h $\Rightarrow$ **A = 02h** |
| ANI 44h | E6 44 | Faz A **AND** 44h, ou, 02h **AND** 44h. $\Rightarrow$ **A = 00h** |
| XRI 23h | EE 23 | Faz A XOR 23h, ou, 00h XOR 23h $\Rightarrow$ **A = 23h** |
| CPI 33h | FE 33 | Faz A - 33h, sem alterar A. Ou, 23 - 33h $\Rightarrow$ **Z = 0 e CY = 1** |
| RLC | 07 | Rotaciona **A** à esquerda. Resultado: **A = 46h e CY = 0** |
| CMP B | B8 | Faz A - B, sem alterar A. Ou, 46h - 46h $\Rightarrow$ **Z = 1 e CY = 0** |
| CMC | 3F | Complementa flag de Carry. **CY = 1** |
| RAR | 1F | Rotaciona A à direita com Carry $\Rightarrow$ **A = A3h e CY = 0** |

## 3.7 Instruções de Desvio

(a) As instruções desse grupo alteram o fluxo normal do programa.

(b) Nenhuma flag é afetada por qualquer das instruções do grupo de instruções de desvio.

(c) As instruções de desvio são divididas em: instruções de **desvio condicional** e instruções de **desvio incondicional**.

(d) As instruções de desvio incondicional simplesmente alteram o fluxo do programa alterando o valor do contador de programa PC.

(e) As instruções de desvio condicional examinam o estado (status) de uma das quatro flags (Z, S, P e CY), para verificar se o desvio indicado deve ser executado. As condições que podem ser especificadas são dadas a seguir:

| **Símbolo** | **Condição indicada** | **bits de identificação** | | |
|---|---|---|---|---|
| | | **C** | **C** | **C** |
| **NZ** | Not Zero (Z = 0) | 0 | 0 | 0 |
| **Z** | Zero (Z = 1) | 0 | 0 | 1 |
| **NC** | No Carry (CY = 0) | 0 | 1 | 0 |
| **C** | Carry (CY = 1) | 0 | 1 | 1 |
| **PO** | Parity odd = Paridade Ímpar (P = 0) | 1 | 0 | 0 |
| **PE** | Parity Even = Paridade Par (P = 1) | 1 | 0 | 1 |
| **P** | Plus ≡ Positivo (S = 0) | 1 | 1 | 0 |
| **M** | Minus ≡ Negativo (S = 1) | 1 | 1 | 1 |

Instruções possíveis: **JNZ, JZ, JNC, JC, JPO, JPE, JP e JM**

Instruções de Desvio:

| Mnemônico Genérico | Simbologia | N$^{O}$ de Ciclos | N$^{o}$ de Estados | Modo de Endereçamento | Flags Afetadas | Comentário |
|---|---|---|---|---|---|---|
| **JMP addr** | (PC) ← (byte 3)(byte 2) | 3 | 10 | imediato | nenhuma | (Jump to address) = O controle é transferido incondicionalmente para a instrução cujo endereço é dado no segundo e no terceiro bytes da instrução de desvio. |
| **$J_{condição}$ addr** | Se (CCC), então, (PC) ← (byte 3)(byte 2) | 2/3 | 7/10 | imediato | nenhuma | (Jump to address if CCC) = Se a condição indicada for verdadeira o controle é transferido para a instrução cujo endereço é dado no segundo e no terceiro bytes da instrução de desvio. Caso a condição seja falsa, o processamento continua sequencialmente. |
| **JNZ addr** | Se (CCC), então, (PC) ← (byte 3)(byte 2) | 2/3 | 7/10 | imediato | nenhuma | (Jump if Not Zero) = Desvia para o endereço indicado se o resultado da operação aritmética anterior a esta instrução não for zero, ou seja, **desvia se Z = 0**. |
| **JZ addr** | Se (CCC), então, (PC) ← (byte 3)(byte 2) | 2/3 | 7/10 | imediato | nenhuma | (Jump if Zero) = Desvia para o endereço indicado se o resultado da operação aritmética anterior a esta instrução for igual a zero, ou seja, **desvia se Z = 1**. |
| **JNC addr** | Se (CCC), então, (PC) ← (byte 3)(byte 2) | 2/3 | 7/10 | imediato | nenhuma | (Jump if No Carry) = Desvia para o endereço indicado se a flag de Carry estiver zerada (**CY = 0**). |
| **JC addr** | Se (CCC), então, (PC) ← (byte 3)(byte 2) | 2/3 | 7/10 | imediato | nenhuma | (Jump if Carry) = Desvia para o endereço indicado se a flag de Carry estiver setada (**CY = 1**). |
| **JPO addr** | Se (CCC), então, (PC) ← (byte 3)(byte 2) | 2/3 | 7/10 | imediato | nenhuma | (Jump if Parity Odd) = Desvia para o endereço indicado se a paridade for Ímpar, ou seja, se a flag de Paridade for zero (**P = 0**). |
| **JPE addr** | Se (CCC), então, (PC) ← (byte 3)(byte 2) | 2/3 | 7/10 | imediato | nenhuma | (Jump if Parity Even) = Desvia para o endereço indicado se a paridade for Par, ou seja, se a flag de Paridade estiver setada (**P = 1**). |
| **JP addr** | Se (CCC), então, (PC) ← (byte 3)(byte 2) | 2/3 | 7/10 | imediato | nenhuma | (Jump if Plus) = Desvia para o endereço indicado se o valor no acumulador for um número positivo, ou seja, tiver o sétimo bit zerado, ou ainda se a flag de Sinal estiver zerada (**S = 0**). |
| **JM addr** | Se (CCC), então, (PC) ← (byte 3)(byte 2) | 2/3 | 7/10 | imediato | nenhuma | (Jump if Minus) = Desvia para o endereço indicado se o valor no acumulador for um número negativo, ou seja, tiver o sétimo bit setado, ou ainda se a flag de Sinal estiver setada (**S = 1**). |
| **CALL addr** | ((SP) - 1) ← (PCH)<br>((SP) - 2) ← (PCL)<br>(SP) ← (SP) - 2<br>(PC) ← (byte 3)(byte 2) | 5 | 18 | imediato e indireto por registrador | nenhuma | (Call address) = Chamada de subrotina. O processamento é desviado para o endereço indicado em **addr**, que é dado pelos bytes 2 e 3 da instrução. Antes do desvio para a subrotina, o endereço da próxima instrução é guardado na pilha. O byte superior do endereço da próxima instrução é guardado na posição SP - 1 da pilha e o byte inferior é guardado na posição SP - 2 da pilha. Ao final da subrotina a instrução RET faz o processamento voltar para o programa principal no endereço que foi guardado na pilha. |
| **$C_{condition}$ addr** | ((SP) - 1) ← (PCH)<br>((SP) - 2) ← (PCL)<br>(SP) ← (SP) - 2<br>(PC) ← (byte 3)(byte 2) | 2/5 | 9/18 | imediato e direto por registrador | nenhuma | (Call address if CCC is true) = Chamada de subrotina condicional. O processamento é desviado para o endereço indicado em **addr** se a condição indicada em **CCC** for verdadeira. O endereço é dado pelos bytes 2 e 3 da instrução. Antes do desvio para a subrotina, o endereço da próxima instrução é guardado na pilha. O byte superior do endereço da próxima instrução é guardado na posição SP - 1 da pilha e o byte inferior é guardado na posição SP - 2 da pilha. Ao final da subrotina a instrução RET faz o processamento voltar para o programa principal no endereço que foi guardado na pilha. |

| | | | | | | |
|---|---|---|---|---|---|---|
| | | | | | | Casos possíveis: **CNZ, CZ, CNC, CC, CPO, CPE, CP, CM** |
| **RET** | (PCL) ← (SP)<br>(PCH) ← (SP + 1)<br>(SP) ← (SP) + 2 | 3 | 10 | indireto por registrador | nenhuma | (Return) = Retorno de Subrotina. O processamento volta para o local de onde partiu (instrução seguinte). O endereço de retorno é buscado na pilha. O conteúdo da posição SP é o byte menos significativo do endereço de retorno. O conteúdo da posição SP + 1 é o byte mais significativo do endereço de retorno. Após essas duas operações de transferência o valor de SP é ainda incrementado novamente de forma que SP terá sido incrementado de 2 ao final da operação de busca da pilha. |
| **$R_{condição}$** | (PCL) ← (SP)<br>(PCH) ← (SP + 1)<br>(SP) ← (SP) + 2 | 1/3 | 6/12 | indireto por registrador | nenhuma | (Conditional Return) = Retorno de Subrotina Condicionado a que CCC seja verdadeiro. O processamento volta para o local de onde partiu (instrução seguinte) se a condição dada em CCC for verdadeira. O endereço de retorno é buscado na pilha. O conteúdo da posição SP é o byte menos significativo do endereço de retorno. O conteúdo da posição SP + 1 é o byte mais significativo do endereço de retorno. Após essas duas operações de transferência o valor de SP é ainda incrementado novamente de forma que SP terá sido incrementado de 2 ao final da operação de busca da pilha. |
| **RST n** | ((SP) - 1) ← (PCH)<br>((SP) - 2) ← (PCL)<br>(SP) ← (SP) - 2<br>(PC) ← 8 * (NNN) | 3 | 12 | indireto por registrador | nenhuma | (Restart) = Reinício. O processamento é desviado para o endereço indicado por 8 * (NNN). No entanto, antes do desvio, o endereço da próxima instrução é guardado na pilha. |
| **PCHL** | (PCH) ← (H)<br>(PCL) ← (L) | 1 | 6 | registrador | nenhuma | (Jump H and L inderect - move H and L to PC) = O conteúdo do registrador H é transferido para o byte mais significativo de PC. O conteúdo do registrador L é transferido para o byte menos significativo de PC. |

## 3.8 Instruções de Controle, Pilha e Entrada e Saída

Obs.: As instruções desse grupo não afetam as flags, a menos que seja indicado.

| Mnemônico Genérico | Simbologia | Nº de Ciclos | Nº de Estados | Modo de Endereçamento | Flags Afetadas | Comentário |
|---|---|---|---|---|---|---|
| **PUSH rp** | ((SP) - 1 ← (rh)<br>((SP) - 2 ← (rl)<br>(SP) ← (SP) - 2 | 3 | 12 | indireto por registrador | nenhuma | (Push) = O conteúdo do registrador de mais alta ordem (byte superior) do par de registradores é guardado na posição de memória indicada por SP - 1. O conteúdo do registrador que contém o byte inferior da instrução é guardado na posição de memória indicada por SP - 2. O registrador par **rp** pode ser B (de BC), D (de DE) ou H (de HL). O registrador SP é decrementado de 2. |
| **PUSH PSW** | ((SP) - 1 ← (A)<br>((SP) - 2 ← (F)<br>(SP) ← (SP) - 2 | 3 | 12 | indireto por registrador | nenhuma | (Push Processor Status Word) = O conteúdo dos registradores A (acumulador) e F (flags) é guardado na pilha. O conteúdo do Acumulador é guardado na posição de memória indicada por **SP - 1**. O conteúdo do registrador F é guardado na posição de memória indicada por **SP - 2**. |
| **POP rp** | (rl) ← ((SP))<br>(rh) ← ((SP) + 1)<br>(SP) ← (SP) + 2 | 3 | 10 | indireto por registrador | nenhuma | O conteúdo da posição de memória indicada por **SP** é movido para o registrador que representa o byte inferior (C, E ou L). O conteúdo da posição de memória indicada por **SP + 1** é movido para o registrador que representa o byte superior da instrução (B, D ou H). O conteúdo do registrador SP é incrementado de 2. |
| **POP PSW** | (F) ← ((SP))<br>(A) ← ((SP) + 1)<br>(SP) ← (SP) + 2 | 3 | 10 | indireto por registrador | todas | O conteúdo da posição de memória indicada por **SP** é movido para o registrador que guarda o estado das Flags (registrador F). O conteúdo da posição de memória indicada por **SP + 1** é movido para o acumulador. O conteúdo do registrador SP é incrementado de 2. |
| **XTHL** | (L) ← ((SP))<br>(H) ← ((SP) + 1) | 5 | 16 | indireto por registrador | nenhuma | (Exchange Stack Top with H and L) = O conteúdo da posição de memória indicada por **SP** é trocado com o conteúdo do registrador L e o conteúdo da posição de memória indicada por **SP + 1** é trocado pelo conteúdo do registrador H. |
| **SPHL** | (SP) ← (H) (L) | 1 | 6 | registrador | nenhuma | (Move HL to SP) = O conteúdo do registrador par HL é movido para o registrador de 16 bits **SP.** |
| **IN** porta | (A) ← (dado 8 bits) | 3 | 10 | direto | nenhuma | (Input = Entrada) = O dado colocado na porta indicada é transferido para o acumulador através do barramento de dados (8 bits). |
| **OUT** porta | (dado 8 bits) ← (A) | 3 | 10 | direto | nenhuma | (Output = Saída) = O dado do acumulador é transferido para a porta indicada através do barramento de dados (8 bits). |

| | | | | | | |
|---|---|---|---|---|---|---|
| **EI** | | 1 | 4 | | nenhuma | (Enable Interrupt = Habilita Interrupções) = A interrupção do sistema é habilitada após a execução da instrução seguinte. Nenhuma interrupção é reconhecida durante a execução da intrução EI. |
| **DI** | | 1 | 4 | | nenhuma | (Disable Interrupt = Desabilita Interrupções) = A interrupção do sistema é desabilitada imediatamente após a execução da instrução DI. Nenhuma interrupção é reconhecida durante a execução da intrução DI. |
| **HLT** | | 1+ | 5 | | nenhuma | A instrução HLT faz o processador parar o processamento. Os registradores e flags permanecem inalterados. |
| **NOP** | | 1 | 4 | | nenhuma | (No Operation) = Nenhuma operação é realizada. Os registradores e flags ficam inalterados. |
| **RIM** | | 1 | 4 | | nenhuma | (Read Interrupt Mask = Lê Máscara de Interrupção) = Esta instrução carrega no acumulador os dados relativos às interrupções e à entrada serial. Será melhor detalhada posteriormente. |
| **SIM** | | 1 | 4 | | nenhuma | (Set Interrupt Mask = Define Máscara de Interrupção) = Esta instrução usa o conteúdo do acumulador para definir as máscaras de interrupção. Será melhor detalhado posteriormente. |

Códigos de Operação das Instruções de Desvio e de Controle

| MNEMÔNICO | OPCODE | MNEMÔNICO | OPCODE | MNEMÔNICO | OPCODE | MNEMÔNICO | OPCODE | MNEMÔNICO | OPCODE | MNEMÔNICO | OPCODE |
|---|---|---|---|---|---|---|---|---|---|---|---|
| CALL adr | CD | CMP M | BE | JM adr | FA | POP B | C1 | RNC | D0 | RST 6 | F7 |
| CC adr | DC | CNC adr | D4 | JMP adr | C3 | POP D | D1 | RNZ | C0 | RST 7 | FF |
| CM adr | FC | CNZ adr | C4 | JNC adr | D2 | POP H | E1 | RP | F0 | RZ | C8 |
| CMA | 2F | CP adr | F4 | JNZ adr | C2 | POP PSW | F1 | RPE | E8 | SIM | 30 |
| CMC | 3F | CPE adr | EC | JP adr | F2 | PUSH B | C5 | RPO | E0 | SPHL | F9 |
| CMP A | BF | CPI D8 | FE | JPE adr | EA | PUSH D | D5 | RST 0 | C7 | STC | 37 |
| CMP B | B8 | CPO adr | E4 | JPO adr | E2 | PUSH H | E5 | RST 1 | CF | XCHG | EB |
| CMP C | B9 | CZ adr | CC | JZ adr | CA | PUSH PSW | F5 | RST 2 | D7 | XRI D8 | EE |
| CMP D | BA | DI | F3 | NOP | 00 | RC | D8 | RST 3 | DF | XTHL | E3 |
| CMP E | BB | EI | FB | ORI D8 | F6 | RET | C9 | RST 4 | E7 | | |
| CMP H | BC | IN D8 | DB | OUT D8 | D3 | RIM | 20 | RST 5 | EF | | |
| CMP L | BD | JC adr | DA | PCHL | E9 | RM | F8 | | | | |

## 3.9 Funcionamento da Pilha

Como já foi dito anteriormente, a Pilha é uma região da memória RAM, definida pelo usuário, para guardar valores que serão usados posteriormente. Assim, o usuário pode guardar o conteúdo de qualquer registrador (dois a dois: A e Flags, B e C, D e E, H e L) na pilha e o microprocessador guarda automaticamente os endereços de retorno de subrotinas comuns e de subrotinas de interrupções. A seguir é ilustrada a região da memória definida como Pilha (Stack).

Observações:

(a) O conteúdo guardado na pilha é sempre de 16 bits. Assim, o microprocessador normalmente guarda o conteúdo de **PC**, que já é de 16 bits, mas o usuário normalmente guarda o conteúdo de registradores de 8 bits, que então são associados 2 a 2;
(b) Os registradores duplos que podem ser guardados na pilha são **PSW** (= A + Flags), **B** (= B + C), **D** (= D + E) e **H** (= H + L);
(c) Para guardar o conteúdo de um desses registradores duplos usa-se a instrução **PUSH rp;**
(d) Para recuperar o conteúdo que foi guardado na pilha usa-se a instrução **POP rp**;
(e) Quando uma informação é enviada para a pilha o **byte mais significativo é guardado primeiro**; isso significa que o byte menos significativo vai ser retirado primeiro porque o último dado armazenado é o primeiro a ser retirado;
(f) A pilha do 8085 evolui do maior endereço para o menor, ou seja, a cada vez que uma informação (2 bytes) é enviada para a pilha, o endereço do topo da pilha é **reduzido de 2**. Ele é acrescido de 2 quando a informação é retirada da pilha;
(g) O apontador de pilha SP aponta sempre para o topo da pilha, mas ele é incrementado de 1 antes de cada byte ser armazenado.

Exemplo de armazenamento na pilha:

Supondo que inicialmente **SP = 2090 h**, **A = 01 h**, **F = 23 h**, **B = 45 h** e **C = 67 h**, as figuras a seguir mostram a evolução da pilha após cada instrução dada. A região em destaque corresponde à posição apontada por SP após a instrução.

**PUSH PSW**

| Endereço da RAM | Conteúdo (HEX) |
|---|---|
| 2089 | |
| 208A | |
| 208B | |
| 208C | |
| 208D | |
| 208E | 23 |
| 208F | 01 |
| 2090 | |
| SP após a instrução: **208E h** | |

**PUSH B**

| Endereço da RAM | Conteúdo (HEX) |
|---|---|
| 2089 | |
| 208A | |
| 208B | |
| 208C | 67 |
| 208D | 45 |
| 208E | 23 |
| 208F | 01 |
| 2090 | |
| SP após a instrução: **208C h** | |

**POP B**

| Endereço da RAM | Conteúdo (HEX) |
|---|---|
| 2089 | |
| 208A | |
| 208B | |
| 208C | 67 |
| 208D | 45 |
| 208E | 23 |
| 208F | 01 |
| 2090 | |
| SP após a instrução: **208E h** | |

**POP PSW**

| Endereço da RAM | Conteúdo (HEX) |
|---|---|
| 2089 | |
| 208A | |
| 208B | |
| 208C | 67 |
| 208D | 45 |
| 208E | 23 |
| 208F | 01 |
| 2090 | |
| SP após a instrução: **2090 h** | |

Observações:

(a) Apesar da região da pilha continuar com o mesmo conteúdo após as instruções **POP rp**, eles não acessíveis porque o Apontador de Pilha SP aponta para a posição original, de quando a pilha estava vazia;
(b) **POP B** vem antes de **POP PSW** porque a ordem de retirada da pilha é inversa à ordem de armazenagem.

## 3.10 Exemplos de Programas em Assembly do 8085

**1.** Exemplo de uso de instruções de desvio e de controle. A instrução **MOV A,B** não afeta nenhuma flag e, portanto, não afeta a flag de zero (**Z**) decorrente da instrução **DCR C.**

| Label | Instruções | Comentário |
|---|---|---|
| | MVI C,10h | Carrega o registrador C com o valor 10 h |
| volta: | DCR C | Decrementa conteúdo do registrador C |
| | MOV A,B | Copia conteúdo de B em A. Não afeta nenhuma flag. |
| | JNZ volta | Se o resultado de DCR C não for ZERO, desvia para "volta" |
| fim: | HLT | Esta instrução pára o processamento do programa |

2. Exemplo similar ao anterior, mas usando a instrução JZ *endereço* e JMP *endereço*.

| Label | Instruções | Comentário |
|---|---|---|
| | MVI C,10h | Carrega o registrador C com o valor 10 h |
| volta: | DCR C | Decrementa conteúdo do registrador C |
| | JZ fim | Se o resultado de DCR C for ZERO, desvia para "fim" |
| | MOV A,B | Copia conteúdo de B em A. Não afeta nenhuma flag. |
| | JMP volta | Desvio incondicional para "volta". |
| fim: | HLT | Esta instrução pára o processamento do programa |

3. Exemplo usando comparação entre registradores (A e B). A operação de comparação e volta é repetida até o valor de B alcançar o valor 08h, quando então a flag de carry é setada, fazendo o processamento encerrar.

| Label | Instruções | Comentário |
|---|---|---|
| | MVI A,07h | Carrega registrador A com valor 07 h |
| | MVI B,00h | Carrega registrador B com valor 00 h. |
| volta: | INC B | Incrementa em "1" o conteúdo do registrador B |
| | CMP B | Compara conteúdo de B com o conteúdo de A, sem alterar valor de A. |
| | JNC volta | Desvia para "volta", se a flag CY = 0. CY =0 se A > B ou A = B |
| | HLT | Pára processamento quando CY = 1, ou seja, quando B=8 (A < B). |

4. Exemplo similar ao anterior, mas usando a instrução JC *endereço*, ao invés de JNC *endereço*.

| Label | Instruções | Comentário |
|---|---|---|
| | MVI A,07h | A operação de comparação e volta é repetida até o valor em B alcançar o valor 08h, quando então a flag de Carry CY é setada, fazendo o processamento encerrar pulando para o comando HLT |
| | MVI B,00h | |
| volta: | INC B | |
| | CMP B | |
| | JC fim | |
| | JMP volta | |
| fim: | HLT | |

5. Programa que faz a multiplicação de 4 por 3. É usada a instrução ADI *Dado8.*

| Label | Mnemônico | Comentário |
|---|---|---|
| | MVI A,00h | Zera acumulador. A = 00 h |
| | MVI C,03h | Carrega registrador C com 03 h. "C" será usado como contador |
| volta: | ADI 04h | Adiciona Imediato 04h ao acumulador. A = A + 04 h |
| | DCR C | Decrementa o contador. C = C - 1. |
| | JNZ volta | "volta" se o resultado de DCR C não for zero (Se Z = 0) |
| | HLT | Encerra processamento quando C = 0, ou seja, flag Z = 1. |

6. Outra versão de programa que faz a multiplicação de 4 por 3. A instrução ADI Dado8 é substituída pela instrução ADD B. O acumulador vai assumir os valores 00 h, 04 h, 08 h e, finalmente, 0C h, isto é 12 decimal.

| Label | Mnemônico | Comentário |
|---|---|---|
| | MVI A,00h | Zera acumulador. A = 00 h. |
| | MVI B,04h | Carrega o registrador B com 04 h. |
| | MVI C,03h | Carrega o registrador C com 03 h. "C" será usado como contador |
| volta: | ADD B | Adiciona B ao acumulador. A = A + B |
| | DCR C | Decrementa o contador. C = C - 1. |
| | JNZ volta | "volta" se o resultado de DCR C não for zero (Se Z = 0) |
| | HLT | Encerra processamento quando C = 0, ou seja, flag Z = 1. |

7. Programa que faz a multiplicação de 4 por 3 usando uma subrotina.

| Label | Mnemônico | Comentário |
|---|---|---|
| | LXI SP, 2090h | Pilha na posição 2090 h |
| | MVI A,00h | Zera acumulador. A = 00 h. |
| | MVI B,04h | Carrega o registrador B com 04 h. |
| | MVI C,03h | Carrega o registrador C com 03 h. "C" será usado como contador |
| | CALL soma | Chama subrotina que adiciona B ao acumulador |
| | HLT | Encerra processamento |
| | | |
| soma: | ADD B | Adiciona B ao acumulador. A = A + B |
| | DCR C | Decrementa o contador. C = C - 1. |
| | JNZ soma | Desvia para "volta" se o resultado de DCR C não for zero (Se Z = 0) |
| | RET | Retorna para programa principal |

8. Programa que gera uma contagem crescente, em hexadecimal, de 00 h a FF h e envia o resultado para a saída 1.

| Label | Mnemônico | Comentário |
|---|---|---|
| | MVI A,00h | Zera acumulador. A = 00 h |
| volta: | OUT 01h | Envia valor de A para a Porta de saída 01h |
| | INR A | Incrementar conteúdo do acumulador. A = A + 1 |
| | JNZ volta | "volta" se conteúdo do acumulador for diferente de zero |
| | HLT | Encerra quando conteúdo do acumulador for zero. |

Obs.: Nesse programa o acumulador inicia com valor 00h e encerra quando o acumulador volta para o valor 00h, após passar por todos os valores de 00 a FFh.

9. Subrotina de atraso de 1 ms. É feita a suposição de que um programa chama a subrotina denominada **atraso**, que é dada logo a seguir. É suposto um tempo de 1μs para cada estado.

_____
_____
CALL atraso
_____
_____

| Label | Mnemônico | Comentário |
|---|---|---|
| atraso: | MVI C,46h | Registrador C recebe valor 46 h ≡ 70 d |
| volta: | DCR C | Decrementa C. C = C - 1 |
| | JNZ volta | Volta, se flag de zero for zero (Z=0) $\Rightarrow$ C ≠ 0 |
| | NOP | Estado inoperante (No Operation). Apenas retardo |
| | HLT | Encerra quando conteúdo do acumulador for zero. |

Tempo gasto em cada instrução e tempo total:

| Instrução | Nº de Vezes que a instrução é executada | Nº de Estados de cada Instrução | Tempo de cada Estado (μs) | Tempo Parcial (μs) |
|---|---|---|---|---|
| MVI C | 1 | 7 | 1 | 7 |
| DCR C | 70 | 4 | 1 | 280 |
| JNZ (verdadeiro) | 69 | 10 | 1 | 690 |
| JNZ (falso) | 1 | 7 | 1 | 7 |
| NOP | 1 | 4 | 1 | 4 |
| RET | 1 | 10 | 1 | 10 |
| Tempo Total | | | | 998 μs ≅ 1 ms |

10. Subrotina de atraso de 10 ms. É suposto um tempo de 1μs para cada estado.

| Label | Mnemônico | Comentário |
|---|---|---|
| atraso: | MVI B,0Ah | Inicia contador B com valor 0Ah ≡ 10d |
| repete1 | MVI C,47h | Inicia contador C com valor 47h ≡ 71h |
| repete2: | DCR C | Decrementa C. C = C - 1 |
| | JNZ repete2 | Repete laço até registrador C = 0 ⇒ Z = 1. |
| | DCR B | Decrementa registrador B |
| | JNZ repete1 | Reinicia laço interno até zerar registrador B ⇒ Z = 1. |
| | RET | |

Tempo gasto em cada instrução e tempo total

| Instrução | Nº de Vezes a instrução é executada | Nº de Estados de cada Instrução | Tempo de cada Estado (μs) | Tempo Parcial (μs) |
|---|---|---|---|---|
| MVI B | 1 | 7 | 1 | 7 |
| MVI C | 10 | 7 | 1 | 70 |
| DCR C | 710 | 4 | 1 | 2840 |
| JNZ repete2 (verdadeiro) | 700 | 10 | 1 | 7000 |
| JNZ repete2 (falso) | 10 | 7 | 1 | 70 |
| DCR B | 10 | 4 | 1 | 40 |
| JNZ repete1 (verdadeiro) | 9 | 10 | 1 | 90 |
| JNZ repete1 (falso) | 1 | 7 | 1 | 7 |
| RET | 1 | 10 | 1 | 10 |
| Tempo Total | | | | 10134 μs ≅ 10 ms |

## 3.11 Exercícios Propostos

1. Cite todas as instruções possíveis a partir da instrução genérica MOV A, r.

2. Se os registradores H e L contêm, respectivamente, os valores 40 h e 50 h, qual o significado da instrução MVI M, 08h?

3. A que grupo de instruções pertence a instrução MVI M, 08h?

4. Explique em poucas palavras como funciona a pilha no 8085. Mostre, através de um mapa de memória, a evolução da pilha quando se realiza as seguintes instruções, na seqüência mostrada:

   PUSH B, PUSH D, ADD B, PUSH PSW, ADD D, POP PSW, OUT 90, POP D, POP B

   Valores iniciais: SP = 20C0 h, A = 33 h, B = 1C h, C = 4B h, D = 10 h, E = FE h e F = 3D h

5. Mostre, através de um mapa de memória, a evolução da pilha quando se executam as seguintes instruções, na seqüência mostrada:

   PUSH PSW, PUSH B, CALL ADIÇÃO, (RET), MOV B, A, POP B, POP PSW

   Valores iniciais: SP = 2090 h, A = 53 h, B = 0F h, C = 05 h, D = 12 h, E = 01 h e F = 55 h

   Endereço da chamada de subrotina "CALL ADIÇÃO": 2020 h

6. Mostre a evolução da pilha na execução das instruções a seguir:

   **a)** PUSH B, PUSH D, PUSH H, LDA 00FF h, POP H, POP D e POP B, sabendo que o valor inicial de SP é 38FC h e que os registradores B, C, D, E, H e L contém os valores 08 h, 1C h, 2A h, 06 h, FE h e 3Dh.
   **b)** CALL 033CH, ADD B e RET (ADD B e RET estão dentro da sub-rotina que se inicia na posição 033CH), nas mesmas condições do exercício anterior, supondo que o endereço da instrução CALL 033C h é 0038 h.

7. Considere o programa abaixo, em mnemônico, e responda as questões a seguir, sabendo que a subrotina no endereço 0200 h provoca um retardo de 1ms e afeta o registrador B.

| Endereço | Instrução |
|---|---|
| 4000 | LXI SP,4050h |
| | LXI H,402Fh |
| | MVI C,07h |
| volta: | INX H |
| | MOV A,M |
| | ANI 01h |
| | JNZ pula |
| | MOV A,M |
| | OUT 01h |
| pula: | CALL 0200h |
| | DCR C |
| | JNZ volta |
| | RST 4 |

| Endereço | Dados |
|---|---|
| 4030 | 10h |
| 4031 | 02h |
| 4032 | 05h |
| 4033 | F2h |
| 4034 | 0Ah |
| 4035 | 19h |
| 4036 | 03h |

   (a) O que o programa acima faz?
   (b) Desenhe uma tabela mostrando os endereços e o conteúdo da pilha após a execução da instrução CALL 0200h.
   (c) Quais os valores enviados pela porta de saída 01?
   (d) Mostre o que deve ser feito se o reg. B for usado no lugar do reg. C

8. Escreva um programa que produz um retardo de 1 s, sabendo que o 8085 é acionado por um cristal de 6 MHz.

9. Escreva, a partir do endereço 4050 h, uma sub-rotina que produza um retardo de 0.5 ms, aproximadamente. Suponha que a freqüência do cristal do 8085 seja 4,096 MHz.

10. Escreva um programa (usando bytes imediatos para os dados) que soma os decimais 500 e 650.

11. Descreva as etapas do ciclo de execução das instruções a seguir e explicar o que ocorre com os sinais de controle envolvidos e os sinais de endereços e dados:

    **a)** MOV B, M
    **b)** LXI D, 4050 h
    **c)** MOV E, B.

12. Faça o comentário de cada linha do programa abaixo e, a seguir, explique qual a finalidade do programa completo. Complete os endereços.

| End. | Mnemônico | Comentários |
|---|---|---|
| 4000 | LXI SP,4100h | |
| | LXI H,4031h | |
| | MVI C,0Ah | |
| | LDA 4030h | |
| volta: | CMP M | |
| | JC pula | |
| | MOV A,M | |
| pula: | INX H | |
| | DCR C | |
| | JNZ volta | |
| | CALL APDIS | |
| | CALL DBYTE | |
| | HLT | |

## 3.12 Referências Bibliográficas

[1] Ziller, Roberto M., "Microprocessadores – Conceitos Importantes," Editora do autor, Florianópolis, SC, 2000.

[2] Intel, "MCS-80/85$^{TM}$ Family User´s Manual," Outubro de 1979.

[3] Digibyte, "Manual do Usuário do Kit Didático KMD85", Abril de 1984.

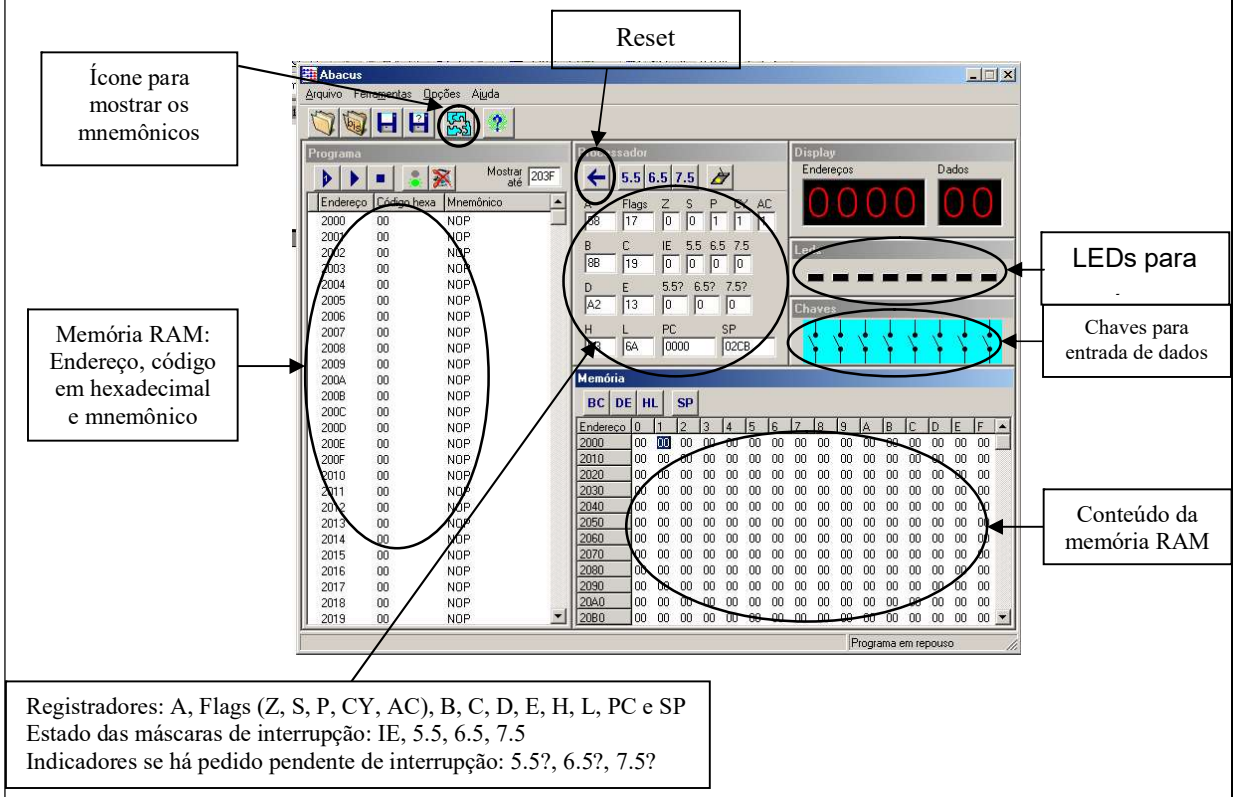
[4] Malvino, Albert P., "Microcomputadores e Microprocessadores," Tradução: Anatólio Laschuk, revisão técnica: Rodrigo Araes Caldas Farias, McGraw-Hill, São Paulo, 1985.

# 4. SIMULADOR DIGITAL ABACUS

## 4.1 Simulador ABACUS para o Microprocessador 8085

Os experimentos, em Engenharia Elétrica, tornam-se mais significativos quando precedidos de uma simulação digital. Na simulação pode-se fazer todas as considerações possíveis de uma forma mais rápida e mais eficiente, sem o risco de danificar quaisquer componentes ou equipamentos. Assim, o experimento seria apenas uma verificação e convalidação do modelo usado na simulação digital.

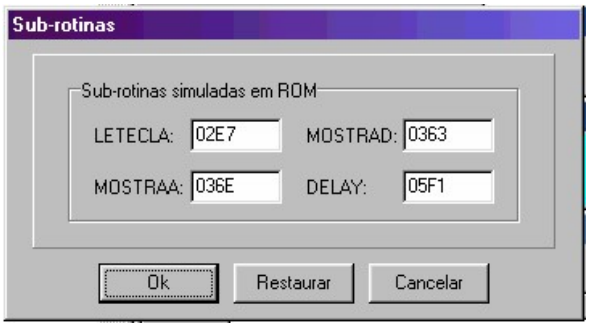
O simulador adotado é o ABACUS, que foi desenvolvido por pesquisadores da Universidade Federal de Santa Catarina (UFSC), sob a coordenação do Prof. Roberto M. Ziller. Esse simulador é apresentado a seguir, de forma simplificada. A página principal do simulador é mostrada a seguir:




O simulador conta com todas as possíveis instruções do 8085, cujos mnemônicos são mostrados quando o ícone "Assembler" é pressionado. Pode também ser mostrado através do "menu" chamado de "ferramentas". Os mnemônicos, do jeito que aparecem no ABACUS, são mostrados a seguir.

| Abacus Assembler | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| ACI | ADC | ADD | ADI | ANA | ANI | CALL | CC | CM | CMA | CMC |
| CMP | CNC | CNZ | CP | CPE | CPI | CPO | CZ | DAA | DAD | DCR |
| DCX | DI | EI | HLT | IN | INR | INX | JC | JM | JMP | JNC |
| JNZ | JP | JPE | JPO | JZ | LDA | LDAX | LHLD | LXI | MOV A | MOV B |
| MOV C | MOV D | MOV E | MOV H | MOV L | MOV M | MVI | NOP | ORA | ORI | OUT |
| PCHL | POP | PUSH | RAL | RAR | RC | RET | RIM | RLC | RM | RNC |
| RNZ | RP | RPE | RPO | RRC | RST | RZ | SBB | SBI | SHLD | SIM |
| SPHL | STA | STAX | STC | SUB | SUI | XCHG | XRA | XRI | XTHL | |

Da mesma forma que o kit didático, o ABACUS tem algumas sub-rotinas já prontas na memória ROM. No caso do ABACUS são apenas 4, mostradas na tela a seguir, cujo acesso, no ABACUS, é através do "menu" "Opções".



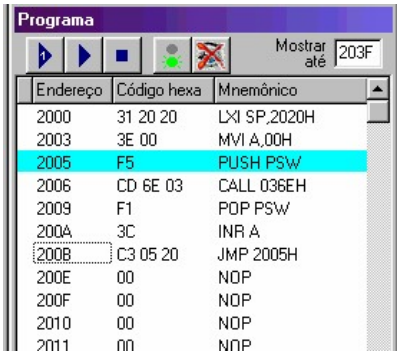

| Subrotina | Endereço (h) | Função | Entrada | Saída | Registradores Afetados |
|---|---|---|---|---|---|
| **LETECLA** | ***02E7*** | Mostra uma entrada de dados no simulador, através da qual um único caractere (0 a F) pode ser digitado | nenhuma | A - valor da tecla lida | A, H e L |
| **MOSTRAA** | ***036E*** | Mostra no display de dados o conteúdo do registrador A | A | nenhuma | todos |
| **MOSTRAD** | ***0363*** | Mostra no display de endereços o conteúdo dos registradores D e E | D - byte mais significativo<br>E - byte menos significativo | nenhuma | todos |
| **DELAY** | ***05F1*** | Provoca atraso de tempo proporcional ao conteúdo de D | D - 1º byte a ser mostrado no display<br>E - 2º byte a ser mostrado no display | nenhuma | A, D e E |

Nesse momento é interessante refazer o exemplo usado com o kit didático. Três detalhes devem ser observados:

1. A memória RAM do ABACUS começa no endereço 2000 h, ao invés de 4000 h.
2. Não há necessidade de usar uma subrotina para apagar o display no ABACUS. No kit essa subrotina é essencial.
3. É fundamental definir a posição da pilha no ABACUS, através do comando LXI SP, xx xx. Caso contrário, pode ocorrer erros, com o simulador tentando acessar endereço fora da faixa permitida.

| Label | Endereço | Mnemônico | Código | Comentário |
|---|---|---|---|---|
| | 2000 | LXI SP, 2020 | 31 20 20 | Define pilha na posição 2020 h da RAM |
| | 2003 | MVI A,00h | 3E 00 | Carrega registrador A com valor 00 h |
| volta: | 2005 | PUSH PSW | F5 | Guarda valor de A e flags na pilha |
| | 2006 | CALL MOSTRAA | CD 6E 03 | Mostra conteúdo de A no display. Afeta "A" |
| | 2009 | POP PSW | F1 | Recupera valor de A guardado na pilha |
| | 200A | INR A | 3C | Incrementa conteúdo de A |
| | 200b | JMP volta | C3 05 20 | Volta para a posição "volta", ou seja, 2005 h |

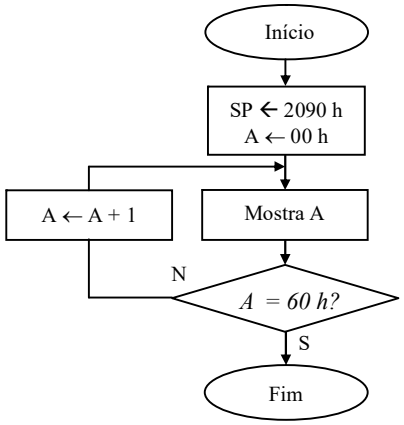
A visualização desse programa na memória RAM do ABACUS é mostrada na figura a seguir.



## 4.2 Exemplos de Programas em Assembly para o ABACUS

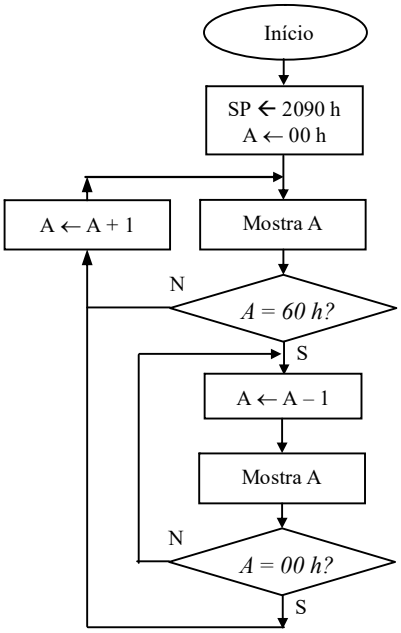
A presente seção mostra dois exemplos de programação voltada para simulação no ABACUS.

Exemplo 1: Faça no ABACUS um programa que executa uma contagem crescente em hexadecimal de 00 h até 60 h.




| Label | Mnemônico |
|---|---|
| | LXI SP,2090H |
| | MVI A,00H |
| volta: | PUSH PSW |
| | CALL MOSTRAA |
| | POP PSW |
| | CPI 60H |
| | JZ fim |
| | INR A |
| | JMP volta |
| fim: | HLT |

Exemplo 2: Faça no ABACUS um programa que executa, de forma ininterrupta, uma contagem hexadecimal crescente de 00h até 60h seguida de uma contagem hexadecimal decrescente de 60h até 00h.




| Label | Mnemônico |
|---|---|
| | LXI SP,2090H |
| | MVI A,00H |
| volta: | PUSH PSW |
| | CALL MOSTRAA |
| | POP PSW |
| | CPI 60H |
| | JZ decresce |
| cresce: | INR A |
| | JMP volta |
| decresce: | DCR A |
| | PUSH PSW |
| | CALL MOSTRAA |
| | POP PSW |
| | CPI 00H |
| | JZ cresce |
| | JMP decresce |

## 4.3 Exercícios Propostos

1. Faça no ABACUS um programa que executa uma contagem decimal de 00 a 60.

2. Faça no ABACUS um programa que executa uma contagem decrescente em hexadecimal de 60 h até 00 h.

3. Repita o problema anterior para uma contagem decimal de 60 até 00.

4. Faça no ABACUS um programa que seleciona e mostra no display o maior número contido em uma tabela inserida na memória. A tabela contém números aleatórios e tem início no endereço 2050h e termine no endereço 205Fh.

5. Repita o problema anterior, selecionando e mostrando o menor número.

6. Faça no ABACUS um programa que seleciona e mostra no display os números ímpares contidos em uma tabela inserida na memória. A tabela contém números aleatórios e tem início no endereço 2050h e termine no endereço 205Fh. Use uma subrotina de atraso com D = 02 h entre os valores mostrados no display.

7. Faça no ABACUS um programa que faz a ordenação em ordem crescente de uma tabela contendo 16 números de 8 bits. Os números já estão na memória a partir do endereço 2050h e devem ser mantidos nessa faixa de endereços, porém, ordenados.

8. Adapte o programa do problema 7 para ordenar os números em ordem decrescente.

9. Faça no ABACUS um programa que seleciona e mostra no display os números maiores ou iguais a 20h e menores que 50h, de uma tabela com 16 números, começando do endereço 2050 h.
   Sugestão de tabela: 05h, 15h, 65h, 95h, 35h, 20h, 50h, 42h, 72h, 10h, 60h, 45h, 33h, 25h, 48h, 49h

10. Repita o problema 9, mostrando os números fora do intervalo.

11. Faça no ABACUS um programa que mostra um número (1 byte), lido pelo teclado, e multiplique este número por outro número, também obtido pelo teclado, mostrando o resultado no display. O resultado deverá conter 2 bytes.

12. Faça no ABACUS um programa que leia do teclado uma temperatura em $^{o}$C (um byte) e converta-a em $^{o}$F (graus Farenheit) mostrando o resultado no display. (o programa só deve aceitar temperaturas de entrada de até 255 graus farenheit, ou seja, um byte na saída). Trabalhar em hexadecimal.

    $$T(^{o}F) = (9/5) * T(^{o}C) + 32 \text{ (números da equação em decimal)}$$

13. Faça no ABACUS um programa que adicione dois números binários de qualquer tamanho (N bytes cada um). O tamanho dos números binários (N) está na posição 2040 h. Os números começam nas posições 2041 e 2061 (1$^{o}$ byte é o menos significativo). O resultado deve substituir o número que começa na posição 2041.

14. Escrever uma sub-rotina que divida dois números de 8 bits. Os dois números estão nas posições 2050 (dividendo) e 2051 (divisor). O quociente deve ser armazenado na posição 2052 e o resto na posição 2053.

15. Faça no ABACUS um programa que lê um número de 1 byte pelo teclado (dividendo), em seguida lê outro número de 1 byte pelo teclado (divisor), faz divisão e mostra o quociente no display de endereços e o resto no display de dados.

16. Faça no ABACUS um programa que lê um número "×" pelo teclado e mostra no display o resultado da operação 3× + 5.

17. Faça no ABACUS um programa que lê um número "×" pelo teclado e mostra no display o resultado da operação 5×- 2.

18. Faça no ABACUS um programa que lê um número "×" pelo teclado e mostra no display o resultado da operação 3(× + 5).

19. Faça no ABACUS um programa que lê um número "×" pelo teclado e mostra no display o resultado da operação 5(× - 2).

20. O trecho de programa dado abaixo é usado para montar uma tabela de 10 números (cada um com dois dígitos).

| LABEL | MNEMÔNICO |
|---|---|
| | LXI SP,2080h |
| | LXI H,2050h |
| | MVI C,0Ah |
| LÊ_PRÓXIMO: | CALL leitura |
| | MOV M,A |
| | PUSH H |
| | PUSH B |
| | CALL MOSTRAA |
| | POP B |
| | POP H |
| | INX H |
| | DCR C |
| | JNZ LÊ_PRÓXIMO |

| LABEL | MNEMÔNICO |
|---|---|
| LEITURA: | PUSH H |
| | CALL LETECLA |
| | RLC |
| | RLC |
| | RLC |
| | RLC |
| | MOV B,A |
| | CALL LETECLA |
| | ORA B |
| | POP H |
| | RET |
| | |
| | |

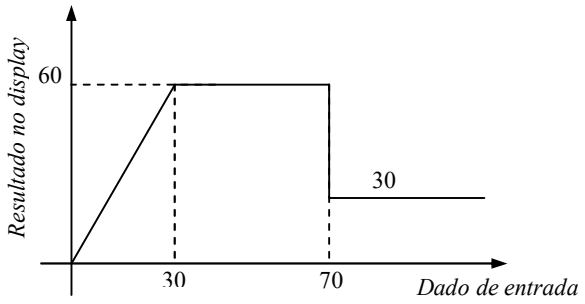
Após analisar o trecho de programa dado, pede-se:

(a) Explique o funcionamento da subrotina "LEITURA";
(b) Faça a continuação do programa acima de modo que o maior número da tabela montada seja mostrado no display de dados;
(c) Faça a continuação do programa acima de modo que o menor número da tabela montada seja mostrado no display de dados;

21. Codifique as instruções dadas a seguir usando o mnemônico do 8085; indique o endereço de cada instrução e indique o conteúdo dos registradores pedidos e das flags de carry e de zero, após a execução da instrução indicada.

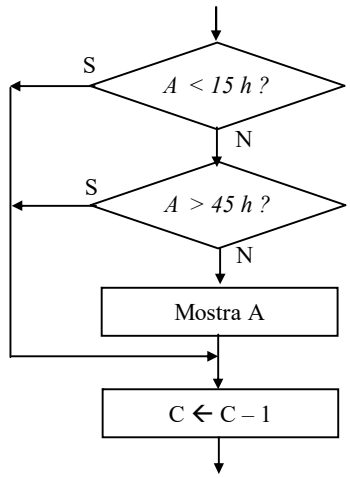
| **OPERAÇÃO** | **END.** | **MNEMÔNICO** | **A** | **C** | **CY** | **Z** |
|---|---|---|---|---|---|---|
| (A) ← 70 h | 2000 | MVI A, 70 h | | | | |
| (C) ← 0F h | | MVI C, 0F h | | | | |
| (H,L) ← 2020 h | | LXI H, 2030 h | | | | |
| ((H)(L)) ← F0 h | | MVI M, F0 h | | | | |
| (A) ← (A) ∧ (C) | | ANA C | | | | |
| (A) – ((H)(L)) | | CMP M | | | | |
| (A) ← (A) + (C) + CY | | ADC C | | | | |
| (C) ← (C) ∨ ((H)(L)) | | MOV A,C | | | | |
| | | ORA M | | | | |
| | | MOV C,A | | | | |
| (A) – FF h | | CPI FF h | | | | |

22. O gráfico a seguir representa a função matemática de um componente não identificado. Escreva em assembly do 8085 (mnemônico do 8085) um programa para ler um dado de 1 byte pelo teclado e mostrar no display de dados o resultado da aplicação da função. Use as sub-rotinas necessárias do ABACUS para a entrada de dados e para mostrar o resultado no display de dados. O programa deve sempre voltar para a entrada de um novo dado.



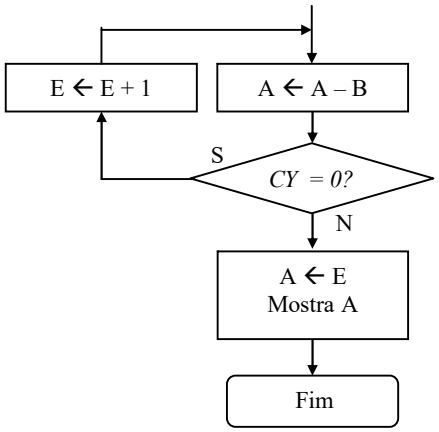
23. Codifique, usando mnemônicos do microprocessador 8085 e recursos do ABACUS, os trechos de programas representados nos fluxogramas dados a seguir:

(a)




| Label | Mnemônico |
|---|---|
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |

(b)




| Label | Mnemônico |
|---|---|
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |

24. Faça, para executar no ABACUS, um programa que lê um número "x" de 1 dígito pelo teclado e mostra no display de dados o resultado de $x^2 - 2$.

## 4.4 Referências Bibliográficas

[1] Ziller, Roberto M., "Microprocessadores – Conceitos Importantes," Editora do autor, Florianópolis, SC, 2000.

[2] Intel, "MCS-80/85™ Family User´s Manual," Outubro de 1979.

[3] Malvino, Albert P., "Microcomputadores e Microprocessadores," Tradução: Anatólio Laschuk, revisão técnica: Rodrigo Araes Caldas Farias, McGraw-Hill, São Paulo, 1985.