



**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, 2006**

## Observações

1. Esta apostila destina-se ao ensino de Microprocessadores e Microcontroladores para alunos da 4ª Série de Engenharia Elétrica da Escola de Engenharia Elétrica e de Computação da Universidade Federal de Goiás
2. Não se pretende, com o uso desta apostila, dispensar os livros didáticos indicados na referência básica, mas apenas facilitar o ensino da disciplina em questão, reunindo num só volume o material a ser usado em sala de aula.

3. A apostila está distribuídas em capítulos que cobrem dois dos três temas principais, que são:

Microprocessador 8085 - Neste tópico são abordados desde a estrutura básica de um microprocessador, até a aplicação do mesmo num sistema mínimo com microprocessador, memória, portas de entrada e saída e outros periféricos. As instruções em assembly do 8085 são empregadas na solução de vários problemas de cunho didático, com auxílio do simulador ABACUS.

Microprocessador 8088 - O objetivo deste tópico é fazer um estudo comparativo entre um microprocessador de 16 bits e o microprocessador 8085 (8 bits) e estudar o princípio de funcionamento do 8088/8086. É utilizado um simulador digital para a execução de alguns programas simples.

O capítulo sobre o microcontrolador 8051 está disponível à parte.

4. Os tópicos abordados são acompanhados de experimentos de laboratório.

## Informações sobre a Disciplina

<i>Disciplina:</i>	<b>MICROPROCESSADORES E MICROCOMPUTADORES</b>
<i>Código:</i>	13.05.035
<i>Carga Horária:</i>	96 horas (teóricas) e 32 horas (de laboratório)
<i>Grade Curricular:</i>	1992 - (disciplina do 4º ano)
<i>Ano Letivo:</i>	2006
<i>Aulas Teóricas:</i>	Quarta-feira: 13:30h às 15:10h (alternando com Controle) Sexta-feira: 15:30h às 17:10h
<i>Aulas de Laboratório:</i>	Segunda-feira à tarde (13:30 às 18:30h)
<i>Professor:</i>	José Wilson Lima Nerys

Obs.: As 96 horas de aulas teóricas serão ministradas em 48 aulas de 2 horas cada uma.  
As 32 horas de aulas práticas serão ministradas em 16 aulas de 2 horas cada uma.

## BIBLIOGRAFIA BÁSICA

### Microprocessador 8085:

1. ZILLER, Roberto M., “Microprocessadores – Conceitos Importantes,” Edição do autor, Florianópolis, 2000. ISBN 85-901037-2-2
2. NERYs, José Wilson L., “Apostila de Microprocessadores”

### Microprocessador 8086:

3. ZILLER, Roberto M., “Microprocessadores – Conceitos Importantes,” Edição do autor, Florianópolis, 2000. ISBN 85-901037-2-2
4. ZELENOVSKY, Ricardo e MENDONÇA, Alexandre, "PC: Um Guia Prático de Hardware e Interfaceamento," Interciência, Rio de Janeiro, 1996. ISBN: 85-7193-001-5
5. NERYs, José Wilson L., “Apostila de Microprocessadores”

### Microcontrolador 8051:

6. SILVA JR., Vidal Pereira da, “Aplicações Práticas do Microcontrolador 8051,” Érica, São Paulo, 1994.
7. GIMENEZ, Salvador P., “Microcontroladores 8051: Teoria do hardware e do software / Aplicações em controle digital / Laboratório e simulação,” Pearson Education do Brasil Ltda, São Paulo, 2002. ISBN: 85.87918-28-1
8. NERYs, José Wilson L., “Apostila de Microprocessadores”

## EMENTA

Conceitos básicos de microprocessadores e microcontroladores. Arquitetura de microprocessadores e microcontroladores. Princípio de funcionamento de microprocessadores e microcontroladores. Modos de endereçamento. Programação de microcontroladores. Entrada/saída. Dispositivos periféricos. Interrupções. Temporizadores. Acesso direto à memória. Barramentos padrões. Expansão e mapeamento de memória. Ferramentas para análise, desenvolvimento e depuração.

## PROGRAMA

### 1 - Introdução aos Computadores e Microprocessadores:

- Histórico
- Computador (definições)
- Estrutura Básica do Computador
- Unidade Central de Processamento - CPU
- Memória
- Unidade de Entrada e Saída
- Microprocessadores e Microcontroladores
- Outros Conceitos Básicos
- Ciclo de Clock, Ciclo de Máquina e Ciclo de Instrução
- Memória ROM e Memória RAM
- Memória RAM Estática e Memória RAM Dinâmica
- Registradores
- Diagrama de Blocos de uma CPU Genérica
- Sistemas de Numeração

### 2 - Arquitetura do 8085:

- Princípio Básico de Operação de um Microprocessador
- Diagrama de Blocos do Microprocessador 8085
- Pinagem;
- Principais Características;
- O Sistema Mínimo;
- Modos de Endereçamento;
- Busca e Execução de Instruções.

### 3 - Conjunto de Instruções do 8085:

- Transferência de Dados;
- Aritméticas e Lógicas;
- Rotação e Deslocamento;
- Desvio;
- Entrada e Saída;
- Controle.

### 4 - Princípios Básicos de Interfaceamento de Micros:

- Interface Paralela
- Interface Serial
- Unidade de Temporização
- Controlador de Interrupção
- Controlador de DMA (DMAC)
- Integrados de Suporte
- Arquitetura atual de um PC

### 5 - Arquitetura do 8086/8088 - Família 80x86:

- Diagrama em Blocos;
- Registradores;
- Segmentação de Memória;
- Pinagem;
- Modos de Endereçamento;
- Evolução dos processadores da família 80X86.

### 6 - Introdução ao Microcontrolador 8051

- Arquitetura da Família do Microcontrolador 8051
- Características Principais
- Programação em Linguagem Assembly

- Interrupções
- Temporizadores
- Comunicação Serial
- Simulação Digital
- Construindo um Sistema Baseado no Microcontrolador 8051

7 - Projeto Experimental usando Microcontrolador da família 8051

8 - Atividades de Laboratório

### LABORATÓRIO

Laboratório	Conteúdo
1	Uso do Kit do 8085 Programa de Simulação ABACUS
2	Instruções de transferência de dados (uso do Kit e do simulador ABACUS)
3	Instruções aritméticas (uso do Kit e do simulador ABACUS)
4	Instruções lógicas (uso do kit e do simulador ABACUS)
5	Programação 8085 (uso do kit e do simulador ABACUS)
6	Programação 8085 com Interrupção
7	Programação 8085 com Interrupção
8	Microprocessador 8086: características básicas
9	Microcontrolador 8051: Sequência de LEDs e Motor de passo (uso de Kit e de simulador)
10	Microcontrolador 8051: Motor de corrente contínua (uso de Kit e de simulador)
11	Microcontrolador 8051: Conversores AD e DA (uso de Kit e de simulador)
12	Projeto usando 8051
13	Projeto usando 8051
14	Projeto usando 8051
15	Projeto usando 8051
16	Projeto usando 8051

### AVALIAÇÃO

Nota	Tipo de Avaliação	Valor Máximo
Nota 1	Prova 1	8,0
	Laboratório	2,0
Nota 2	Prova 2	8,0
	Laboratório	2,0
Nota 3	Prova 3	8,0
	Laboratório	2,0
Nota 4	Prova 4	5,0
	Projeto Experimental	5,0

<b>1. INTRODUÇÃO AOS COMPUTADORES E MICROPROCESSADORES.....</b>	<b>9</b>
1.1 HISTÓRICO SOBRE COMPUTADORES.....	9
1.2 NÚMERO DE TRANSISTORES EM UM MICROPROCESSADOR .....	13
1.3 DEFINIÇÕES E CLASSIFICAÇÕES BÁSICAS .....	14
1.4 ESTRUTURA BÁSICA DE UM COMPUTADOR .....	15
1.4.1 UNIDADE CENTRAL DE PROCESSAMENTO (CPU) .....	16
1.4.2 MEMÓRIA .....	16
1.4.3 UNIDADE DE ENTRADA E SAÍDA (I/O) .....	16
1.4.4 BARRAMENTO .....	16
1.5 ÍNDICE DE DESEMPENHO DE PROCESSADORES .....	17
1.6 MICROPROCESSADOR x MICROCONTROLADOR .....	17
1.7 OUTROS CONCEITOS BÁSICOS: .....	17
1.8 SISTEMAS DE NUMERAÇÃO.....	19
1.8.1 SISTEMA DECIMAL .....	19
1.8.2 SISTEMA BINÁRIO .....	19
1.8.3 SISTEMA BCD (BINARY-CODED DECIMAL) .....	19
1.8.4 SISTEMA OCTAL .....	19
1.8.5 SISTEMA HEXADECIMAL .....	20
1.9 EXERCÍCIOS PROPOSTOS .....	20
1.10 REFERÊNCIAS BIBLIOGRÁFICAS.....	21
<b>2. ARQUITETURA E PRINCÍPIO DE FUNCIONAMENTO DO 8085.....</b>	<b>22</b>
2.1 DIAGRAMA DE BLOCOS DO MICROPROCESSADOR 8085.....	22
2.2 UNIDADES INTERNAS E REGISTRADORES DO 8085.....	23
2.3 FREQUÊNCIA DE CLOCK .....	25
2.4 PINAGEM DO 8085.....	25
2.5 SISTEMA BÁSICO DE TEMPORIZAÇÃO E PRINCÍPIO DE OPERAÇÃO .....	27
2.6 FORMATO DAS INSTRUÇÕES .....	30
2.7 EXERCÍCIOS PROPOSTOS .....	31
2.8 REFERÊNCIAS BIBLIOGRÁFICAS.....	32
<b>3. CONJUNTO DE INSTRUÇÕES DO MICROPROCESSADOR 8085.....</b>	<b>33</b>
3.1 SIMBOLOGIA DAS INSTRUÇÕES.....	33
3.2 MODOS DE ENDEREÇAMENTO .....	33
3.3 GRUPOS DE INSTRUÇÕES.....	34
3.4 INSTRUÇÕES DE TRANSFERÊNCIA DE DADOS .....	35
3.5 INSTRUÇÕES ARITMÉTICAS .....	39
3.6 INSTRUÇÕES LÓGICAS.....	43
3.7 INSTRUÇÕES DE DESVIO.....	46
3.8 INSTRUÇÕES DE CONTROLE, PILHA E ENTRADA E SAÍDA .....	49
3.9 FUNCIONAMENTO DA PILHA .....	51
3.10 EXEMPLOS DE PROGRAMAS EM ASSEMBLY DO 8085.....	52
3.11 EXERCÍCIOS PROPOSTOS .....	54
3.12 REFERÊNCIAS BIBLIOGRÁFICAS.....	56
<b>4. SIMULADOR DIGITAL ABACUS .....</b>	<b>57</b>
4.1 SIMULADOR ABACUS PARA O MICROPROCESSADOR 8085.....	57
4.2 EXEMPLOS DE PROGRAMAS EM ASSEMBLY PARA O ABACUS .....	59
4.3 EXERCÍCIOS PROPOSTOS .....	60

<b>4.4</b>	<b>REFERÊNCIAS BIBLIOGRÁFICAS.....</b>	<b>62</b>
<b>5.</b>	<b>KIT DIDÁTICO E ROTEIROS DE LABORATÓRIO .....</b>	<b>63</b>
<b>5.1</b>	<b>SUB-ROTINAS DO PROGRAMA MONITOR DO KIT DIDÁTICO .....</b>	<b>63</b>
<b>5.2</b>	<b>PRINCIPAIS COMANDOS E ORIENTAÇÕES PARA USO DO KIT DIDÁTICO .....</b>	<b>64</b>
<b>5.3</b>	<b>ROTEIROS DE EXPERIMENTOS.....</b>	<b>66</b>
<b>5.4</b>	<b>REFERÊNCIAS BIBLIOGRÁFICAS.....</b>	<b>74</b>
<b>6.</b>	<b>INTERRUPÇÕES E OPERAÇÕES DE ENTRADA E SAÍDA .....</b>	<b>75</b>
<b>6.1</b>	<b>INSTRUÇÕES DE RECOMEÇO.....</b>	<b>75</b>
<b>6.2</b>	<b>INTERRUPÇÕES .....</b>	<b>75</b>
<b>6.3</b>	<b>CIRCUITOS DE INTERRUPÇÃO.....</b>	<b>76</b>
<b>6.4</b>	<b>SETAGEM E LEITURA DA MÁSCARA DE INTERRUPÇÃO .....</b>	<b>79</b>
<b>6.5</b>	<b>AMPLIANDO A CAPACIDADE DE INTERRUPÇÃO .....</b>	<b>80</b>
<b>6.6</b>	<b>DISPOSITIVOS DE ENTRADA E SAÍDA.....</b>	<b>81</b>
<b>6.7</b>	<b>EXERCÍCIOS PROPOSTOS .....</b>	<b>83</b>
<b>6.8</b>	<b>REFERÊNCIAS BIBLIOGRÁFICAS.....</b>	<b>84</b>
<b>7.</b>	<b>ENTRADA E SAÍDA SERIAL E PASTILHAS INTEGRADAS DE SUPORTE .....</b>	<b>85</b>
<b>7.1</b>	<b>INTERFACE SERIAL NO PC.....</b>	<b>85</b>
<b>7.2</b>	<b>INTEGRADOS DE SUPORTE .....</b>	<b>88</b>
7.2.1	MEMÓRIAS RAM E ROM.....	88
7.2.2	DECODIFICADOR 74LS138.....	94
<b>7.3</b>	<b>EXERCÍCIOS PROPOSTOS .....</b>	<b>95</b>
<b>7.4</b>	<b>REFERÊNCIAS BIBLIOGRÁFICAS.....</b>	<b>97</b>
<b>8.</b>	<b>INTERFACE ANALÓGICA .....</b>	<b>98</b>
<b>8.1</b>	<b>CONVERSOR DIGITAL-ANALÓGICO .....</b>	<b>98</b>
<b>8.2</b>	<b>CONVERSOR ANALÓGICO-DIGITAL .....</b>	<b>101</b>
<b>8.3</b>	<b>EXERCÍCIOS PROPOSTOS .....</b>	<b>102</b>
<b>8.4</b>	<b>REFERÊNCIAS BIBLIOGRÁFICAS.....</b>	<b>103</b>
<b>9.</b>	<b>MICROCONTROLADORES 8086/8088.....</b>	<b>104</b>
<b>9.1</b>	<b>INTRODUÇÃO .....</b>	<b>104</b>
<b>9.2</b>	<b>DIAGRAMA DE BLOCOS DO 8088 .....</b>	<b>106</b>
<b>9.3</b>	<b>OS REGISTRADORES DO 8088 .....</b>	<b>108</b>
9.3.1	REGISTRADORES DE DADOS.....	109
9.3.2	REGISTRADORES APONTADORES E ÍNDICES .....	109
9.3.3	REGISTRADORES DE SEGMENTO .....	110
9.3.4	APONTADOR DE INSTRUÇÕES .....	111
<b>9.4</b>	<b>A PINAGEM DO 8088.....</b>	<b>112</b>
<b>9.5</b>	<b>CICLOS DE BARRAMENTO DO 8086 .....</b>	<b>114</b>
<b>9.6</b>	<b>ENDEREÇAMENTO DE MEMÓRIA.....</b>	<b>116</b>
<b>9.7</b>	<b>LINGUAGEM DE PROGRAMAÇÃO ASSEMBLY DO 8086 .....</b>	<b>118</b>
<b>9.8</b>	<b>MODOS DE ENDEREÇAMENTO E ENDEREÇOS DE MEMÓRIA EFETIVO.....</b>	<b>119</b>
<b>9.9</b>	<b>INTERRUPÇÕES DO 8086.....</b>	<b>121</b>
9.9.1	ESTRUTURA DE INTERRUPÇÃO DO 8086: .....	121

9.9.2	INTERRUPÇÕES PRÉ-DEFINIDAS, RESERVADA OU EXCEÇÕES.....	122
9.9.3	INTERRUPÇÕES POR HARDWARE .....	122
9.9.4	INTERRUPÇÕES POR SOFTWARE.....	122
<b>9.10</b>	<b>CONJUNTO DE INSTRUÇÕES DO 8088/86 .....</b>	<b>123</b>
9.10.1	INSTRUÇÕES DE TRANSFERÊNCIA DE DADOS .....	123
9.10.2	INSTRUÇÕES DE STRINGS .....	126
9.10.3	INSTRUÇÕES LÓGICAS.....	128
9.10.4	INSTRUÇÕES ARITMÉTICAS.....	129
9.10.5	INSTRUÇÕES DE DESVIO.....	129
9.10.6	INSTRUÇÕES DE CONTROLE .....	130
<b>9.11</b>	<b>CARACTERÍSTICAS DE I/O DO PC-XT.....</b>	<b>130</b>
9.11.1	DIVISÃO DOS ENDEREÇOS DOS DISPOSITIVOS DE I/O DO PC-XT .....	130
9.11.2	ENDEREÇOS DOS DISPOSITIVOS DA PLACA PRINCIPAL (PLACA MÃE) .....	131
9.11.3	ENDEREÇOS DOS DISPOSITIVOS DOS SLOTS.....	132
9.11.4	O SLOT DO PC (ISA 8 BITS) .....	132
<b>9.12</b>	<b>DECODIFICAÇÃO DE ENDEREÇOS.....</b>	<b>133</b>
<b>9.13</b>	<b>EXEMPLOS GERAIS.....</b>	<b>133</b>
<b>9.14</b>	<b>PROBLEMAS PROPOSTOS .....</b>	<b>134</b>
<b>9.15</b>	<b>REFERÊNCIAS BIBLIOGRÁFICAS.....</b>	<b>135</b>



# 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.

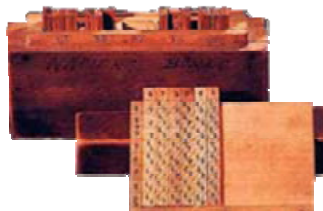
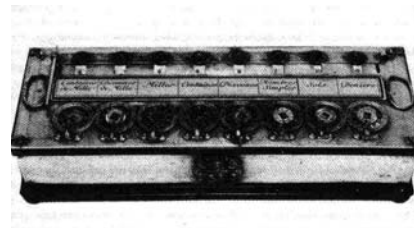


Tabela de logaritmos

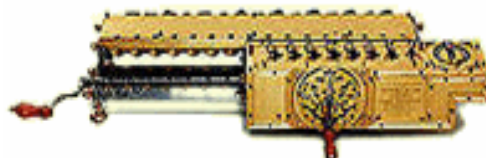


Pascaline

**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.

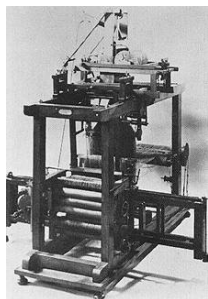
**1642 – PASCALINE** – O cientista francês **BLAISE PASCAL** criou uma calculadora capaz de realizar operações de adição e subtração. A máquina implementada utilizava rodas e engrenagens, com as quais era possível representar números decimais de 0 a 9. Pascal desenvolveu essa máquina para ajudar seu 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](http://www.automates-anciens.com)). Em **1749** ele construiu o primeiro **TEAR AUTOMÁTICO**, que aceitava comandos através de um cilindro de metal perfurado.



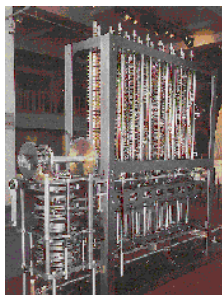
Tear de Vaucanson



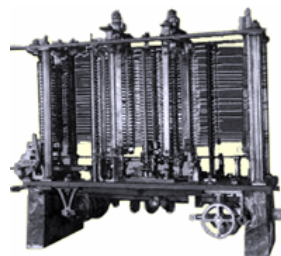
Tear de Jacquard

**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 (*Tabulation Machine Company*).

Para ampliar seus negócios, a **TMC** se uniu com duas pequenas empresas para formar a **CTRC** (*Computing Tabulation Recording Company*), em 1914. Em 1924, a **CTRC** se tornou uma empresa internacional e mudou seu nome para **IBM** (*Internacional Business Machine*).



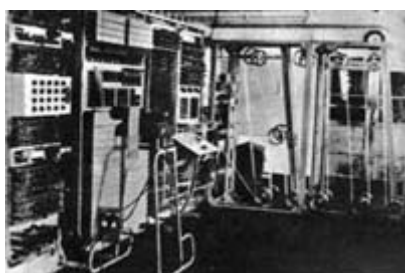
Máquina de Herman Hollerith



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 decodificar 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 Turing” 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, unidade 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 MultiMedia 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.

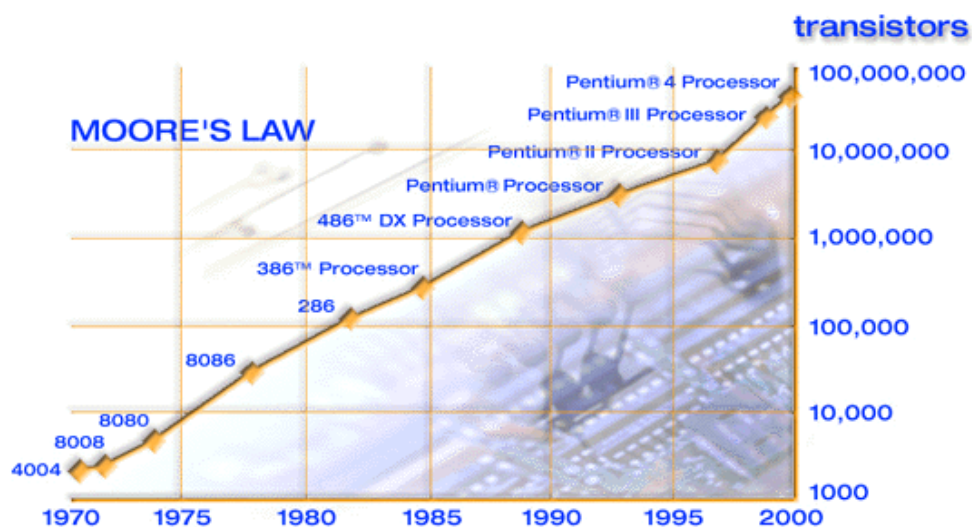


Fig. 1.1: Evolução do número de transistores no microprocessador



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 que 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>).

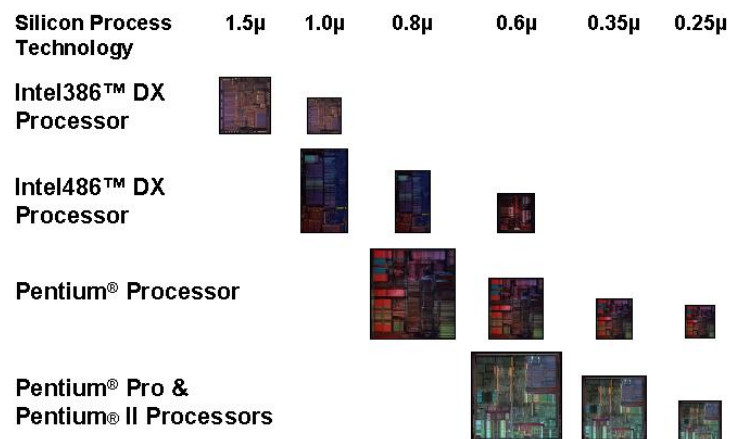


Fig. 1.2: Redução do tamanho físico dos processadores

### 1.3 Definições e Classificações Básicas

#### ➤ COMPUTADOR (definições):

- Máquina destinada a reduzir cálculos (cálculos) e tomadas de decisões;
- Máquina que executa uma sequência 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

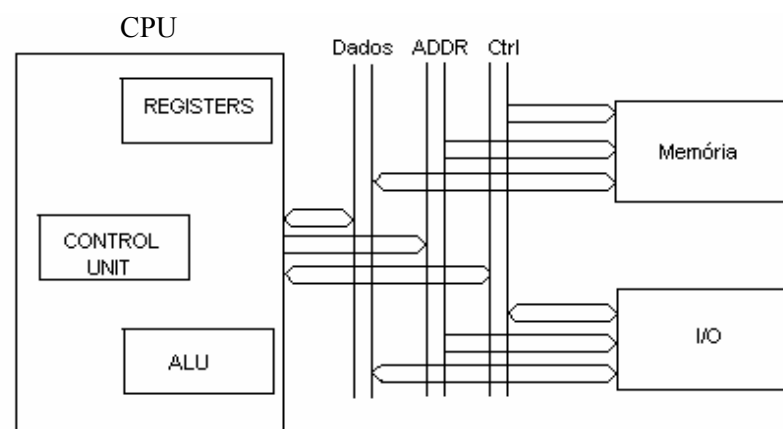


Fig. 1.3: Estrutura básica de um computador

### 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 (Arithmetic 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 torne-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 ( $4.294.967.296 = 4 \times 1024 \times 1024 \times 1024 = 4 \text{ GB}$ ). 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 (Industry Standard Architecture)** 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 (Peripheral Component Interconnect)** foi desenvolvido 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 (Plug and Play)**. Seu clock é geralmente de 33 MHz, para valores de clock interno acima de 150 MHz.

**Barramento AGP** - O barramento **AGP (Accelerated Graphics Port)** foi desenvolvido 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.

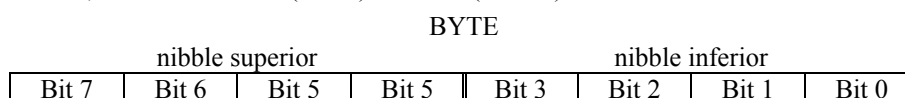
**FLOPS** - *FLOating point instructions Per Seconds* (Instruções com Ponto Flutuante Por Segundo). É também uma unidade de desempenho do microprocessador. Indica a capacidade de trabalhar com números decimais.

**Representação em Ponto Fixo** - Sistema numérico no qual o ponto está implicitamente fixo (à direita do dígito 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 0 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).



**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 instruções.

**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<sup>2</sup>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 equivale 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.

$$1 \text{ kbyte} = 2^{10} = 1.024 \text{ bytes};$$

$$1 \text{ Mbyte} = 2^{10} \times 2^{10} = 1.048.576 \text{ bytes} = 1.024 \text{ kbytes};$$

$$1 \text{ Gbyte} = 2^{10} \times 2^{10} \times 2^{10} = 1.073.741.824 \text{ bytes} = 1.024 \text{ Mbytes}$$

### 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<sub>16</sub>**

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<sub>16</sub> = 0111 1101 0011 1111<sub>2</sub>**

**Ex.: 1010000110111000<sub>2</sub> = 1010 0001 1011 1000<sub>2</sub> = A 1 B 8<sub>16</sub>**

### 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 frequência 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)  $345_{10} = \underline{\hspace{2cm}}_h$
- (b)  $10100111_2 = \underline{\hspace{2cm}}_h$
- (c)  $10101010_8 = \underline{\hspace{2cm}}_h$
- (d)  $255_{10} = \underline{\hspace{2cm}}_h$
- (e)  $128_{10} = \underline{\hspace{2cm}}_h$
- (f)  $10100111011_2 = \underline{\hspace{2cm}}_h$

### 1.10 Referências Bibliográficas

- [1] Ziller, Roberto M., “Microprocessadores – Conceitos Importantes,” Edição do Autor, Florianópolis, SC, 2000.
- [2] <http://www.intel.com/research/silicon/moorespaper.pdf>
- [3] [www.maebee.com.br](http://www.maebee.com.br)
- [4] [http://www.intel.com/intel/intelis/museum/exhibits/hist\\_micro/hof/](http://www.intel.com/intel/intelis/museum/exhibits/hist_micro/hof/)
- [5] <http://www.computerhistory.org/timeline/>

## 2. ARQUITETURA E PRINCÍPIO DE FUNCIONAMENTO DO 8085

### 2.1 Diagrama de Blocos do Microprocessador 8085

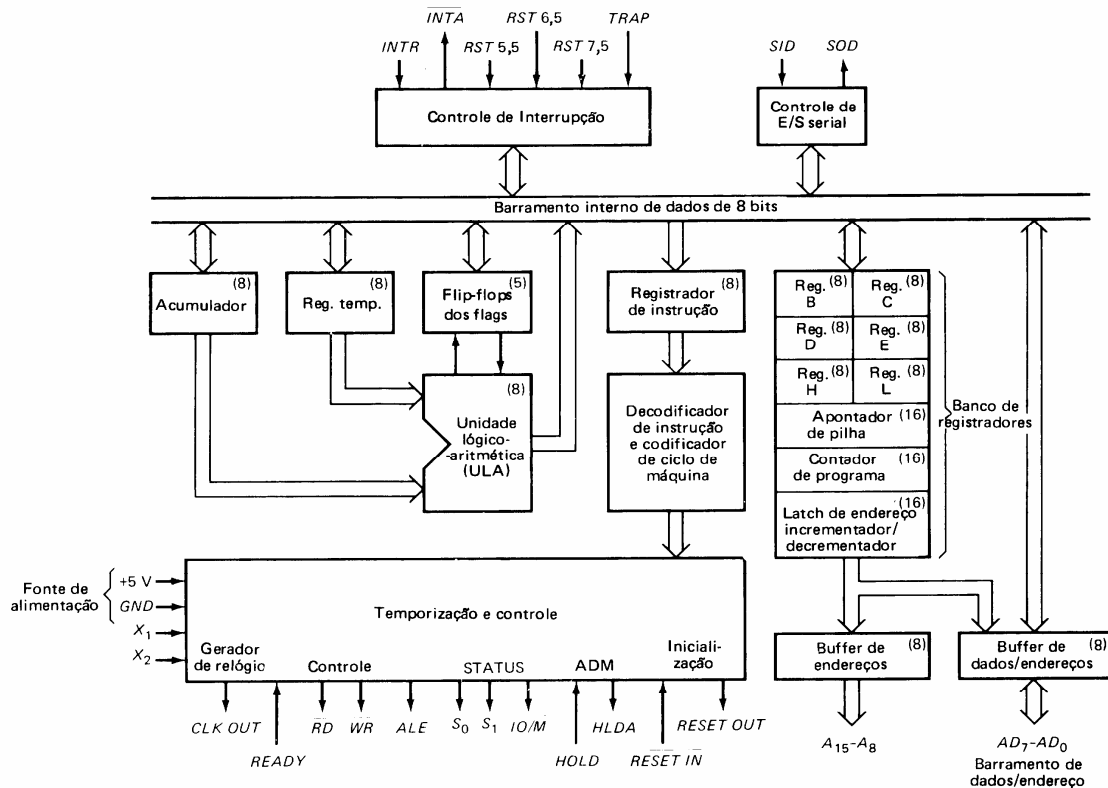


Fig. 2.1: Diagrama de blocos do microprocessador 8085

#### 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 64kbytes = 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<sub>0</sub> a A<sub>7</sub>);
- 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 Input 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). Frequentemente 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 (*Program Status Word*). 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

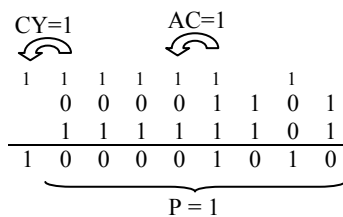
**Z = Flag de Zero** - assume o valor 1 quando o resultado de uma operação é zero

**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

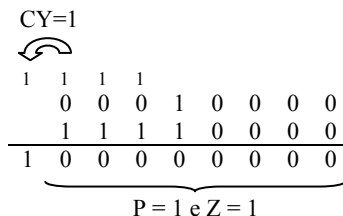
**P = Flag de Paridade** = assume valor 1 quando há um número de par de algarismos 1 no acumulador

**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 quantidade 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 quantidade 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 registrador 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 semelhança 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** - *Last In First*



**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 (Program Counter)** 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 frequência de clock do **8085A** deve ficar na faixa de **500 kHz a 3,125 MHz**. Já a frequência de clock do **8085A-2** deve ficar entre 500 kHz e 5 MHz. A frequência de clock é metade da frequência 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.

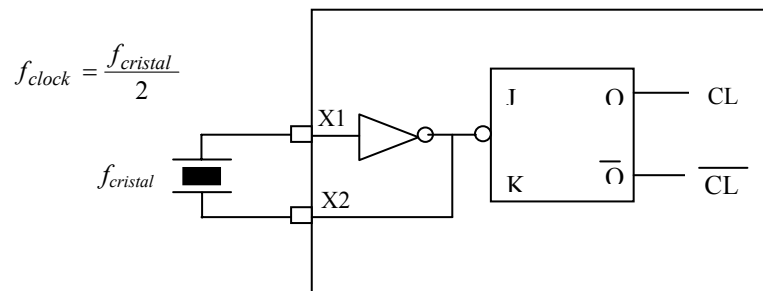


Fig. 2.2: Conexão do cristal oscilador para a geração da frequência de clock

## 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 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

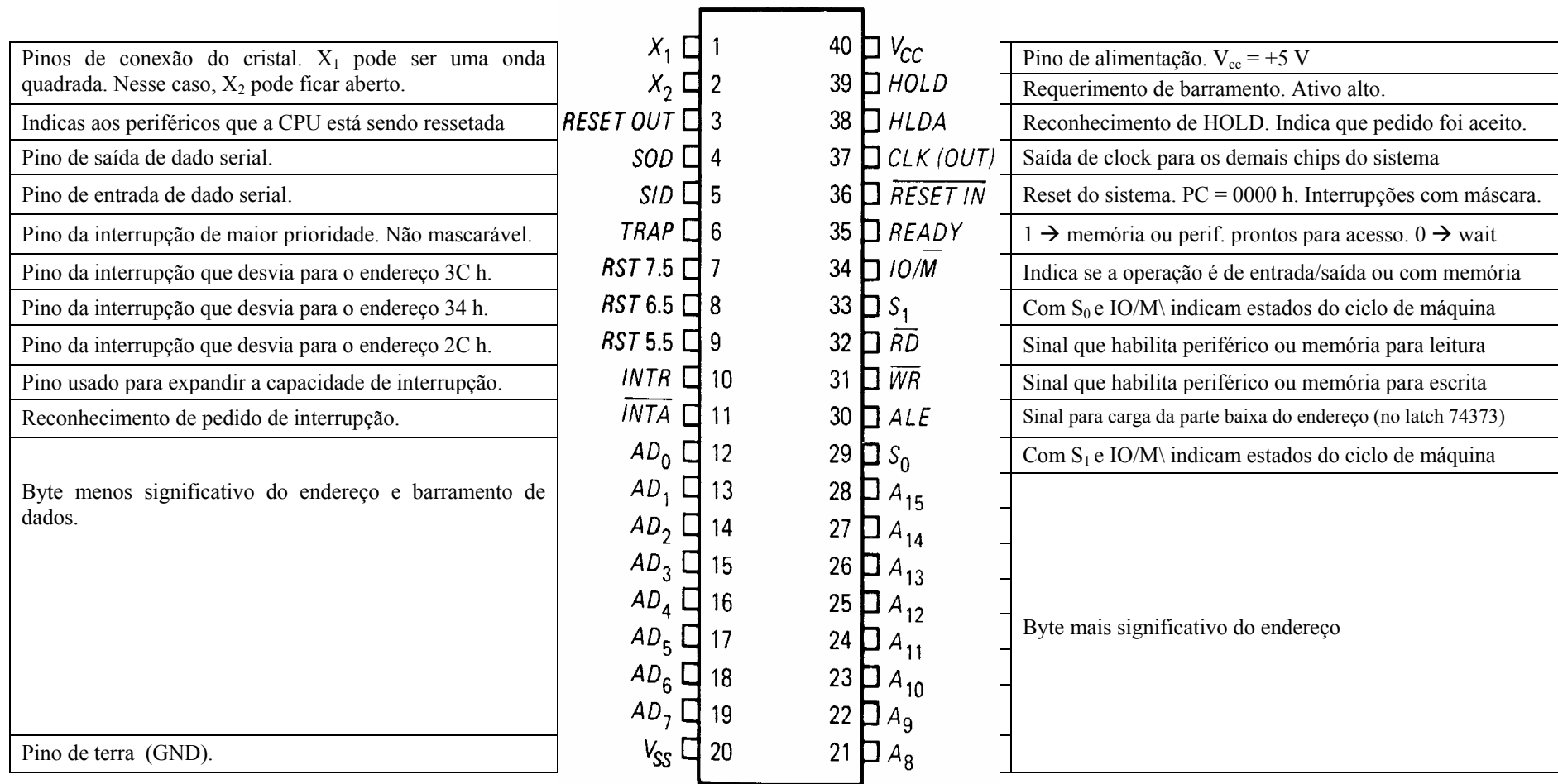


Fig. 2.3: Pinagem do microprocessador 8085

## 2.5 Sistema Básico de Temporização e Princípio de Operação

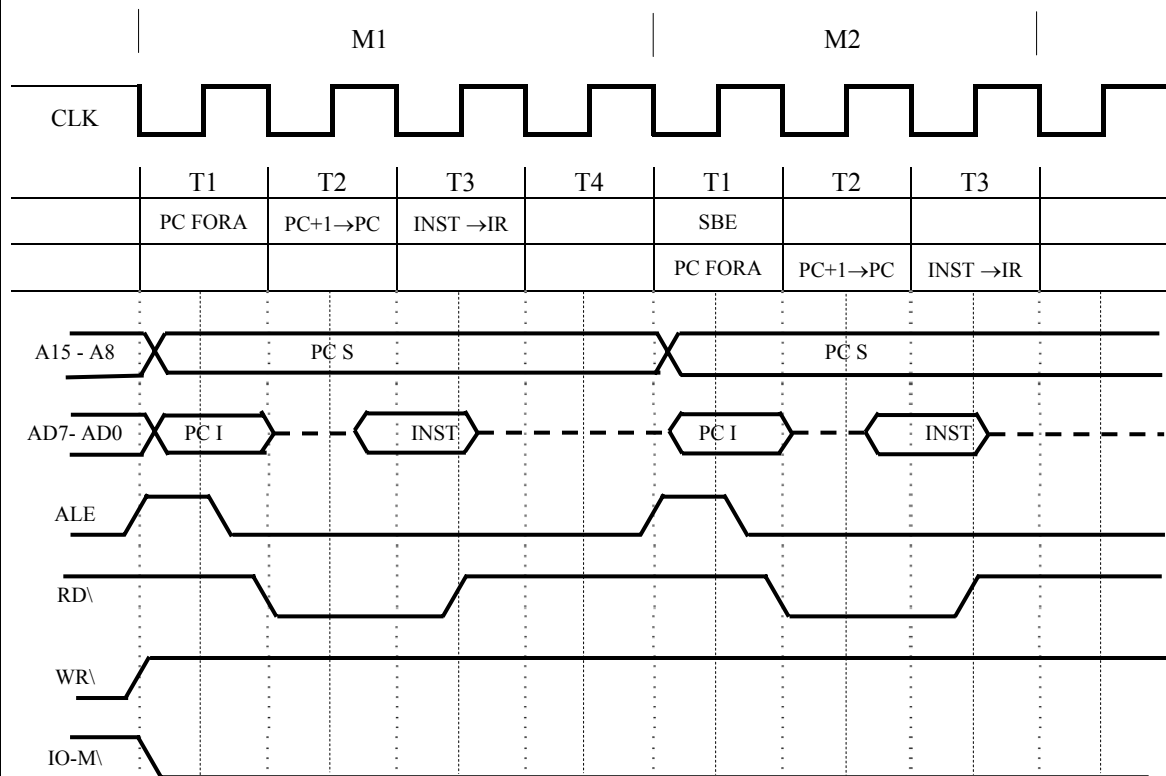


Fig. 2.4: Diagrama básico de temporização do 8085

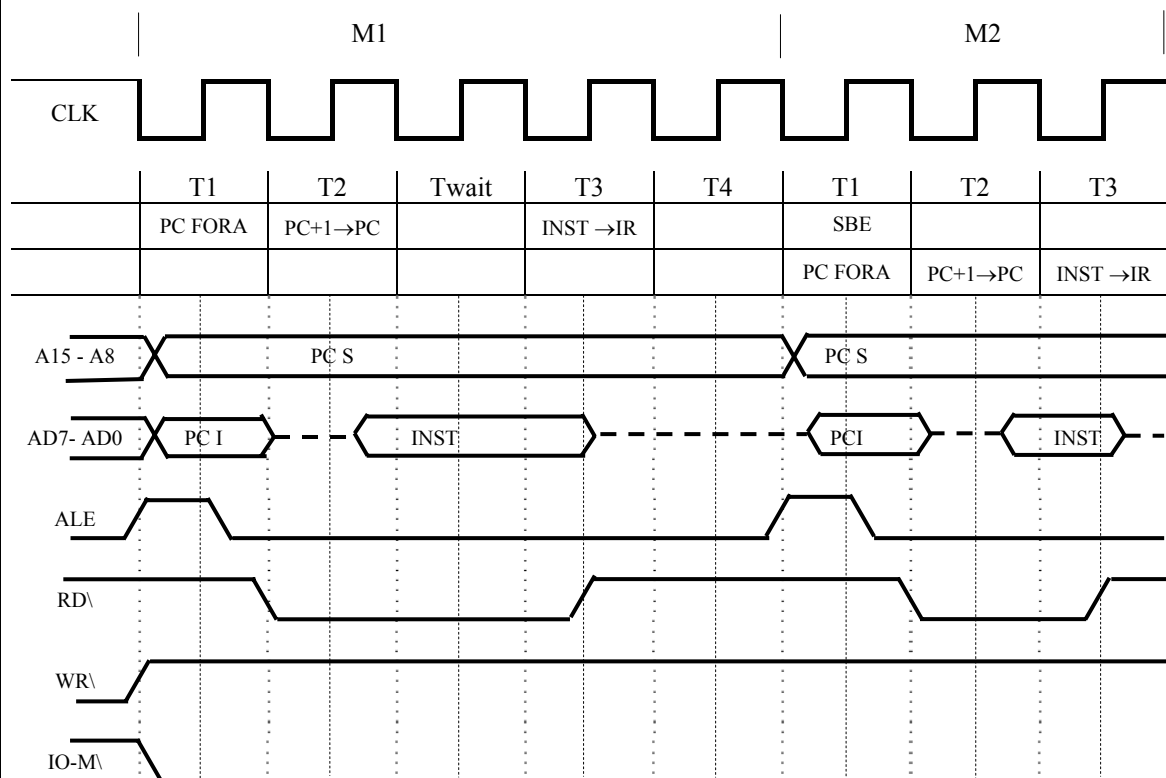


Fig. 2.5: Diagrama básico de temporização com período de espera Twait

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.

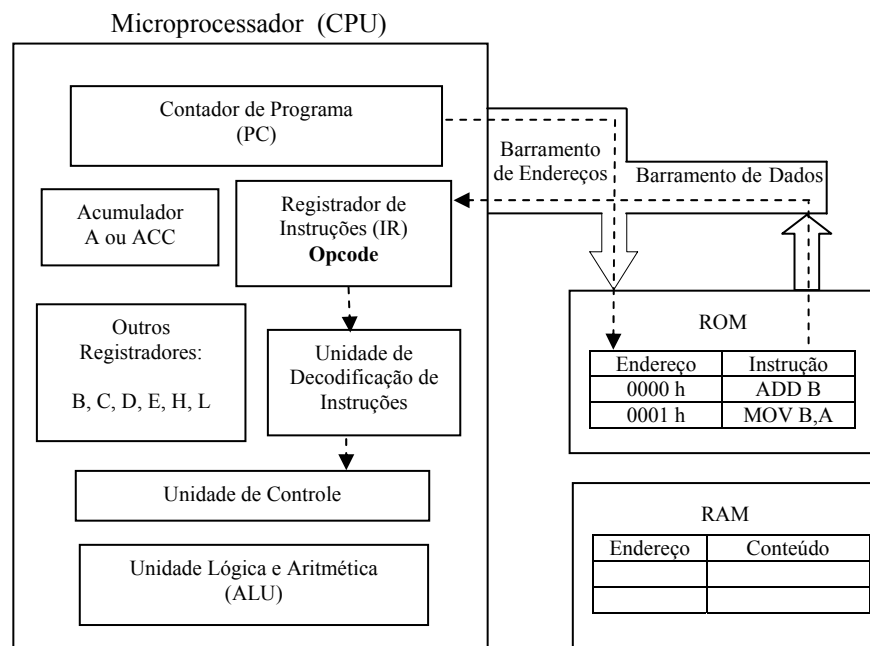
O sinal **RD** vai a zero após o estado **T1** indicando que haverá uma operação de leitura. Como o sinal **IO-M** permanece zero durante todo o tempo, figura mostrada, significa que trata-se de leitura de memória, e não de qualquer dispositivo de entrada e saída. Assim, a memória **ROM** é lida nesse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



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.

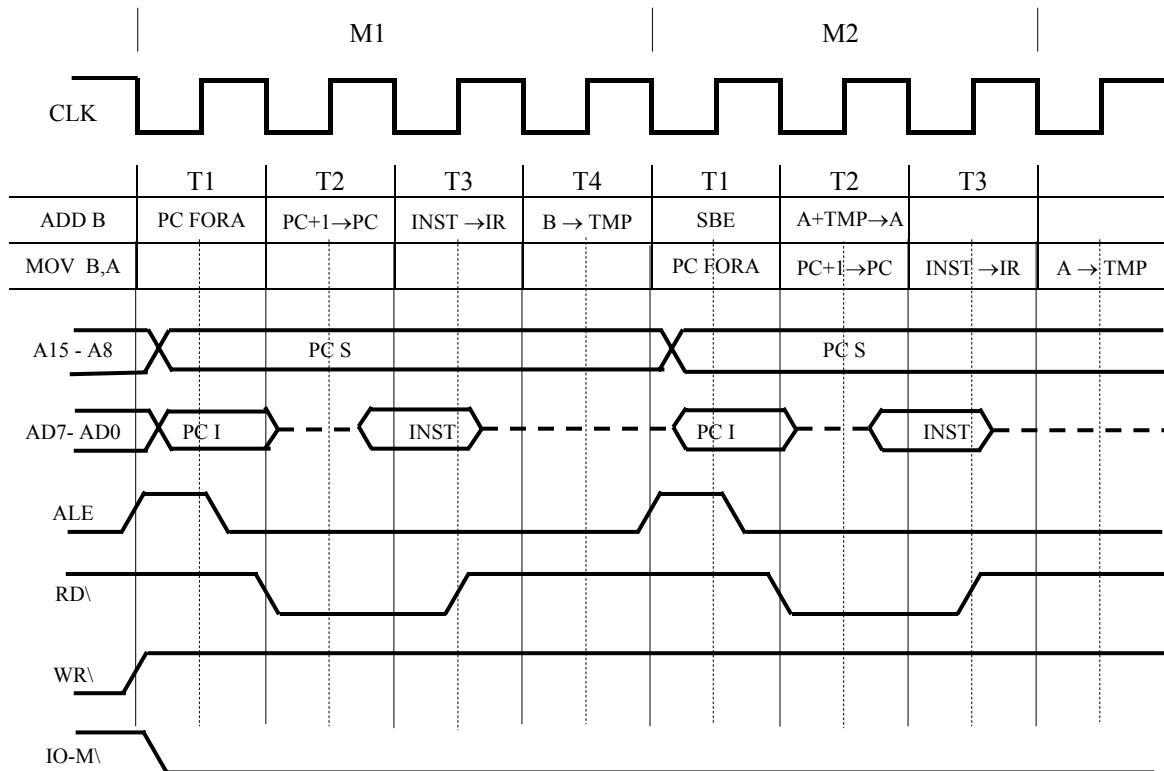


Fig. 2.7: Diagrama de temporização das instruções de adição e transferência

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 sequência 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 **ADD B** E BUSCA DE **MOV B,A**:

- 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  
 $(A) \leftarrow (A) + (B)$

Opcode (byte 1)	1	0	0	0	0	0	0	0
-----------------	---	---	---	---	---	---	---	---

= 80 h

b) Instruções de 2 bytes:

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

Exemplo: **MVI A, 32h** - Move imediatamente o conteúdo 32h para A  
 $(A) \leftarrow (\text{byte 2})$

Opcode (byte 1)	0	0	1	1	1	1	1	0
Operando (byte 2)	0	0	1	1	0	0	1	0

= 3E h  
= 32 h

c) Instruções de 3 bytes:

Opcode (byte 1)	D7	D6	D5	D4	D3	D2	D1	D0
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 demais ciclos de máquina contêm 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

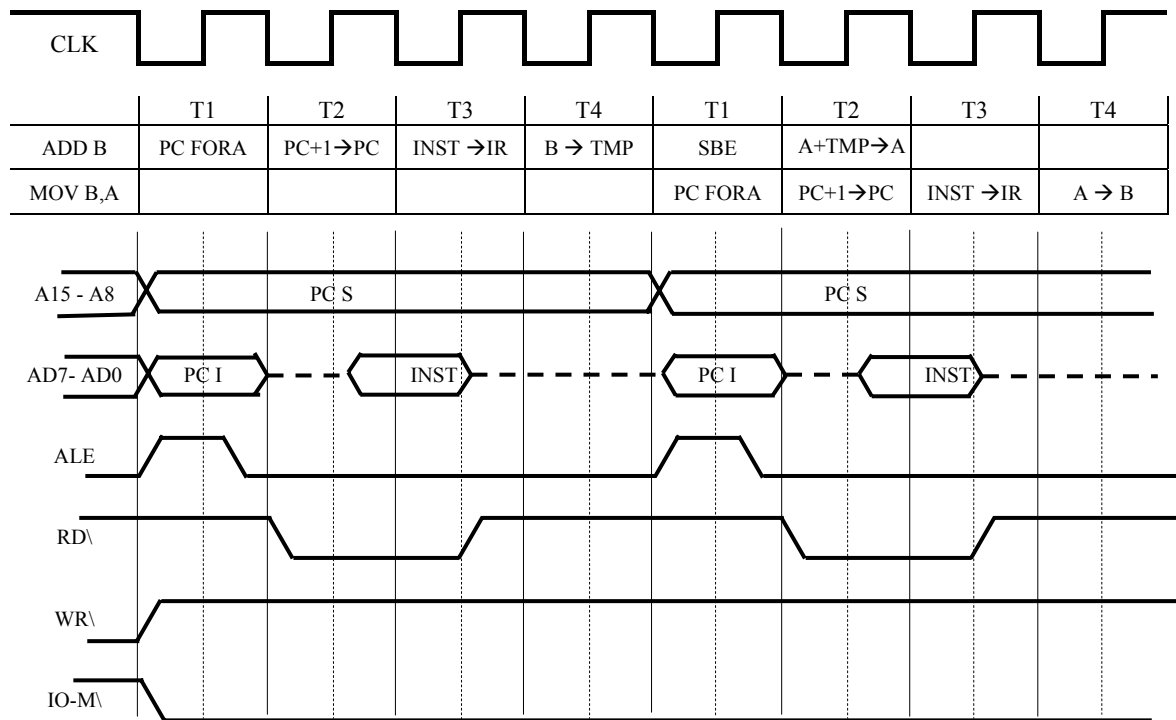
## 2.7 Exercícios Propostos

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 :  $(A) \leftarrow ((\text{byte } 3)(\text{byte } 2))$

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  
 $(r) \leftarrow (\text{byte } 2)$

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

**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

$$(A) \leftarrow ((\text{byte } 3)(\text{byte } 2))$$

**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

$$(r1) \leftarrow (r2)$$

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

$$(A) \leftarrow (A) + (r)$$

**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

$$(r) \leftarrow ((H)(L))$$

### 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.

$$((H)(L)) \leftarrow (\text{byte } 2)$$

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

$$(A) \leftarrow (A) - (r)$$

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.

$$(A) \leftarrow (A) \wedge (r)$$

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

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

$$(PC) \leftarrow (\text{byte } 3)(\text{byte } 2)$$

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

$$(A) \leftarrow (\text{data})$$

### 3.4 Instruções de Transferência de Dados

Mnemônico Genérico	Simbologia	Nº de Ciclos	Nº de Estados	Modo de Endereçamento	Flags Afetadas	Comentário
<b>MVI r, dado8</b>	$(r) \leftarrow (\text{byte } 2)$	2	7	Imediato	nenhuma	move o dado para o registrador <b>r</b> indicado
<b>MOV r1, r2</b>	$(r1) \leftarrow (r2)$	1	4	Registrador	nenhuma	move o conteúdo do registrador <b>r2</b> para o registrador <b>r1</b>
<b>MOV r, M</b>	$(r) \leftarrow ((H)(L))$	2	7	Indireto por registrador	nenhuma	move para o registrador <b>r</b> o dado presente no endereço de memória especificado pelo registrador par H-L
<b>MOV M, r</b>	$((H)(L)) \leftarrow (r)$	2	7	Indireto por registrador	nenhuma	move o conteúdo do registrador <b>r</b> para a posição de memória especificada pelo registrador par H-L.
<b>MVI M, dado8</b>	$((H)(L)) \leftarrow (\text{byte } 2)$	3	10	Indireto por registrador e imediato	nenhuma	Carrega o dado na posição de memória especificada pelo registrador par H-L.
<b>LXI rp, dado16</b>	$(rh) \leftarrow (\text{byte } 3)$ $(rl) \leftarrow (\text{byte } 2)$	3	10	Imediato	nenhuma	Carrega o dado de 16 bits no registrador par indicado em <b>rp</b> . O byte 2 da instrução é colocado no registrador de menor ordem, ou byte menos significativo, <b>rl</b> . O byte 3 da instrução é colocado no registrador de maior ordem, ou byte mais significativo, <b>rh</b>
<b>LDA addr</b>	$(A) \leftarrow ((\text{byte } 3)(\text{byte } 2))$	4	13	direto	nenhuma	Carrega acumulador com o dado armazenado na posição de memória indicada pelo endereço <b>addr</b> . O byte 2 armazena o byte inferior do endereço. O byte 3 da instrução armazena o byte superior do endereço.
<b>STA addr</b>	$((\text{byte } 3)(\text{byte } 2)) \leftarrow (A)$	4	13	direto	nenhuma	Move o conteúdo do acumulador para a posição de memória indicada pelo endereço <b>addr</b> . O byte 2 armazena o byte inferior do endereço. O byte 3 da instrução armazena o byte superior do endereço.
<b>LHLD addr</b>	$(L) \leftarrow ((\text{byte } 3)(\text{byte } 2))$ $(H) \leftarrow ((\text{byte } 3)(\text{byte } 2) + 1)$	5	16	direto	nenhuma	Carrega o conteúdo da posição de memória dada por <b>addr</b> ((byte 3)(byte 2)) no registrador L. Carrega o conteúdo da posição subsequente ((byte 3)(byte 2) + 1) no registrador H.
<b>SHLD addr</b>	$((\text{byte } 3)(\text{byte } 2)) \leftarrow (L)$ $((\text{byte } 3)(\text{byte } 2) + 1) \leftarrow (H)$	5	16	direto	nenhuma	Move o conteúdo do registrador L para a posição de memória dada por <b>addr</b> ((byte 3)(byte 2)). Move o conteúdo do registrador H para a posição subsequente de memória ((byte 3)(byte 2) + 1).
<b>LDAX rp</b>	$(A) \leftarrow ((rp))$	2	7	Indireto por registrador	nenhuma	Carrega acumulador com o conteúdo da posição de memória indicada pelo registrador par <b>rp</b> . <b>rp</b> pode ser B (do registrador para BC) ou D (do registrador par DE).
<b>STAX rp</b>	$((rp)) \leftarrow (A)$	2	7	Indireto por registrador	nenhuma	Move o conteúdo do acumulador para a posição de memória indicada pelo registrador par <b>rp</b> . <b>rp</b> pode ser B (do registrador para BC) ou D (do registrador par DE).
<b>XCHG</b>	$(H) \leftrightarrow (D)$ $(L) \leftrightarrow (E)$	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 adr	22
MOV B,B	40	MOV D,H	54	MOV L,C	69	STA adr	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). 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
<b>ADD r</b>	$(A) \leftarrow (A) + (r)$	1	4	registrador	todas	O conteúdo do registrador <b>r</b> é adicionado ao conteúdo do acumulador.
<b>ADD M</b>	$(A) \leftarrow (A) + ((H) + (L))$	2	7	registrador	todas	O conteúdo da posição de memória indicado pelo par HL é adicionado ao conteúdo do acumulador.
<b>ADI dado8</b>	$(A) \leftarrow (A) + (\text{byte } 2)$	2	7	imediato	todas	O valor dado em data é adicionado ao conteúdo do acumulador.
<b>ADC r</b>	$(A) \leftarrow (A) + (r) + (CY)$	1	4	registrador	todas	O conteúdo do registrador <b>r</b> é adicionado com carry ao conteúdo do acumulador.
<b>ADC M</b>	$(A) \leftarrow (A) + ((H)(L)) + (CY)$	2	7	imediato por registrador	todas	O conteúdo da posição indicada pelo par HL é adicionado com carry ao conteúdo do acumulador.
<b>ACI dado8</b>	$(A) \leftarrow (A) + (\text{byte } 2) + (CY)$	2	7	imediato	todas	O valor dado em data é adicionado com carry ao conteúdo do acumulador.
<b>SUB r</b>	$(A) \leftarrow (A) - (r)$	1	4	registrador	todas	O conteúdo do registrador <b>r</b> é subtraído do conteúdo do acumulador.
<b>SUB M</b>	$(A) \leftarrow (A) - ((H)(L))$	2	7	indireto por registrador	todas	O conteúdo da posição de memória indicado pelo par HL é subtraído do conteúdo do acumulador.
<b>SUI dado8</b>	$(A) \leftarrow (A) - (\text{byte } 2)$	2	7	imediato	todas	O valor dado em data é subtraído do conteúdo do acumulador.
<b>SBB r</b>	$(A) \leftarrow (A) - (r) - (CY)$	1	4	registrador	todas	O conteúdo do registrador <b>r</b> é subtraído com carry do conteúdo do acumulador.
<b>SBB M</b>	$(A) \leftarrow (A) - ((H)(L)) - (CY)$	2	7	indireto por registrador	todas	O conteúdo da posição indicada pelo par HL é subtraído com carry do conteúdo do acumulador.
<b>SBI dado8</b>	$(A) \leftarrow (A) - (\text{byte } 2) - (CY)$	2	7	imediato	todas	O valor dado em data é subtraído com carry do conteúdo do acumulador.
<b>INR r</b>	$(r) \leftarrow (r) + 1$	1	4	registrador	Z, S, P e AC	O conteúdo do registrador <b>r</b> é adicionado de 1. Todas as <b>Flags</b> são afetadas, exceto CY.
<b>INR M</b>	$((H)(L)) \leftarrow ((H)(L)) + 1$	3	10	indireto por registrador	Z, S, P e AC	O conteúdo da posição apontada pelo par HL é incrementado de 1.
<b>DCR r</b>	$(r) \leftarrow (r) - 1$	1	4	registrador	Z, S, P e AC	O conteúdo do registrador <b>r</b> é decrementado. Todas as <b>Flags</b> são afetadas, exceto CY.
<b>DCR M</b>	$((H)(L)) \leftarrow ((H)(L)) - 1$	3	10	indireto por registrador	Z, S, P e AC	O conteúdo da posição apontada pelo par HL é decrementada de 1.
<b>INX rp</b>	$(rh)(rl) \leftarrow (rh)(rl) + 1$	1	6	registrador	nenhuma	O conteúdo do registrador par <b>rp</b> é adicionado de 1. Nenhuma <b>Flag</b> é afetada.

### Instruções Aritméticas - Continuação

Mnemônico Genérico	Simbologia	Nº de Ciclos	Nº de Estados	Modo de Endereçamento	Flags Afetadas	Comentário
<b>DCX rp</b>	$(rh)(rl) \leftarrow (rh)(rl) - 1$	1	6	registrador	nenhuma	O conteúdo do registrador par <b>rp</b> é decrementado de 1.
<b>DAD rp</b>	$((H)(L)) \leftarrow ((H)(L)) + (rh)(rl)$	3	10	registrador	CY	O conteúdo do registrador par <b>rp</b> é adicionado ao conteúdo do registrador par HL. Somente a Flag de Carry (CY) é afetada.
<b>DAA</b>		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. A regra seguida é a seguinte: <ol style="list-style-type: none"> <li>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.</li> <li>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.</li> </ol>

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 instruções 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 subtração 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

$$\begin{array}{r}
 1\ 1\ 1\ 1\ 0\ 0\ 0\ 1 \\
 +\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0 \\
 \hline
 1\ 1\ 1\ 1\ 1\ 0\ 0\ 0 \\
 \text{Carry} = 0 \rightarrow CY = 0
 \end{array}$$

$$\begin{array}{r}
 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1 \\
 +\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1 \\
 \hline
 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0 \\
 \text{Carry} = 1 \rightarrow CY = 1
 \end{array}$$

##### Exemplo 2: Operação de subtração

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

$$\begin{array}{r}
 \text{subtração normal} \\
 0\ 0\ 0\ 0\ 1\ 1\ 1\ 1 \\
 -\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1 \\
 \hline
 0\ 0\ 0\ 0\ 1\ 1\ 1\ 0 \\
 \text{Carry} = 0 \rightarrow CY = 0
 \end{array}$$

$$\begin{array}{r}
 \text{subtração usando complementar de 2} \\
 0\ 0\ 0\ 0\ 1\ 1\ 1\ 1 \\
 +\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1 \\
 \hline
 1\ 0\ 0\ 0\ 0\ 1\ 1\ 0 \\
 \text{Carry} = 1 \rightarrow CY = 0
 \end{array}$$

**Exemplo 3: Operação de subtração**

$$\begin{array}{r}
 \text{subtração normal} \\
 0\ 0\ 0\ 0\ 1\ 1\ 0\ 0 \\
 -\ 0\ 0\ 0\ 1\ 0\ 0\ 1\ 0 \\
 \hline
 1\ 1\ 1\ 1\ 1\ 1\ 0\ 1\ 0 \\
 \text{Carry} = 1 \rightarrow \text{CY} = 1
 \end{array}$$

$$\begin{array}{r}
 \text{subtração usando complementar de 2} \\
 0\ 0\ 0\ 0\ 1\ 1\ 0\ 0 \\
 +\ 1\ 1\ 1\ 0\ 1\ 1\ 1\ 0 \\
 \hline
 1\ 1\ 1\ 1\ 1\ 0\ 1\ 0 \\
 \text{Carry} = 0 \rightarrow \text{CY} = 1
 \end{array}$$

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
9 2 6	3	9	E			1	1	1	0	0	1
1 3 7 8	5	6	2			1	0	1	1	0	0

### 3.6 Instruções Lógicas

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

Mnemônico Genérico	Simbologia	Nº de Ciclos	Nº de Estados	Modo de Endereçamento	Flags Afetadas	Comentário
<b>ANA r</b>	$(A) \leftarrow (A) \wedge (r)$	1	4	registrador	todas	O conteúdo do acumulador passa por uma operação lógica <b>AND</b> com o conteúdo do registrador indicado em <b>r</b> . <b>A flag CY é zerada e a flag AC é setada.</b>
<b>ANA M</b>	$(A) \leftarrow (A) \wedge ((H)(L))$	2	7	indireto por registrador	todas	O conteúdo do acumulador passa por uma operação lógica <b>AND</b> com o conteúdo da posição de memória apontada pelo registrador par HL. <b>A flag CY é zerada e a flag AC é setada.</b>
<b>ANI dado8</b>	$(A) \leftarrow (A) \wedge (\text{byte } 2)$	2	7	imediato	todas	o conteúdo do acumulador passa por uma operação lógica <b>AND</b> com o dado fornecido no byte 2 da instrução. <b>A flag CY é zerada e a flag AC é setada.</b>
<b>XRA r</b>	$(A) \leftarrow (A) \vee (r)$	1	4	registrador	todas	O conteúdo do acumulador passa por uma operação lógica <b>XOR</b> com o conteúdo do registrador <b>r</b> indicado. <b>As flags CY e AC são zeradas.</b>
<b>XRA M</b>	$(A) \leftarrow (A) \vee ((H)(L))$	2	7	indireto por registrador	todas	O conteúdo do acumulador passa por uma operação lógica <b>XOR</b> com o conteúdo do endereço de memória apontado pelo registrador par HL. As flags CY e AC são zeradas.
<b>XRI dado8</b>	$(A) \leftarrow (A) \vee (\text{byte } 2)$	2	7	imediato	todas	o conteúdo do acumulador passa por uma operação lógica <b>XOR</b> com o dado fornecido no byte 2 da instrução. As flags CY e AC são zeradas.
<b>ORA r</b>	$(A) \leftarrow (A) \vee (r)$	1	4	registrador	todas	O conteúdo do acumulador passa por uma operação lógica <b>OR</b> com o conteúdo do registrador indicado em <b>r</b> . As flags CY e AC são zeradas.
<b>ORA M</b>	$(A) \leftarrow (A) \vee ((H)(L))$	2	7	indireto por registrador	todas	O conteúdo do acumulador passa por uma operação lógica <b>OR</b> com o conteúdo da posição de memória apontada pelo registrador par HL. As flags CY e AC são zeradas.
<b>ORI dado8</b>	$(A) \leftarrow (A) \vee (\text{byte } 2)$	2	7	imediato	todas	O conteúdo do acumulador passa por uma operação lógica <b>OR</b> com o dado presente no byte 2 da instrução. As flags CY e AC são zeradas.
<b>CMP r</b>	$(A) - (r)$	1	4	registrador	todas	Compara o conteúdo do registrador <b>r</b> com o conteúdo do acumulador. Na operação o conteúdo de <b>r</b> é subtraído do conteúdo de <b>A</b> , 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, $CY = 0$ .
<b>CMP M</b>	$(A) \leftarrow (A) - ((H)(L))$	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 <b>A</b> , 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.
<b>CPI dado8</b>	$(A) \leftarrow (A) - (\text{byte } 2)$	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 <b>A</b> , sem que o resultado seja guardado em A. Se o resultado da subtração for zero, ou seja $(A) = (\text{byte } 2)$ , a flag de zero Z = 1. Caso contrário, Z = 0. Se $(A) < (\text{byte } 2)$ , a flag de carry CY = 1. Caso contrário, CY = 0.
<b>RLC</b>	$(A_{n+1}) \leftarrow (A_n)$ $(A_0) \leftarrow (A_7)$ $(CY) \leftarrow (A_7)$	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'. <b>Somente a flag de carry é afetada.</b>
<b>RRC</b>	$(A_n) \leftarrow (A_{n+1})$ $(A_7) \leftarrow (A_0)$ $(CY) \leftarrow (A_0)$	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'. <b>Somente a flag de carry CY é afetada.</b>
<b>RAL</b>	$(A_{n+1}) \leftarrow (A_n)$ $(CY) \leftarrow (A_7)$ $(A_0) \leftarrow (CY)$	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). <b>Somente a flag de carry é afetada.</b>
<b>RAR</b>	$(A_n) \leftarrow (A_{n+1})$ $(CY) \leftarrow (A_0)$ $(A_7) \leftarrow (CY)$	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. <b>Somente a flag de carry é afetada.</b>
<b>CMA</b>	$(A) \leftarrow (A)$	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. <b>Nenhuma flag é afetada.</b>
<b>CMC</b>	$(CY) \leftarrow (CY)$	1	4		CY	(Complement Carry) = (Complementa Carry) = O conteúdo da flag de Carry complementado. <b>Nenhuma outra flag é afetada.</b>
<b>STC</b>	$(CY) \leftarrow 1$	1	4		CY	(Set Carry) = (Seta o bit de Carry) = A flag de Carry (CY) é feita igual a 1. <b>Nenhuma outra flag é afetada.</b>

## 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 <b>AND</b> C, ou, 0F <b>AND</b> 52h $\Rightarrow$ <b>A = 02h</b>
ANI 44h	E6 44	Faz A <b>AND</b> 44h, ou, 02h <b>AND</b> 44h. $\Rightarrow$ <b>A = 00h</b>
XRI 23h	EE 23	Faz A <b>XOR</b> 23h, ou, 00h <b>XOR</b> 23h $\Rightarrow$ <b>A = 23h</b>
CPI 33h	FE 33	Faz A - 33h, sem alterar A. Ou, 23 - 33h $\Rightarrow$ <b>Z = 0 e CY = 1</b>
RLC	07	Rotaciona A à esquerda. Resultado: <b>A = 46h e CY = 0</b>
CMP B	B8	Faz A - B, sem alterar A. Ou, 46h - 46h $\Rightarrow$ <b>Z = 1 e CY = 0</b>
CMC	3F	Complementa flag de Carry. <b>CY = 1</b>
RAR	1F	Rotaciona A à direita com Carry $\Rightarrow$ <b>A = A3h e CY = 0</b>

### 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
<b>NZ</b>	Not Zero ( $Z = 0$ )	0	0	0
<b>Z</b>	Zero ( $Z = 1$ )	0	0	1
<b>NC</b>	No Carry ( $CY = 0$ )	0	1	0
<b>C</b>	Carry ( $CY = 1$ )	0	1	1
<b>PO</b>	Parity odd = Paridade Ímpar ( $P = 0$ )	1	0	0
<b>PE</b>	Parity Even = Paridade Par ( $P = 1$ )	1	0	1
<b>P</b>	Plus $\equiv$ Positivo ( $S = 0$ )	1	1	0
<b>M</b>	Minus $\equiv$ Negativo ( $S = 1$ )	1	1	1

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

## Instruções de Desvio:

Mnemônico Genérico	Simbologia	Nº de Ciclos	Nº de Estados	Modo de Endereçamento	Flags Afetadas	Comentário
<b>JMP addr</b>	(PC) $\leftarrow$ (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.
<b>J<sub>condição</sub> addr</b>	Se (CCC), então, (PC) $\leftarrow$ (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.
<b>JNZ addr</b>	Se (CCC), então, (PC) $\leftarrow$ (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, <b>desvia se Z = 0</b> .
<b>JZ addr</b>	Se (CCC), então, (PC) $\leftarrow$ (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, <b>desvia se Z = 1</b> .
<b>JNC addr</b>	Se (CCC), então, (PC) $\leftarrow$ (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 ( <b>CY = 0</b> ).
<b>JC addr</b>	Se (CCC), então, (PC) $\leftarrow$ (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 ( <b>CY = 1</b> ).
<b>JPO addr</b>	Se (CCC), então, (PC) $\leftarrow$ (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 ( <b>P = 0</b> ).
<b>JPE addr</b>	Se (CCC), então, (PC) $\leftarrow$ (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 ( <b>P = 1</b> ).
<b>JP addr</b>	Se (CCC), então, (PC) $\leftarrow$ (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 ( <b>S = 0</b> ).
<b>JM addr</b>	Se (CCC), então, (PC) $\leftarrow$ (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 ( <b>S = 1</b> ).
<b>CALL addr</b>	((SP) - 1) $\leftarrow$ (PCH) ((SP) - 2) $\leftarrow$ (PCL) (SP) $\leftarrow$ (SP) - 2 (PC) $\leftarrow$ (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 <b>addr</b> , 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.
<b>C<sub>condition</sub> addr</b>	((SP) - 1) $\leftarrow$ (PCH) ((SP) - 2) $\leftarrow$ (PCL) (SP) $\leftarrow$ (SP) - 2 (PC) $\leftarrow$ (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 <b>addr</b> 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: <b>CNZ, CZ, CNC, CC, CPO, CPE, CP, CM</b>
<b>RET</b>	$(PCL) \leftarrow (SP)$ $(PCH) \leftarrow (SP + 1)$ $(SP) \leftarrow (SP) + 2$	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.
<b>R<sub>condição</sub></b>	$(PCL) \leftarrow (SP)$ $(PCH) \leftarrow (SP + 1)$ $(SP) \leftarrow (SP) + 2$	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.
<b>RST n</b>	$((SP) - 1) \leftarrow (PCH)$ $((SP) - 2) \leftarrow (PCL)$ $(SP) \leftarrow (SP) - 2$ $(PC) \leftarrow 8 * (NNN)$	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.
<b>PCHL</b>	$(PCH) \leftarrow (H)$ $(PCL) \leftarrow (L)$	1	6	registrador	nenhuma	(Jump H and L indirect - 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
<b>PUSH rp</b>	$((SP) - 1 \leftarrow (rh))$ $((SP) - 2 \leftarrow (rl))$ $(SP) \leftarrow (SP) - 2$	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 <b>SP - 1</b> . O conteúdo do registrador que contém o byte inferior da instrução é guardado na posição de memória indicada por <b>SP - 2</b> . O registrador par <b>rp</b> pode ser B (de BC), D (de DE) ou H (de HL). O registrador <b>SP</b> é decrementado de 2.
<b>PUSH PSW</b>	$((SP) - 1 \leftarrow (A))$ $((SP) - 2 \leftarrow (F))$ $(SP) \leftarrow (SP) - 2$	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 <b>SP - 1</b> . O conteúdo do registrador F é guardado na posição de memória indicada por <b>SP - 2</b> .
<b>POP rp</b>	$(rl) \leftarrow ((SP))$ $(rh) \leftarrow ((SP) + 1)$ $(SP) \leftarrow (SP) + 2$	3	10	indireto por registrador	nenhuma	O conteúdo da posição de memória indicada por <b>SP</b> é movido para o registrador que representa o byte inferior (C, E ou L). O conteúdo da posição de memória indicada por <b>SP + 1</b> é movido para o registrador que representa o byte superior da instrução (B, D ou H). O conteúdo do registrador <b>SP</b> é incrementado de 2.
<b>POP PSW</b>	$(F) \leftarrow ((SP))$ $(A) \leftarrow ((SP) + 1)$ $(SP) \leftarrow (SP) + 2$	3	10	indireto por registrador	todas	O conteúdo da posição de memória indicada por <b>SP</b> é movido para o registrador que guarda o estado das Flags (registrador F). O conteúdo da posição de memória indicada por <b>SP + 1</b> é movido para o acumulador. O conteúdo do registrador <b>SP</b> é incrementado de 2.
<b>XTHL</b>	$(L) \leftarrow ((SP))$ $(H) \leftarrow ((SP) + 1)$	5	16	indireto por registrador	nenhuma	(Exchange Stack Top with H and L) = O conteúdo da posição de memória indicada por <b>SP</b> é trocado com o conteúdo do registrador L e o conteúdo da posição de memória indicada por <b>SP + 1</b> é trocado pelo conteúdo do registrador H.
<b>SPHL</b>	$(SP) \leftarrow (H) (L)$	1	6	registrador	nenhuma	(Move HL to SP) = O conteúdo do registrador par HL é movido para o registrador de 16 bits <b>SP</b> .
<b>IN porta</b>	$(A) \leftarrow (\text{dado 8 bits})$	3	10	direto	nenhuma	(Input = Entrada) = O dado colocado na porta indicada é transferido para o acumulador através do barramento de dados (8 bits).
<b>OUT porta</b>	$(\text{dado 8 bits}) \leftarrow (A)$	3	10	direto	nenhuma	(Output = Saída) = O dado do acumulador é transferido para a porta indicada através do barramento de dados (8 bits).

<b>EI</b>		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 instrução EI.
<b>DI</b>		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 instrução DI.
<b>HLT</b>		1+	5		nenhuma	A instrução HLT faz o processador parar o processamento. Os registradores e flags permanecem inalterados.
<b>NOP</b>		1	4		nenhuma	(No Operation) = Nenhuma operação é realizada. Os registradores e flags ficam inalterados.
<b>RIM</b>		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.
<b>SIM</b>		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:

- 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;
- Os registradores duplos que podem ser guardados na pilha são **PSW** (= A + Flags), **B** (= B + C), **D** (= D + E) e **H** (= H + L);
- Para guardar o conteúdo de um desses registradores duplos usa-se a instrução **PUSH rp**;
- Para recuperar o conteúdo que foi guardado na pilha usa-se a instrução **POP rp**;
- 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;
- 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;
- 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		PUSH B		POP B		POP PSW	
Endereço da RAM	Conteúdo (HEX)	Endereço da RAM	Conteúdo (HEX)	Endereço da RAM	Conteúdo (HEX)	Endereço da RAM	Conteúdo (HEX)
2089		2089		2089		2089	
208A		208A		208A		208A	
208B		208B		208B		208B	
208C		208C	67	208C	67	208C	67
208D		208D	45	208D	45	208D	45
208E	23	208E	23	208E	23	208E	23
208F	01	208F	01	208F	01	208F	01
2090		2090		2090		2090	
SP após a instrução: <b>208E h</b>		SP após a instrução: <b>208C h</b>		SP após a instrução: <b>208E h</b>		SP após a instrução: <b>2090 h</b>	

Observações:

- 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;
- 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 $\equiv$ 70 d
volta:	DCR C	Decrementa C. C = C - 1
	JNZ volta	Volta, se flag de zero for zero (Z=0) $\Rightarrow$ C $\neq$ 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 $\cong$ 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 $\cong$ 10d
repete1	MVI C,47h	Inicia contador C com valor 47h $\cong$ 71h
repete2:	DCR C	Decrementa C. C = C - 1
	JNZ repete2	Repete laço até registrador C = 0 $\Rightarrow$ Z = 1.
	DCR B	Decrementa registrador B
	JNZ repetel	Reinicia laço interno até zerar registrador B $\Rightarrow$ 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 repetel (verdadeiro)	9	10	1	90
JNZ repetel (falso)	1	7	1	7
RET	1	10	1	10
Tempo Total				10134 µs $\cong$ 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 sequência 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 frequência 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<sup>TM</sup> 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.

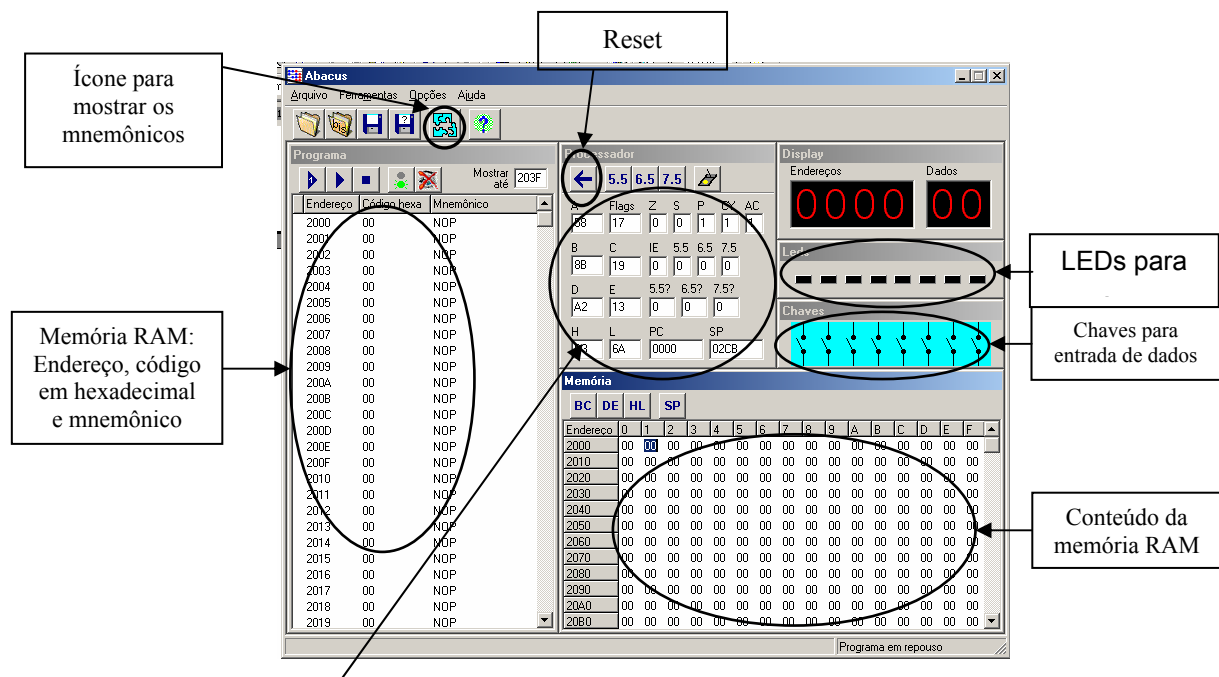


## 4. SIMULADOR DIGITAL ABACUS

### 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:

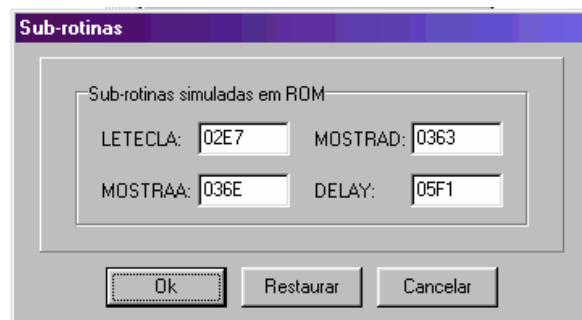


Registradores: A, Flags (Z, S, P, CY, AC), B, C, D, E, H, L, PC e SP  
Estado das máscaras de interrupção: IE, 5.5, 6.5, 7.5  
Indicadores se há pedido pendente de interrupção: 5.5?, 6.5?, 7.5?

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".



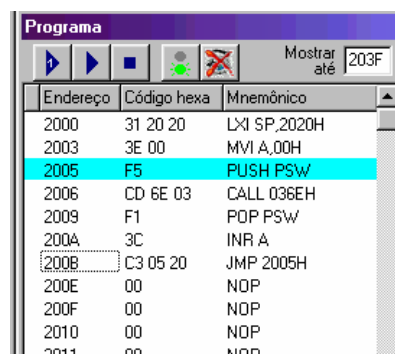
Subrotina	Endereço (h)	Função	Entrada	Saída	Registradores Afetados
<b>LETECLA</b>	<b>02E7</b>	Mostra uma entrada de dados no simulador, através da qual um único caractere (0 a F) pode ser digitado	nenhuma	A - valor da tecla lida	A, H e L
<b>MOSTRAA</b>	<b>036E</b>	Mostra no display de dados o conteúdo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

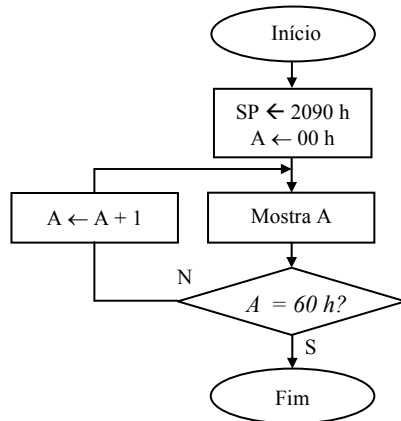
A visualização desse programa na memória RAM do ABACUS é mostrada na figura a seguir.



## 4.2 Exemplos de Programas em Assembly para o ABACUS

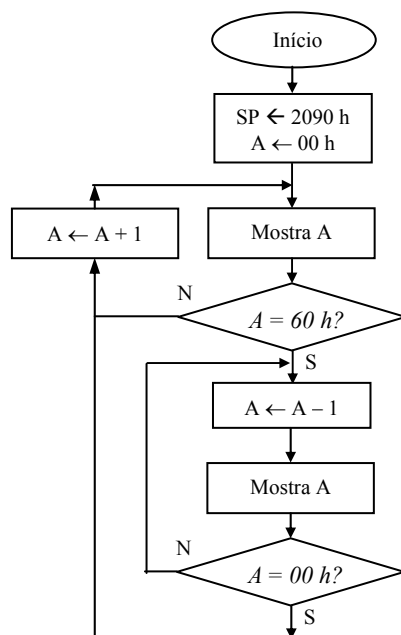
A presente seção mostra dois exemplos de programação voltada para simulação no ABACUS.

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 °C (um byte) e converta-a em °F (graus Fahrenheit) mostrando o resultado no display. (o programa só deve aceitar temperaturas de entrada de até 255 graus fahrenheit, ou seja, um byte na saída). Trabalhar em hexadecimal.  
$$T(^{\circ}\text{F}) = (9/5) * T(^{\circ}\text{C}) + 32$$
 (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 (1º 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 "x" pelo teclado e mostra no display o resultado da operação  $3x + 5$ .
17. Faça no ABACUS um programa que lê um número "x" pelo teclado e mostra no display o resultado da operação  $5x - 2$ .
18. Faça no ABACUS um programa que lê um número "x" pelo teclado e mostra no display o resultado da operação  $3(x + 5)$ .

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

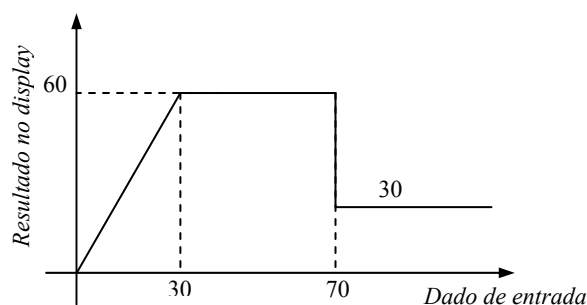
LABEL	MNEMÔNICO
LEITURA:	PUSH H
	CALL LETECLA
	RLC
	RLC
	RLC
	RLC
	MOV B,A
	CALL LETECLA
	ORA B
	POP H
	RET

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

- Explique o funcionamento da subrotina "LEITURA";
  - Faça a continuação do programa acima de modo que o maior número da tabela montada seja mostrado no display de dados;
  - 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.

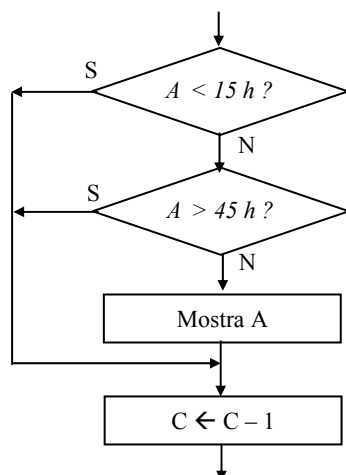
OPERAÇÃO	END.	MNEMÔNICO	A	C	CY	Z
$(A) \leftarrow 70 \text{ h}$	2000	MVI A, 70 h				
$(C) \leftarrow 0F \text{ h}$		MVI C, 0F h				
$(H,L) \leftarrow 2020 \text{ h}$		LXI H, 2030 h				
$((H)(L)) \leftarrow F0 \text{ h}$		MVI M, F0 h				
$(A) \leftarrow (A) \wedge (C)$		ANA C				
$(A) \leftarrow ((H)(L))$		CMP M				
$(A) \leftarrow (A) + (C) + CY$		ADC C				
$(C) \leftarrow (C) \vee ((H)(L))$		MOV A,C				
		ORA M				
		MOV C,A				
$(A) \leftarrow FF \text{ h}$		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.



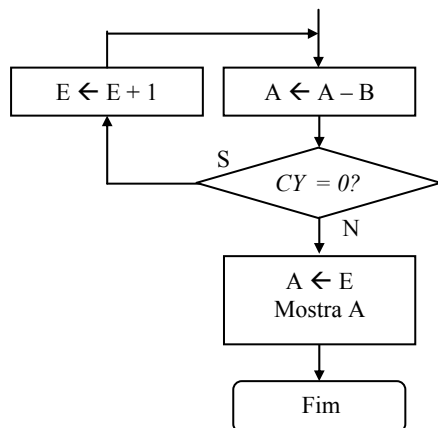
23. Codifique, usando mnemônicos do microprocessador 8085 e recursos do ABACUS, os trechos de programas representados nos fluxogramas dados a seguir:

(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.

## 5. KIT DIDÁTICO E ROTEIROS DE LABORATÓRIO

### 5.1 Sub-rotinas do Programa Monitor do Kit Didático

O laboratório de microprocessadores conta com um Kit Didático para realização de experimentos usando o microprocessador 8085. Esse Kit já possui várias sub-rotinas no programa residente (chamado de programa monitor). Elas são dadas a seguir, juntamente com uma breve descrição e com os endereços onde elas estão armazenadas.

Subrotina	Endereço (h)	Função	Entrada	Saída	Registradores Afetados
<b>APDIS</b>	<b>070B</b>	apagar display (e inicializar WRITE DISPLAY RAM na posição 0) (LEFT entry)	nenhuma	nenhuma	nenhum
<b>CI</b>	<b>05C1</b>	leitura da tecla apertada	nenhuma	A - código da tecla lida	A e flags
<b>CO</b>	<b>05CB</b>	mostra no display o caracter contido no registrador C	C - código do caracter a ser mostrado no display	A - código do caracter mostrado no display	A e flags
<b>DADD</b>	<b>05B8</b>	mostra 2 bytes no display (LEFT entry)	D - 1º byte a ser mostrado no display E - 2º byte a ser mostrado no display	nenhuma	A, BC, DE e flags
<b>DBYTE</b>	<b>05A1</b>	mostra 1 byte no display (LEFT entry)	A - byte a ser mostrado no display	nenhuma	A, BC e flags
<b>PRINT</b>	<b>05D6</b>	imprime mensagem no display (mensagem localizada no endereço dado por M e que termina com um byte FF)	M - endereço da memória a partir de onde começa a mensagem	nenhuma	A, C, HL e flags
<b>RADD</b>	<b>0580</b>	lê 2 bytes do teclado e mostra no display (p. ex.: endereços)	nenhuma	M - 2 bytes lidos	A, BC, HL e flags
<b>RBYTE</b>	<b>0558</b>	lê 1 byte do teclado e mostra no display	nenhuma	A = B = byte lido e mostrado no display	A, BC e flags
<b>RDATA</b>	<b>0593</b>	mostra 2 bytes no display	nenhuma	M = 2 bytes mostrados no display	DE
<b>TECLA</b>	<b>06BA</b>	guarda em A a posição (na TABTEC) da tecla lida	nenhuma	A = C = posição na TABTEC do código da tecla lida	A, C e flags
<b>DELAY</b>	<b>077A</b>	temporizador (tempo = 0,6 ms)	nenhuma	nenhuma	A e flags
<b>DEL 1</b>	<b>0770</b>	temporizador (tempo = 50 ms)	nenhuma	nenhuma	A, B e flags
<b>DEL 2</b>	<b>0766</b>	temporizador (tempo = 850 ms)	nenhuma	nenhuma	A, BC e flags

#### Exemplo de programa usando sub-rotinas da tabela anterior:

O programa a seguir mostra uma contagem hexadecimal crescente de 00 h até FF h no display do Kit didático. O programa deve ser digitado a partir da posição de memória 4000 h.

Label	Endereço	Mnemônico	Código	Comentário
	4000	MVI A,00h	3E 00	Carrega registrador A com valor 00 h
volta:	4002	CALL APDIS	CD 0B 07	Apaga display. É importante para que os números da contagem sejam mostrados na mesma posição
	4005	PUSH PSW	F5	Guarda valor de A e flags na pilha
	4006	CALL DBYTE	CD A1 05	Mostra conteúdo de A no display. Afeta "A"
	4009	POP PSW	F1	Recupera valor A guardado na pilha
	400A	INR A	3C	Incrementa conteúdo de A
	400B	JMP volta	C3 02 40	Volta para a posição "volta", ou seja, 4002 h

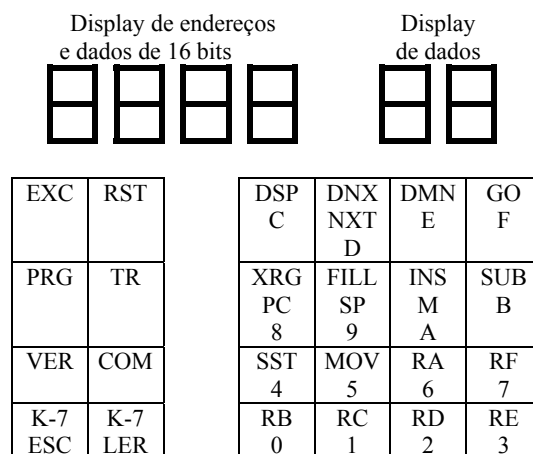
## 5.2 Principais comandos e orientações para uso do Kit Didático

O programa a ser executado pelo Kit deve ser digitado através do teclado e deve ser usado o código hexadecimal de cada instrução. No exemplo anterior aparece uma coluna com os códigos das instruções e com os dados em hexadecimal. Por exemplo, o código “3Eh” corresponde à instrução “MVI A”; logo a seguir aparece 00 h, que é o dado a ser transferido para o acumulador.

Na instrução seguinte “CALL APDIS”, verifica-se que “CD” é o código da instrução de chamada de subrotina “CALL”, “A1” o byte menos significativo do endereço da subrotina chamada e “05” é o byte mais significativo do endereço.

Obs: Em toda instrução que contém endereço ou dado de 16 bits o byte menos significativo é digitado primeiro.

A figura a seguir ilustra o teclado e o display do Kit didático. Cada uma das teclas do teclado da direita apresenta 2 ou 3 funções. Uma delas é um dígito hexadecimal, que vai de 0 a F. A outra é uma das funções explicadas a seguir.



RB, RC, RD, RE, RA e RF → Estas teclas permitem visualizar no display de dados o conteúdo dos registradores B, C, D, E, A e F. Elas devem ser precedidas da tecla XRG. Ou seja, quando se deseja visualizar, por exemplo, o conteúdo do acumulador (registrador A), usa-se as teclas <XRG> <RA>

PC → Permite visualizar o conteúdo do contador de programa, ou seja, a posição atual do programa em execução. Essa posição é visualizada no display de endereços. Usa-se as teclas <XRG> <PC>

SP → Permite visualizar o conteúdo do apontador de pilha, ou seja, o endereço atual de armazenagem ou leitura da pilha. Usa-se as teclas <XRG> <SP>.

M → Permite visualizar o conteúdo de memória cujo endereço é dado pelo par HL. Ao digitar-se <XRG> <M>, o conteúdo dos registradores H e L é mostrado no display de endereços e o conteúdo da posição de memória correspondente é mostrado no display de dados.



SUB → Substitui-se o conteúdo de uma determinada posição de memória por outro byte. Usa-se digitando essa tecla, seguida do endereço desejado. O conteúdo atual aparece no display de dados quando, então, pode-se substituído. Em seguida digita-se <EXC>. Resumindo:

<SUB> <endereço>  
<novo conteúdo> <EXC>

DSP → Permite mostrar (Display) o conteúdo de uma determinada posição de memória. Uso:

<DSP> <endereço desejado>

DNX → Permite mostrar no display o conteúdo do próximo endereço de memória, após o uso da tecla DSP. Usando repetidas vezes essa tecla, pode-se percorrer todo o programa digitado.

DMN → Permite mostrar no display de dados o conteúdo de memória do endereço anterior ao atual. É usado da mesma forma que a tecla DNX.

GO → É usada para executar o programa. Uso: <GO> <endereço de início do programa> <EXC>

SST → Permite executar o programa passo-a-passo. Uso: <SST> <endereço de início do programa> <NXT>

Cada próximo passo é executado após a tecla NXT.

RST → É usada para “ressetar” o programa, ou seja, zerar o conteúdo do contador de programa e de outros registradores.

INS → É usada para a inserção de códigos e dados no Kit Didático. Uso:

<INS> <endereço de início do programa>  
<código 1>  
<código / dado 2>  
<código / dado 3>  
.  
.  
<último código / dado > <EXC>

Como exemplo, mostra-se como o programa usado na seção anterior é digitado no kit didático.

<INS> <4000>, <3E>, <00>, <CD>, <0B>, <07>, <F5>, <CD>, <A1>, <05>, <F1>, <3C>, <C3>, <02>, <40>  
<EXC>

Convém, neste ponto, usar a tecla <RST> para zerar o contador de programa.

Obs.: Somente o endereço inicial é digitado. A cada novo dado digitado o endereço é automaticamente incrementado.

Para executar o programa faz-se: <GO> <4000> <EXC>

Obs.: Provavelmente a contagem será tão rápida que não será vista no display porque nenhuma subrotina de atraso de tempo foi usada.

Sugere-se, como exercício, adicionar uma subrotina de atraso de tempo após a instrução CALL DBYTE. Usar, por exemplo, DEL 1, cujo endereço é 0770 h. Assim, acrescenta-se essa subrotina CALL DEL 1 da seguinte forma:

<INS> <4009>, <CD>, <70>, <07>, <F1>, <3C>, <C3>, <02>, <40> <EXC>

Obs.: No kit didático não há como deslocar os comandos que já tinham sido digitados depois da instrução CALL DBYTE. Assim, todos eles foram digitados novamente após a instrução CALL DEL 1. Observar sempre que o byte menos significativo de um endereço é digitado primeiro. O endereço 4009 h foi usado porque ele é o primeiro endereço depois da instrução CALL DBYTE.

É importante ainda frisar como é feita a distribuição de memória no Kit Didático. A memória **ROM** consiste em uma pastilha de memória **2716**, de 2 kbytes, o que significa 2048 posições de 8 bits cada uma. Sendo  $2048 = 800\text{ h}$ , os endereços vão de 000 h até 7FF h. A memória RAM consiste em 6 pastilhas de memória 2114, de 1 kbyte. Porém, cada posição de memória da pastilha 2114 contém apenas 4 bits, o que torna necessário o uso de duas a duas para formar 1 byte (8 bits). Em cada conjunto de 2 pastilhas em paralelo o de acesso vai de 000 h até 3FF h.

A figura a seguir ilustra a distribuição de memória no Kit Didático. Procura-se mostrar a memória do modo como ela está colocada no kit.

<b>Memória ROM</b>		<b>Memória RAM</b>	
Embora essa memória possa ir de 0000 h até 3FFF h, somente uma pastilha de memória é usada e o endereço vai de <b>0000 h até 07FF h</b>		Embora essa memória possa ir de 4000 h até 5FFF h, os três conjuntos de memória permitem acessar dados de <b>4000 h até 4BFF h</b>	
Espaço reservado	2716	2114	2114
0800 h – 0FFF h	0000 h – 07FF h	4000 h – 43FF h	4000 h – 43FF h
Espaço reservado	Espaço reservado	2114	2114
1800 h – 1FFF h	1000 h – 17FF h	4400 h – 47FF h	4400 h – 47FF h
Espaço reservado	Espaço reservado	2114	2114
2800 h – 2FFF h	2000 h – 27FF h	4800 h – 4BFF h	4800 h – 4BFF h
Espaço reservado	Espaço reservado	Espaço reservado	Espaço reservado
3800 h – 3FFF h	3000 h – 37FF h	4C00 h – 4FFF h	4C00 h – 4FFF h
		Espaço vazio	Espaço reservado
		5000 h – 53FF h	5000 h – 53FF h
		Espaço reservado	Espaço reservado
		5400 h – 57FF h	5400 h – 57FF h
		Espaço reservado	Espaço reservado
		5800 h – 5BFF h	5800 h – 5BFF h
		Espaço reservado	Espaço reservado
		5C00 h – 5FFF h	5C00 h – 5FFF h

Observar que o último endereço de memória ROM é 07FF h e que o último endereço de memória RAM é 4BFF h.

### 5.3 Roteiros de Experimentos

A presente seção tem como objetivo propor experimentos de laboratório que possam ser implementados com o simulador ABACUS e com o kit didático. Esses experimentos serão executados na medida em que a parte teórica for ministrada.

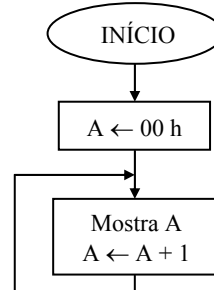
## EXPERIMENTO 1

Objetivo: Familiarização com o simulador ABACUS e com o kit didático

Material: computador com o simulador ABACUS e kit didático

1. Inclua no ABACUS e execute o programa da seção anterior, repetido a seguir.

Endereço	Mnemônico
2000	LXI SP, 2020
2003	MVI A, 00h
2005	PUSH PSW
2006	CALL MOSTRAA
2009	POP PSW
200A	INR A
200b	JMP 2005



2. Inclua, após a instrução CALL MOSTRAA, uma subrotina de atraso de tempo com D = 05 h, execute o programa novamente e observe o aumento no tempo entre cada valor mostrado no display. O trecho a ser incluído é mostrado a seguir:

Endereço	Mnemônico
2009	MVI D, 05
200B	CALL DELAY
200E	POP PSW
200F	INR A
2010	JMP 2005

3. Inclua no Kit didático e execute o programa da seção anterior, repetido a seguir.

Endereço	Mnemônico	Código
4000	MVI A, 00h	3E 00
4002	CALL APDIS	CD 0B 07
4005	PUSH PSW	F5
4006	CALL DBYTE	CD A1 05
4009	POP PSW	F1
400A	INR A	3C
400B	JMP 4002	C3 02 40

4. Inclua, após a instrução CALL DBYTE, a subrotina de atraso de tempo DEL 1 (50 ms), execute o programa novamente e observe o aumento no tempo entre cada valor mostrado no display. O trecho a ser incluído é mostrado a seguir:

Endereço	Mnemônico	Código
4009	CALL DEL1	CD 70 07
400C	POP PSW	F1
400D	INR A	3C
400E	JMP 4002	C3 02 40

5. Questões do Experimento 1

- (a) O que ocorre quando se diminui o valor de D no ABACUS?
- (b) Por que foram usadas as instruções PUSH PSW e POP PSW no ABACUS e no Kit Didático?
- (c) Sendo SP = 2020 h no ABACUS, em que endereço será armazenado o primeiro dado da pilha?
- (d) A subrotina DEL1 provoca um atraso de 50 ms. Mostre como ela pode ser usada para gerar um atraso de tempo de aproximadamente 200 ms.

## EXPERIMENTO 2

**Objetivo:** Familiarização com as instruções de transferência de dados

**Material:** Computador com o simulador ABACUS e kit didático

**Roteiro:**

1. Codifique as operações dadas na tabela a seguir e insira no ABACUS os mnemônicos resultantes. Depois, execute cada instrução PASSO-A-PASSO, observando o que ocorre no registrador, ou nos registradores e posições de memória envolvidas.

Operação	End.	Mnemônico	Comentário
(A) ← 99h	2000	MVI A, 99	Carrega acumulador com valor 99
(H,L) ← 2050h	2002		
(2050) ← (A)			
(D,E) ← 2060h			
(E) ← 70h			
(B) ← (E)			
((H,L)) ← (B)			
((D,E)) ← (A)			
(H,L) ↔ (D,E)			
(SP) ← 20C0 h			
(L) ← AA h			
((H,L)) ← 22 h			
(D) ← ((H,L))			
(A) ← (D)			
(H) ← (L)			

2. Complete a tabela dada a seguir, fazendo uma adaptação do que foi feito na tabela anterior. Acrescente os códigos das operações e insira-os no kit didático. Execute cada instrução PASSO-A-PASSO, observando o que ocorre no registrador, ou nos registradores e posições de memória envolvidas.

Operação	End.	Mnemônico	Código	Comentário
(A) ← 99h	4000	MVI A, 99	3E 99	Carrega acumulador com valor 99
(H,L) ← 4050h	4002			
(4050) ← (A)				
(D,E) ← 4060h				
(E) ← 70h				
(B) ← (E)				
((H,L)) ← (B)				
((D,E)) ← (A)				
(H,L) ↔ (D,E)				
(SP) ← 4B00h				
(L) ← AA h				
((H,L)) ← 22 h				
(D) ← ((H,L))				
(A) ← (D)				
(H) ← (L)				

3. Questões do Experimento 2

- (a) Qual o conteúdo final dos registradores A, D, E e H no ABACUS?
- (b) Qual o conteúdo final do endereço 2050 h no ABACUS?
- (c) Em que endereço foi inserido o opcode da instrução (SP) ← 4B00 h?
- (d) Qual o endereço apontado pelo par HL após a instrução (H,L) ↔ (D,E) no kit didático?
- (e) Em que endereço o valor 22 h foi armazenado após a instrução ((H,L)) ← 22h no kit didático?

## EXPERIMENTO 3

Objetivo: Familiarização com as instruções aritméticas do 8085

Material: Computador com o simulador ABACUS e kit didático

Roteiro:

1. Codifique as operações dadas na tabela a seguir e insira no ABACUS os mnemônicos resultantes. Depois, execute cada instrução PASSO-A-PASSO, observando o que ocorre no registrador, ou nos registradores e posições de memória envolvidas.

Operação	End.	Mnemônico	2Comentário
$(H,L) \leftarrow 2050h$	2000	LXI H, 2050	
$((H,L)) \leftarrow 10h$	2003		
$(E) \leftarrow 05h$			
$(A) \leftarrow 01h$			
$(B) \leftarrow FFh$			
$(A) \leftarrow (A) + (B)$			
$(A) \leftarrow (A) + ((H,L)) + CY$			
$(A) \leftarrow (A) + 21h$			
$(A) \leftarrow (A) - (E)$			
$(A) \leftarrow (A) - (B) - CY$			
$(A) \leftarrow (A) - 33h$			
$(B,C) \leftarrow 2F01h$			
$(B) \leftarrow (B) + 1$			
$(C) \leftarrow (C) - 1$			
$(B,C) \leftarrow (B,C) - 1$			
$(D) \leftarrow (C) + (H)$			

2. Complete a tabela dada a seguir, fazendo uma adaptação do que foi feito na tabela anterior. Acrescente os códigos das operações e insira-os no kit didático. Execute cada instrução PASSO-A-PASSO, observando o que ocorre no registrador, ou nos registradores e posições de memória envolvidas.

Operação	End.	Mnemônico	Código	Comentário
$(H,L) \leftarrow 4050h$	4000	LXI H, 4050	21 50 40	
$((H,L)) \leftarrow 10h$	4003			
$(E) \leftarrow 05h$				
$(A) \leftarrow 01h$				
$(B) \leftarrow FFh$				
$(A) \leftarrow (A) + (B)$				
$(A) \leftarrow (A) + ((H,L)) + CY$				
$(A) \leftarrow (A) + 21h$				
$(A) \leftarrow (A) - (E)$				
$(A) \leftarrow (A) - (B) - CY$				
$(A) \leftarrow (A) - 33h$				
$(B,C) \leftarrow 2F01h$				
$(B) \leftarrow (B) + 1$				
$(C) \leftarrow (C) - 1$				
$(B,C) \leftarrow (B,C) - 1$				
$(D) \leftarrow (C) + (H)$				

### 3. Questões do Experimento 3

- (a) Qual o conteúdo do registrador “A” após a operação  $(A) \leftarrow (A) + 21h$  no ABACUS?
- (b) Qual o valor da flag de carry ao final da tabela do ABACUS e do kit didático?
- (c) Qual o conteúdo dos registradores A, B, C e D ao final da tabela?
- (d) Escreva um programa (usando bytes imediatos para os dados) que soma os decimais 500 e 650.

## EXPERIMENTO 4

Objetivo: Familiarização com as instruções lógicas do 8085

Material: Computador com o simulador ABACUS e kit didático

Roteiro:

1. Codifique as operações dadas na tabela a seguir e insira no ABACUS os mnemônicos resultantes. Depois, execute cada instrução PASSO-A-PASSO, observando o que ocorre no registrador, ou nos registradores e posições de memória envolvidas.

Operação	End	Mnemônico	Comentário
$(A) \leftarrow 0F\text{ h}$	2000		
$(B) \leftarrow EA\text{ h}$			
$(C) \leftarrow 52\text{ h}$			
$(A) \leftarrow (A) \wedge (C)$			
$(H,L) \leftarrow 2040\text{ h}$			
$((H,L)) \leftarrow FF\text{ h}$			
$(A) \leftarrow (C) \wedge ((H,L))$			
$(A) \leftarrow (A) \vee 44\text{ h}$			
$(A) \leftarrow (A) \vee 23\text{ h}$			
$(A) - 33\text{ h}$			
$(A_{n+1}) \leftarrow (A_n)$ $(A_0) \leftarrow (A_7)$			
$(CY) \leftarrow (\overline{CY})$			
$(A) - (B)$			
$(B) \leftarrow (B) \vee (C)$			

2. Complete a tabela dada a seguir, fazendo uma adaptação do que foi feito na tabela anterior. Acrescente os códigos das operações e insira-os no kit didático. Execute cada instrução PASSO-A-PASSO, observando o que ocorre no registrador, ou nos registradores e posições de memória envolvidas.

Operação	End	Mnemônico	Código	Comentário
$(A) \leftarrow 0F\text{ h}$	4000			
$(B) \leftarrow EA\text{ h}$				
$(C) \leftarrow 52\text{ h}$				
$(A) \leftarrow (A) \wedge (C)$				
$(H,L) \leftarrow 4040\text{ h}$				
$((H,L)) \leftarrow FF\text{ h}$				
$(H,L) \leftarrow 0000\text{ h}$				
$(A) \leftarrow (C) \wedge ((H,L))$				
$(A) \leftarrow (A) \vee 44\text{ h}$				
$(A) \leftarrow (A) \vee 23\text{ h}$				
$(A) - 33\text{ h}$				
$(A_{n+1}) \leftarrow (A_n)$ $(A_0) \leftarrow (A_7)$				
$(CY) \leftarrow (\overline{CY})$				
$(A) - (B)$				
$(B) \leftarrow (B) \vee (C)$				

3. Questões do Experimento 4

- (a) Qual o conteúdo final dos registradores A, B, C, H e L no ABACUS?
- (b) Qual o valor final das flags de carry CY e de zero, no ABACUS, após a instrução de comparação  $(A) - 33\text{ h}$ ? E o conteúdo de A?
- (c) Qual o conteúdo das flags de carry CY e de zero Z, no kit didático, após a instrução de comparação  $(A) - (B)$ ? E o conteúdo de A?

## EXPERIMENTO 5

Objetivo: Programação com o assembly do 8085

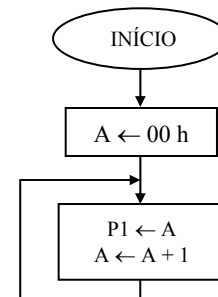
Material: Computador com o simulador ABACUS

Roteiro:

1. O programa a seguir gera os múltiplos de 4, a partir de 4 até 20 h. Esses valores são armazenados na memória a partir do valor dado para HL. Insira os mnemônicos no ABACUS e execute o programa. Observe o resultado.

Label	Mnemônico
	LXI SP, 2090 h
	LXI H, 2050 h
	MVI C, 01 h
	MVI B, 04 h
volta:	MVI A, 00 h
	PUSH B
	CALL produto
	MOV M, A
	CPI 20h
	JNC fim

Label	Mnemônico
	POP B
	INX H
	INR C
	JMP volta
fim:	HLT
produto:	ADD B
	DCR C
	JNZ subrotina
	RET



2. O programa a seguir é uma nova versão do programa anterior incluindo a subrotina para mostrar os múltiplos de 4 no display de dados e retirando as instruções para armazenamento na memória.

Label	Mnemônico
	LXI SP, 2090 h
	MVI C, 01 h
	MVI B, 04 h
volta:	MVI A, 00 h
	PUSH B
	CALL produto
	PUSH PSW
	CALL MOSTRAA
	POP PSW
	CPI 50h

Label	Mnemônico
	JNC fim
	POP B
	INR C
	JMP volta
fim:	HLT
produto:	ADD B
	DCR C
	JNZ subrotina
	RET

3. O programa a seguir é uma nova versão dos programas anteriores, onde o número para o qual se deseja os múltiplos é inserido através do teclado.

Label	Mnemônico
	LXI SP, 2090 h
	MVI C, 01 h
	CALL LETECLA
	MOV B, A
volta:	MVI A, 00 h
	PUSH B
	CALL produto
	PUSH PSW
	CALL MOSTRAA
	POP PSW
	CPI 50h

Label	Mnemônico
	JNC fim
	POP B
	INR C
	JMP volta
fim:	HLT
produto:	ADD B
	DCR C
	JNZ subrotina
	RET

### 4. Questões do Experimento 5

- (a) Por que foram usadas as instruções PUSH B e POP B no primeiro programa?
- (b) Por que foram usadas as instruções PUSH PSW e POP PSW no segundo e no terceiro programas?
- (c) Observando o programa durante a execução, verifica-se que ele vai ficando mais lento na medida em que os múltiplos maiores vão sendo mostrados. Por que isso acontece?
- (d) Você pode sugerir uma maneira de resolver o problema citado no item anterior?
- (e) Qual a finalidade da instrução CPI 50 h no segundo e no terceiro programas?

## EXPERIMENTO 6

Objetivo: Programação com o assembly do 8085

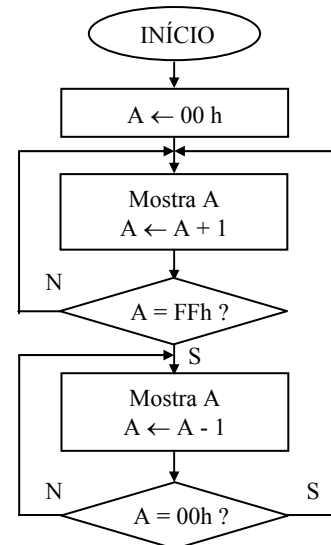
Material: Computador com o simulador ABACUS e kit didático

Roteiro:

1. O programa a seguir gera e mostra no display uma sequência crescente de 00 h a FF h, seguida de uma sequência decrescente de FF h a 00 h, de forma ininterrupta. Insira os mnemônicos no ABACUS e execute o programa. Observe o resultado.

Label	Mnemônico
	LXI SP, 2090 h
	MVI A, 00 h
cresce:	PUSH PSW
	CALL MOSTRAA
	POP PSW
	INR A
	CPI FF h
	JNZ cresce

Label	Mnemônico
decesce:	PUSH PSW
	CALL MOSTRAA
	POP PSW
	DCR A
	CPI 00 h
	JNZ decresce
	JMP cresce
	HLT



2. O programa a seguir é uma nova versão do programa anterior, mas com contagem em decimal que vai de 00 a 99 e volta para 00, de forma ininterrupta.

Label	Mnemônico
	LXI SP, 2090 h
	MVI A, 00 h
cresce:	PUSH PSW
	CALL MOSTRAA
	POP PSW
	ADI 01 h
	DAA
	CPI 99 h
	JNZ cresce

Label	Mnemônico
decesce:	PUSH PSW
	CALL MOSTRAA
	POP PSW
	ADI 99 h
	DAA
	CPI 00 h
	JNZ decresce
	JMP cresce
	HLT

3. O programa a seguir é uma adaptação do programa anterior (contagem decimal) para ser executado no kit didático. Codifique o programa e insira-o no kit didático.

Label	Mnemônico	Código
	LXI SP, 4090 h	
	MVI A, 00 h	
cresce:	CALL APDIS	
	PUSH PSW	
	CALL DBYTE	
	CALL DEL1	
	POP PSW	
	ADI 01 h	
	DAA	
	CPI 99 h	
	JNZ cresce	

Label	Mnemônico	Código
decesce:	CALL APDIS	
	PUSH PSW	
	CALL DBYTE	
	CALL DEL1	
	POP PSW	
	ADI 99 h	
	DAA	
	CPI 00 h	
	JNZ decresce	
	JMP cresce	
	HLT	

4. Questões do Experimento 6

- (a) Por que a instrução INR A foi substituída pela instrução ADI 01 h no programa 2?
- (b) Por que foi usada a instrução ADI 99 h nos programas 2 e 3, ao invés de DCR A, como no programa 1?
- (c) Por que foi usada a instrução CALL APDIS no programa 3?
- (d) Por que foi usada a instrução CALL DEL1 no programa 3?
- (e) Substitua a instrução CALL DEL1 por CALL DEL2 e observe o efeito?



## EXPERIMENTO 7

Objetivo: Programação com o assembly do 8085 e uso de INTERRUPÇÕES

Material: Computador com o simulador ABACUS e kit didático

1. Execute os programas dados a seguir usando o simulador ABACUS e usando o módulo didático.

Programa 1: Usa subrotinas para gerar de forma ininterrupta uma sequência crescente de 00 h a FF h no display de dados, seguida de uma sequência decrescente (FF h a 00 h). Complete os códigos.

<i>ABACUS</i>		
Label	Mnemônico	Código
2000	LXI SP,2040h	31 40 20
	MVI A,00h	3E 00
repete:	CALL crescente	CD _____
	CALL decresc	CD _____
	JMP repete	C3 _____
crescente:	PUSH PSW	F5
	CALL MOSTRAA	CD 6E 03
	MVI D,01h	16 01
	CALL DELAY	CD F1 05
	POP PSW	F1
	CPI FF	FE FF
	RZ	C8
	INR A	3C
	JMP crescente	C3 _____
decresc:	DCR A	3D
	PUSH PSW	F5
	CALL MOSTRAA	CD 6E 03
	MVI D,01h	16 01
	CALL DELAY	CD F1 05
	POP PSW	F1
	CPI 00	FE 00
	RZ	C8
	JMP decresc	C3 _____

<i>KIT DIDÁTICO</i>		
Label	Mnemônico	Código
4000	LXI SP,4200h	31 00 42
	MVI A,00h	3E 00
repete:	CALL crescente	CD _____
	CALL decresc	CD _____
	JMP repete	C3 _____
crescente:	PUSH PSW	F5
	CALL APDIS	CD 0B 07
	CALL DBYTE	CD A1 05
	CALL DEL1	CD 70 07
	POP PSW	F1
	CPI FF	FE FF
	RZ	C8
	INR A	3C
	JMP crescente	C3 _____
decresc:	DCR A	3D
	PUSH PSW	F5
	CALL APDIS	CD 0B 07
	CALL DBYTE	CD A1 05
	CALL DEL1	CD 70 07
	POP PSW	F1
	CPI 00	FE 00
	RZ	C8
	JMP decresc	C3 _____

### ENDEREÇO DAS INTERRUPÇÕES

Interrupção	Pino do 8085	Endereço na ROM	Endereço no ABACUS	Endereço no KIT
TRAP	6	0024 h	20D1h	4BE0 h
RST 7.5	7	003C h	20CE h	4BD7 h
RST 6.5	8	0034 h	20CB h	4BDA h
RST 5.5	9	002C h	20C8 h	4BDD h

Programa 2: Adapte o programa 1 para que a contagem crescente seja chamada através da interrupção **RST 7.5** e a contagem decrescente seja chamada pela interrupção **RST 6.5**. Desabilite a interrupção **RST 5.5**.

<b>ABACUS</b>		
<b>Label</b>	<b>Mnemônico</b>	<b>Código</b>
2000	LXI SP,2050h	31 50 20
	MVI A,09h	3E 09
	SIM	30
	MVI A,00h	3E 00
loop:	EI	FB
	JMP loop	C3 _____
crescente:	DI	F3
	PUSH PSW	F5
	CALL MOSTRAA	CD 6E 03
	MVI D,01h	16 01
	CALL DELAY	CD F1 05
	POP PSW	F1
	CPI FF	FE FF
	RZ	C8
	INR A	3C
	JMP crescente	C3 _____
decresc:	DI	F3
	DCR A	3D
	PUSH PSW	F5
	CALL MOSTRAA	CD 6E 03
	MVI D,01h	16 01
	CALL DELAY	CD F1 05
	POP PSW	F1
	CPI 00	FE 00
	RZ	C8
	JMP decresc	C3 _____
20CB h:	JMP decresc	C3 _____
20CE h:	JMP crescente	C3 _____

<b>KIT DIDÁTICO</b>		
<b>Label</b>	<b>Mnemônico</b>	<b>Código</b>
4000	LXI SP,4200h	31 00 42
	MVI A,09h	3E 09
	SIM	30
	MVI A,00h	3E 00
loop:	EI	FB
	JMP loop	C3 _____
crescente:	DI	F3
	PUSH PSW	F5
	CALL APDIS	CD 0B 07
	CALL DBYTE	CD A1 05
	CALL DEL1	CD 70 07
	POP PSW	F1
	CPI FF	FE FF
	RZ	C8
	INR A	3C
	JMP crescente	C3 _____
decresc:	DI	F3
	DCR A	3D
	PUSH PSW	F5
	CALL APDIS	CD 0B 07
	CALL DBYTE	CD A1 05
	CALL DEL1	CD 70 07
	POP PSW	F1
	CPI 00	FE 00
	RZ	C8
	JMP decresc	C3 _____
4BDA h:	JMP decresc	C3 _____
4BD7 h:	JMP crescente	C3 _____

Perguntas:

1. Por que há necessidade de desabilitar a interrupção RST 5.5?
2. Quais os comandos a serem usados para bloquear a interrupção RST 7.5 e usar a interrupção RST 6.5 para a contagem crescente e a RST 5.5 para a contagem decrescente?
3. Qual a finalidade da instrução RZ usada nas subrotinas?

## 5.4 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] Digibyte, “Manual do Usuário do Kit Didático KMD85”, Abril de 1984.

## 6. INTERRUPTÕES E OPERAÇÕES DE ENTRADA E SAÍDA

### 6.1 Instruções de Recomeço

O 8085 tem 8 instruções de **recomeço por software** (restart), que são: RST 0, RST 1, RST 2, RST 3, RST 4, RST 5, RST 6 e RST 7. Estas instruções são para chamadas de subrotinas que são usadas com frequência. O efeito de uma instrução RST é o mesmo de uma chamada de subrotina, com uso da pilha para guardar o endereço de retorno. A diferença é que cada instrução RST desvia o processamento para um endereço predeterminado, como mostra a tabela abaixo. Um outro detalhe é que a chamada de uma RST é feita com apenas 1 byte de código, ao contrário de uma chamada de subrotina normal, que é feita com 3 bytes de código. Do mesmo modo que no caso de uma subrotina comum, quando uma instrução RET é encontrada o processamento volta para o programa principal.

INSTRUÇÕES DE RECOMEÇO

Instrução	Efeito	Código de Operação	Posição do Vetor
RST 0	CALL 0000h	C7	0000h
RST 1	CALL 0008h	CF	0008h
RST 2	CALL 0010h	D7	0010h
RST 3	CALL 0018h	DF	0018h
RST 4	CALL 0020h	E7	0020h
RST 5	CALL 0028h	EF	0028h
RST 6	CALL 0030h	F7	0030h
RST 7	CALL 0038h	FF	0038h

Verifica-se da tabela acima que há apenas 8 posições (8 bytes) disponíveis para cada instrução de recomeço. Sendo assim, normalmente essas posições vetoradas são usadas apenas para desviar para uma subrotina que é frequentemente usada e que não caberia em apenas 8 posições. Uma instrução de desvio incondicional JMP addr é então usada. A chamada da subrotina através de uma dessas instruções de recomeço é mais eficiente que a chamada direta da subrotina.

### 6.2 Interrupções

Além das 8 instruções de recomeço por software, o 8085 possui 4 instruções de **recomeço por hardware** (**Interrupções Externas**). Essas instruções são chamadas de **TRAP** (pino 6), **RST 7.5** (pino 7), **RST 6.5** (pino 8) e **RST 5.5** (pino 9). Quando qualquer desses pinos é ativado, os circuitos internos do 8085 produzirão uma chamada de subrotina (CALL) por hardware que desviarão o processamento para o endereço predeterminado. A tabela das posições de recomeço por hardware (interrupções externas) é dada a seguir.

As operações de entrada e saída por interrupção usam uma das interrupções por hardware dadas na tabela, de forma que, quando o periférico está pronto para a transferência de dados a interrupção selecionada é acionada, permitindo a transferência de dados de uma forma mais rápida que a transferência Programada.

Interrupções do 8085

Instrução	Efeito	Código de Operação	Posição do Vetor	Pino
TRAP	CALL 0024h	não tem	0024 h	6
RST 5.5	CALL 002Ch	não tem	002C h	9
RST 6.5	CALL 0034h	não tem	0034 h	8
RST 7.5	CALL 003Ch	não tem	003C h	7

As interrupções externas tem uma ordem de prioridade, de forma que, no caso de dois ou mais pedidos simultâneos de interrupção, a ordem de atendimento é:

Interrupção	Prioridade	Posição do Vetor
TRAP	1	0024 h
RST 7.5	2	003C h
RST 6.5	3	0034 h
RST 5.5	4	002C h
INTR	5	Nenhuma

Quando um pedido de interrupção é feito, o microprocessador termina a execução da instrução em curso para somente depois atender a interrupção pendente. A interrupção **INTR**, também presente na tabela de prioridades é um canal que permite expandir a capacidade de interrupção. É então usado um CI extra para essa expansão, como será visto numa seção posterior.

Como já foi mencionado, as interrupções são acionadas por sinais externos nos pinos correspondentes. Para o caso do 8085 esses sinais de acionamento de interrupção podem ser nível alto no pino (acionamento por nível) ou transição do nível baixo para o nível alto (acionamento por borda, ou por transição). A figura a seguir ilustra cada uma delas.

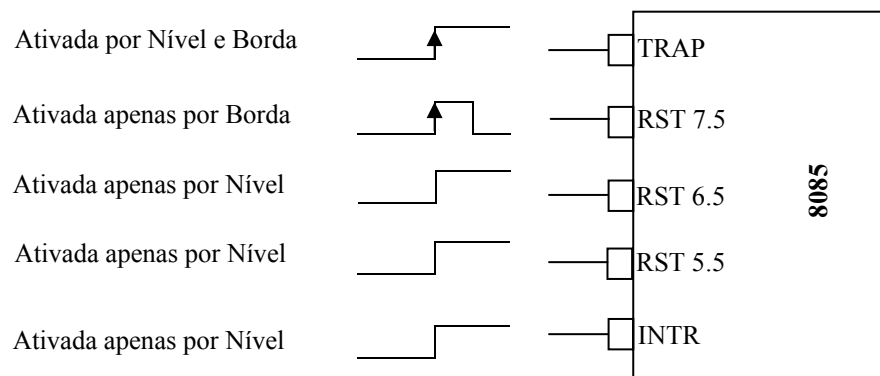


Fig. 6.1: Formas de se ativar cada interrupção

- TRAP - acionada simultaneamente por nível e por transição. É usada para eventos catastróficos tais como falhas na alimentação elétrica e erros de paridade.
- RST 7.5 - acionada por transição somente
- RST 6.5 - acionada apenas por nível
- RST 5.5 - acionada apenas por nível
- INTR - acionada apenas por nível. Essa interrupção em particular, permite a expansão da capacidade de interrupções do 8085. Através desse pino um dispositivo de suporte pode ser usado para aumentar os pinos de interrupção.

### 6.3 Circuitos de Interrupção

As interrupções são melhor entendidas através dos circuitos lógicos mostrados a seguir.

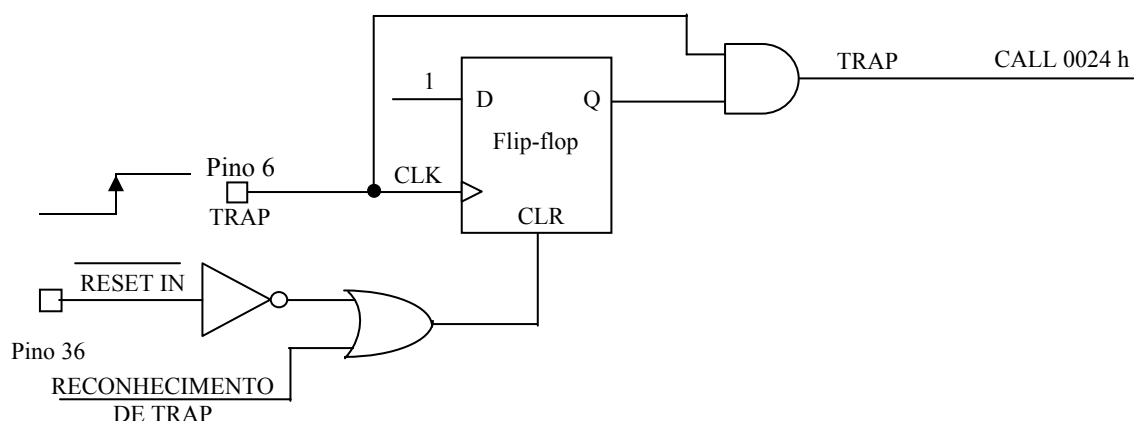


Fig. 6.2: Circuito da interrupção TRAP

No circuito da Fig. 6.2 o sinal alto de clock habilita a porta AND na saída do Flip-Flop, enquanto que a transição do clock do nível baixo para o nível alto faz com que a saída do Flip-Flop assumo o valor 1 e, consequentemente, a saída geral do circuito da interrupção **TRAP** assume o valor 1, fazendo com que o processamento seja desviado para o endereço 0024 h.

Um sinal baixo no pino **RESET-IN** ou um sinal alto de RECONHECIMENTO DE TRAP limpa o pedido de interrupção.

A Fig. 6.3 mostra os circuitos para as interrupções **RST 7.5**, **RST 6.5** e **RST 5.5**. Observe que as portas AND de saída dos circuitos das interrupções tem três entradas. O sinal IE estando alto, todas as interrupções estão habilitadas. Esse sinal fica alto quando a instrução **EI** (entrada do flip-flop) é executada. Se nenhuma das interrupções estiver com máscara, então os sinais M7.5, M6.5 e M5.5 permanecem em nível baixo e, consequentemente as respectivas entradas das portas AND estarão altas. Dessa forma, todas as interrupções estarão aptas a serem acionadas.

Um pulso ascendente (transição do nível baixo para o nível alto) no pino 7 faz com que a saída **I7.5** do flip-flop correspondente à interrupção RST 7.5 fique alta, acionando essa interrupção.

Um nível alto no pino 8 faz com que a interrupção RST 6.5 seja executada e um nível alto no pino 9 faz com que a interrupção RST 5.5 seja executada. Os sinais I7.5, I6.5 e I5.5 quando estão em nível alto indicam que há interrupção pendente.

Um sinal R7.5 alto limpa uma interrupção RST 7.5 pendente. Um sinal de Reconhecimento de interrupção RST 7.5 também faz com que a saída do flip-flop seja zerada, limpando qualquer interrupção RST 7.5 pendente.

O bloqueio total das interrupção mostradas na Fig. 6.3 pode ser feito de três maneiras distintas: (a) através da instrução **DI** (**D**isable **I**nterrupt), que desabilita todas as interrupções, com exceção da TRAP; (b) através de um sinal alto no pino 36 (**RESET-IN**) ou (c) através de um sinal alto de reconhecimento de qualquer interrupção.

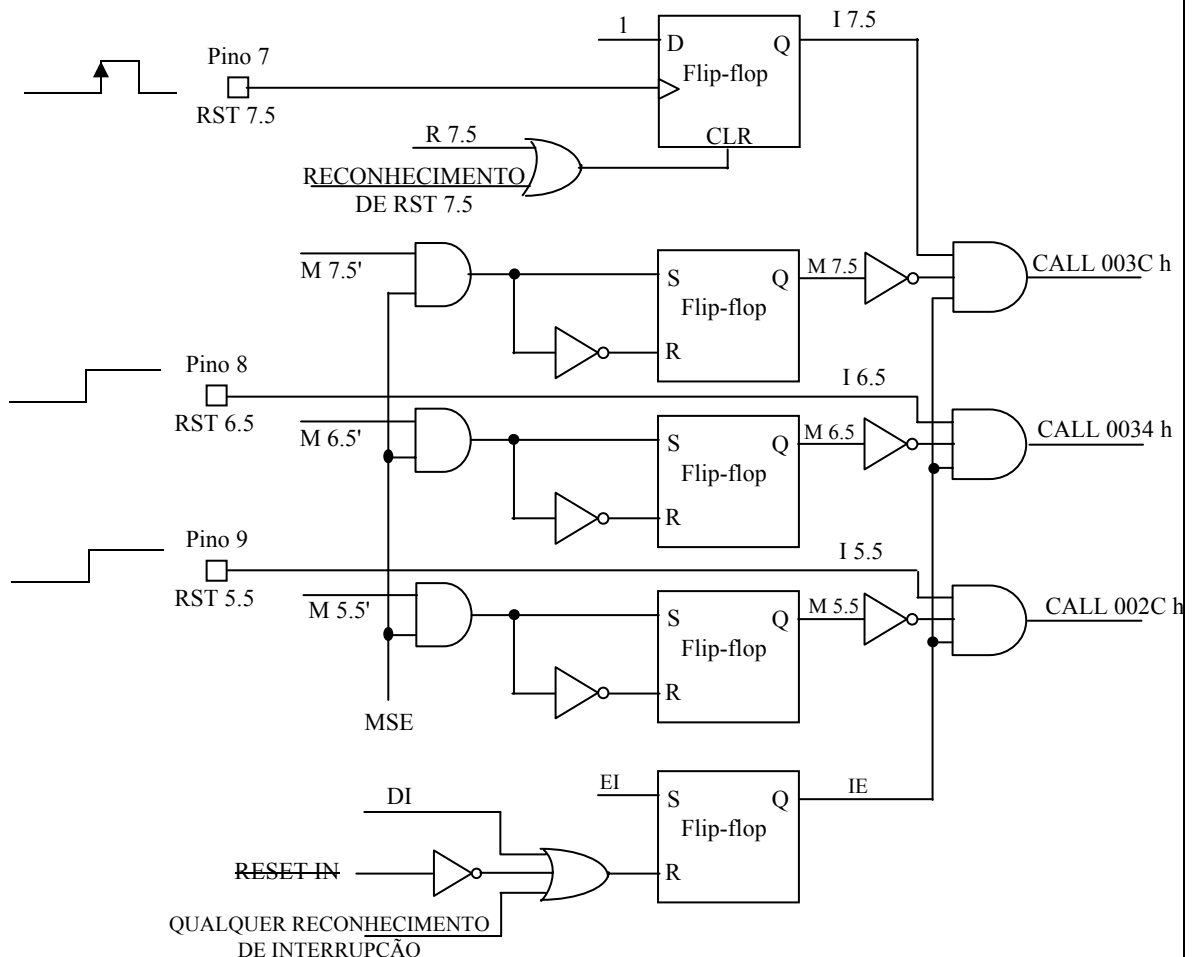


Fig. 6.3: Circuitos das interrupções RST 7.5, RST 6.5 e RST 5.5

A setagem de máscara também é mostrada na Fig. 6.3. Verifica-se que um sinal alto **MSE** (**M**ask **S**et **E**nable) é necessário para habilitar qualquer das máscaras de interrupção. A partir daí, desejando-se mascarar (não permitir a execução) a interrupção RST 7.5, o sinal M7.5' deve ser levado ao nível alto. Desejando-se

mascarar a interrupção RST 6.5 o sinal M6.5' deve estar em nível alto. E, finalmente, se a interrupção que se deseja mascarar é a RST 5.5, o sinal M5.5' deve ficar em nível alto.

SÍMBOLO	SIGNIFICADO/COMENTÁRIO
I7.5, I6.5, I5.5	Interrupções Pendentes - assumem valor 1 quando há interrupção pendente
IE	Flag que indica (com valor 1) quando as interrupções estão habilitadas
M7.5, M6.5 e M5.5	Máscaras de Interrupção (Sinal baixo habilita as portas AND) - podem desabilitar uma interrupção pendente. Sinal alto → interrupção bloqueada
RST 7.5, RST 6.5 e RST 5.5	interrupções mascaráveis → podem ser bloqueadas via <i>software</i>
TRAP	não é mascarável nem passível de habilitação/desabilitação por EI/DI
EI	Enable Interrupt - Instrução usada para habilitar todas as interrupções, exceto a TRAP. Ativo alto.
DI	Disable Interrupt - Instrução que Desabilita as Interrupções, exceto a TRAP (baixo)
MSE	Mask Set Enable - Habilita a setagem de máscara. Quando alto as máscaras M7.5', M6.5' e M5.5' podem ser reconhecidas
RESET IN \	A inicialização do sistema desabilita as interrupções pendentes e zera o flip-flop da TRAP
QUALQUER RECONHECIMENTO DE INTERRUPÇÃO	Quando qualquer das interrupções é atendida, um sinal alto desabilita as interrupções pendentes, menos a TRAP, evitando que a interrupção em execução seja interrompida.
SIM	Set Interrupt Mask - Instrução que seta as máscaras de interrupção, fazendo com que determinada interrupção não seja ativada.
RIM	Read Interrupt Mask - Instrução que lê o estado das máscaras das interrupções.

Embora os endereços de desvio sejam os indicados na Fig. 6.2 e 6.3, os endereços de desvio no ABACUS e no kit didático são diferentes. Na verdade, como o usuário final não pode inserir instruções/dados na ROM do ABACUS nem do kit didático, o criador dessas ferramentas incluiu no programa monitor uma instrução de desvio para a RAM, de forma que a cada interrupção o programa seja automaticamente desviado para um endereço acessível pelo usuário final. Esses endereços são dados na tabela a seguir.

Interrupção	Endereço na ROM	Endereço no ABACUS	Endereço no Kit
TRAP	0024 H	20D1 H	4BE0 H
RST 5.5	002C H	20C8 H	4BDD H
RST 6.5	0034 H	20CB H	4BDA H
RST 7.5	003C H	20CE H	4BD7 H

Observações.:

- (a) É comum usar a instrução **EI** como penúltima instrução de uma subrotina, tendo em vista que o microprocessador automaticamente desabilita todas as interrupções com o bit QUALQUER RECONHECIMENTO DE INTERRUPÇÃO.

Subrotina: -----

EI

RET

- (b) Quando se deseja evitar que uma parte do programa principal seja interrompida por qualquer interrupção, exceto a TRAP, usa-se o comando **DI** no começo do trecho a ser protegido.

Programa principal: -----

DI

EI

Exemplo:

O programa a seguir, a ser implementado no ABACUS, a cada interrupção RST 5.5, mostra uma contagem hexadecimal crescente de 00 h até FF h e dá mais um passo e pára em 00 h.

Rótulo/endereço	Mnemônico	Comentário
	LXI SP, 2090H	Topo da pilha na posição 2090 h
	MVI A, 00H	Acumulador começa com valor 00 h
	PUSH PSW	Guarda A e Flags na pilha
	CALL MOSTRAA	Mostra conteúdo de A no display de dados
	POP PSW	Recupera conteúdo de A e Flags da pilha
	EI	Habilita interrupções
espera:	JMP espera	Laço de espera de pedido de interrupção
cresce:	INR A	Incrementa acumulador
	PUSH PSW	Guarda conteúdo de A e F na pilha
	CALL MOSTRAA	Mostra conteúdo de A no display
	MVI D, 01H	Faz D = 01 h, para usar na subrotina de tempo
	CALL DELAY	Chama subrotina de atraso de tempo
	POP PSW	Recupera conteúdo de A e F da pilha
	CPI 00 H	Compara A com 00 h.
	JNZ cresce	Se A ≠ 00 h, volta para INR A. Se A = 00 h, sai da subrotina
	EI	Habilita novamente as interrupções
	RET	Retorna da subrotina de interrupção
20C8:	JMP cresce	Vai para o início da subrotina "cresce"

#### 6.4 Setagem e Leitura da Máscara de Interrupção

A máscara de interrupção é usada para que as interrupções sejam usadas de forma seletiva, onde as interrupções não desejadas sejam bloqueadas.

A máscara de uma interrupção mascarável só é reconhecida após a execução do comando **SIM** (Set Interrupt Mask = Setar a Máscara de Interrupção). E antes da execução deste comando alguns valores do circuito de interrupção (Fig. 6.3) devem ser carregados no acumulador de forma que a instrução SIM faz a transferência desses valores para os pontos necessários. Os valores no acumulador devem ser como mostrado a seguir:

Acumulador	SOD	SOE	×	R7.5	MSE	M7.5'	M6.5'	M5.5'
	Serial Output Data - Dado da Saída Serial	Serial Output Enable - Habilita Saída Serial	Irrelevante	Zera flip-flop da RST 7.5	Mask Set Enable - Habilita Setagem de Máscara	Seta máscara da RST 7.5	Seta máscara da RST 6.5	Seta máscara da RST 5.5

Exemplo: Desejando-se habilitar apenas a interrupção **RST 5.5** e mascarar as demais, o conteúdo do acumulador antes da instrução SIM é dado a seguir:

A:	SOD	SOE	×	R7.5	MSE	M7.5'	M6.5'	M5.5'
	0	0	0	0	1	1	1	0

A execução do comando **RIM** (Read Interrupt Mask) faz com que o estado das máscaras de interrupção sejam transferidos para o acumulador.

A seguir é mostrado o significado de cada bit do acumulador após a execução da instrução RIM. É também dado um exemplo, onde é mostrado o conteúdo do acumulador e a interpretação de cada bit, após a execução da instrução RIM.

Acumulador	SID	I7.5	I6.5	I5.5	IE	M7.5	M6.5	M5.5
	Serial Input Data - Dado de Entrada Serial	Interrupção RST 7.5 pendente	Interrupção RST 6.5 pendente	Interrupção RST 5.5 pendente	Interrupt Enable - Habilita Interrupção	Máscara da interrupção RST 7.5	Máscara da interrupção RST 6.5	Máscara da interrupção RST 5.5

Exemplo: Após a instrução RIM o conteúdo do acumulador ficou:

A:	SID	I7.5	I6.5	I5.5	IE	M7.5	M6.5	M5.5
	0	1	0	0	1	1	0	0

Significado:

- Há uma interrupção RST 7.5 pendente ( I7.5 = 1 )
- As interrupções estão habilitadas ( IE = 1 )
- A máscara da interrupção RST 7.5 está setada, significando que ela não será atendida

## 6.5 Ampliando a Capacidade de Interrupção

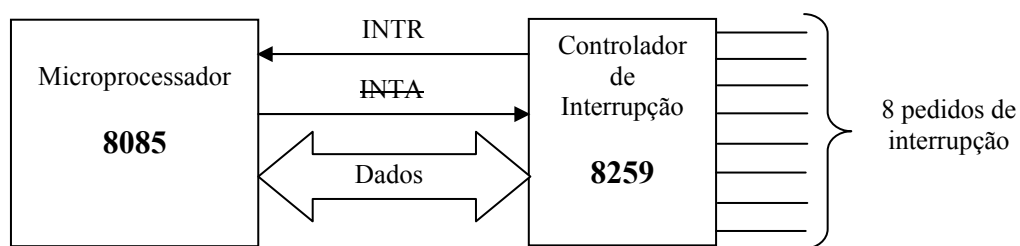


Fig. 6.4: Controlador de interrupção

O Controlador de interrupção (8259) mostrado acima permite expandir a capacidade de interrupção do microcontrolador 8085. Cada partilha 8259 permite acrescentar em 8 as linhas de interrupções, permitindo, portanto a conexão de um número maior de periféricos ao microprocessador.

Nesse circuito, o 8259 contém os endereços iniciais de oito subrotinas de atendimento de interrupção, um para cada dispositivo periférico. Para atender a interrupção ele envia uma *CALL endereço* para o 8085, o que faz com que o 8085 interrompa o processamento normal para o atendimento da interrupção solicitada.

A cada solicitação de interrupção o controlador de interrupção envia um pulso alto para o 8085 através do pino INTR. O 8085 envia de volta um sinal baixo *INTA* (Interrupt Acknowledge) de reconhecimento de interrupção. A seguir o 8259 envia o código de operação da subrotina, ao qual o 8085 responde com outro sinal *INTA* baixo solicitando o byte inferior do endereço da subrotina. Após o envio do byte inferior de endereço um outro sinal *INTA* baixo é enviado pelo 8085 solicitando o byte superior do endereço da subrotina. A seguir, o conteúdo do contador de programa (PC) é enviado para a pilha e o processamento é desviado para o endereço indicado. A Fig. 6.5 mostra o diagrama de blocos do controlador 8259.



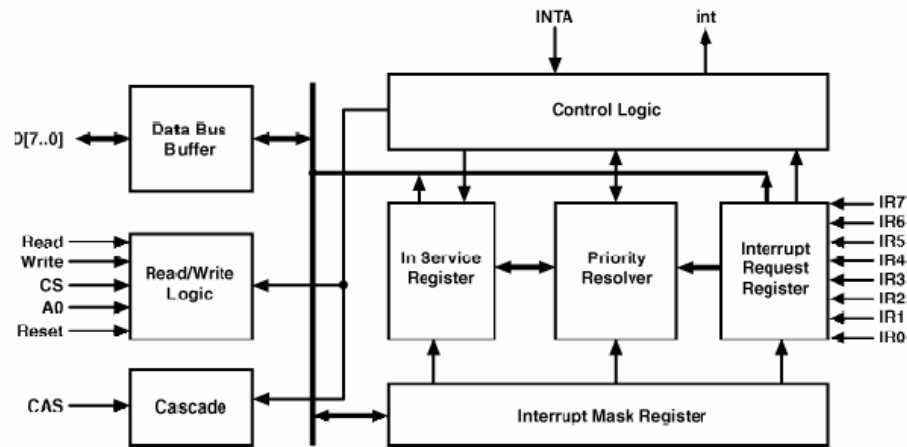


Fig. 6.5: Diagrama de blocos do controlador de interrupção 8259

O procedimento para uso de interrupções através do 8259 é:

1. O nível lógico de um ou mais dos pinos IR (IR0 a IR7) é levado a 1, setando o correspondente registrador de pedido de interrupção IRR;
2. O 8259 envia um pulso alto para o 8085 através do pino INTR (pino 10);
3. O 8085 envia de volta um sinal baixo INTA (Interrupt Acknowledge) de reconhecimento de pedido de interrupção;
4. Após receber o sinal de OK, o 8259 o bit ISR (In Service Register) do pedido de maior prioridade, limpa o registro de interrupção IRR correspondente e envia para o 8085 o código de chamada de subrotina;
5. O 8085 responde com outro sinal INTA baixo solicitando o byte inferior do endereço da subrotina;
6. O 8259 envia o byte inferior de endereço correspondente ao pedido de interrupção;
7. Um outro sinal INTA baixo é enviado pelo 8085 solicitando o byte superior do endereço da subrotina;
8. O 8259 envia o byte superior do endereço da subrotina para o 8085 e, em seguida, limpa o bit ISR correspondente;
9. A seguir, o conteúdo do contador de programa (PC) é enviado para a pilha e o processamento é desviado para o endereço indicado

## 6.6 Dispositivos de Entrada e Saída

Os dispositivos de Entrada e Saída (ou dispositivos de I/O) são necessários para possibilitar a comunicação entre a CPU e o resto do sistema.

Este capítulo tem o objetivo de fornecer maiores informações sobre o hardware que se associa à CPU (microprocessador) em um microcomputador, assim como conhecimentos de alguns recursos que podem ser utilizados em interfaces de microcomputadores.

### FORMAS DE ENTRADA E SAÍDA

Há basicamente duas formas de comunicação entre o computador e um periférico:

- **Comunicação Serial:** quando é enviado um bit de cada vez. É empregada, por exemplo, na comunicação via linha telefônica (através de um modem), no mouse, câmara fotográfica digital, impressora serial, instrumentos eletrônicos, agendas eletrônicas. A taxa de transmissão é medida em bit por segundo (bps ou bit/s).
- **Comunicação Paralela:** quando é enviado um grupo de bits, simultaneamente. É empregada, por exemplo, na comunicação entre um computador e uma impressora paralela. A taxa de transmissão é, em geral, medida

em bytes por segundo; OBS: a transmissão paralela é utilizada em substituição à transmissão serial quando a distância entre transmissor/receptor é pequena e deseja-se maior velocidade de transferência.

### MÉTODOS DE CONTROLE DE ENTRADA E SAÍDA

Referem-se a técnicas de hardware e software utilizadas para controlar o fluxo de dados entre o computador e seus periféricos. Os três métodos mais frequentemente encontrados são:

- **Varredura** (ou E/S por consulta ou *Pooling*):
  - ⇒ Técnica de comunicação na qual o processador interroga periodicamente o periférico para determinar seu estado;
  - ⇒ Pode ser síncrona ou assíncrona;
  - ⇒ Desvantagem: perda de tempo: o processador abandona o programa principal para fazer a varredura mesmo se nenhum periférico deseja serviço.
- **Interrupção**:
  - ⇒ Técnica de comunicação na qual o processador somente é ocupado se há pedido de serviço de algum periférico;
  - ⇒ Mecanismo assíncrono;
  - ⇒ Transferência de dados é feita por software (mais lento que DMA);
- **DMA** (*Direct Memory Access*):
  - ⇒ Técnica de comunicação na qual não se interrompe o processamento do microprocessador (se a instrução em curso não faz acesso ao barramento);
  - ⇒ O DMAC (controlador de DMA) envia o sinal HOLD requerendo o barramento e a CPU responde com HLDA. O DMAC passa a controlar as linhas de dados, linhas de endereço, e sinais de controle (RD, WR, etc.);
  - ⇒ A transferência de dados é feita por hardware: é particularmente interessante quando necessita-se transferência de dados em alta velocidade.

#### Exemplo de Entrada Programada:

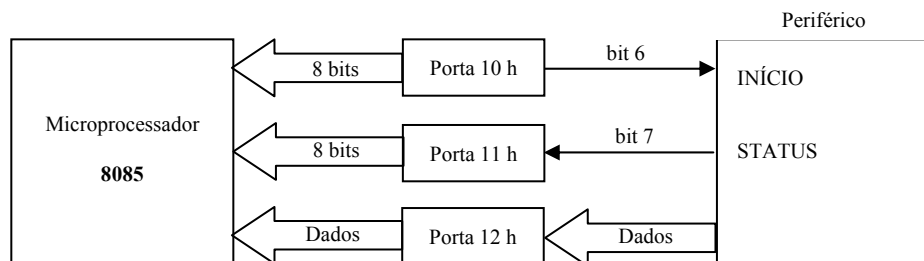


Fig. 6.6: Entrada de dados programada

Nesse exemplo, quando a CPU está pronta para receber dados ela envia um bit alto de INÍCIO (bit 6) para o periférico. Quando os dados estão prontos, um bit alto de STATUS (bit 7) é enviado de volta à CPU. Em seguida os dados são transferidos do periférico para a CPU. O grande problema nesse tipo de transferência de dados é que a CPU fica aguardando o bit alto de STATUS, o que pode significar um tempo considerável, uma vez que a maior parte do tempo é gasto esperando o periférico terminar o processamento de dados.

#### Exemplo de Transferência por Interrupção:

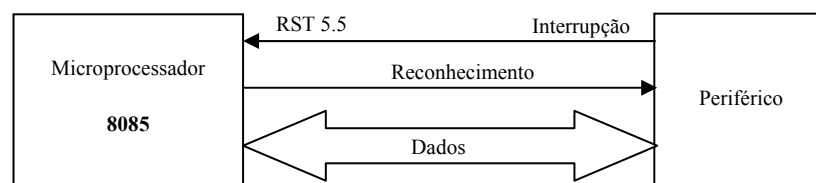


Fig. 6.7: Transferência de dados por interrupção

Nesse exemplo a CPU não precisa aguardar o processamento do periférico. Quando o periférico está pronto para transferir ou receber os dados um bit de INTERRUPÇÃO é enviado à CPU. O processamento normal da CPU é interrompido para atender o pedido externo. Um bit de RECONHECIMENTO de interrupção é enviado para o periférico, havendo então a transferência dados para a CPU.

#### Exemplo de Transferência por DMA:

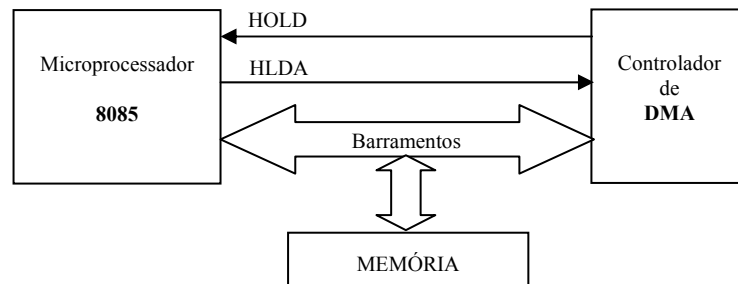


Fig. 6.8: Transferência de dados por Acesso Direto à Memória (DMA)

Nesse tipo de operação as transferências de dados são mais rápidas porque o acumulador não é utilizado e não há necessidade de leituras da memória ROM. O dados vão diretamente da memória para o periférico, ou vice-versa. O controlador de DMA é fisicamente conectado aos barramentos de dados, endereços e controle, dependendo apenas da liberação dos mesmos pela CPU. Dessa forma, quando o periférico está pronto para a transferência de dados, um sinal de HOLD alto é enviado à CPU solicitando a liberação do barramentos. A CPU responde com um sinal HLDA, informando que o processamento foi interrompido e os barramentos liberados. O controlador de DMA realiza então a transferência de dados em alta velocidade e devolve o controle dos barramentos para a CPU, enviando um sinal de HOLD baixo.

## 6.7 Exercícios Propostos

25. Explique como funciona uma interrupção no 8085. Explique habilitação e máscara de interrupção, interrupção vetorada, interrupção por nível, interrupção por transição. Exemplifique escrevendo as linhas de programa para mascarar as interrupções RST 5.5 e RST 7.5.
26. Para que servem as instruções EI e DI? Descreva as instruções SIM e RIM.
27. Qual a finalidade do bit MSE?
28. Insira os bits adequados do acumulador para mascarar a interrupção RST 6.5 e deixar as demais livres.

	SOD	SOE	×	R7.5	MSE	M7.5'	M6.5'	M5.5'
A:								

29. Faça a leitura do acumulador mostrado abaixo, após uma instrução RIM.

	SID	I7.5	I6.5	I5.5	IE	M7.5	M6.5	M5.5
A:	0	0	1	1	1	1	1	0

30. Zerar o flip-flop da interrupção RST 7.5, setar e habilitar as interrupções RST 7.5 e RST 6.5 e mascarar a interrupção RST 5.5. Comente o valor de cada bit utilizado na instrução de setagem de máscara. Desenhe um circuito simples com fototransistor que possa ser usado para acionar a interrupção RST 7.5 com a interrupção de um sinal luminoso.
31. Faça um programa no ABACUS, usando o assembly do 8085, que mostre uma contagem decimal crescente no display de zero a 50. Ao alcançar o valor 50, a interrupção RST 7.5 deve ser habilitada e o programa deve ficar aguardando um pedido de interrupção. Quando a interrupção RST 7.5 for solicitada, a contagem no display deve se tornar decrescente (e a interrupção desabilitada) até alcançar o valor zero. Alcançado o valor zero, a interrupção deve ser habilitada novamente para iniciar outra contagem decimal até 50, e assim sucessivamente.

32. Faça um programa no ABACUS que mostre uma contagem decimal no display, sendo que esta contagem deve ser crescente até que ocorra uma interrupção RST 6.5, quando então a contagem deverá se tornar decrescente. Caso ocorra uma nova interrupção RST 6.5 o programa deve voltar a contar na forma crescente, ou seja, a cada interrupção muda-se o modo de contagem alternando entre crescente e decrescente.
33. Como é feita a comunicação entre o microprocessador 8085 e o controlador de interrupção 8259?
34. Como é feita a transferência por interrupção? E por DMA?
35. Faça um programa no ABACUS usando a linguagem assembly do 8085 que mostre um valor zero no display enquanto aguarda um pedido de interrupção RST 6.5. Ocorrida a interrupção, o programa deve fazer uma contagem decimal crescente no display de zero a 60, sem interrupções. Ao alcançar o valor 60, o programa deve ficar esperando um pedido de interrupção RST 7.5. Ocorrida a interrupção, o programa deve fazer uma contagem decimal decrescente no display até zero. A contagem deve se repetir caso ocorra novamente a interrupção RST 6.5 e, depois, a RST 7.5.
36. Faça o comentário de cada linha do programa a seguir e explique qual a finalidade do programa completo. Complete os endereços.

Rótulo	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	

37. É mostrado a seguir o que representa cada bit do acumulador quando se deseja mascarar interrupções. Escreva as instruções para mascarar as interrupções RST7.5 e RST6.5, em seguida, habilitar a RST5.5.

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
Acumulador	SOD	SOE	×	R7.5	MSE	M7.5'	M6.5'	M5.5'

38. Faça um programa seguindo os passos a seguir:
- Programa principal: Escreva as instruções para habilitar as chaves do ABACUS como entrada de dados e os LEDs como saída de dados, mascarar as interrupções RST6.5 e RST5.5, habilitar a interrupção RST7.5 e ficar aguardando num "loop" um pedido de interrupção;
  - Subrotina: Escreva uma subrotina para rotacionar, de dois em dois (0-1, 1-2, 2-3, ..., 6-7), os LEDs para a esquerda, quando há um pedido da interrupção RST7.5. O processamento volta para o "loop" quando forem alcançados os LEDs 6 e 7.

## 6.8 Referências Bibliográficas

- Ziller, Roberto M., "Microprocessadores – Conceitos Importantes," Editora do autor, Florianópolis, SC, 2000.
- Intel, "MCS-80/85™ Family User's Manual," Outubro de 1979.
- Malvino, Albert P., "Microcomputadores e Microprocessadores," Tradução: Anatólio Laschuk, revisão técnica: Rodrigo Araes Caldas Farias, McGraw-Hill, São Paulo, 1985.

## 7. ENTRADA E SAÍDA SERIAL E PASTILHAS INTEGRADAS DE SUPORTE

A entrada de dados seriais para o 8085 pode ser feita através do pino 5 do microprocessador, que é o pino **SID** (Serial Input Data). Os dados seriais que chegam a esse pino são transferidos para o bit 7 do acumulador a cada execução da instrução RIM.

O pino 4, **SOD** (Serial Output Data) pode ser usado para a transferência serial de dados para um periférico. Nesse caso, a instrução SIM é usada para transferir o conteúdo do bit 7 do acumulador para o pino 4, mas apenas se o bit 6 do acumulador, SOE (Serial Output Enable) estiver setado.

Exemplo de entrada serial para o 8085 usando o pino SID:

Label	Mnemônico	Comentário
	MVI B,00h	Zerar o registrador B
	MVI C,08h	Inicializar o contador com 8
Laço:	RIM	Obter o bit <b>SID</b>
	ANI 80h	Isolar o bit 7 do acumulador (bit SID)
	ORA B	Atualizar o conteúdo do acumulador
	RRC	Rotacionar acumulador à direita
	MOV B,A	Guardar acumulador em B
	DCR C	Decrementar contador C
	JNZ Laço	Retornar para o laço se o contador não for zero
	RLC	Rotacionar acumulador para a esquerda
	HLT	Encerrar

O programa acima repete o *Laço* 8 vezes, recebendo, portanto 8 bits para formar um byte. Esses bits chegam através do pino **SID**, que é transferido para o bit 7 do acumulador. Cada novo bit que é carregado é isolado através da instrução **ANI 80 h**. A seguir, esse novo bit é "juntado" (usando a instrução **ORA B**) aos bits anteriores que já estão guardados em **B**. O acumulador, nesse momento contém todos os bits lidos. A rotação à direita é necessária para receber um novo dado no bit 7. O conteúdo rotacionado do acumulador é guardado em **B**.

Supondo, por exemplo, que os bits 1 1 0 0 0 0 1 e 0 são carregados no 8085 usando o programa anterior, tem-se:

Na primeira execução de MOV B,A o conteúdo de B será 0 1 0 0 0 0 0 0, porque o primeiro dado (1) foi recebido no bit 7 do acumulador, mas foi rotacionado à direita antes da transferência para B.

Após a segunda execução de MOV B,A o conteúdo de B será 0 1 1 0 0 0 0 0.

Após a oitava execução de MOV B,A o conteúdo de B será 1 0 1 0 0 0 0 1. Vê-se aqui que o primeiro dado (1) a ser carregado já voltou para o bit 7 de B, sendo, portanto necessário fazer uma rotação à esquerda para encerrar o primeiro byte de entrada. Assim, após sair do Laço, e após a instrução RLC, o conteúdo do acumulador (mas não o de B) conterá o byte de entrada, que é: 0 1 0 0 0 0 1 1. A leitura foi feita do bit menos significativo para o bit mais significativo.

### 7.1 Interface Serial no PC

Os microcomputadores atuais utilizam uma pastilha especial para a comunicação serial. Os primeiros PCs usavam o **CI 8250:UART** (Universal Asynchronous Receiver-Transmitter), que permitia o armazenamento de apenas um byte por vez, limitando a velocidade de comunicação serial. Atualmente o CI utilizado é o **16550**, que permite o armazenamento simultâneo de até 16 bytes de informação. A seguir são dadas as características da Interface Serial no PC:

- Possui 2 interfaces seriais;
- São interfaces assíncronas;
- São baseadas no CI 16550 **UART** (Universal Asynchronous Receiver/Transmitter); Este CI é um upgrade do **CI 8250** da National utilizado PC original (com taxa de transmissão máxima de 9.600bit/s);
- Características do CI 16550 UART:
  - ⇒ Taxa de transmissão de até 115.200bit/s;

- ⇒ FIFO de 16 bytes; Esta FIFO armazena dados que foram recebidos pela interface serial mas que não foram lidos pelo microprocessador; evita erro de *overrun* (perda de informação por sobreescrita);
- ⇒ Função Loopback (diagnóstico);
- ⇒ Funções para controle de modem.

A placa mãe de um microcomputador Pentium possui 2 interfaces seriais: cada uma delas pode ser configurada de duas formas:

- Interface 1: COM1 (IRQ4) ou COM3 (IRQ4);
- Interface 2: COM2 (IRQ3) ou COM4 (IRQ4).

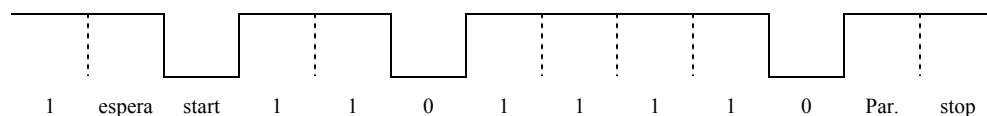
### Transmissão Serial Assíncrona:

Processo de comunicação no qual os caracteres são enviados independentemente uns dos outros.

Uma das aplicações mais comuns deste tipo de comunicação é na interface entre teclado e computador. Neste circuito cada tecla gera um código ASCII de 7 bits que é transmitido bit a bit para o computador por intermédio de dois ou três condutores. Para verificar a eficiência deste tipo de transmissão serial deve-se observar o desempenho de um bom digitador: sabe-se que ele não supera uma taxa de 60 palavras por minuto (cerca de 8 caracteres/seg). Logo a transmissão serial, relativamente lenta, é aceita neste caso.

Observe uma característica importante nesta interface: em alguns momentos a porta serial será utilizada para transferir dados a uma taxa de 10 a 20 caracteres/seg, em outros esta pode cair para 1 ou 2 caracteres/seg, e ainda, em grande parte do tempo, o teclado não é utilizado chegando a uma taxa de zero caracteres/seg. Por causa deste tipo de característica de transmissão de dados, um protocolo de comunicação assíncrono deve ser estabelecido entre estes dispositivos.

A técnica aceita para comunicação serial assíncrona é a de segurar a linha de saída da transmissão serial no nível lógico 1 (marca) até que chegue dados para transmissão. Cada caractere será iniciado com um nível lógico 0 (start bit). Este 1º bit, o start bit, é utilizado para sincronizar a transmissão e a recepção. O dado é transmitido colocando-se o bit menos significativo em primeiro lugar após o start bit. No final do dado são transmitidos stop bits (1, 1 1/2 ou 2) com nível lógico 1. Opcionalmente pode-se transmitir um bit de paridade após o MSB do caractere e antes do stop bit (ver figura abaixo: esta figura considera apenas 1 stop bit).



Os bits de start e de stop não carregam informações, e são necessários pela natureza da comunicação serial assíncrona. Para se configurar uma porta serial vários parâmetros devem ser especificados. Os parâmetros mais comuns são:

- tamanho do caractere ou bits por caractere (usualmente de 5 a 8 bits);
- stop bits (1, 1 1/2 ou 2);
- bit de paridade, usado para detectar erros simples: pode ser especificado como ímpar, par ou nenhum;
- taxa de transmissão, etc.

Uma das considerações que devem ser feitas ao se utilizar a comunicação serial entre periféricos e computador (ou entre dois computadores) é a seleção de parâmetros de comunicação compatíveis. Um dos problemas mais comuns é o de configuração de taxas de transmissão distintas entre o transmissor e o receptor.

### Transmissão Serial Síncrona:

Processo de comunicação no qual o transmissor e o receptor estão sincronizados pelo mesmo sinal de clock. O transmissor envia continuamente caracteres de sincronismo sempre que não existir dado a ser transmitido.

Na transmissão serial assíncrona os bits de start e stop representam um grande *overhead* nos dados transmitidos, reduzindo a taxa de transferência de informações. Na transmissão serial síncrona também



## 7.2 Integrados de Suporte

### 7.2.1 Memórias RAM e ROM

Como já foi enfatizado anteriormente, o microprocessador 8085 precisa de alguns componentes auxiliares que, além de conterem memórias ROM e RAM, possibilitem a comunicação do 8085 com ambiente externo. Dois desses componentes são os Circuitos Integrados 8156 e 8355.

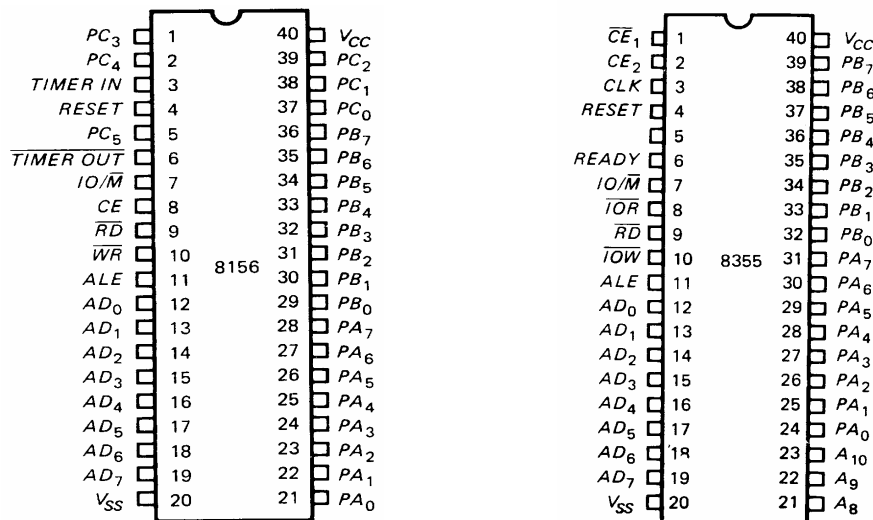


Fig. 7.1: Pastilhas de memória ROM e RAM

CI	Tipo de Memória e Capacidade	Número de Portas de Entrada e Saída	Características
8156	RAM 256 bytes	3	2048 bits organizados como 256 palavras de 8 bits cada
8355	ROM 2 Kbytes	2	Possui 16384 bits organizados como 2048 palavras de 8 bits cada

A Fig. 6.2 mostra um sistema mínimo baseado no 8085 e contendo uma pastilha de memória RAM (8156) e uma de memória ROM (8355).

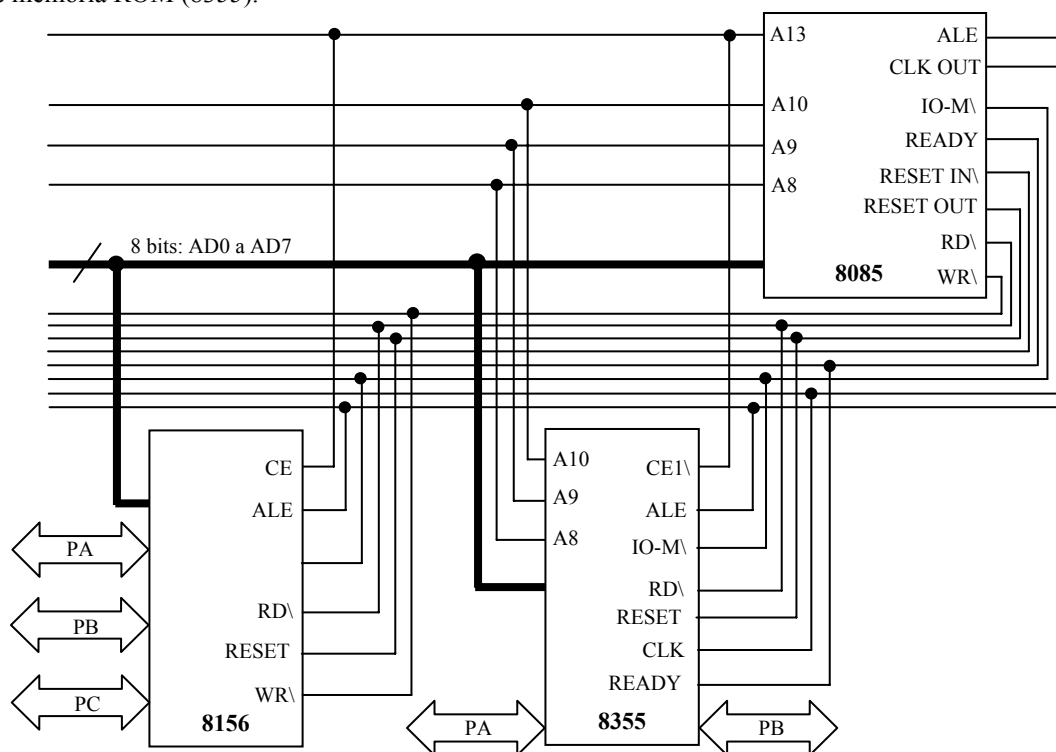


Fig. 7.2: Sistema mínimo baseado no microprocessador 8085



Pino **A<sub>13</sub>** – Pino usado para habilitar/desabilitar a RAM e a ROM.

<b>A<sub>13</sub></b>	<b>CE</b>	<b>CE\</b>	<b>Pastilha Habilitada</b>
0	0	0	CI 8156
1	1	1	CI 8355

No sistema mínimo mostrado na Fig. 7.2 o pino de endereço **A<sub>13</sub>** do 8085 é usado para habilitar/desabilitar a RAM e a ROM. Ele é conectado ao pino **CE** (Chip Enable) da RAM e ao pino **CE\** da ROM. Isso significa que, quando o pino **A<sub>13</sub>** está em nível baixo, a ROM fica habilitada e a RAM desabilitada. Quando em nível alto a ROM fica desabilitada e RAM fica habilitada. Dessa forma, o endereço inicial da ROM é 0000 h e o da RAM é 2000 h, como demonstrado abaixo.

A <sub>15</sub>	A <sub>14</sub>	A <sub>13</sub>	A <sub>12</sub>	A <sub>11</sub>	A <sub>10</sub>	A <sub>9</sub>	A <sub>8</sub>	AD <sub>7</sub>	AD <sub>6</sub>	AD <sub>5</sub>	AD <sub>4</sub>	AD <sub>3</sub>	AD <sub>2</sub>	AD <sub>1</sub>	AD <sub>0</sub>
Endereço Inicial da <b>ROM</b>															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0				0				0				0			
Endereço Final da <b>ROM</b>															
0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1
0				7				F				F			
Endereço Inicial da <b>RAM</b>															
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
2				0				0				0			
Endereço Final da <b>RAM</b>															
0	0	1	0	0	0	0	0	1	1	1	1	1	1	1	1
2				0				F				F			

Como a ROM 8355 tem 2048 bytes (800 h) de memória, as posições ocupadas por estes bytes vão de 0 a 2047, que em hexadecimal é 0000 h até 07FF h. O número de linhas de endereço necessárias são 11 (de AD<sub>0</sub> a A<sub>10</sub>), porque  $2^{11} = 2048$ .

Como a RAM 8156 tem apenas 256 bytes de memória, os endereços desses bytes vão de 0 a 255 (00 h a FF h). Daí, como o endereço inicial escolhido para a memória RAM é 2000 h na conexão do sistema mínimo dado, o endereço final será 20FF h. O número de linhas de endereço necessárias são 8 (de AD<sub>0</sub> a AD<sub>7</sub>), porque  $2^8 = 256$ .

A memória RAM do Kit didático usado no laboratório é a RAM 2114, de 1 K de memória, mas de apenas 4 bits. Assim, utiliza-se duas-a-duas para forma 1 byte por endereço. No kit o endereço inicial da RAM é 4000 h. São necessárias 10 linhas de endereço:  $2^{10} = 1024$  (= 400h). Posições de memória: 0 a 1023 (000 h a 3FF h)

A <sub>15</sub>	A <sub>14</sub>	A <sub>13</sub>	A <sub>12</sub>	A <sub>11</sub>	A <sub>10</sub>	A <sub>9</sub>	A <sub>8</sub>	AD <sub>7</sub>	AD <sub>6</sub>	AD <sub>5</sub>	AD <sub>4</sub>	AD <sub>3</sub>	AD <sub>2</sub>	AD <sub>1</sub>	AD <sub>0</sub>
Endereço Inicial da primeira <b>RAM 2114</b> do Kit Didático															
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4				0				0				0			
Endereço Final da primeira <b>RAM 2114</b> do Kit Didático															
0	1	0	0	0	0	1	1	1	1	1	1	1	1	1	1
4				3				F				F			

A Segunda RAM 2114 do Kit didático deve ser acessada a partir do primeiro endereço após a primeira RAM, ou seja, endereço 4400 h.

A <sub>15</sub>	A <sub>14</sub>	A <sub>13</sub>	A <sub>12</sub>	A <sub>11</sub>	A <sub>10</sub>	A <sub>9</sub>	A <sub>8</sub>	AD <sub>7</sub>	AD <sub>6</sub>	AD <sub>5</sub>	AD <sub>4</sub>	AD <sub>3</sub>	AD <sub>2</sub>	AD <sub>1</sub>	AD <sub>0</sub>
Endereço Inicial da SEGUNDA RAM 2114 do Kit Didático															
0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0
4				4				0				0			
Endereço Final da SEGUNDA RAM 2114 do Kit Didático															
0	1	0	0	0	1	1	1	1	1	1	1	1	1	1	1
4				7				F				F			

A Terceira RAM 2114 do Kit didático deve ser acessada a partir do primeiro endereço após a segunda RAM, ou seja, endereço 4800 h.

A <sub>15</sub>	A <sub>14</sub>	A <sub>13</sub>	A <sub>12</sub>	A <sub>11</sub>	A <sub>10</sub>	A <sub>9</sub>	A <sub>8</sub>	AD <sub>7</sub>	AD <sub>6</sub>	AD <sub>5</sub>	AD <sub>4</sub>	AD <sub>3</sub>	AD <sub>2</sub>	AD <sub>1</sub>	AD <sub>0</sub>
Endereço Inicial da SEGUNDA <b>RAM 2114</b> do Kit Didático															
0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0
4				8				0				0			
Endereço Final da SEGUNDA <b>RAM 2114</b> do Kit Didático															
0	1	0	0	1	0	1	1	1	1	1	1	1	1	1	1
4				B				F				F			

Dessa forma, um circuito lógico deve ser usado quando há um conjunto de pastilhas de memória no sistema mínimo. A Fig. 7.3 mostra o circuito lógico para conectar três conjuntos de RAMs.

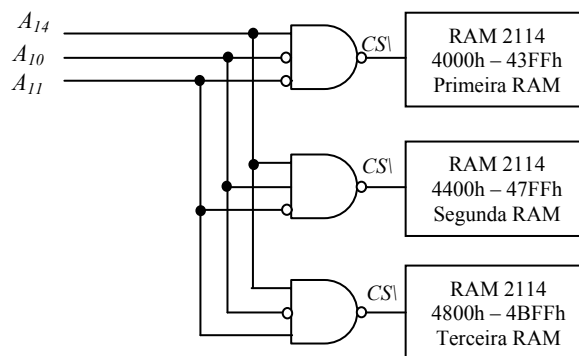


Fig. 7.3: Circuito lógico para conexão de 3 conjuntos de RAMs 2114 ao sistema mínimo

Desejando-se conectar um quarto conjunto de RAMs, tem-se o circuito lógico da Fig. 7.4, a seguir.

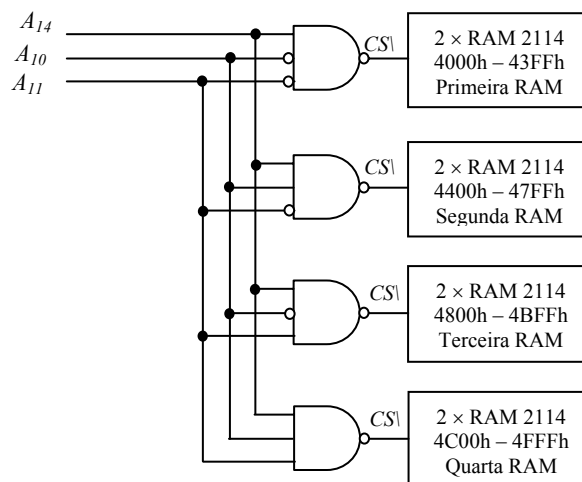
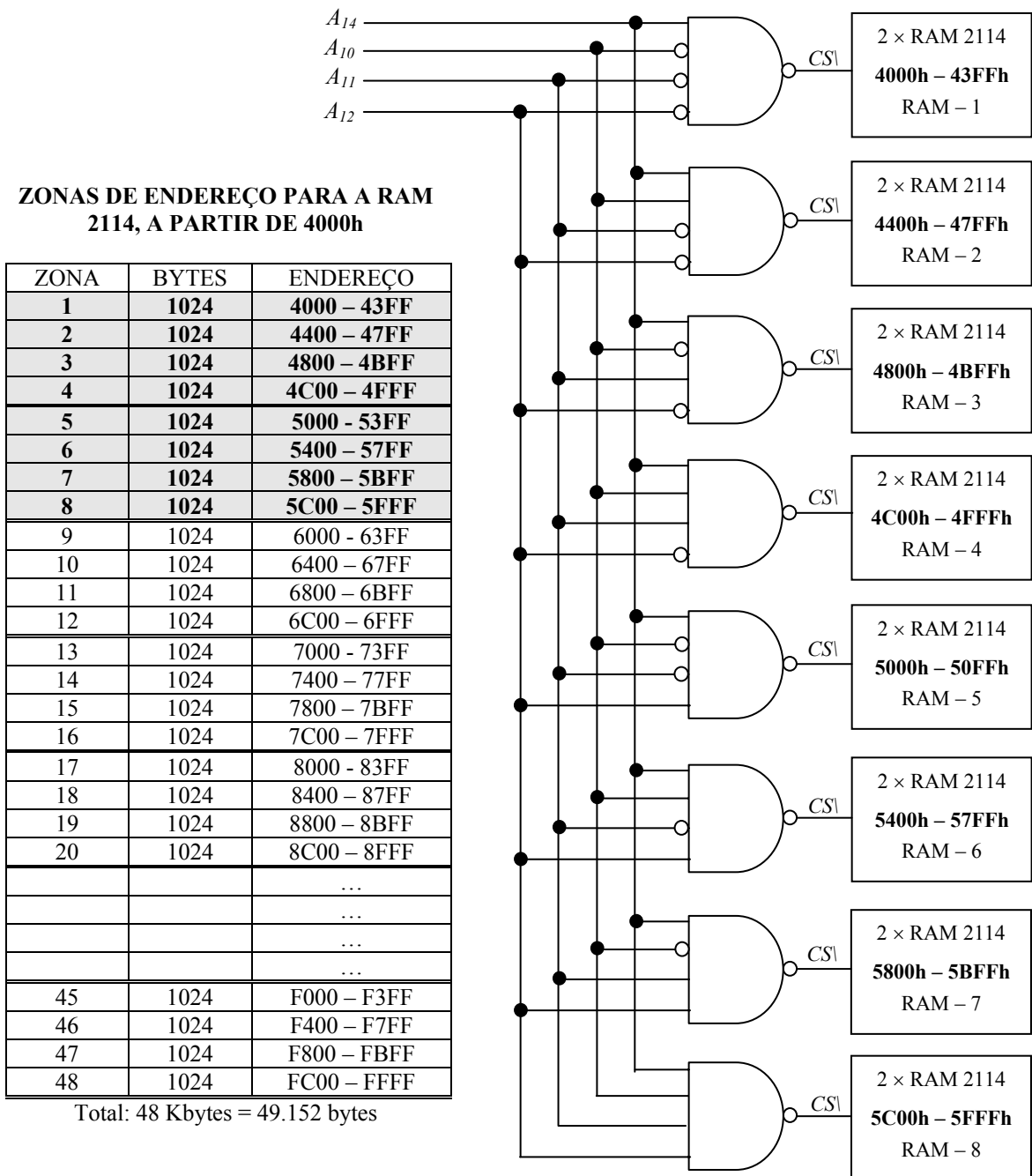


Fig. 7.4: Circuito lógico para conexão de 4 conjuntos de RAMs 2114 ao sistema mínimo

A Quarta RAM 2114 é acessada a partir do endereço 4C00h

A <sub>15</sub>	A <sub>14</sub>	A <sub>13</sub>	A <sub>12</sub>	A <sub>11</sub>	A <sub>10</sub>	A <sub>9</sub>	A <sub>8</sub>	AD <sub>7</sub>	AD <sub>6</sub>	AD <sub>5</sub>	AD <sub>4</sub>	AD <sub>3</sub>	AD <sub>2</sub>	AD <sub>1</sub>	AD <sub>0</sub>
Endereço Inicial da QUARTA <b>RAM 2114</b> do Kit Didático															
0	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0
4				C				0				0			
Endereço Final da QUARTA <b>RAM 2114</b> do Kit Didático															
0	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1
4				F				F				F			

Desejando-se acrescentar mais 4 conjuntos de RAMs, de forma a ter um sistema mínimo com 8 Kbytes de memória, precisa-se de mais um pino de seleção, que no caso é o pino  $A_{12}$ .



$A_{15}$	$A_{14}$	$A_{13}$	$A_{12}$	$A_{11}$	$A_{10}$	$A_9$	$A_8$	$AD_7$	$AD_6$	$AD_5$	$AD_4$	$AD_3$	$AD_2$	$AD_1$	$AD_0$
----------	----------	----------	----------	----------	----------	-------	-------	--------	--------	--------	--------	--------	--------	--------	--------

Fig. 7.5: Circuito lógico para conexão de 8 conjuntos de RAMs 2114 ao sistema mínimo

As pastilhas 8156 e 8355 não são apenas memórias RAM e ROM; elas também contêm portas de entrada e saída paralela, além de temporizadores.

As três portas de Entrada e Saída da 8156 são: Porta A (8 bits), Porta B (8 bits) e Porta C (6 bits), que são numeradas respectivamente como 21h, 22h e 23h no sistema mínimo da Fig. 7.2. Internamente, tem-se ainda acesso a três outras portas: Porta 20h, usada como registrador de status e de comando, Porta 24h, usada como byte inferior do contador de temporização e Porta 25h, usada como modo e contador (6 bits superiores) do temporizador.

O endereçamento das portas da 8156 são dados na tabela a seguir:

AD <sub>2</sub>	AD <sub>1</sub>	AD <sub>0</sub>	Seleção
0	0	0	Registadores de comando e de status
0	0	1	Porta A
0	1	0	Porta B
0	1	1	Porta C
1	0	0	8 bits menos significativos do temporizador
1	0	1	2 bits de modo do temporizador e 6 bits superiores do temporizador

As duas portas de Entrada e Saída do 8355 são: Porta A (8 bits) e Porta B (8 bits), que são numeradas respectivamente como 00h e 01h no sistema mínimo da Fig. 7.2. O endereçamento é dado na tabela a seguir:

AD <sub>1</sub>	AD <sub>0</sub>	Seleção	Número da Porta
0	0	Porta A	00h
0	1	Porta B	01h
1	0	Registrador de Direção de Dados da Porta A (DDR A)	02h
1	1	Registrador de Direção de Dados da Porta A (DDR A)	03h

O temporizador da 8156 possui é decrementador pré-setável, controlado pelas portas 20h, 24h e 25h. O contador pode operar em 4 modos, mostrados nas figuras a seguir:

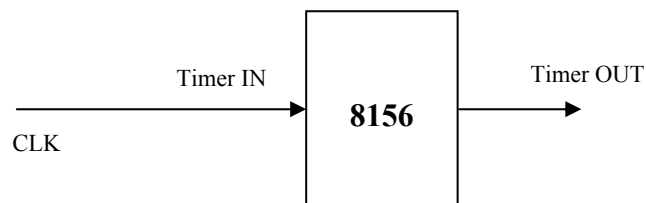


Fig. 7.6: Temporizador do CI 8156

Comando	D7	D6	D5	D4	D3	D2	D1	D0
	Temporizador		Habilitação de interrupção B	Habilitação de interrupção A	Porta C		Porta B	Porta A

Porta 20h - Registro de comando para as portas A, B e C do 8156 e para o temporizador

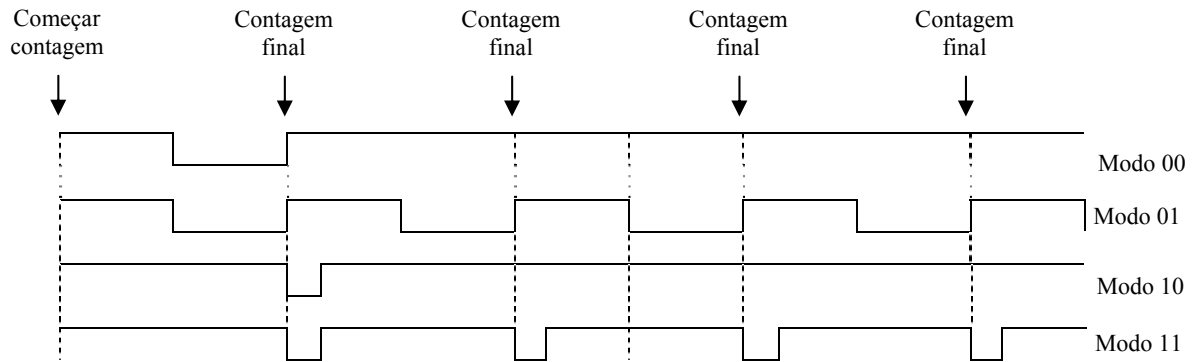
D <sub>7</sub>	D <sub>6</sub>	D <sub>5</sub>	D <sub>4</sub>	D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>
Temp	Temp	Int. B	Int. A	C	C	B	A

Porta 24h - Byte menos significativo do Temporizador

T <sub>7</sub>	T <sub>6</sub>	T <sub>5</sub>	T <sub>4</sub>	T <sub>3</sub>	T <sub>2</sub>	T <sub>1</sub>	T <sub>0</sub>
----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------

Porta 25h - Byte mais significativo do Temporizador e modo do temporizador

M <sub>2</sub>	M <sub>1</sub>	T <sub>13</sub>	T <sub>12</sub>	T <sub>11</sub>	T <sub>10</sub>	T <sub>9</sub>	T <sub>8</sub>
----------------	----------------	-----------------	-----------------	-----------------	-----------------	----------------	----------------



<b>M<sub>2</sub></b>	<b>M<sub>1</sub></b>	Onda de saída no pino Timer OUT
0	0	Onda quadrada simples (um ciclo)
0	1	Onda quadrada contínua
1	0	Pulso simples (único)
1	1	Pulso contínuo

<b>D<sub>7</sub></b>	<b>D<sub>6</sub></b>	Estado do Temporizador
0	0	Não opera
0	1	Parar imediatamente
1	0	Parar depois de atingir contagem final
1	1	Começar contagem

Exemplo: Gerar uma onda quadrada de 1 kHz, sendo a frequência de clock de 3 MHz

$$\text{Contagem final} = \frac{3\text{MHz}}{1\text{kHz}} = 3000 = 0\text{BB8 h}$$

Porta 24h - LSB: B8h

1	0	1	1	1	0	0	0
---	---	---	---	---	---	---	---

Porta 25h - MSB e Temporizador: 01 + B8h

0	1	0	0	1	0	1	1
---	---	---	---	---	---	---	---

Onda quadrada contínua

Porta 20h - Registro de comando para as portas A, B e C do 8156 e para o temporizador

1	1	0	0	1	1	1	0
---	---	---	---	---	---	---	---

Int. A e B des.

C saída

B saída

A Ent.

Começar contagem do temporizador

Instruções	Comentário
MVI A,B8h	Onda quadrada contínua de 1kHz
OUT 24h	
MVI A,4Bh	
OUT 25h	
MVI A,Ceh	Começa contagem e desabilita interrupções de A e B
OUT 20h	

## 7.2.2 Decodificador 74LS138

O CI 74LS138 é um decodificador usado tanto para expansão de memória quanto para o endereçamento de muitas portas de entrada e saída. É um decodificador 1 de 8, ou seja, cada CI permite a seleção de 8 saídas diferentes. Três pinos (C, B e A) são usados para selecionar uma das 8 linhas de saída ( $Y_0$  a  $Y_7$ ).

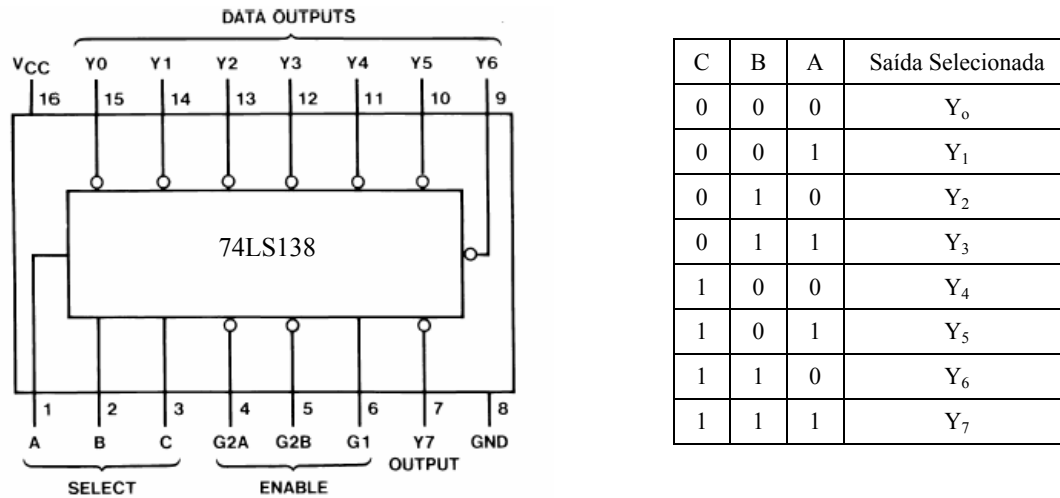


Fig. 7.7: Decodificador 74LS138 e tabela verdade

Inputs					Outputs							
Enable		Select										
G1	G2 (Note 8)	C	B	A	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7
X	H	X	X	X	H	H	H	H	H	H	H	H
L	X	X	X	X	H	H	H	H	H	H	H	H
H	L	L	L	L	L	H	H	H	H	H	H	H
H	L	L	L	H	H	L	H	H	H	H	H	H
H	L	L	H	L	H	H	L	H	H	H	H	H
H	L	L	H	H	H	H	L	H	H	H	H	H
H	L	H	L	L	H	H	H	H	L	H	H	H
H	L	H	L	H	H	H	H	H	L	L	H	H
H	L	H	H	L	H	H	H	H	H	L	L	H
H	L	H	H	H	H	H	H	H	H	H	L	H

H = High Level, L = Low Level, X = Don't Care

Note 8:  $G_2 = G_{2A} + G_{2B}$

A saída está ativa quando está em nível zero. A pastilha 74LS138 é habilitada fazendo  $G_1 = 1$ ,  $G_{2A} = 0$ ,  $G_{2B} = 0$ . Quando qualquer desses pinos está desativado o CI está desabilitado.

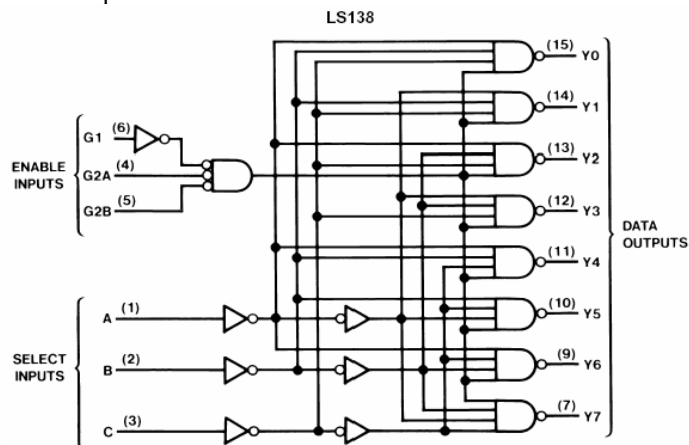


Fig. 7.8: Diagrama de blocos do decodificador 74LS138

Voltando às 8 pastilhas de memória RAM do exercício anterior, pode-se fazer o endereçamento usando um decodificador 74LS138. O pino  $A_{14}$  é usado para habilitação geral (G1) e os pinos  $A_{10}$ ,  $A_{11}$  e  $A_{12}$  são usados para seleção da pastilha correspondente a cada faixa de endereço.

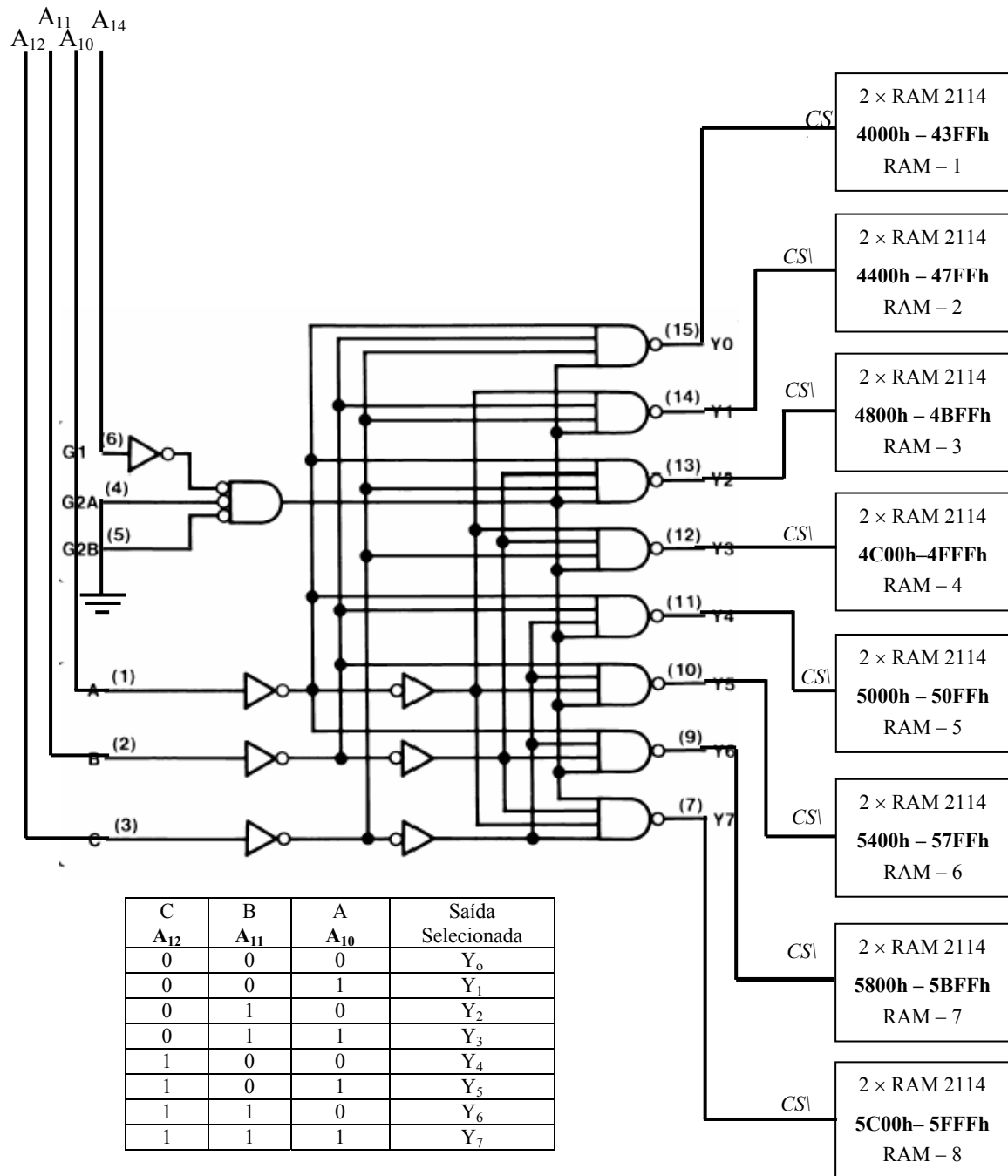


Fig. 7.9: Uso do decodificador 74LS138 para conexão de 8 RAMs ao sistema mínimo

Cada 74LS138 pode endereçar 8 zonas de endereço  $\Rightarrow$  São necessários 6 decodificadores 74LS138 para cobrir toda a memória possível a partir da posição 4000 h até FFFF h.

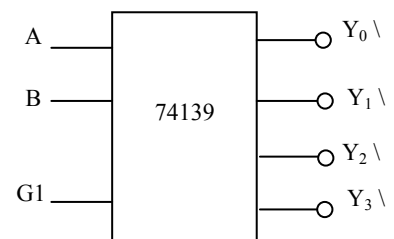
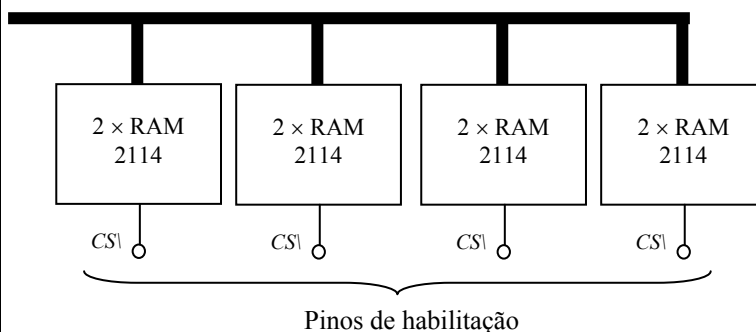
### 7.3 Exercícios Propostos

1. Considere um sistema mínimo composto por um microprocessador 8085, uma memória ROM de 2 K e uma memória RAM de 4 K. O pino CE da RAM e  $CE\backslash$  da ROM estão conectados ao pino  $A_{14}$  do 8085. Responda as questões abaixo:

(a) Quais os endereços inicial e final da ROM e da RAM?

- (b) Qual a finalidade dos pinos CE, WR e IO/M de uma memória RAM?
- (c) Quantas linhas de endereço são necessárias para a ROM e para a RAM?
- (d) Conceitue ROM, EPROM, RAM estática e RAM dinâmica
2. Gerar um pulso contínuo na saída do temporizador da pastilha 8156, em uma frequência de 1 kHz, sabendo que a frequência de clock é de 1 MHz. Dê uma sugestão de como o temporizador pode ser usado para a geração de sinal PWM (Modulação de Largura de Pulso).
  3. Explicar para que serve e como é utilizado o decodificador 74139 num sistema com o microprocessador 8085. Esse decodificador tem duas entradas (A e B) e quatro saídas ( $\bar{Y}_0$ ,  $\bar{Y}_1$ ,  $\bar{Y}_2$ ,  $\bar{Y}_3$ ). Os habilitadores do CI 74139 são  $G_1$ ,  $\bar{G}_2$  e  $\bar{G}_3$ , como no caso do CI 74138. Faça um esquema mostrando uma aplicação.
  4. Desenhar o circuito de conexão de 4 pastilhas de memória RAM, cada uma de 2 kbytes, a partir do endereço 1000 h, usando o decodificador 74139. Esse decodificador tem duas entradas (A e B) e quatro saídas ( $\bar{Y}_0$ ,  $\bar{Y}_1$ ,  $\bar{Y}_2$  e  $\bar{Y}_3$ ). O habilitador da memória RAM é o pino CE. Os habilitadores do CI 74139 são  $G_1$ ,  $\bar{G}_2$  e  $\bar{G}_3$ , como no caso do CI 74138. Inclua as linhas de endereços / dados.
  5. Faça um programa que leia um byte da porta serial e o mostre no display. A cada interrupção RST 7.5 o programa deverá ler outro byte da porta serial e mostrá-lo no display.
  6. Faça um programa que leia um dado do teclado, e então, envie este byte pela saída serial por 5 vezes seguidas, com um intervalo de 1s entre cada repetição (destas 5 vezes). Em outras palavras, enviar o byte 5 vezes pela porta serial, parar por 1 segundo e tornar a enviar o byte por mais 5 vezes, parar por mais 1s... e assim por diante. O bit menos significativo deve ser enviado primeiramente.
  7. Faça um programa que leia dois números pelo teclado, *some* estes números e mostre o resultado no display, repetindo esta operação indefinidamente. Caso ocorra uma interrupção RST 6,5, o programa passará a *subtrair* esses números ( $1^o - 2^o$ ), mostrando o resultado no display. Após uma nova interrupção RST 6,5 o programa volta a somar os dois números, e assim sucessivamente. Caso ocorra uma interrupção RST 5,5 o último resultado mostrado no display deverá ser enviado pelo pino SOD, uma única vez, após o que o processamento volta ao programa principal.
  8. A RAM 2114 é de 1 K, mas possui apenas 4 bits cada uma. Por isso são usadas de duas em duas de modo a formar 1 byte para cada endereço. Deseja-se associar 4 grupos de RAMs 2114 (ver figura a seguir), de modo a formar uma memória RAM de 4 Kbytes. Pede-se:
    - (a) Quais são as linhas de endereço para a operação das RAMs 2114?
    - (b) Qual a faixa de endereços (endereços contínuos) de cada grupo de RAMs?
    - (c) Use portas lógicas para desenhar o circuito de habilitação dos grupos de RAMs a partir do endereço 1000h;
    - (d) Use o decodificador 74139 (figura a seguir) para a habilitação dos grupos de RAMs a partir do endereço 1000h. Os pinos A e B são usados para selecionar a saída e o pino  $G_1$  é usado para habilitar a pastilha.

Linhas de endereço: \_\_\_\_\_



A	B	Y
0	0	$Y_0\backslash$
0	1	$Y_1\backslash$
1	0	$Y_2\backslash$
1	1	$Y_3\backslash$



9. A RAM 2114 é de 1 K, mas possui apenas 4 bits cada uma. Por isso são usadas de duas em duas de modo a formar 1 byte para cada endereço . Deseja-se associar 4 grupos de RAMs 2114 (ver figura abaixo), de modo a formar uma memória RAM de 4 Kbytes. Pede-se:
- (e) Quais são as linhas de endereço para a operação das RAMs 2114?
  - (f) Qual a faixa de endereços (endereços contínuos) de cada grupo de RAMs?
10. Faça um programa para o ABACUS com a seguinte característica: Quando a chave CH0 OU a chave CH1 está ligada os LEDs ficam piscando em intervalos regulares. Quando estão desligadas, o programa fica num loop de espera.

#### **7.4 Referências Bibliográficas**

- [1] Ziller, Roberto M., “Microprocessadores – Conceitos Importantes,” Editora do autor, Florianópolis, SC, 2000.
- [3] Malvino, Albert P., “Microcomputadores e Microprocessadores,” Tradução: Anatólio Laschuk, revisão técnica: Rodrigo Araes Caldas Farias, McGraw-Hill, São Paulo, 1985.

## 8. INTERFACE ANALÓGICA

Como os dados de um microprocessador estão na forma digital e os dados do mundo exterior estão na forma analógica (contínua), é necessário fazer a conversão entre esses dados. Assim, tem-se o Conversor Analógico-Digital (ADC), que faz a conversão de sinal analógico para sinal digital e o Conversor Digital-Analógico (DAC), que faz a conversão de sinal digital para sinal analógico.

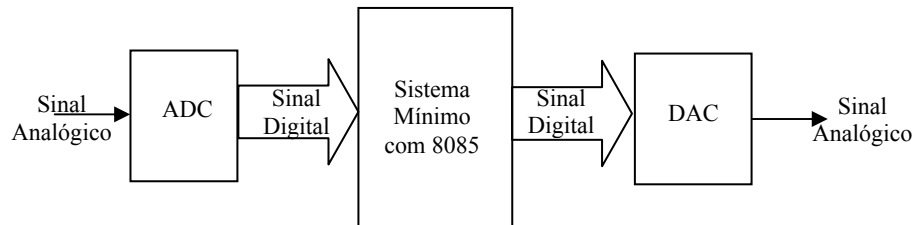


Fig. 8.1: Processamento de sinal analógico pelo Microprocessador

### 8.1 Conversor Digital-Analógico

Um circuito somador com Amplificador Operacional pode ser usado para construir um conversor digital-analógico, bastando para isso escolher valores ponderados dos resistores, como na figura 8.2 a seguir. A figura mostra um conversor DA de 4 bits.

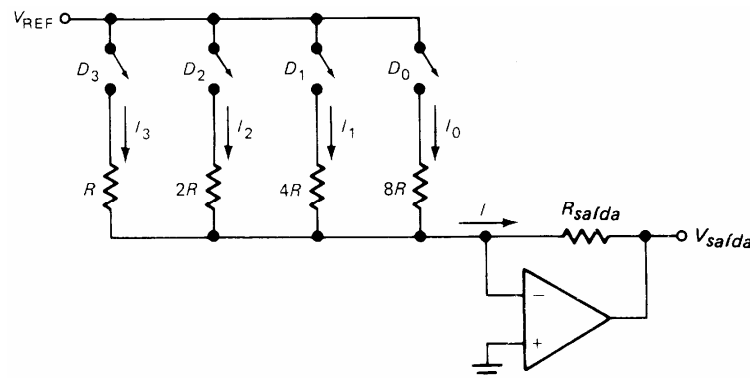


Fig. 8.2: Circuito Básico de um Conversor Analógico-Digital

Estas chaves podem ser substituídas por transistores que trabalharão na região de corte quando  $D = 0$ , na região de saturação quando  $D = 1$ .

Correntes nas resistências quando as chaves  $D_3$ ,  $D_2$ ,  $D_1$  e  $D_0$  estão ligadas:

$$I_3 = \frac{V_{REF}}{R} \quad I_2 = \frac{V_{REF}}{2R} \quad I_1 = \frac{V_{REF}}{4R} \quad I_0 = \frac{V_{REF}}{8R}$$

A corrente total na resistência de saída  $R_{saida}$  é  $I = I_3 + I_2 + I_1 + I_0$ , ou seja,

$$I = \frac{V_{REF}}{R} (1 * D_3 + 0.5 * D_2 + 0.25 * D_1 + 0.125 * D_0),$$

onde os valores de  $D$  podem ser 0 para chave desligada e 1 para chave ligada. Assim, forma-se a tabela abaixo, supondo  $V_{REF}/R = 1$ :

Tabela de valores para o conversor de 4 bits

D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>	Corrente de saída (mA)	Fração do Máximo
0	0	0	0	0	0
0	0	0	1	0.125	1/15
0	0	1	0	0.250	2/15
0	0	1	1	0.375	3/15
0	1	0	0	0.500	4/15
0	1	0	1	0.625	5/15
0	1	1	0	0.750	6/15
0	1	1	1	0.875	7/15
1	0	0	0	1.000	8/15
1	0	0	1	1.125	9/15
1	0	1	0	1.250	10/15
1	0	1	1	1.375	11/15
1	1	0	0	1.500	12/15
1	1	0	1	1.625	13/15
1	1	1	0	1.750	14/15
1	1	1	1	1.875	15/15

Um vez que o conversor tem apenas 4 bits, existem 16 ( $2^4$ ) possíveis valores de saída, sendo o menor valor 0 (zero) e o valor máximo correspondente ao decimal 15, ou binário 1111, e hexadecimal 'F'. A tensão de saída é dada por  $V_{saída} = R_{saída} * I$ . Os resistores ponderados são necessários para que a saída corresponda à fração indicada na última coluna da tabela.

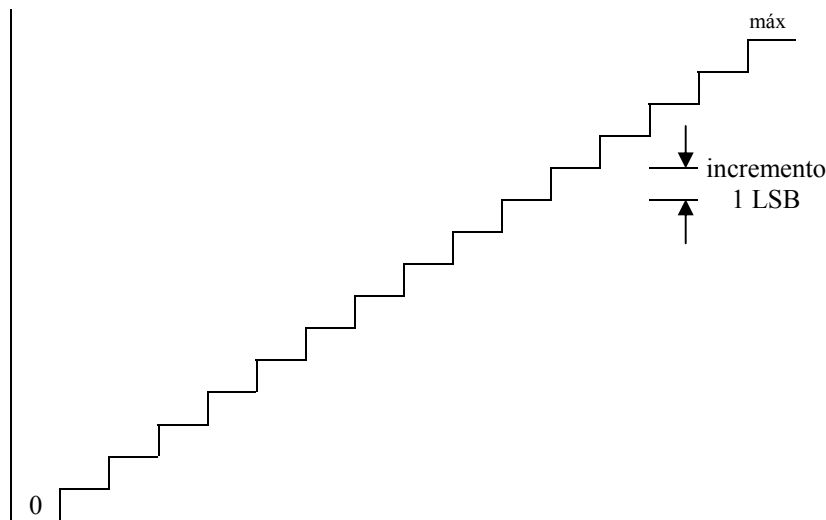


Fig. 8.3: Forma da corrente de saída para uma variação de corrente de 0 a 15

Características de um Conversor DA:

- (a) Cada degrau corresponde a 1 LSB (bit menos significativo)
- (b) Número de degraus de um conversor:  $2^n - 1$ , onde  $n$  é o número de bits. Para  $n = 4 \Rightarrow 15$  degraus.

1. Resolução: É a relação entre o menor incremento possível, 1 LSB e a saída máxima.

$$\text{Resolução} = \frac{1}{2^n - 1}, \quad n \text{ é número de bits. No caso de } n = 4 \Rightarrow \text{Resolução} = \frac{1}{15}$$

Resolução Percentual = Resolução x 100%

No caso de 4 bits  $\Rightarrow$  Resolução Percentual = 6.67%

Resolução do DA *versus* Número de bits

Número de bits	Resolução	Resolução Percentual (%)
4	1/15	6.67
8	1/255	0.392
12	1/4095	0.0244
16	1/65535	0.000381

2. **Precisão Absoluta:** Refere-se a quão próxima cada corrente de saída está de seu valor ideal. A precisão depende da tolerância dos resistores, do descasamento dos transistores e da tensão de referência.
3. **Precisão Relativa:** Refere-se a quão próximo cada nível de saída está de sua fração ideal de saída total. A precisão relativa depende principalmente da tolerância dos resistores ponderados. Se eles forem exatamente iguais a  $R$ ,  $2R$ ,  $4R$  e  $8R$  no caso do conversor de 4 bits, todos os degraus serão iguais a 1 incremento LSB. Se os resistores não estiverem corretos os degraus poderão ser maiores ou menores que 1 incremento LSB.
4. **Monotonicidade:** Um conversor DA monotônico é aquele que produz um aumento na corrente de saída para cada entrada digital sucessiva, ou seja, cada aumento no sinal de entrada produz um aumento no sinal de saída. Se os resistores ponderados não estiverem corretos, pode-se ter um conversor não monotônico. O erro máximo da saída do conversor deve ser de  $\pm 1/2$  LSB para garantir que o conversor seja monotônico.
5. **Tempo de Resolução (ou de Posicionamento):** Tempo que a saída do conversor leva para se estabilizar dentro de  $1/2$  LSB de seu valor final. Esse tempo depende, dentre outros fatores, das capacitâncias espúrias e do tempo de retardo de saturação dos transistores.

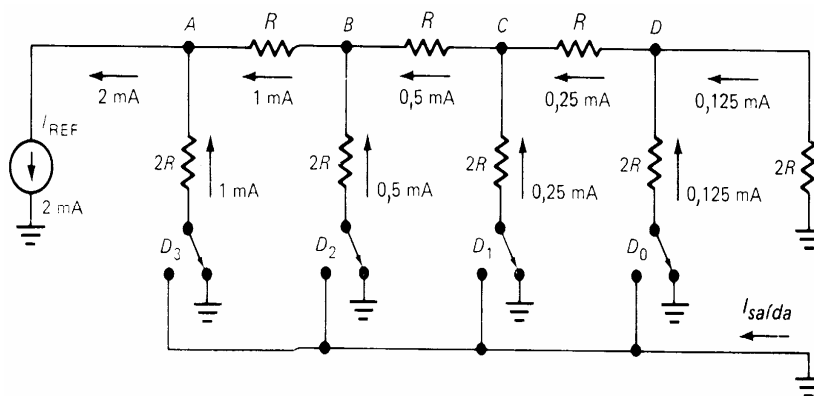


Fig. 8.4: Conversor Digital-Analógico em escada

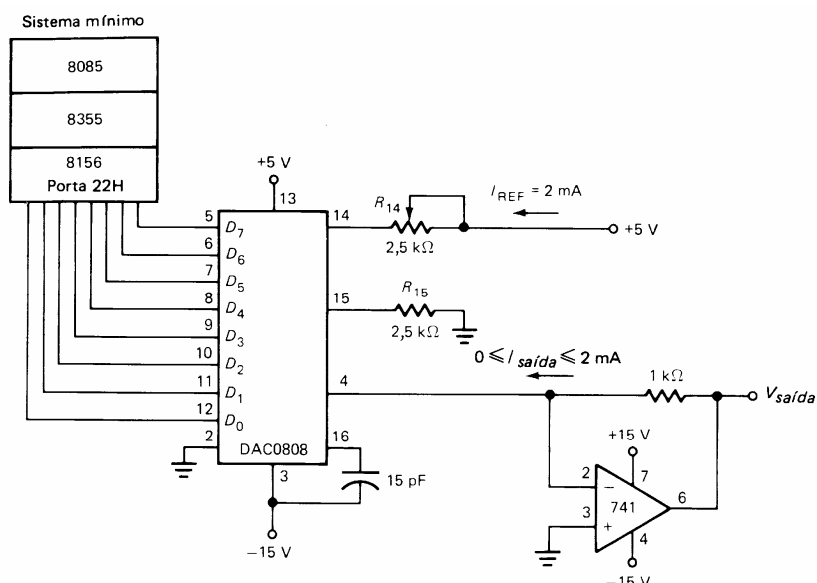


Fig. 8.5: Conexão do conversor DAC 0808 com um sistema mínimo

Para que o conversor seja monotônico a tolerância dos resistores ponderados deve ser inferior à resolução percentual. Assim, a tolerância dos resistores deve ser no máximo de 6.67% para o conversor de 4 bits e de 0.4% para o conversor de 8 bits. Portanto, quanto maior o número de bits do conversor, maior a dificuldade de construção usando o modelo de resistores ponderados, além da dificuldade com os diferentes valores. A solução encontrada foi o modelo da Escada  $R-2R$ , mostrado na Fig. 8.4. Nesse circuito a corrente permanece constante em cada ramo. O que muda é a posição do ponto de terra, onde o terra da corrente de saída é um terra virtual do amplificador operacional. Dessa forma, a corrente de saída varia de acordo com o fechamento das chaves  $D$ .

A Fig. 8.5 mostra a conexão de um conversor Digital-Analógico (DAC0808) com um sistema mínimo formado pelo microprocessador 8085, memória ROM 8355 e memória RAM 8156.

## 8.2 Conversor Analógico-Digital

A Fig. 8.6 mostra um circuito que converte um sinal analógico em digital. O processo de conversão usa um conversor digital-analógico e um contador. No início da conversão um sinal de controle zera a saída do contador. Como o valor digital inicial é zero, a saída do conversor DA é também zero, o que resulta um sinal alto na saída do amplificador operacional que compara o sinal analógico de entrada  $V_{ent}$  com o sinal analógico  $V_{saida}$  de saída do conversor DA. Esse sinal alto dá início ao processo de conversão.

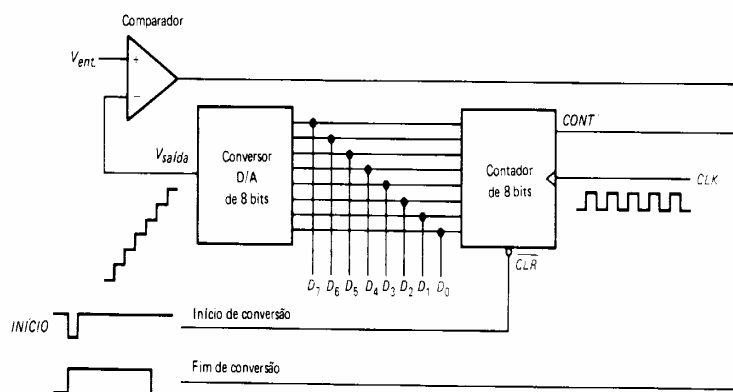


Fig. 8.6: Circuito básico de um Conversor Analógico-Digital

A conversão se encerra quando o sinal de saída do comparador estiver em nível baixo, ou seja, quando o sinal  $V_{saida}$  for maior que o sinal que se deseja converter,  $V_{ent}$ . O resultado da conversão é obtido nas linhas de transferência de dados do contador para o conversor DA. Observe que o sinal na saída do comparador pode ser usado para indicar o fim da conversão. Durante a conversão esse sinal permanece em nível alto.

A desvantagem principal de um conversor do tipo mostrado na Fig. 8.6 é que o tempo de conversão pode durar até 255 períodos de clock para um conversor de 8 bits e até 65535 períodos de clock para um conversor de 16 bits. Uma solução para reduzir o tempo de conversão é o conversor mostrado na Fig. 8.7, do tipo aproximação sucessiva.

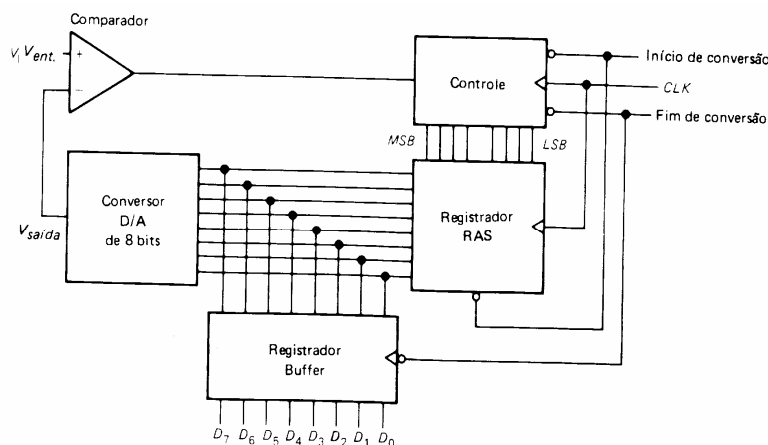


Fig. 8.7: Conversor AD por aproximação sucessiva

O processo de conversão desse tipo de conversor é diferente do anterior. Nesse conversor o número máximo de períodos de clock utilizados na conversão corresponde ao número de bits do conversor, ou seja, o conversor de 8 bits leva até 8 ciclos de clock para completar uma conversão e o de 16 bits leva até 16 ciclos de clock.

Um pulso alto de “início de conversão” enviado ao Registrador de Aproximação Sucessiva (RAS) dá início ao processo. O registrador RAS começa setando o bit mais significativo  $D_7$  (MSB) que alimenta o conversor DA. Se a saída do DA for maior que o sinal a ser convertido  $V_{ent}$ , o bit MSB é zerado. Caso o bit  $D_7$  não resulte em saída analógica  $V_{saída}$  maior que o sinal a ser convertido, o bit  $D_7$  é mantido. A seguir o segundo bit mais significativo,  $D_6$ , e os demais bits restantes são testados seguindo o mesmo procedimento, até atingir o valor correto de conversão. Ao final da conversão, um sinal baixo é emitido para o Registrador Buffer, liberando a saída digital e, ao mesmo tempo, conservando esse valor durante a próxima conversão.

### 8.3 Exercícios Propostos

1. Explique de forma resumida o princípio de operação de um conversor Analógico/Digital (conversor AD) com aproximação sucessiva.
2. Explique a característica monotonicidade de um conversor digital-analógico (conversor DA).
3. Diferencie precisão absoluta de precisão relativa de um conversor DA.
4. Como se calcula a resolução de um conversor DA? Qual a resolução de um conversor DA de 12 bits?
5. Qual a tensão na saída de um conversor DA de 8 bits, cujo sinal de entrada seja 40 h e cuja tensão de referência seja 5 V?
6. Considere o conversor DA de 4 bits da Fig. 8.2, com resistências ponderadas, onde a resistência  $R = 2 \text{ k}\Omega$  e a resistência de saída seja  $R_{saída} = 10 \text{ k}\Omega$ . Calcule a corrente e a tensão de saída do circuito. A tensão de referência é  $V_{REF} = 5 \text{ V}$ .
7. Faça um programa para o ABACUS, que lê a tensão de alimentação de um motor de corrente contínua e mostra no display de 7-segmentos correspondente ao endereço (usar CALL MOSTRAD) e lê a velocidade de rotação do motor e mostra nos LEDs. A tensão (que varia de 0 a 255 V) é lida através de um conversor Analógico-Digital de 8 bits e entrada de 0 a 5 V, que é conectado à porta A (21h) do sistema mínimo do 8085. Cada fim de conversão desse AD gera uma interrupção RST 6.5 para leitura da tensão. A tensão é lida em base hexadecimal e mostrada em base decimal.

A velocidade (que varia de 0 a 2000 rpm) é lida também através de um conversor Analógico-Digital de 8 bits e entrada de 0 a 5 V, que é conectado à porta C (23h) do sistema mínimo. Cada fim de conversão desse AD gera uma interrupção RST 5.5 para leitura da velocidade. A indicação de velocidade nos LEDs é feita da seguinte forma: até 250 rpm apenas o LED0 fica ligado; de 250 rpm até 500 rpm, os LEDs 0 e 1 ficam ligados e assim sucessivamente, até chegar no intervalo de 1750 rpm até 2000 rpm, quando todos os LEDs ficam ligados. Seguir os passos dados.

#### Programa principal:

- a. Habilita as interrupções RST 6.5 e RST 5.5 e mascara a RST 7.5;
- b. Habilita a porta A (21 h) como entrada de dados e a porta B (22 h) como saída de dados (igual ao caso do ABACUS). Considere que a porta C já está habilitada;
- c. Zera registradores A, D e E;
- d. Entra num loop que mostra continuamente o conteúdo de DE (tensão) no display (subrotina do ABACUS) e o conteúdo de A (velocidade) nos LEDs.

#### Subrotina 1

(atendimento da RST 6.5 a cada fim de conversão do AD da tensão)

- a. Lê o valor da tensão, presente na porta 21h;
- b. Converte o valor lido (hexadecimal) em valor decimal, guardando o resultado em DE.
- c. Retornar ao programa principal.

Observação: Não é o caso de usar DAA e sim fazer a conversão da base hexadecimal para a base decimal. Essa conversão é feita da seguinte forma:

- Passo 1: Divide-se o número hexadecimal pelo decimal 10 (hexadecimal 0A). O resto dessa primeira divisão é o dígito menos significativo do número decimal resultante (d0). Se o quociente for zero o processo é encerrado.
- Passo 2: Se o quociente da divisão anterior for diferente de zero, divide-se esse quociente novamente pelo decimal 10 até obter o segundo resto, que é o segundo dígito menos significativo (d1). Se o quociente dessa segunda divisão for zero, o processo é encerrado.
- Passo 3: Se o quociente da divisão anterior for diferente de zero, divide-se esse quociente novamente pelo decimal 10 até obter o terceiro resto, que é o dígito mais significativo (d2). Como o valor máximo possível é FF h, não há possibilidade de um Passo 4.

O decimal resultante é: **d2d1d0**. d2 deve ser mostrado em D e d1d0 deve ser mostrado em E.

### Subrotina 2

(atendimento da RST 5.5 a cada fim de conversão do AD da velocidade)

- (a) Lê o valor da velocidade, presente na Porta 23h;
- (b) Verifica o valor da velocidade (através de seu equivalente digital) e carrega em A o valor hexadecimal que liga os LEDs desejados, de acordo com o que foi explicado no item (d) da parte inicial desta avaliação.
- (c) Retorna ao programa principal com o valor de A definido no item anterior.

## 8.4 Referências Bibliográficas

- [1] Ziller, Roberto M., “Microprocessadores – Conceitos Importantes,” Editora do autor, Florianópolis, SC, 2000.
- [3] Malvino, Albert P., “Microcomputadores e Microprocessadores,” Tradução: Anatólio Laschuk, revisão técnica: Rodrigo Araes Caldas Farias, McGraw-Hill, São Paulo, 1985.

## 9. MICROCONTROLADORES 8086/8088

### 9.1 Introdução

O 8088 é o microprocessador do PC-XT (*Personal Computer - eXtended Technology*), produzido inicialmente pela Intel, sendo equivalente ao microprocessador 8086 (que tem como principal diferença a extensão do barramento externo de dados). O 8086 tem os barramentos de dados externo e interno de 16 bits enquanto o 8088 apresenta o barramento interno de 16 bits e o externo de 8 bits.

Os processadores 80186 e 80188 são *upgrades* do 8086 e do 8088 respectivamente, onde foram incorporados de 10 a 15 componentes (gerador de clock, controlador de DMA, unidade de chip select, controlador de interrupção, temporizadores, etc.) mais comuns em sistemas baseados no 8088/86. Todas as instruções do 8086 são incluídas no 80186, além de 10 novas instruções.

Ainda, dentro da família dos microprocessadores Intel de 16 bits, temos o processador 80286 (registradores de 16 bits, 24 linhas de endereço). O 80286 incorporou um gerenciador de memória e agregou recursos para multitarefa (há novas instruções com este objetivo). A partir do 80286 os PCs passaram a se chamar PC-AT (PC - *Advanced Technology*).

A diferença no barramento externo de dados entre o 8088 e o 8086 poderia nos levar a supor este último é duas vezes mais rápido que o primeiro; tal fato, em geral, não é verdadeiro. Quando a Unidade de Execução do microprocessador (chamada **EU**: *Execution Unit*) executa uma instrução, outra unidade independente (chamada **BIU**: *Bus Interface Unit*) busca uma nova instrução na memória e a guarda em uma memória interna do microprocessador (*queue*). Com isso, há uma sobreposição de ciclos de execução e ciclos de busca de instrução. No caso do 8088, se a instrução é de 16 bits a BIU deve fazer duas leituras e a EU deverá esperar a instrução estar completa. No 8086 com apenas uma leitura na memória a instrução já estaria disponível para execução.

#### Portanto:

- para aplicações orientadas a 8 bits (byte) o 8088 desenvolve a tarefa tão bem quanto o 8086;
- para aplicações orientadas a 16 bits (word), o 8088 será menos eficiente que o 8086 mas não com metade da velocidade.

#### **Principais Características do 8088**

- Barramento de endereços com 20 linhas: proporciona endereçamento para até 1MB (1.048.576 bytes) de memória.
- Capacidade de uso de coprocessador (8087).
- Todos os registradores internos possuem 16 bits (alguns podem ser acessados pela metade - primeiros ou últimos 8 bits).
- Possui 2 modos de funcionamento: Mínimo e Máximo: no modo mínimo equivale a um 8085 acelerado. O modo máximo é usado em ambientes multiprocessados (quando o 8088 convive com outros 8088) ou em ambientes coprocessados (quando há processador aritmético). Desta forma, como o PC-XT tem soquete para utilização do 8087, sua CPU funciona com o 8088 no modo máximo.
- A CPU é dividida em 2 blocos: Unidade de Execução (**EU**) e Unidade de Interface com o Barramento (**BIU**).
- BIU com fila de instruções (*queue*) de 4 bytes (8088) ou 6 bytes (8086).
- Realiza instruções de divisão e multiplicação (para números com ou sem sinal), assim como operações com strings (blocos de bytes).
- Possui registradores de segmento, possibilitando segmentação de memória.
- No RESET o endereço de memória acessado é FFFF0H.

Para ilustrar melhor as características do 8088, a tabela a seguir foi criada, onde é feita uma comparação entre o 8085, o 8086 e o 8088. A seguir faz-se um paralelo entre os registradores do 8085 e os registradores do 8088 (os quais serão mais detalhadamente explicados nas próximas seções).



## Características dos Microprocessadores 8085, 8086 e 8088

Característica	Microprocessador 8085	Microprocessador 8088	Microprocessador 8086
Barramento de endereço	16 bits	20 bits	20 bits
Capacidade de endereçamento de memória	65.536 ( 64 kB )	1.048.576 ( 1 MB )	1.048.576 ( 1 MB )
Barramento de dados	8 bits	Interno: 16 bits Externo: 8 bits	Interno: 16 bits Externo: 16 bits
Manipulação de STRINGS	NÃO	SIM	SIM
Registradores Internos	8 bits e 16 bits	16 bits	16 bits
Uso de segmentação para endereçamento	NÃO	SIM	SIM
Aritmética Decimal completa	NÃO	SIM	SIM
Etapas de Busca e Execução	Em sequência:  <b>Busca → Executa</b>	Unidades Independentes: Unidade de Interfaceamento com Barramento ( <b>BIU</b> ) – responsável pela <b>Busca</b> e Unidade de Execução ( <b>EU</b> )	Unidades Independentes: Unidade de Interfaceamento com Barramento ( <b>BIU</b> ) – responsável pela <b>Busca</b> e Unidade de Execução ( <b>EU</b> )

## Registradores dos microprocessadores 8085, 8086 e 8088

## Registradores do 8085

	A	Acumulador
H	L	Apontador de dados
B	C	
D	E	
SP		Apontador de pilha

PC	Contador de Programa
----	----------------------

FLAGS	Registrador de Flags
-------	----------------------

## Registradores do 8088 / 8086

AH	AL (A)	AX – Acumulador Primário
BH	BL	BX – Acumulador e Registrador Base
CH	CL	CX – Acumulador e Contador
DH	DL	DX – Acumulador e Endereçador de I/O
SP		Apontador de pilha
BP		Apontador base – usado na pilha
SI		Índice da Fonte – usado para indexação
DI		Índice de Destino – usado para indexação
IP		Ponteiro de Instrução
CS	Segmento de Código	Registradores de segmento. São usados para a formação do endereço absoluto.
DS	Segmento de Dados	
SS	Segmento de Pilha	
ES	Segmento Extra	
FLAGS		Registrador de Flags

## 9.2 Diagrama de Blocos do 8088

Uma das principais inovações do 8088 foi a separação entre lógica de execução e lógica de controle do barramento, criando dois blocos que funcionam de forma assíncrona: a Unidade de Execução (**EU**) e a Unidade de Interface com o Barramento (**BIU**).

A EU tem como função processar (decodificar e executar) instruções obtidas da BIU. A EU é constituída de:

- Registradores de Dados;
- Registradores de Endereços;
- ALU;
- Unidade de Controle.

A BIU tem apenas funções de hardware: controla o acesso ao barramento (linhas de endereçamento, linhas de dados e sinais de controle). A BIU é constituída de:

- Lógica de interface com o barramento;
- Registradores de segmento;
- Lógica para endereçamento de memória (somador);
- Fila de instruções (4 bytes para o 8088 e 6 bytes para o 8086).

### Procedimento de trabalho do 8088:

1. A BIU coloca o conteúdo do IP (que é somado ao registrador CS) no barramento para efetuar a busca de instrução;
2. O registrador IP é incrementado (aponta para a próxima instrução);
3. A instrução lida é passada para a fila;
4. A EU pega a primeira instrução da fila;
5. Enquanto a EU executa esta instrução a BIU faz uma nova busca de instrução para preencher a fila. Se a instrução a ser executada pela EU for muito demorada a BIU preenche toda a fila.

Há 2 situações em que não são aproveitadas as instruções contidas na fila. São elas:

- Na execução de instruções de desvio. Neste caso a fila é descartada (ou seja, é sobreescrita);
- Quando a instrução faz referência à memória.

### Definições:

- ⇒ **Pipelining**: introdução de paralelismo para executar programas de natureza sequencial. OBS: Esta tecnologia é utilizada no projeto de processadores RISC.
- ⇒ **Periférico**: qualquer equipamento ou dispositivo que provê à CPU comunicação com o resto do sistema.
- ⇒ **Interface**: Conjunto de componentes capaz de controlar um dispositivo (*device*). OBS: uma placa de expansão pode conter uma ou mais interfaces.
- ⇒ **IP (Instruction Pointer)**: tem a mesma função do PC (*Program Counter*) no 8085.
- ⇒ **Registrador de Segmento**: registrador que armazena o endereço base. Deve ser somado ao endereço de deslocamento (ou *offset*, ou ainda endereço lógico) para obter o endereço físico (ou endereço absoluto). No 8088 os segmentos definem blocos de 64Kbytes de memória. Há 4 registradores de segmento:
  - \* Code Segment (CS): área destinada a código;
  - \* Data Segment (DS): área destinada a dados;
  - \* Extra Segment (ES): área extra destinada a dados;
  - \* Stack Segment (SS): área destinada a armazenar endereços de retorno de rotinas de interrupções e de sub-rotinas;

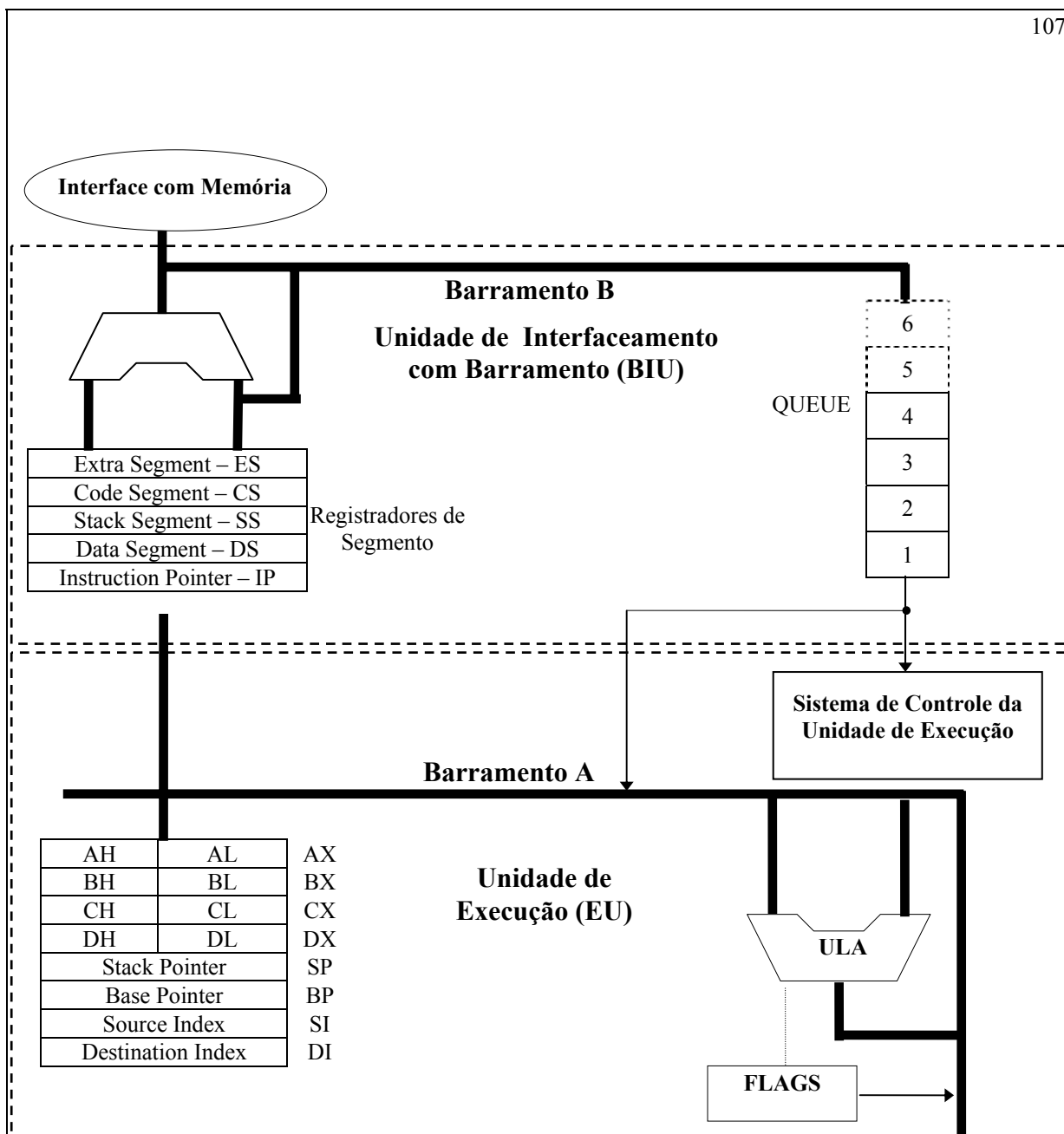


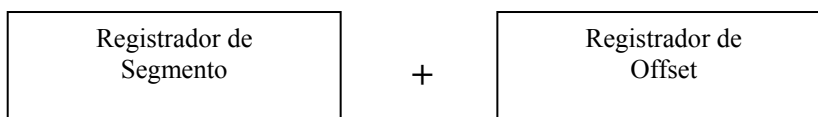
Diagrama de blocos do 8088/8086

### Vantagens da Utilização de Memória Segmentada

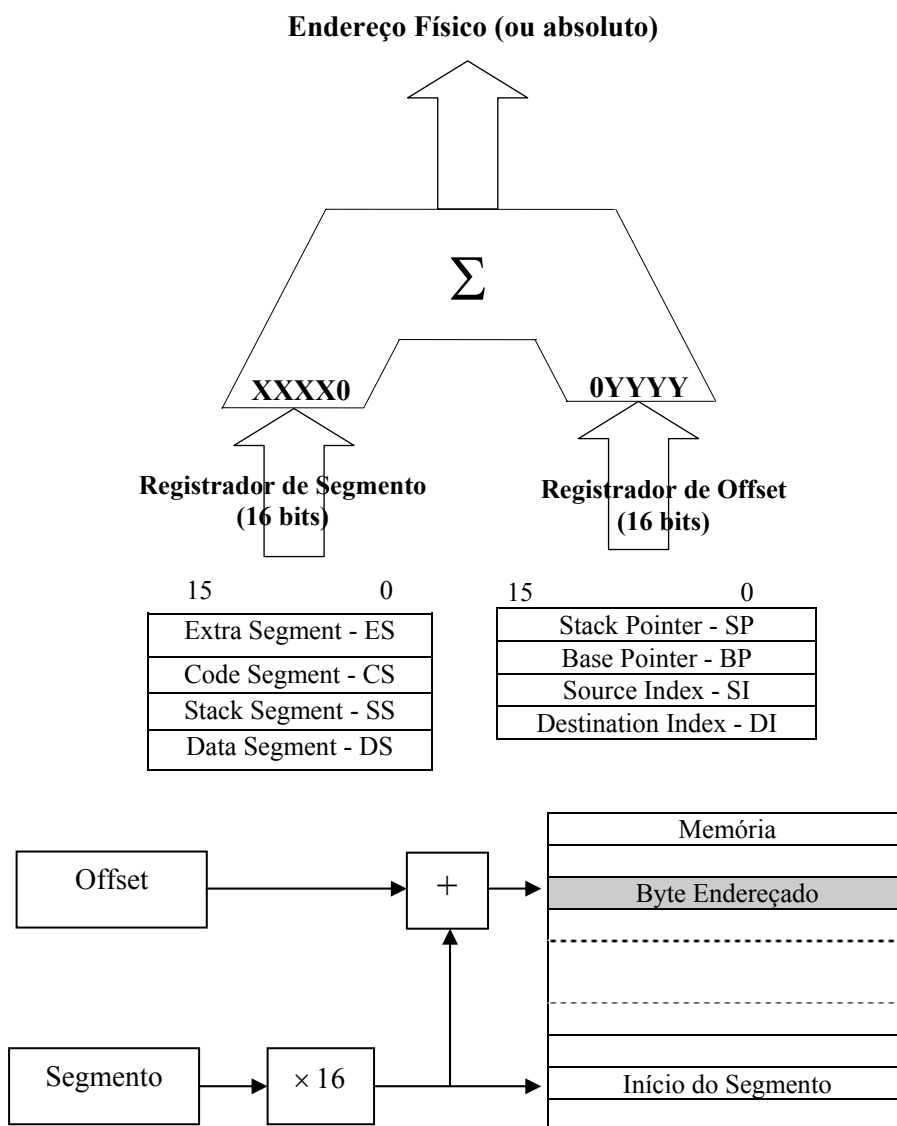
- Por haver uma área específica para armazenamento de código e outras áreas para armazenamento de dados, pode-se trabalhar com tipos diferentes de conjuntos de dados (por exemplo, em um ambiente multitarefa onde um programa atende várias entradas de dados);
- Programas que referenciam endereços lógicos (0000 a FFFF no caso do 8088) podem ser carregados em qualquer espaço (físico) da memória (00000 a FFFFF): possibilita a realocação de programas.

### SEGMENTAÇÃO

Consiste em combinar 2 registradores de 16 bits para gerar um endereço de memória de 20 bits



Endereço Físico = (Conteúdo do Registrador de Segmento)  $\times$  16 + (Conteúdo do Registrador de Offset)



Exemplo: SEGMENTO = 2000H; OFFSET = 2000H

Representação: 2000h:2000h

Endereço Físico = 20000 + 02000h = 22000h

Se o segmento for 4000h, tem-se:

Representação: 4000h:2000h e Endereço Físico = 40000h + 02000h = 42000h

### 9.3 Os Registradores do 8088

O 8088 tem 14 registradores de 16 bits, que podem ser classificados em grupos:

- ⇒ grupo de registradores de dados (4), (*Data register*);
- ⇒ grupo de registradores apontadores (*Pointer*) (2) e índices (*Index*) (2);

- ⇒ grupo de registradores de segmento (4) (*Segment register*);
- ⇒ um registrador apontador de instruções (*Instruction Pointer register*);
- ⇒ um registrador de flags (*Flags register*).

### 9.3.1 Registradores de Dados

Os Registradores de Dados são também chamados Registradores de Propósito Geral. São 4 registradores de 16 bits de uso geral, normalmente utilizados pelo conjunto de instruções para realizar operações lógicas e aritméticas. Podem também ser usados como registradores de 8 bits para instruções envolvendo 1 byte.

Registradores de Dados					
	15	8	7	0	
AX	AH		AL		Accumulator
BX	BH		BL		Base
CX	CH		CL		Counter
DX	DH		DL		Data

**AX** – Acumulador primário – Todas as operações de I/O são realizadas através deste registrador. Operações que utilizam dados imediatos necessitam de menos memória quando feitos através de **AX**. Algumas operações com *'strings'* e instruções aritméticas pedem o uso deste registrador. Geralmente é usado pelos compiladores como hospedeiro para valores retornados de subrotinas.

**BX** – Registrador Base – Parece com registrador **HL** do 8085. É o único registrador de finalidade geral que pode ser utilizado no cálculo de endereço de memória. Todas as referências à memória que usam esse registrador no cálculo do endereço usam o registrador **DS**, como segmento *'default'*.

**CX** – Contador – Parece com o registrador **BC** do 8085. É decrementado durante operações com *'loops'* e *'strings'*. Tipicamente, é usado para controlar o número de repetições do *'loop'*. Também é usado para rotações e deslocamentos de vários bits.

**DX** – Endereçador de I/O e Registrador de Dados – Parece com o registrador **DE** do 8085. Recebe o nome de registrador de dados, principalmente por força dos mnemônicos. Em algumas operações de I/O, fornece o endereço, coisa que nenhum outro registrador pode fazer. Também é usado em operações aritméticas, incluindo multiplicação e divisão, com o resultado a 32 bits. Pode ser usado por compiladores, juntamente com **AX**, para retornar valores de subrotinas.

### 9.3.2 Registradores Apontadores e Índices

Este grupo de 4 registradores é tipicamente usado para gerar endereços de memória efetivo (nome dado à porção offset do endereço físico). Apenas são usados em 16 bits. Podem ainda ser utilizados em operações aritméticas e lógicas para gerar novos endereços efetivos.

<b>Registradores Apontadores e Índices</b>			
	15		0
Stack Pointer	SP		
Base Pointer	BP		
Source Index	SI		
Destination Index	DI		

**SP** – Ponteiro de Pilha – Parece com o registrador **SP** do 8085. Armazena o offset do endereço do topo da pilha. Todas as referências ao **SP**, por definição, usam o registrador **SS**.

**BP** – Ponteiro da Base – Permite acessar dados no segmento da pilha. Tipicamente é usado para acessar parâmetros que foram passados pela pilha.

**SI** e **DI** – Registradores de Indexação – São usados para acessar dados na memória de dados. São extensivamente usados nas operações com *'strings'*. Podem ser usados como operandos em todas as operações lógicas e aritméticas de 16 bits.

No conjunto de instruções do 8088 nem todos os registradores são especificados. Em muitos casos, uma instrução pode usar um só registrador ou conjunto de registradores específicos. Para outras instruções, os registradores têm uso implícito. A tabela a seguir lista as operações que implicam no uso específico (implícito) de um determinado registrador.

Registradores	Operações
AX	multiplicação de word (16 bits), divisão de word, I/O de word
AL	multiplicação de byte, divisão de byte, I/O de byte, aritmética decimal
BX	referência de memória (semelhante ao reg. par HL do 8085)
CX	operação de strings, loops
CL	deslocamento, rotação
DX	multiplicação de word, divisão de word, end. indireto de I/O (0-65535)
SP	operação de stack
SI	operação de string (origem)
DI	operação de string (destino)

### 9.3.3 Registradores de Segmento

#### *Registradores de Segmento*

	15	0
Code Segment	CS	
Data Segment	DS	
Stack Segment	SS	
Extra Segment	ES	

Estes 4 registradores de 16 bits são utilizados para alocar segmentos de 64 kB de memória.

**CS** – Segmento de Código – Para a busca (*'fetch'*) de cada instrução, o offset, definido por **IP**, é adicionado ao endereço base definido por **CS** para o endereço da instrução.

**DS** – Segmento de Dados – Todo acesso a dados usa este registrador como referência, mas existem 3 exceções: (a) endereços para acessos à pilha são calculados usando o registrador de segmento de pilha (**SS**); (b) endereços para acessos a dados que usam o **BP** são calculados usando o **SS** e (c) operações com *'strings'*, que usam o **DI** no cálculo do endereço, são feitas usando **ES**.

**SS** – Segmento de Pilha – Todos os acessos a dados que usam os registradores **SP** ou **BP** tomam como referência o registrador de segmento de pilha (**SS**).

**ES** – Segmento Extra – Operações com *'strings'*, que usam **DI** para calcular o endereço, são feitas usando o registrador **ES** para definir o segmento.

A tabela a seguir sintetiza o uso dos registradores de segmento como *'default'*.

#### *Uso dos registradores de segmento*

Tipos de Referência à Memória	Segmento Base (default)	Alternativo	Offset
Instrução de busca	CS	nenhum	IP
Operação de pilha	SS	nenhum	SP
Variável	DS	CS, ES, SS	endereço efetivo
Fonte string	DS	CS, ES, SS	SI
Destino string	ES	nenhum	DI
BP usado como reg. pointer	SS	CS, DS, ES	endereço efetivo

BX usado como reg. pointer	DS	CS, SS, ES	endereço efetivo
----------------------------	----	------------	------------------

O registrador de segmento *default* é selecionado pelo hardware do 8088. Em alguns casos é possível não levar em conta o registrador default e especificar um registrador de segmento diferente (ver exemplo abaixo).

#### **EXEMPLO:**

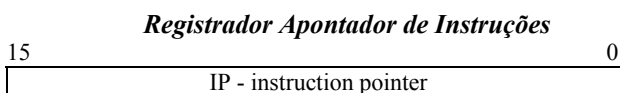
A instrução **MOV [BX],CX** move o conteúdo de CX para a posição de memória calculada utilizando o registrador de segmento **DS** (segmento default para dados) e o registrador de dados **BX**. Mas, se escrevermos **MOV CS:[BX],CX** forçamos o uso do registrador de segmento **CS** para o cálculo da posição de memória destino (que será **CS:BX**).

Da mesma forma, a instrução **MOV AX,[BP]** acessa posição de memória diferente da instrução **MOV AX,ES:[BP]**.

Notar que não há registrador de segmento e registrador de offset alternativos para a busca de instruções (que deve sempre ser o par **CS:IP**) e para operações de pilha (que deve sempre ser o par **SS:SP**).

### 9.3.4 Apontador de Instruções

Este registrador contém o endereço offset da próxima instrução a ser executada pelo microprocessador. Tem a mesma função do PC utilizado no 8085.



### 3.5 - O Registrador de Flags

É um registrador de 16 bits mas apenas 9 bits são usados. Seis deles são bits de status que refletem os resultados de operações aritméticas e lógicas.

Os outros três são bits de controle. O conjunto de instruções do 8088 possui instruções específicas setar e resetar seus flags (que será visto mais adiante).

#### **Registrador de Flags = Registrador de Estado do Programa (PSW)**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
×	×	×	×	O	D	I	T	S	Z	×	A	×	P	×	C

#### **Flags de Status**

C – Flag de carry – reflete o ‘vai um’ do bit mais significativo, nas operações aritméticas (de 8 ou 16 bits). Ele também é modificado por algumas instruções de rotação e deslocamento.

Obs.: As operações de subtração usam aritmética em complemento dois e, por isso, o ‘carry’ é invertido e passa a funcionar como ‘borrow’. Se, após uma operação de subtração, obtém-se C = 1, isso indica que não houve ‘borrow’, mas C=0, indica que houve ‘borrow’.

P – Flag de Paridade – indica a paridade (par), dos 8 bits menos significativos, do resultado da operação realizada.

P = 1 → número par de ‘1’ nos 8 bits menos significativos

P = 0 → número ímpar de ‘1’ nos 8 bits menos significativos

A – Flag Auxiliar de Carry – reflete o ‘vai um’ do bit 3, em uma operação de 8 bits.

Z – Flag de Zero – indica se uma operação teve zero como resultado.

Z = 1 → se o resultado da operação for igual a zero

$Z = 0 \rightarrow$  se o resultado da operação for diferente de zero

S – Flag de Sinal – é igual ao bit de mais alta ordem do resultado de uma operação aritmética.

$S = 0 \rightarrow$  resultado positivo

$S = 1 \rightarrow$  resultado negativo

O – Flag de Overflow – seu conteúdo é obtido através de uma operação XOR do 'carry in' com o 'carry out' do bit de mais alta ordem do resultado de uma operação aritmética. Ele indica um 'overflow' de magnitude, em aritmética binária com sinal. Indica que o resultado é muito grande para o campo destino.

### Flags de Controle

T – Flag de Trap (armadilha) – usada para a depuração de programas. Coloca o 8086 no modo passo a passo. Após cada instrução uma interrupção é gerada automaticamente.

I – Flag de Interrupção – habilita ou desabilita a interrupção externa (pedida pelo pino INTR). Ao contrário do 8085, onde as interrupções RST 7.5, RST 6.5 e RST 5.5 podem ser habilitadas/desabilitadas individualmente, no 8086 todas são habilitadas ou desabilitadas ao mesmo tempo. A habilitação/desabilitação individual pode ser feita através do controlador de interrupção 8259.

$I = 1 \rightarrow$  interrupção habilitada

$I = 0 \rightarrow$  interrupção desabilitada

D – Flag de Direção – determina se as operações com 'strings' vão incrementar ou decrementar os registradores de indexação (SI e DI).

$D = 1 \rightarrow$  os registradores SI e DI serão decrementados, ou seja, a 'string' será acessada a partir do endereço mais alto em direção ao mais baixo.

$D = 0 \rightarrow$  os registradores SI e DI serão incrementados, ou seja, a 'string' será acessada a partir do endereço mais baixo em direção ao mais alto.

### EXEMPLO:

Descreva o estado dos flags após a execução da instrução **ADD AL,1** (Adiciona 1 ao conteúdo do registrador de 8 bits AL), sabendo que inicialmente  $AL = 7Fh$ .

Resultado: AL terá o valor 80h ( $7Fh + 01h$ ) e seus flags serão:

$C \rightarrow 0$

$P \rightarrow 0$

$A \rightarrow 1$

$Z \rightarrow 0$

$S \rightarrow 1$

$O \rightarrow 1$

## 9.4 A Pinagem do 8088

O microprocessador 8088 é um CI com 40 pinos. Na figura a seguir alguns pinos apresentados tem duas definições. O 8088 tem dois modos de operação que são selecionados pelo pino 33 ( $MN/\overline{MX}$ ). Quando este pino é ligado um nível alto, o  $\mu P$  fica no modo mínimo e é compatível com o 8085, podendo substituir diretamente este  $\mu P$ .

Vcc	A15	A16/ S3	A17/ S4	A18/ S5	A19/ S6	Vcc	MN/ MX	RD	RQ/ GT0	RQ/ GT1	lock	S2	S1	S0	QS0	QS1	Test	RDY	RST
40	39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20



Gnd A14 A13 A12 A11 A10 A9 A8 AD7 AD6 AD5 AD4 AD3 AD2 AD1 AD0 NMI INTR CLK GND

No PC, o 8088 é usado no modo máximo ( $\overline{MN}/\overline{MX}=0$ ). As definições a seguir são relativas a este modo.

- (a) **AD0-AD7 (9 -16):** transmitem os 8 bits menos significativos dos endereços multiplexados com o byte de dados. Os bits de endereço (A0-A7) são apresentados no início do ciclo (T1) e os dados a partir de T2. Ambos sinais são bufferizados para fazerem parte do barramento do sistema.
- (b) **A8-A15 (2-8 e 39):** é a outra parte dos bits de endereços. Não são multiplexados (no 8086 são), permanecendo estáveis durante grande parte do ciclo de acesso a barramento. Após bufferização fazem parte do barramento do sistema.
- (c) **A16/S3 - A19/S6 (35-38):** sinais de status e endereços multiplexados. No início do ciclo de barramento estão presentes os bits dos endereços A16 a A19. Durante o restante do ciclo (a partir de T2) informa o status interno do 8088.
  - S6** = 0 (permanece em nível baixo no modo máximo);
  - S5** indica o status do flag de interrupção (se = 1 a interrupção está habilitada);
  - S3 e S4** são decodificados para indicar qual registrador de segmento está sendo usado no momento (ver tabela abaixo). Não são usados no PC-XT da IBM.

S3	S4	Reg. Segmento
0	0	Dados Extra (ES)
0	1	Pilha (SS)
1	0	Código (CS)
1	1	Dados (DS)

- (d) **Clock (19):** sinal de entrada responsável pela geração da temporização no  $\mu P$ . No projeto do PC este sinal é gerado pelo CI 8284A, tendo frequência de 4,77Mhz e ciclo de trabalho (*duty cycle*) de 33%.
- (e)  **$\overline{RQ}/\overline{GT}_0$  (Request / Grant) (31):** sinal bidirecional, usado por outros controladores de barramento local para requerer o uso do barramento (faz o papel dos pinos HOLD e HLDA do 8085). No PC este sinal é ligado ao co-processador 8087.
- (f)  **$\overline{RQ}/\overline{GT}_1$  (30):** idem ao anterior, com grau de prioridade menor. No PC não é usado.
- (g) **Lock (29):** é ativado pela instrução LOCK e permanece ativo até o fim da próxima instrução. É usado para indicar a outros controladores de barramento que estes não devem tentar obter o controle do barramento durante a execução da instrução seguinte à LOCK. O PC não é projetado para multi-controladores e portanto, este pino não é usado.
- (h) **NMI (17):** sinal de entrada, usado para gerar uma interrupção não mascarável no  $\mu P$ . No PC este sinal é mascarado externamente por uma lógica programável (um FlipFlop). É equivalente ao pino TRAP do 8085.
- (i) **INTR (18):** entrada para solicitação de interrupção mascarável. No PC este pino é ligado ao CI 8259A que expande esta entrada de 1 para 8, adicionando, dentre outras funções, níveis de prioridade.
- (j) **Ready (22):** sinal de entrada usado para inserir estados de espera no ciclo de barramento, estendendo-os. No PC este sinal vem do chip do clock 8284A que o sincroniza com o clock do sistema.
- (l) **Reset (21):** usado para reiniciar o  $\mu P$ . No PC este sinal vem do chip de clock 8284A que recebe uma entrada da fonte do sistema. A fonte envia um sinal chamado "Power Good" indicando que os níveis de tensão estão apropriados e o sinal Reset pode ser então removido para iniciar o  $\mu P$ . A situação dos registradores do 8088 após o RESET é a seguinte:

- Reg. IP = 0000 e reg. CS = FFFF resultando no end. de início de programa = FFFF0h;
- Registradores DS, SS, ES e IP zerados;
- Registrador Flags zerado.

(m) **QSO, QS1 (24, 25) - Queue Status:** sinais de saída que fornecem o status da fila de instrução (queue) do 8088. No PC são ligados ao co-processor de modo que este possa acompanhar o status da fila 8088.

(n) **TEST (23):** pino de entrada, utilizado em conjunto com a instrução WAIT (não confundir com estado de espera, "wait state"). Se após a instrução WAIT o pino  $\overline{TEST}$  estiver em nível alto, o 8088 espera em estado inativo até este pino obter sinal baixo. No PC é ligado a um pino de saída do 8087.

(o)  **$\overline{S_0}$ ,  $\overline{S_1}$ ,  $\overline{S_2}$  (26 - 28) (status):** apresentam informações de status sobre o tipo de ciclo de barramento que está sendo desenvolvido. Este status é válido no início de cada ciclo de barramento (durante o período T1). No PC estes pinos são ligados ao 8288, chip controlador de barramento, onde são decodificados. Os sinais de saída decodificados no 8288 formam as linhas de controle do sistema. Os sinais de controle gerados pelo 8288 a partir das informações de status são:  $\overline{IOR}$ ,  $\overline{IOW}$ ,  $\overline{MemR}$ ,  $\overline{MemW}$  e ALE. Os bits de status podem ser vistos na tabela abaixo.

$\overline{S_2}$	$\overline{S_1}$	$\overline{S_0}$	Tipo de ciclo no BUS
0	0	0	Reconhecimento de interrupção ( $\overline{INTA}$ )
0	0	1	Leitura I/O ( $\overline{IOR}$ )
0	1	0	Escrita I/O ( $\overline{IOW}$ )
0	1	1	Halt
1	0	0	Busca de instrução
1	0	1	Leitura memória ( $\overline{MemR}$ )
1	1	0	Escrita memória ( $\overline{MemW}$ )
1	1	1	Passivo

## 9.5 Ciclos de Barramento do 8086

O 8086 COMUNICA-SE COM AMBIENTE EXTERNO ATRAVÉS DE UM SISTEMA DE BARRAMENTOS. ASSIM, EXISTEM CICLOS DE BARRAMENTO PARA BUSCAR ("FETCH") INSTRUÇÕES E PARA TRANSFERIR DADOS (ESCREVER OU LER). AS FIGURAS A SEGUIR (ZELENOVSKY, PÁG. 22) MOSTRAM OS CICLOS DE LEITURA E ESCRITA, RESPECTIVAMENTE.

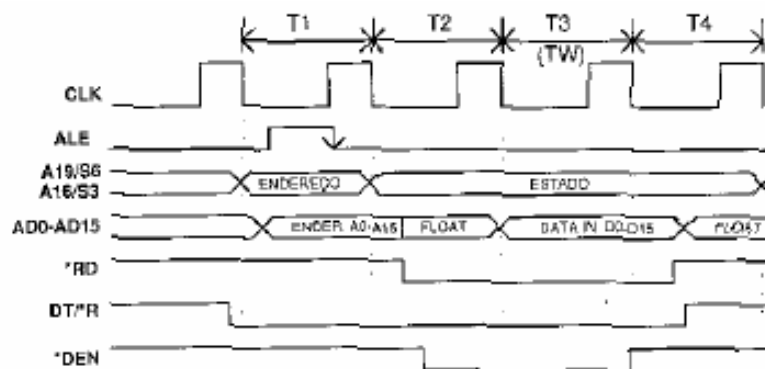


Figura 2.4. Ciclo de barramento para a leitura de dados ou instrução.

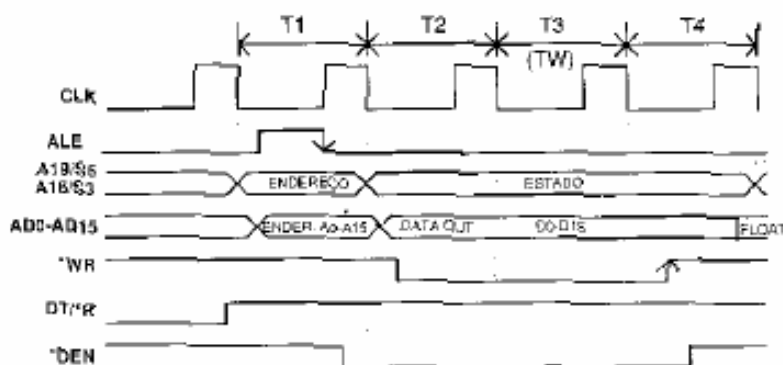


Figura 2.5. Ciclo de barramento para a escrita de dados.

T1 a T4 → períodos de relógio

ALE – Address Latch Enable – O endereço só é reconhecido depois que este sinal vai para nível alto, mas o endereço só é efetivamente usado quando este sinal volta para o nível lógico baixo. Durante o ciclo T2 os sinais de endereço são removidos.

AD0 – AD15 → Barramento de dados. No ciclo de leitura esse barramento é colocado em estado de alta impedância no estado T2, em preparação para a leitura de dados. No ciclo de escrita a CPU coloca os dados no barramento no estado T2.

\*RD → Ativo baixo. Indica leitura de dados da memória ou de um dispositivo de entrada e saída. Ativado em T2.

\*WR → Ativo baixo. Indica escrita de dados na memória ou em um dispositivo de entrada e saída. Ativado em T2.

DT/\*R → Indica a direção da transferência de dados.

\*DEN → Ativo baixo. Data Enable. Indica que vai haver tráfego de dados no barramento.

A19/S6, A16/S3 → Barramento de dados durante ciclo T1. Barramento de sinais de estado de T2 a T4.

A17/S4	A16/S3	Significado
0	0	Segmento Extra
0	1	Segmento de Pilha
1	0	Segmento de Código (ou nenhum)
1	1	Segmento de Dados
S5 = IF	Estado de habilitação da interrupção	
S6 = 0	CPU 8086 tem controle do barramento	

TW → Wait State – Tempo de espera. No período de leitura a CPU lê o dado do barramento no período T3, após o dispositivo lido entregar esses dados. Se o dispositivo não conseguir entregar os dados até o final do ciclo T3, o pino READY do 8086 é colocado em nível lógico baixo, fazendo com que a CPU fique estado de espera, repetindo do estado T3 quantas vezes forem necessárias, enquanto aguarda o dispositivo entregar os dados. No período de escrita pode acontecer o mesmo. A CPU está pronta para entregar os dados, mas o dispositivo não está pronto para recebe-los. O pino READY é colocado em nível baixo, gerando períodos de espera.

## 9.6 Endereçamento de Memória

Já foi explicado numa seção anterior como é feita a composição do endereço físico do 8088. Uma das características desse microprocessador é a sua capacidade de endereçar mais que 64 kbytes de memória. O 8088 possui 20 bits de endereços, permitindo ter uma memória física de 1.048.576 bytes (ou 1MB).

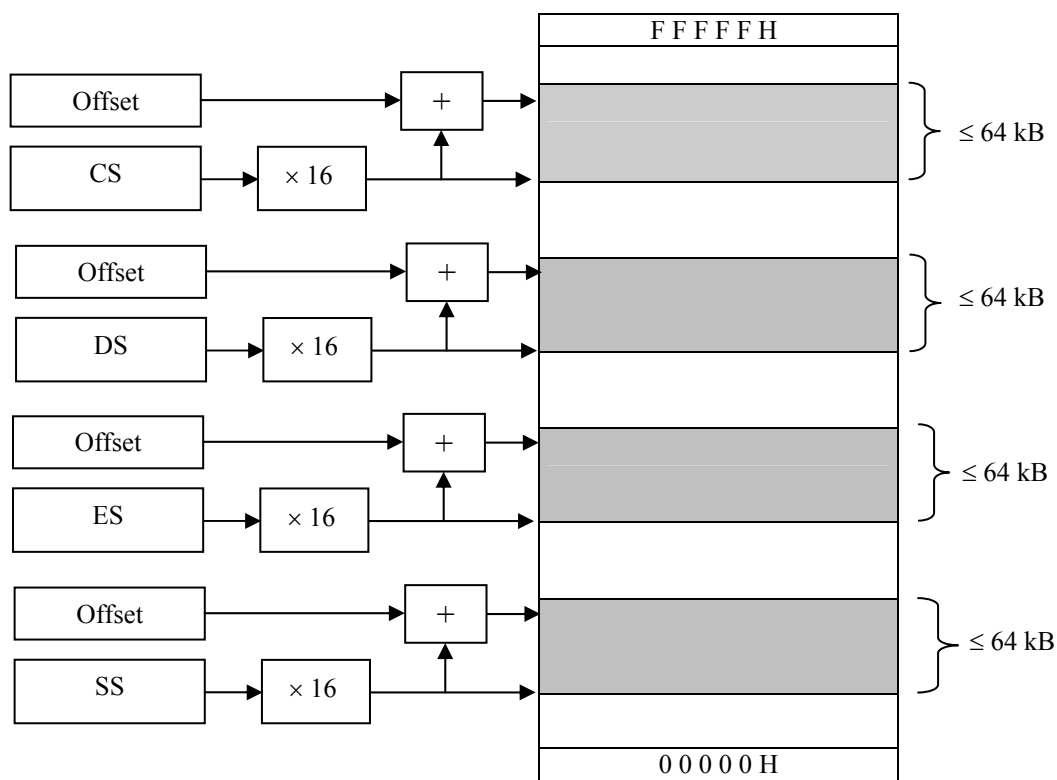
A geração de 20 bits, já explicada, é feita manipulando o conteúdo de registradores especiais chamados *Registradores de Segmento*. O valor carregado no registrador de segmento é usado para localizar, no espaço de 1MB, a região de 64kB onde as instruções do 8088 irão operar. Um outro registrador (que pode ser um reg. apontador, um reg. índice ou o BX) pode especificar 64kB dentro de um segmento. O endereço de memória física será formado pelo deslocamento do conteúdo do registrador de segmento de 4 bits para a esquerda (acrescentado-se 4 bits 0 à direita), somado a outro valor de 16 bits (acrescentado-se 4 bits 0 à esquerda), resultando num endereço físico com 20 bits. É interessante observar que diferentes combinações de endereços lógicos (offsets) e registradores de segmento podem oferecer o mesmo endereço físico, como exemplificado a seguir:

**EXEMPLO:** supondo que **SS=8F00h**, **SP=21F1h**, **CS=9020h**, **IP=0FF1h**, os endereços físicos para um acesso à pilha e para a busca da próxima instrução são dados por:

SS = 8F00h	→	8F000	
SP = 21F1h	→	+ 021F1	
<b>endereço físico</b>	→	911F1h	←
CS = 9020h	→	90200	
IP = 0FF1h	→	+ 00FF1	
<b>endereço físico</b>	→	911F1h	←

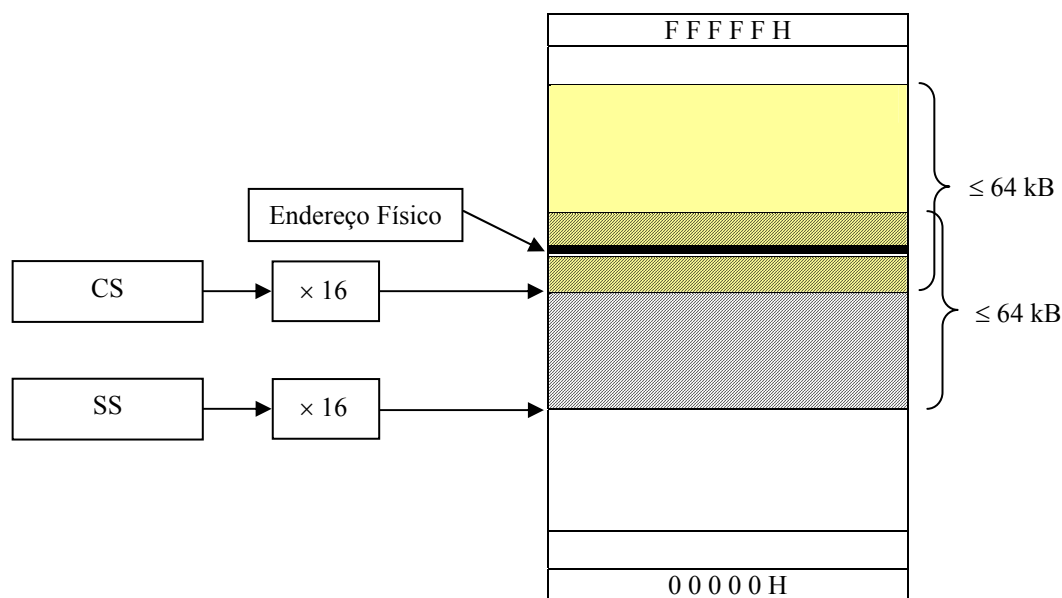
Na realidade os quatro registradores de segmento no 8088 (CS, DS, SS e ES) podem assumir o mesmo valor, ou valores próximos, ou seja, podem se sobrepor parcial ou totalmente. O programador deve tomar cuidado para que isto apenas ocorra se for estritamente necessário.

Estes 4 registradores podem apontar para qualquer região de 64 kB, no espaço de 1 MB. Uma vez setados, haverá uma região de 64kB para o código (CS), 64kB para dados (CS), 64kB para pilha (SS) e 64kB para dados extra (ES). Em qualquer momento que o programa necessite manipular a memória física, fora das atuais regiões de 64 kB, ele deve manipular o registrador de segmento apropriado (alterando-o através de instruções de transferência de dados. As figuras a seguir ilustram as regiões definidas pelos registradores de segmento sem e com sobreposição.



### MULTIPLICIDADE DE ENDEREÇOS

Um mesmo endereço pode ser acessado usando diferentes registradores de segmentos e diferentes registradores de offset



Exemplo: Endereço Físico = 10020 h

Possibilidades de pares SEGMENTO:OFFSET

$$1000h:0020h = 1000h * 10h + 20h$$

$$1001h:0010h = 1001h * 10h + 10h$$

$$1002h:0000h = 1002h * 10h + 00h$$

Um segmento tem até 64 kBytes

Como o segmento é multiplicado por 16 na formação do endereço

↓

O espaço mínimo entre dois segmentos consecutivos é 16 bytes

↓

No intervalo de 64 kBytes há  $65536 \div 16 = 4096$  possibilidades diferentes de endereçar a mesma posição física de memória.

## 9.7 Linguagem de Programação ASSEMBLY do 8086

Conceito de tipo → Cada símbolo (nome de variável, endereço, constante) tem um determinado tipo.

Tipos da linguagem assembly ASM-86:

**BYTE PTR** → referencia uma variável de 1 byte

**WORD PTR** → referencia uma variável de 1 word

**DWORD PTR** → referencia uma variável de 2 words

**NEAR PTR** → referencia o endereço de destino de uma instrução de desvio do tipo *near*

**FAR PTR** → referencia o endereço de destino de uma instrução de desvio do tipo *far*

**NUMBER** → constante de 16 bits

Obs.: near → altera somente IP (intra segmento)

far → altera IP e CS (inter segmento)

Exemplos:

**MOV AX, [BX]** → copia o word endereçado por **BX** para **AX** → o tipo do símbolo usado já está implícito (uma vez que AX é de 16 bits), não havendo necessidade de informar ao assembly.

**INC [BX]** → é necessário informar se deve ser incrementado o BYTE de offset BX ou o WORD de offset BX:

**INC BYTE PTR [BX]** → incrementa o byte cujo offset é o valor contido em **BX**

**INC WORD PTR [BX]** → incrementa word que se encontra nos endereços **BX** e **BX+1**.

### FORMATO DAS INSTRUÇÕES DO 8086

Instruções compactas → otimização do uso da memória e da velocidade de leitura das instruções

↓

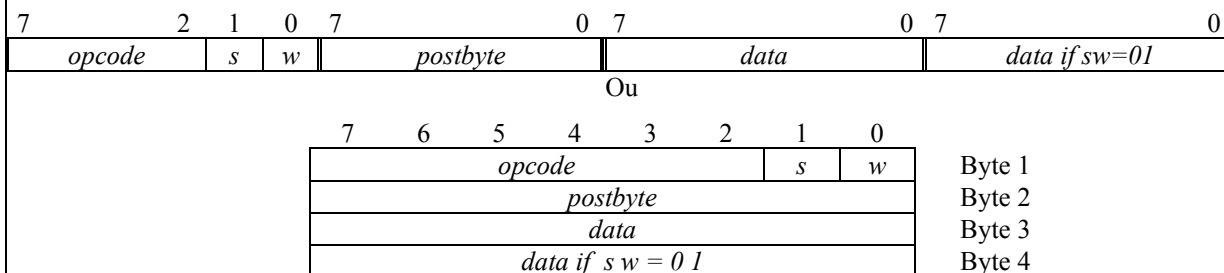
codificação mais complexa do que a do 8085

instruções com comprimento de 1 a 6 bytes

instruções não precisam ocupar exatamente 1 byte

bits restantes podem ser usados

Formato das instruções de transferência de dados:



*opcode* → código da operação

$w \rightarrow$  se  $w = 0 \Rightarrow$  instrução manipula byte; se  $w = 1 \Rightarrow$  instrução manipula word

*postbyte*:

7	6	5	4	3	2	1	0
mod		reg			r/m		

Se  $mod = 11 \Rightarrow$  o operando da instrução é um registrador, que é identificado em “r/m”

Se a instrução envolver dois registradores o campo “reg” identifica o segundo registrador.

## 9.8 Modos de Endereçamento e Endereços de Memória Efetivo

Como observado anteriormente, um endereço físico (20 bits) é constituído de duas partes:

- um segmento ou valor base;
- um offset ou endereço efetivo.

Um Endereço Efetivo (**EA**: *Effective Address*) é a parte offset do endereço físico. Ele é formado pela soma de um deslocamento (um número) com um registrador base (**BX** ou **BP**) e com um registrador índice (**SI** ou **DI**) (exemplo: **MOV AX, [BP+SI+4]**). A referência de memória com omissão de qualquer uma destas partes é também chamada de Endereço Efetivo (por exemplo, a parte offset do exemplo anterior poderia ser **[BP+4]** ou **[BP+SI]**).

Uma das vantagens de se escrever instruções com endereços efetivos é que um endereço de memória pode ser modificado baseado em condições do programa. Isto é particularmente útil quando se trabalha com tabelas (ou matrizes). Por exemplo, **BP** pode ser setado para apontar para a base (o início) da tabela e **SI** (que armazena o resultado de um cálculo) localiza um elemento na tabela.

A tabela abaixo sumariza os modos de endereçamento do 8088 e os registradores que estão disponíveis em cada modo.

Modo Endereçamento	Exemplo Mnemônico	Registrador de Segmento acessado	Operação
Imediato	MOV AX, 1000H	Code $\rightarrow$ CS	AX $\leftarrow$ 1000H
Registrador	MOV DX, CX	-	DX $\leftarrow$ CX
Direto	MOV AH, [1000H]	Data $\rightarrow$ DS	AH $\leftarrow$ [1000H]
Indireto por Registro (BX, BP, SI, DI)	MOV AX, [SI]	Data $\rightarrow$ DS	AL $\leftarrow$ [SI]; AH $\leftarrow$ [SI+1]
Indexado (SI ou DI)	MOV AX, [SI+6]	Data $\rightarrow$ DS	AL $\leftarrow$ [SI+6]; AH $\leftarrow$ [SI+7]
Baseado (BP ou BX)	MOV AX, [BP+2]	Stack $\rightarrow$ SS	AL $\leftarrow$ [BP+2]; AH $\leftarrow$ [BP+3]
Baseado e Indexado	MOV AX, [BX+SI]	Data $\rightarrow$ DS	AL $\leftarrow$ [BX+SI]; AH $\leftarrow$ [BX+SI+1]
Baseado e Indexado com deslocamento	MOV AX, [BX+SI+5]	Stack $\rightarrow$ SS	AL $\leftarrow$ [BX+SI+5]; AH $\leftarrow$ [BX+SI+6]
Strings	MOVSX	Extra, Dado $\rightarrow$ ES, DS	[ES:DI] $\leftarrow$ [DS:SI] se DF = 0 então: SI $\leftarrow$ SI+1 e DI $\leftarrow$ DI+1 se DF = 1 então: SI $\leftarrow$ SI-1 e DI $\leftarrow$ DI-1

### ENDEREÇAMENTO POR REGISTRADOR

Operando está em um registrador (de 16 bits AX a DI ou de 8 bits AH a DL)

Ex.: MOV AX, BX  $\rightarrow$  copia conteúdo de BX para AX

CMP AL, DL  $\rightarrow$  compara conteúdos de AL e DL (e seta flags)

### ENDEREÇAMENTO IMEDIATO

Operando faz parte da própria instrução. Usado para atribuir valores de 8 ou 16 bits a registradores

Ex.: MOV AX, 1000H → faz AX = 1000H

CMP SI, 0000H → compara conteúdo de SI com 0000H (e seta flags)

Obs.: Modo imediato não pode ser usado para os registradores de segmento, nem para as instruções PUSH

### ENDEREÇAMENTO ABSOLUTO OU DIRETO

O operando está na memória. A instrução contém o offset do endereço do operando (o segmento é DS).

Ex.: MOV AX, [1000H] → copia para AX o conteúdo localizado nos offsets 1000H e 1001 H, ou seja, AX = 1234 H.

Endereço	Dado
1000H	34H
1001H	12H
1002H	25H

### ENDEREÇAMENTO INDIRETO

O operando está na memória. A instrução, ao invés de conter o offset do endereço, como no caso direto, contém um registrador, que contém o offset do endereço do operando. Os registradores de offset podem ser BX, BP, DI ou SI (o segmento é DS).

Ex.: MOV AX, [BX] → copia para AX o conteúdo localizado no offset dado em BX. Se BX = 1000h, então AX = 1234h

Endereço	Dado
1000H	34H
1001H	12H
1002H	25H

Exemplo de aplicação: Tabelas, onde o endereço de leitura da tabela é dado através do registrador de offset, que é incrementado para varredura da tabela.

### ENDEREÇAMENTO INDEXADO

Uma constante, denominada base, e um registrador de índice são usados. A base e o índice são somados para a obtenção do offset do endereço. Os registradores de índice podem ser BX, BP, DI ou SI

Ex.: MOV AX, 0100H[BX] → copia para AX o conteúdo localizado no offset dado por BX + 0100H. Se BX = 1000h, então BX + 0100H = 1100H e AX = 1234h

Endereço	Dado
1100H	34H
1101H	12H
1102H	25H

Exemplo de aplicação: Tabelas, onde a posição inicial da tabela pode ser dada através da base; o índice pode então ser variado para varredura da tabela.



## ENDEREÇAMENTO BASEADO

É o contrário do endereçamento indexado, ou seja, o conteúdo de um registrador é copiado para uma posição variável indicada por um registrador e uma constante. Registradores usados: BX, BP, DI ou SI

Ex.: MOV [BX + 0100H], AX → copia conteúdo de AX para o endereço BX + 0100H. Se BX = 1000H, então BX + 0100H = 1100H. Se AX = 1234H, então o valor 34H será armazenado em 1100H e o valor 12H será armazenado em 1101H.

Endereço	Dado
1100H	34H
1101H	12H
1102H	xxH

Exemplo de aplicação: Tabelas, onde a posição inicial da tabela pode ser dada através da base; o índice pode então ser variado para varredura da tabela.

## ENDEREÇAMENTO RELATIVO

As instruções de desvio e chamadas de subrotina do tipo near ou short somam ou subtraem um valor de deslocamento para o desvio, ao invés de usar o endereço longo.

Offset de destino = offset da instrução seguinte + deslocamento

## 9.9 Interrupções do 8086

### 9.9.1 Estrutura de interrupção do 8086:

Enquanto o 8085 possui apenas 5 canais de interrupção, o 8086 é capaz de tratar de 256 interrupções diferentes, numeradas de 00h a FFh (ou 0 a 255 decimal). Ao contrário do 8085, a localização de cada interrupção não vem pré-definida no microprocessador; existe uma tabela, mostrada a seguir, com os vetores de interrupção, onde o endereço físico de cada interrupção é colocado.

Número da Interrup.	Endereço do vetor de interrupção	Endereço do tratador da interrupção
0	0000h:0000h	IP low h
	0000h:0001h	IP high h
	0000h:0002h	CS low h
	0000h:0003h	CS high h
1	0000h:0004h	IP low h
	0000h:0005h	IP high h
	0000h:0006h	CS low h
	0000h:0007h	CS high h
2	0000h:0008h	IP low h
	0000h:0009h	IP high h
	0000h:000Ah	CS low h
	0000h:000Bh	CS high h
<i>i</i>	0000h:4 <i>i</i>	IP low h
	0000h:(4 <i>i</i> +1)	IP high h
	0000h:(4 <i>i</i> +2)	CS low h
	0000h:(4 <i>i</i> +3)	CS high h
255	0000h:03FCh	IP low h

Endereço efetivo do vetor de interrupção	Endereço do tratador da interrupção
0000h:0000h	CS:IP
0000h:0004h	CS:IP
0000h:0008h	CS:IP
0000h:4 <i>i</i>	CS:IP
0000h:03FCh	CS:IP

	0000h:03FDh	IP high h
	0000h:03FEh	CS low h
	0000h:03FFh	CS high h


Obs.: O endereço efetivo da interrupção **INT  $n$**  é: **0000h:4n**

### 9.9.2 Interrupções Pré-definidas, reservada ou exceções

O 8086 possui três tipos de interrupção: Interrupções Pré-definidas, reservadas ou exceções, Interrupções por Hardware e Interrupções por Software.

As interrupções de 0 a 31 (ou 00h a 1Fh) são reservadas. Elas têm finalidades específicas para o 8086. Dentre as reservadas 5 interrupções já são pré-definidas no 8086:

Número da Interrup.	Endereço do vetor	Finalidade
00 H	0000h:0000h	Divisão por zero. É chamada sempre que o divisor de uma operação de divisão for zero.
01 H	0000h:0004h	Execução passo a passo. É chamada sempre que a Flag de Trap (T) estiver setada. Facilita a depuração de um programa.
02 H	0000h:0008h	NMI = Interrupção Não-Mascarável. Equivale à TRAP do 8085. Pino 17 do 8086.
03 H	0000h:000Ch	Breakpoints. Permite a execução de programas para depuração até um endereço especificado pelo programador.
04 H	0000h:0010h	Overflow. É chamada sempre que a Flag de Overflow estiver setada, indicando que o conteúdo do resultado não cabe no registrador de destino.
05 H – 1F H	0000H:0014H até 0000H:007C H	Interrupções que não são pré-definidas, mas são reservadas para o 8086. As diversas versões de sistemas usando o 8086 usam essas interrupções para finalidades específicas, mas não necessariamente idênticas entre os sistemas.

### 9.9.3 Interrupções por Hardware

São interrupções solicitadas através do pino INTR (pino 18). Nesse pino é conectado um controlador de interrupção que amplia para 8 os pedidos de interrupção. São interrupções mascaráveis através do bit IF do registrador de flags. A sequência de atendimento de uma interrupção por hardware é:

- 8086 envia primeiro pulso \*INTA para periférico, após receber pedido de interrupção. Após esse sinal reconhecimento de pedido de interrupção o periférico prepara-se para enviar o número da interrupção
- 8086 envia segundo pulso \*INTA para periférico, tendo como resposta do periférico o número da interrupção colocada no barramento de dados
- 8086 salva PSW na pilha
- Apaga flag TF (Trap – passo a passo) e, se for reservada, também IF (Interrupt Flag – desabilita interrupções por INTR)
- Salva CS (segmento) e IP (offset) na pilha
- Carrega novo IP a partir do endereço  $4*nn$
- Carrega novo CS a partir do endereço  $4*nn + 2$

### 9.9.4 Interrupções por Software

São interrupções solicitadas através da instrução **INT  $nn$** , onde  **$nn$**  deve estar entre 32 e 255 (as interrupções de 0 a 31 são reservadas). As interrupções geradas pela instrução **INT  $nn$**  não são mascaráveis. A sequência de atendimento de uma interrupção por software e também de uma interrupção reservada é:

- Salva PSW na pilha

- Apaga flag TF (Trap – passo a passo) e, se for reservada, também IF (Interrupt Flag – desabilita interrupções por INTR)
- Salva CS (segmento) e IP (offset) na pilha
- Carrega novo IP a partir do endereço  $4*nn$
- Carrega novo CS a partir do endereço  $4*nn + 2$

Embora existam 256 interrupções por software, cada uma delas pode ter até 256 sub-funções diferentes, ou seja, através das interrupções por software, pode-se executar  $256*256 = 65.536$  funções diferentes. Normalmente o registrador **AH** é usado para passar para a interrupção qual a função a ser executada. A tabela a seguir exemplifica algumas dessas funções.

Interrupção	Finalidade “macro”	Subfunção	Registadores	Descrição da subfunção
INT 10 H	Serviços de vídeo  (É uma função do BIOS)	00 H	AH = 00 H	Seta o modo vídeo
			AL = modo	AL = 03 $\Rightarrow$ VGA 80 x 25
		01 H	AH = 01 H	Seta tamanho do cursor
			CH: bit 7 = 0	
			Bits 6 e 5	00 $\rightarrow$ normal; 01 $\rightarrow$ invisível
			Bits 4 a 0	Linha mais alta do caractere para o cursor
			CL: bits 4 a 0	Linha mais baixa do caractere para o cursor
		02 H	AH = 02 H	Seta posição do cursor
			BH = 0 a 3	Número da página de vídeo
			DH = linha	00 H $\rightarrow$ linha superior
			DL = coluna	00 H $\rightarrow$ coluna da esquerda
INT 21 H	Serviços Gerais  (É uma função do DOS)	01 H	AH = 01 H	Lê caractere da entrada padrão, com eco.
			AL	Retorna o código ASCII do caractere lido
		02 H	AH = 02 H	Escreve caractere na saída padrão
			DL	Contém o código ASCII do caractere
		08 H	AH = 08 H	Lê caractere da entrada padrão, sem eco
			AL	Retorna o código ASCII do caractere lido
		09 H	AH = 09 H	Escreve na saída padrão textos terminados em “\$”
			DS:DX	Endereço de início do texto
		25 H	AH = 25 H	Seta vetor de interrupção
			AL	Número do vetor de interrupção a ser modificado
			DS:DX	Novo valor para o vetor a ser modificado
		31 H	AH = 31 H	Encerra programa e deixa residente (TSR)
			AL	Código de retorno
			DX	Número de parágrafos que devem ficar residentes
		4C H	AH = 4C H	Encerra programa
			AL	Código de retorno

## 9.10 Conjunto de Instruções do 8088/86

Há mais de 3000 opcodes distintos se considerados os vários modos de endereçamento. Abaixo estão listadas algumas instruções relativas a cada grupo funcional.

### 9.10.1 Instruções de Transferência de Dados

- movimentação de dados entre dois registradores,
- entre um registrador e posição de memória,
- entre registradores e portas [I/O].

**FORMATO:** *destino, fonte*

Formas genéricas possíveis:

MOV REG, memória	Copia conteúdo da posição de memória para o Registrador indicado
MOV memória, REG	Copia conteúdo do registrador indicado para a posição de memória
MOV REG, REG	Copia conteúdo de um registrador para outro
MOV memória, imediato	Carrega posição de memória com valor indicado
MOV REG, immediate	Carrega registrador com valor indicado

Formas genéricas possíveis para os registradores de segmento:

MOV SREG, memória	Copia conteúdo da posição de memória para o Registrador de segmento indicado
MOV memória, SREG	Copia conteúdo do registrador de segmento indicado para a posição de memória
MOV SREG, REG	Copia conteúdo de um registrador comum para um de segmento
MOV REG, SREG	Copia conteúdo de um registrador de segmento para um comum

Podem ser movimentados byte ou word, sendo que os registradores Ponteiros (BP e SP), Índices (SI e DI) e de Segmento (CS, DS, SS, ES) apenas podem ser acessados como word (16 bits).

**DEFINIÇÃO DE POSIÇÃO DE MEMÓRIA:** Ao escrever programas é conveniente dar nomes a posições de memória: para isto utiliza-se pseudo-instruções.

MEMBY DB ? ; variável MEMBY (do tipo "byte") com valor indefinido  
MEMBY DB 3A H ; variável MEMBY com valor definido e igual a 3A H

MEMWO DW ? ; variável MEMWO (do tipo "word") com valor indefinido  
MEMWO DW 21AB H ; variável MEMWO com valor definido e igual a 21AB H

Exemplos de mnemônicos com instruções de transferência

Mnemônico	Descrição
MOV AX, 0100 H	Carrega registrador de 16 bits AX com valor 0100 h
MOV DL, 23 H	Carrega registrador de 8 bits DL com valor 23 h
MOV CL, 'char'	Carrega registrador de 8 bits com código ASCII do caractere
MOV BP, A8	Carrega registrador de 16 bits BP com valor 00A8 h
MOV DS, AX	Registrador de segmento DS é carregado com conteúdo de AX
MOV AX, BX	Carrega AX com conteúdo de BX
MOV AL, BL	Carrega registrador de 8 bits AL com conteúdo de BL
MOV AX, CS	Carrega registrador AX com conteúdo do segmento CS
MOV AX, [BX]	Carrega AX com conteúdo do endereço de offset BX
MOV [BP], CL	Carrega posição de offset BP com valor contido em CL
MOV AH, [SI]	Carrega AH com valor da posição cujo offset é SI
MOV AX, [1000 H]	Faz $AL \leftarrow [1000 H]$ e $AH \leftarrow [1001 H]$
MOV CL, MEMBY	Carrega CL com posição da variável de 1 byte MEMBY
MOV CX, MEMWO	Carrega CX com posição da variável de 1 word MEMWO
MOV CX, [MEMWO]	Carrega CX com posição da variável de 1 word MEMWO
MOV BX, [BX+2]	Carrega BX com conteúdo da posição BX+2 e BX+2+1
MOV [BP+2], BH	Carrega posição BP+2 com conteúdo do registrador BH
MOV AX, [SI-6]	$AL \leftarrow [SI - 6]$ e $AH \leftarrow [SI - 5]$
MOV AX, [BX+SI]	$AL \leftarrow [BX + SI]$ e $AH \leftarrow [BX + SI + 1]$
MOV BX, [BP+SI+3]	$BL \leftarrow [BP + SI + 3]$ e $BH \leftarrow [BP + SI + 3 + 1]$
MOV BX, MEMBY[SI]	O mesmo que MOV BX, [SI+MEMBY]

Observações.:

- Um registrador de segmento **não** pode ser carregado usando o modo imediato. Exemplos **incorretos**: MOV CS, 1000, MOV DS, 3A00.
- Somente o registrador BX e os ponteiros e índices podem referenciar memória. Exemplos **incorretos**: MOV AX, [DX], MOV CX, [AX]
- Destino e fonte não podem especificar posições de memória ao mesmo tempo. Exemplo **incorreto**: MOV [BX], [SI]

#### Instruções Especiais de Transferência de Dados:

Mnemônico	Descrição
XCHG AX, BX	permuta registradores AX e BX $\rightarrow$ (AX $\leftrightarrow$ BX)
XCHG AL, CL	Permuta registradores AL e CL
XCHG DX, [SI]	Permuta DX com o conteúdo da posição de memória endereçada por SI. Neste exemplo DL $\leftrightarrow$ [SI] e DH $\leftrightarrow$ [SI + 1]
LAHF	AH $\leftarrow$ Flags low
SAHF	Flags low $\leftarrow$ AH
IN AH, 26	IN direto
IN AL, DX	IN indireto para as 65536 portas (0 - 65535 ou 0000 - FFFF).
OUT 26, AX	Aplica-se as mesmas regras da instrução IN.
OUT DX, AX	
LEA BX, MEMBY	Load Effective Address: carrega endereço efetivo. No exemplo, BX é carregado com o endereço efetivo de MEMBY (notar a diferença desta com a instrução "MOV BX, MEMBY")
LEA BX, [1000]	BX $\leftarrow$ 1000
LDS BX, dword ptr [SI]	Load Pointer using DS. No exemplo BX $\leftarrow$ [SI+1:SI], DS $\leftarrow$ [SI+3:SI+2]. Esta instrução é útil quando é preciso estabelecer novo endereço absoluto (composição endereço base + endereço de offset)
LES BX, dword ptr [SI]	Load Pointer using ES. No exemplo BX $\leftarrow$ [SI+1:SI], ES $\leftarrow$ [SI+3:SI+2]
XLAT	AL $\leftarrow$ [BX+AL]. Carrega registrador AL com o conteúdo (byte) da tabela iniciada em [BX] e com offset AL

Obs.:

- (a) A instrução IN somente pode ser usada para as primeiras 256 portas (0 a 255). Portas c/ endereço > 255 devem utilizar somente reg. DX (referência de I/O). Exemplo não permitido: IN AX, 3400.
- (b) A instrução IN deve utilizar somente o registrador AX. Exemplo não permitido: IN BL,DX.

**Exemplo 1:** O programa a seguir demonstra o uso de instruções de transferência de dados. O caractere "A" é escrito diretamente na memória de vídeo.

Mnemônico	Descrição
#MAKE COM#	Diretiva para o compilador gerar um arquivo ".com"
ORG 100h	Diretiva que indica o endereço inicial do programa: 0100 H
MOV AX, 0B800h	Carrega registrador AX com valor B800 H
MOV DS, AX	Copia valor de AX para DS, definindo segmento de dados
MOV CL, 'A'	Carrega CL com o código ASCII do caractere 'A', isto é, 41 H
MOV CH, 01011111b	Carrega CH com o valor binário 01011111 b = 5F H
MOV BX, 15Eh	Carrega registrador BX com valor 015E H
MOV [BX], CX	Copia conteúdo de CX na posição DS:BX, ou seja, B800:015E
HLT	Pára programa

**Exemplo 2:** Programa que demonstra o uso de variáveis

Mnemônico	Descrição
#MAKE COM#	Diretiva para o compilador gerar um arquivo ".com"
ORG 100h	Diretiva que indica o endereço inicial do programa: 0100 H

MOV AL, var1	Carrega registrador de 8 bits AL com valor 7 (variável var1)
MOV BX, var2	Carrega registrador BX de 16 bits com valor 1234 H (var2)
RET	Pára o programa
var1 DB 7	Define variável var1 como byte de valor 7 decimal
var2 DW 1234h	Define variável var2 como Word de valor 1234 hexadecimal

### 9.10.2 Instruções de Strings

- instruções para movimentação de blocos de dados (ou seja, vários bytes).
- reg. de seg. DS é utilizado como base e o reg. índice SI (*Source Index*) como offset para o cálculo do endereço fonte. Simbologia: **DS:SI**. (**OBS**: aceita registrador de segmento alternativo, por exemplo, CS, SS e ES).
- reg. de seg. ES é utilizado como base e o reg. índice DI (*Destination Index*) como offset para o cálculo do endereço destino. Simbologia: **ES:DI** (**OBS**: não aceita registrador de segmento alternativo. Deverá ser sempre ES).
- o flag DF (*Direction Flag*) é consultado nestas operações.

#### Instruções para movimentação de “strings”

Mnemônico	Descrição
MOVS BYTE PTR ES:[DI],[SI]	Copia bloco de dados de uma região para outra da memória. Origem: região DS:SI (a origem é sempre do segmento DS) Destino: região ES:DI (o destino é ES, mas poderia ser CS, DS ou SS)
MOVS WORD PTR ES:[DI],[SI]	Idêntico ao anterior, mas a movimentação é um Word por vez.
STOS BYTE PTR ES:[DI], AL	Armazena o byte em AL no endereço ES:DI. (Poderia ser CS, DS ou SS) ES:[DI] ← AL. Se D = 0 incrementa DI; se D = 1 decrementa DI
STOS WORD PTR ES:[DI], AX	Armazena o word em AX no endereço ES:DI. (Poderia ser CS, DS ou SS) ES:[DI] ← AL. Se D = 0 incrementa DI; se D = 1 decrementa DI
LODS BYTE PTR AL, ES:[DI]	Carrega AL com byte da posição ES:DI AL ← DS:[SI]. Se D = 0 incrementa SI; Se D = 1 decrementa SI
LODS WORD PTR AX, ES:[DI]	Carrega AX com word da posição ES:DI AL ← DS:[SI]. Se D = 0 incrementa SI; Se D = 1 decrementa SI
CMPS BYTE PTR ES:[DI],[SI]	Compara bloco de bytes de duas regiões de memória
CMPS WORD PTR ES:[DI],[SI]	Compara bloco de word de duas regiões de memória

#### Instruções para movimentação de “strings”

Mnemônico	Descrição
STOSB	Armazena o byte em AL no endereço padrão ES:DI. ES:[DI] ← AL. Se D = 0 incrementa DI; se D = 1 decrementa DI
STOSW	Armazena o word em AX no endereço padrão ES:DI. ES:[DI] ← AL ; ES:[DI + 1] ← AH. Se D = 0 DI ← DI + 2; Se D = 1 DI ← DI -2
LODSB	Copia em AL o byte localizado no endereço padrão DS:SI. AL ← DS:[SI]. Se DF=0 incrementa SI; Se DF=1 decrementa SI
LODSW	Copia em AX o word localizado no endereço padrão DS:SI. AL ← DS:[SI]; AH ← DS:[SI+1]. Se DF=0 SI ← SI+2; Se DF=1 SI ← SI -2
MOVSB	Copia bytes da região de origem padrão (DS:SI) para a região de destino padrão (ES:DI). Sendo a origem e o destino as regiões padrões (ao contrário da instrução MOVS), a instrução não precisa de argumentos. ES:[DI] ← DS:[SI]. Se D = 0, incrementa SI e DI; Se D = 1, decrementa SI e DI.
MOVSW	Copia words da região de origem padrão (DS:SI) para a região de destino padrão (ES:DI).

	Sendo a origem e o destino as regiões padrões (ao contrário da instrução MOVS), a instrução não precisa de argumentos. $ES:[DI] \leftarrow DS:[SI]; ES:[DI+1] \leftarrow DS:[SI+1]$ . Se $D = 0$ , $DI \leftarrow DI + 2$ e $SI \leftarrow SI + 2$ ; Se $D = 1$ , $DI \leftarrow DI - 2$ e $SI \leftarrow SI - 2$
CMPSB	Compara os bytes das posições DS:SI e ES:DI e atualiza Flags. $DS:[SI] - ES:[DI]$ .
CMPSW	Compara os words das posições DS:SI e ES:DI e atualiza Flags. $DS:[SI+1:SI] - ES:[DI+1:DI]$ .

### Uso do Prefixo REP (Repeat)

Precedendo as instruções de transferência de strings com "REP" faz com que estas instruções sejam repetidas o número de vezes igual ao conteúdo do registrador CX.

#### EXEMPLO 1: Uso de MOVSB e REP

Mnemônico	Descrição
#make_COM#	Diretiva para o compilador
ORG 100h	Diretiva para o compilador → programa começa na posição 100h
LEA SI, a1	Offset SI assume o endereço do primeiro valor da variável a1.
LEA DI, a2	Offset DI assume o endereço da variável a2.
MOV CX, 5	Registrador CX recebe valor 5
REP MOVSB	Instrução MOVSB é repetida até o registrador CX alcançar 0.
RET	Encerra programa
a1 DB 1,2,3,4,5	Variáveis
a2 DB 5 DUP(0)	A variável a2 terá 5 valores zero, conforme declarado em 5 DUP(0).

**EXEMPLO 2:** Um bloco de memória de 10 bytes com caractere "A3" é armazenado na memória, a partir do endereço físico E0000 H.

Mnemônico	Descrição
#make_COM#	Diretiva para o compilador
ORG 100h	Diretiva para o compilador → programa começa na posição 100h
MOV AX, 0E000 H	$AX = E000\text{ h}$
MOV ES, AX	Segmento especial $ES = AX = E000\text{ h}$
MOV DI, 0000	$DI = 0000\text{ h} \rightarrow ES:DI = E0000 + 00000 = E0000\text{ h}$
MOV AL, 0A3 H	Byte a ser armazenado é colocado em <b>AL</b>
CLD	<i>clear direction flag</i> (resseta flag "direção": modo auto-incremento)
MOV CX, 10	Faz $CX = 10 \rightarrow$ serão armazenado 10 bytes a partir de E0000 h
REP STOSB	Repete instrução 10 vezes
HLT	Pára programa

**EXEMPLO 3:** Copiar um bloco de memória de 1000 bytes do endereço físico A1000H para o endereço físico E1000 H.

Mnemônico	Descrição
MOV AX, 0A000 H	$AX = A000\text{ H} \rightarrow$ valor que será passado para registrador de segmento DS
MOV DS, AX	Segmento de dados $DS = AX = A000\text{ h}$
MOV SI, 1000 H	$SI = 1000\text{ h} \rightarrow DS:SI = E0000 + 00000 = A1000\text{ h}$
MOV AX, 0E000 H	$AX = E000\text{ h} \rightarrow$ valor que será passado para registrador de segmento ES
MOV ES, AX	Segmento especial $ES = AX = E000\text{ h}$
MOV SI, 1000 H	Registrador de offset $SI = 1000\text{ h} \rightarrow DS:SI = A1000\text{ H}$
MOV DI, 1000 H	Registrador de offset $DI = 1000\text{ h} \rightarrow ES:DI = E1000\text{ H}$
CLD	<i>clear direction flag</i> (resseta flag "direção": modo auto-incremento)
MOV CX, 03E7 H	Faz $CX = 03E7\text{ h} \rightarrow$ serão armazenado 1000 bytes a partir de E0000 h
REP MOVSB	Repete instrução 1000 vezes → serão copiados 1000 bytes

### 9.10.3 Instruções Lógicas

- referem-se a funções de lógica booleanas;
- cada instrução é realizada bit a bit;
- Há também instruções de rotação e deslocamento;
- Há 5 instruções booleanas: NOT, AND, OR, XOR, TEST.

Mnemônico	Descrição
NOT BX	BX é complementado (seus bits são invertidos)
NOT BYTE PTR [SI]	Byte apontado por SI é complementado
NOT WORD PTR [SI]	Word apontado por SI é complementado
AND CX, DX	Função AND entre CX e DX
AND BL, BYTE PTR [SI]	Função AND entre BL e conteúdo do Byte apontado por SI
AND AX, 8000 H	Função AND entre AX e 8000 H
OR CX, DX	Função OR entre CX e DX
OR BL, BYTE PTR [SI]	
OR AX, 8000	
XOR CL, DH	Função XOR entre CL e DH
XOR BX, WORD PTR [SI]	Função XOR entre BX e o Word apontado por SI
XOR AX, 8000	
TEST CX, DX	Semelhante ao AND, apenas não altera os operandos. É utilizado quando se deseja testar vários bits: se o teste do 1º bit fracassa, pode-se testar o 2º bit.
TEST AX, 3000	

**EXEMPLO:** Determine o estado do registrador AL e dos flags após a sequência de instruções:

```
MOV AL, 6D
MOV BH, 40
AND AL, BL
```

Resultado das Flags:

C = 0 → não houve transporte do bit 7 para o 8 na operação de 8 bits executada

Z = 1 → o resultado da operação é zero

S = 0 → o bit mais significativo após a operação é zero (número positivo)

O = 0 → não houve “overflow” na operação.

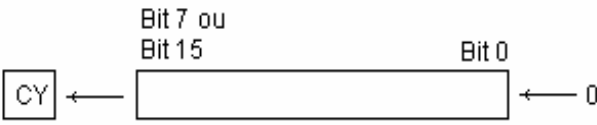
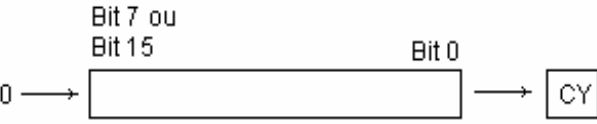
P = 1 → há um número par de bits “1” nos 8 primeiros bits, após a operação (número de 1s = 0)

A = 0 → não houve transporte do bit 3 para o bit


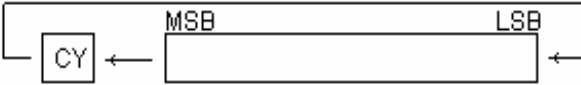
I = 1 → interrupção desabilitada

D = 0 → os registradores SI e DI serão incrementados nas operações com “string”

### Instruções de Deslocamento e de Rotação

SHL AL, 1 SHL BX, CL	<p>Deslocar à esquerda uma vez. Formato: SHL <i>destino</i>, <i>contador</i></p> <p>Deslocar à esquerda CL vezes.</p>  <p><b>Instrução SHL</b></p>
SHR AX, 1 SHR BL, CL	<p>Deslocar à direita uma vez.</p> <p>Deslocar à direita CL vezes</p>  <p><b>Instrução SHR</b></p>



ROL AL, 1 ROR CX, CL	Rodar à esquerda. Rodar à direita CL vezes.
	 <p><b>Instrução ROL</b></p>
RCL AL, 1	rodar a esquerda através do CY
	 <p><b>Instrução RCL</b></p>
RCR DL, CL	Rodar à direita através do CY CL vezes.

#### 9.10.4 Instruções Aritméticas

Mnemônico	Descrição
ADD SI, DX	soma de registradores
ADD BYTE PTR [BX], CH	
ADD DI, 6000	
ADC SI, DX	soma de registradores com carry
ADC BYTE PTR [BX], CL	
SUB AX, BX	subtração de registradores
SUB BL, 34	subtração de cte imediata
SBB AX, CX	subtração de registradores com borrow
SBB BL, 34	subtração de cte imediata com borrow
DAA	ajuste decimal para adição
DAS	ajuste decimal para subtração
AAA	ajuste ASCII para adição
AAS	ajuste ASCII para subtração
INC CL	incremento de registrador
INC WORD PTR [SI]	incremento de byte apontado por SI
DEC CX	decremento de registrador
DEC WORD PTR [SI]	
NEG DX	complemento de 2 de registrador
NEG WORD PTR [SI]	
CMP BL, BH	comparação entre dois registradores, BL e BH
CMP [BX], CX	comparação entre word apontado por BX e CX
MUL BL	multiplicação de byte: $AX \leftarrow AL * BL$
MUL CX	multiplicação de word: $DX:AX \leftarrow AX * CX$
IMUL BL	multiplicação de número com sinal
DIV BL	divisão de byte: $AX \leftarrow AL / BL$ (AL: quociente; AH: resto)
DIV CX	divisão de word: $DX:AX \leftarrow AX / CX$ (AX: quoc.; DX: resto)
IDIV BL	divisão de número com sinal

#### 9.10.5 Instruções de Desvio

Mnemônico	Descrição
Desvio Incondicional	
JMP [BX]	$IP \leftarrow [BX+1:BX]$
JMP BX	$IP \leftarrow BX$
JMP DWORD PTR [BX];	$IP \leftarrow [BX+1:BX] ; CS \leftarrow [BX+3:BX+2]$
Desvio condicional	

JNC LABEL	desvia para LABEL se flag carry = 0 (se não há carry)
JS LABEL	desvia para LABEL se MSB =1 (flag de sinal =1)
LOOP LABEL	decrementa CX e desvia para LABEL se CX ≠ 0
LOOPE LABEL	decrementa CX e desvia para LABEL se CX ≠ 0 e ZF = 1
PUSH e POP → Só pode ser aplicado a registrador de 16 bits	
PUSH CX	salva registrador CX na pilha
PUSH [DI+2]	salva posição de memória apontada por DI+2 e DI+3
POP DS	Recupera DS da pilha
PUSHF	Salva registrador de Flags na pilha
POPF	Recupera registrador de Flags da pilha
CALL e RETURN	
CALL DELAY	chamada de sub-rotina "DELAY"
CALL [BX]	chamada de sub-rotina iniciada pelo conteúdo de memória apontado por BX: IP ← [BX+1:BX]
CALL BX	chamada de sub-rotina iniciada pelo BX: IP ← BX
RET	retorno de sub-rotina
IRET	retorno de rotina de interrupção (restaura CS e IP)
Interrupções por Software	
INT <i>nn</i>	Guarda na pilha o endereço de retorno (CS:IP), antes de chamar a interrupção <i>nn</i> .

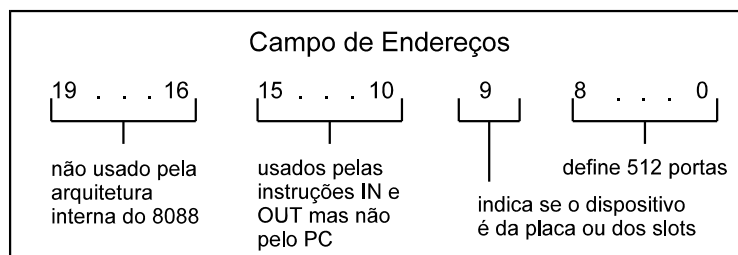
### 9.10.6 Instruções de Controle

Mnemônico	Descrição
STI	seta flag Interrupção
CLI	reseta flag Interrupção
STC	seta carry flag
CLC	reseta carry flag
CMC	complementa carry flag
STD	seta flag direção
CLD	reseta flag direção
HLT	para a CPU
WAIT	para a CPU até o pino test ficar ativo
NOP	sem operação (utilizado para gerar delay)
LOCK <i>instruction</i>	coloca Pino LOCK em "0" durante a execução da próxima instrução ( <i>instruction</i> )

## 9.11 Características de I/O do PC-XT

### 9.11.1 Divisão dos Endereços dos Dispositivos de I/O do PC-XT

Apesar de possuir 20 bits de endereço, a arquitetura do 8088 suporta apenas 64KB (65.536) endereços de dispositivos de I/O pois os bits de endereçamento A16 a A19 não são utilizados. A configuração usada no PC também não usa toda esta capacidade. Somente os 10 bits menos significativos do barramento de endereços são utilizados. Como  $2^{10} = 1024$ , este é o número de endereços de I/O disponíveis no PC (0000h a 03FFh). A figura abaixo mostra a divisão de endereços no PC.



O bit A9 tem um significado especial no projeto do PC: quando está inativo (0), os dados não podem ser recebidos (pelo micro) vindo dos slots. São habilitados apenas dados vindos de dispositivos de I/O instalados na própria placa mãe (*mother board*, *main board* ou *system board*). Quando ativo (1), são habilitados a recepção apenas de dados provenientes dos slots.

Tem-se assim 512 portas para uso interno do PC e 512 portas para uso de dispositivos conectados nos slots (ver figura abaixo).

A A 9 8	A A A A 7 6 5 4	A A A A 3 2 1 0	A A A A 9 8 7 6 5 4	A A A A 3 2 1 0
0 X	X X X X	X X X X	1 X	X X X X
000 - 1FF: PARA PLACA			200 - 3FF: PARA SLOTS	

Esta divisão aplica-se apenas à instrução IN (instrução de entrada de I/O). Para a instrução OUT (instrução de saída de I/O), todos os 1024 endereços podem ser utilizados nos slots.

Em sintonia com a restrição da instrução IN, o PC usa a primeira metade dos endereços para os dispositivos da placa principal (placa mãe) e a metade restante para os dispositivos dos slots (ver figura abaixo).

Endereço	Quantidade de Endereços	Função
0000 h 01FF h	512	Dispositivos da Placa Principal
0200 h 03FF h	512	Dispositivos dos Slots
0400 h FFFF h	64.512	Não usado no projeto do PC

A seguir serão apresentados os mapas de endereços de I/O do PC-XT. Deve-se observar que novas placas (por exemplo, placas de som ou de rede) poderão decodificar endereços que atualmente estão livres.

### 9.11.2 Endereços dos Dispositivos da Placa Principal (Placa Mãe)

Dispositivos de I/O residentes na placa principal fazem o serviço de interrupção, transferência de dados (DMA), contagem, etc. Na figura abaixo pode ser visto o mapa de endereçamento de I/O para a placa principal.

Endereço	Quantidade de Endereços	Função
0000 h 001F h	32	Chip de DMA (8239)
0020 h 003F h	32	Chip de interrupção (8259)
0040 h 005F h	32	Chip temporizador (8253)
0060 h 007F h	32	Chip PPI (8255)
0080 h 009F h	32	Registrador de páginas DMA
00A0 h 00BF h	32	Bit de interrupção NMI
00C0 h	320	Não codificado ou usado

01FF h

### 9.11.3 Endereços dos Dispositivos dos Slots

O conhecimento destes endereços torna-se a base de qualquer projeto de interface, já que representam o ponto de entrada e saída de dados (ver figura na página 16).

Nem todos os 512 endereços disponíveis no slots podem ser utilizados para interfaceamento pois vários endereços são previamente reservados para dispositivos comuns ao sistema. Os dispositivos de I/O que normalmente são conectados nos slots são joysticks, portas paralelas (impressoras), portas seriais (RS-232), adaptador gráfico (CGA, MDA, EGA, VGA), acionador de disquetes e de disco rígido. Para evitar problemas de incompatibilidade de placa de interface entre diferentes micros deve-se evitar a duplicação desses endereços.

O modo mais simples de decodificar um endereço para uma porta de I/O ou um grupo de endereços para projetos de interface é inspecionar o mapa de endereçamento e encontrar um bloco de endereços não utilizado, e então, construir um circuito decodificador adequado. O endereço decodificado deve ser acionado logicamente com os sinais de barramento  $\overline{\text{IOR}}$  (leitura de I/O) ou  $\overline{\text{IOW}}$  (escrita de I/O) para gerar entradas ou saídas de dados, respectivamente.

Outro sinal importante é o AEN (*Address ENable*), gerado pelo controlador de DMA. Quando ativo (1), o DMA assume o controle dos barramentos e o microprocessador é desabilitado. Portanto, tem a função de desabilitar a decodificação de endereços de I/O durante o ciclo de DMA: deve fazer parte da lógica de decodificação. Como mostrado mais à frente, a faixa de endereços de 0300h à 031Fh está livre para uso em placas de interface.

### 9.11.4 O Slot do PC (ISA 8 bits)

O slot do PC apresenta todos os sinais de interesse para a interface com qualquer dispositivo de I/O. Os sinais disponíveis nos slots são os sinais de endereçamento, de dados, clock, tensões, etc. A configuração dos sinais no slot ISA 8 bits pode ser visualizada na figura da página 17. Os sinais mais utilizados são dados a seguir:

**Alimentação:** Servem para alimentar os CI's digitais e analógicos). São eles:

<b>GND</b>	(B1, B10, B31)
<b>+5 V</b> (B3, B29)	$I_{\text{máx}} = 700 \text{ mA}$
<b>- 5 V</b> (B5)	$I_{\text{máx}} = 30 \text{ mA}$
<b>+12 V</b> (B9)	$I_{\text{máx}} = 100 \text{ mA}$
<b>- 12 V</b> (B7)	$I_{\text{máx}} = 50 \text{ mA}$

**Sinais de Controle de I/O:**

Servem para controlar o acesso aos CI's de I/O). Alguns principais sinais de controle são:

**IOW** (B13) - habilita escrita nos periféricos de I/O quando em nível lógico baixo;

**IOR** (B14) - habilita leitura dos periféricos pelo microprocessador quando em nível lógico baixo;

**RESET DRV** (B2) - sinal de reset utilizado para ressetar portas programáveis da placa de interface;

**AEN** (A11): utilizado na lógica de decodificação de endereços para indicar quando está ocorrendo um DMA;

**CLOCK** (B20): sinal de clock fornecido pela placa principal, frequência = 4,77 MHz;

**DADOS:** de D0 à D7 (A9 - A2): sinais provenientes do barramento de dados;

**ENDEREÇOS:** de A0 à A9 (A31 - A22): sinais provenientes do barramento de endereços.

Todos os demais sinais não são usados frequentemente, pois apenas placas de interface mais sofisticadas necessitam destes outros recursos.

Endereço	Quantidade de Endereços	Função
0200 h 020F h	16	Adaptador de Jogos
0210 h 021F h	16	Unidades de Expansão

0220 h 026F h	80	Reservada ou Não Documentada
0270 h 027F h	16	Porta Paralela ou Impressora # 2
0280 h 02EF h	112	Reservada ou Não Documentada
02F0 h 02FF h	16	Porta Serial #2
0300 h 031F h	32	Placa de Prototipo
0320 h 032F h	16	Adaptador de Disco Rígido
0330 h 036F h	64	Não Documentada
0370 h 037F h	16	Impressora Paralela Principal
0380 h 03AF h	48	Reservada ou Não Documentada
03B0 h 03BF h	16	Placa de Vídeo Monocromática
03C0 h 03CF h	16	Placa de Vídeo EGA
03D0 h 03DF h	16	Placa de Vídeo CGA
03E0 h 03EF h	16	Reservado
03F0 h 03FF h	16	Disco Flexível e Porta Serial

## 9.12 Decodificação de Endereços

A decodificação de endereços de I/O é semelhante ao mapeamento de memória e é utilizada em qualquer placa de interface. Para acessar periféricos de I/O, o PC-XT utiliza-se apenas 10 bits de endereços, de A0 a A9. Pode, portanto endereçar 1024 endereços diferentes. Decodificar um endereço ou grupo de endereços é gerar um único sinal, geralmente baixo, a partir dos sinais de endereço e controle. Este sinal é chamado de habilitação e é ligado nos CI's periféricos no pino CS (*Chip Select*) ou CE (*Chip Enable*). Para gerar este sinal pode-se utilizar CI's decodificadores e portas lógicas mais simples, decodificando os sinais de controle ( IOR, IOW e AEN) e os sinais de Endereços 99(A0 até A9). O decodificador 74LS138 é um dos decodificadores usados nesse processo. Um decodificador é 74LS154, cuja principal diferença do 74LS138 é possuir dois sinais de habilitação e quatro sinais de seleção, gerando, portanto 16 saídas diferentes.

## 9.13 Exemplos Gerais

A maioria dos exemplos dados a seguir são exemplos do emulador do 8086 chamado **Emu8086 v2.58**. Eles podem ser verificados através desse emulador.

1. Adiciona o conteúdo de dois registradores

Mnemônico	Descrição
#make BIN#	Diretiva do compilador para gerar um arquivo “.bin”
MOV AX, 5	Registrador AX = 0005 (decimal)
MOV BX, 10	Registrador BX = 0010 (decimal)
ADD AX, BX	Faz $AX \leftarrow AX + BX \rightarrow AX = 0005\text{ h} + 000A\text{ h} = 000F\text{ h}$ (15 dec)
SUB AX,1	Faz $AX \leftarrow AX - 0001\text{ h} \rightarrow AX = 000F - 0001 = 000E\text{ h}$ (14 dec)
HLT	Pára programa

2. Calcula a soma dos elementos do vetor V1 e armazena o resultado na variável V2.

Rótulo	Mnemônico	Descrição
--------	-----------	-----------

	#make BIN#	Diretiva do compilador para gerar um arquivo “.bin”
	MOV CX, 5	Número de elementos CE = 5 (decimal)
	MOV AL, 0	Registrador AL registrará a soma dos elementos (valor inicial = 0)
	MOV BX, 0	BX é o indexador. Valor inicial = 0.
Next:	ADD AL, V1[BX]	Faz $AL \leftarrow AL + [V1 + BX]$ . Conteúdo da posição V1 + BX.
	MOV V1[BX], BL	Modifica o conteúdo da posição V1 + BX com o valor de BL
	INC BX	Incrementa BX
	LOOP Next	Retorna para “Next” até o contador CX = 0. Decrementa CX automaticamente
	MOV V2, AL	Armazena conteúdo de AL na variável V2
	HLT	Pára programa
	V1 DB 4, 3, 2, 1, 0	Valores do vetor V1
	V2 DB 0	Valor inicial da variável V2

3. Carrega registradores com valores em notação binária, hexadecimal e octal.

Rótulo	Mnemônico	Descrição
	#make BIN#	Diretiva do compilador para gerar um arquivo “.bin”
	MOV AL, 00000101b	Carrega AL com valor binário correspondente a 5
	MOV BL, 0Ah	Carrega registrador com o hexadecimal de 10
	MOV CL, 10o	Carrega CL com o valor octal que corresponde a 8
	ADD AL, BL	Adicional o conteúdo de BL ao conteúdo de AL ( $5 + 10 = 15$ )
	SUB AL, CL	Subtrai o conteúdo de CL do conteúdo de AL ( $15 - 8 = 7$ )
	HLT	

## 9.14 Problemas Propostos

1. Rodar o programa “int21” do Emu8086 e verificar a entrada de textos pelo usuário e a impressão do mesmo no vídeo.
2. Rodar o programa “Advanced\_io” do Emu8086 e verificar o uso do display de 7-segmentos.
3. Criar um programa para fazer uma contagem hexadecimal crescente de 00 a FFh e mostrar a contagem no vídeo e no display de 7- segmentos.
4. Repetir o programa anterior para uma contagem hexadecimal de 0000h até FFFF h, ininterrupta.
5. Criar um programa para fazer uma contagem decimal crescente de 0 a 50, ininterrupta. Mostrar resultado no display de 7-segmentos.
6. Repetir o programa anterior para uma contagem decimal de 0 a 50000.
7. Criar um programa para fazer uma contagem hexadecimal decrescente de FFh a 00 ininterrupta e mostrar a contagem no vídeo e no display de 7- segmentos.
8. Criar um programa para fazer um contagem decimal decrescente ininterrupta de 50000 a 0. Mostrar no display de 7-segmentos.
9. Fazer um programa que simula um cronômetro decrescente que conta de 20:00 min até 00:00. Mostrar o resultado no vídeo, no formato apresentado.
10. Fazer um programa que simula um relógio no formato hh:mm:ss. Mostrar resultado no vídeo.
11. Criar um programa para mostrar três diferentes frases no vídeo, em linhas diferentes.
12. Fazer um programa para somar os elementos de dois vetores de 5 valores, cada. O resultado deve ser mostrado no vídeo.

13. Fazer um programa para ler dois valores pelo teclado, adiciona-los e mostrar o resultado no display de 7-segmentos.
14. Adapte o programa “stepper\_motor” do Emu8086 para que o motor fique girando 10 vezes para um lado e 10 vezes para o outro lado, ininterruptamente.
15. Adapte o programa “traffic\_lights” do Emu8086 de forma a simular um cruzamento simples, ou seja, permissão para seguir em frente ou virar à direita para as duas avenidas (sinal de dois tempos).
16. Adapte o programa “traffic\_lights” do Emu8086 de forma a simular um cruzamento onde os dois sentidos da avenida horizontal tenha permissão para seguir em frente, virar à esquerda ou virar à direita e os veículos da avenida vertical tenham permissão apenas para seguir em frente ou virar à direita (sinal de três tempos).
17. Adapte o programa “traffic\_lights” do Emu8086 de forma a simular um cruzamento onde os veículos de todas as pistas possam seguir em frente, virar à direita ou virar à esquerda (sinal de quatro tempos).

### 9.15 Referências Bibliográficas

- [1] ZILLER, Roberto M., ‘Microprocessadores: Conceitos Importantes’, Edição do autor, Florianópolis, SC, Brasil, 2000.
- [2] ZELENOVSKY, Ricardo, MENDONÇA, Alexandre, ‘PC: Um Guia Prático de Hardware e Interfaceamento’, Interciência, Rio de Janeiro, RJ, 1996.
- [3] Tutorial do Emulador emu8086 – <http://www.emu8086.com>