P. Mélusson

ABC DO MICRO PROCESSADOR

TEMPOS LIVRES

CULTURA E TEMPOS LIVRES

1. ABC do Xadrez, *Petar Trifunovitch e Sava Vukovitch*
2. Fisher/Spassky — Campeonato Mundial de Xadrez de 1972, *Petar Trifunovitch*
4. ABC do Bridge, *Pierre Jais e H. Lahana*
5. Guia Prático da Fotografia, *W. D. Emanuel*
6. ABC do Judo, *E. J. Harrison*
7. Como Fazer Cinema, *Paul Petzold*
8. Bridge Moderno, *Pierre Jais e H. Lahana*
9. Fotografia — Técnicas e Truques I, *Edwin Smith*
10. Estilos do Mobiliário, *A. Aussel*
11. Fotografia — Técnicas e Truques II, *Edwin Smith*
12. A Pesca Submarina, *António Ribera*
13. Teoria dos Finais de Partida, *Yuri Averbach*
14. Aprenda Rádio, *B. Fighiera*
15. Guia do Cão, *Louise Laliberté-Robert e Jean-Pierre Robert*
16. ABC do Aquário, *Anthony Evans*
17. Iniciação à Electricidade e à Electrónica, *Fernand Huré*
18. Os Transístores, *Fernand Huré*
19. Karaté I, *Albrecht Pflüger*
20. Iniciação ao Radiocomando dos Modelos Reduzidos, *C. Péricone*
21. Construa o seu Receptor, *B. Fighiera*
22. Montagens Electrónicas, *B. Fighiera*
23. O Berbequim Eléctrico, *Villy Dreier*
24. Cactos, *J. Nilaus Jensen*
25. Iniciação à Alta Fidelidade, *Peter Turner*
26. O Aquário de Água Doce, *Paulo de Oliveira*
27. ABC do Ténis, *Fonseca Vaz*
28. Karaté II, *Albrecht Pflüger*
29. ABC da Criação de Canários, *Curt Af Anehjelm*
30. Ginástica Feminina, *Sonja Helmer Jensen*
31. Cartomancia, *Rhea Koch*
32. Calculadoras Electrónicas de Bolso, *E. Dam Ravn*
33. O Pastor Alemão, *Gilles Legrand*
34. Xadrez – Teoria do Meio Jogo, *I. Bondarevsky*
35. Manual do Super 8 – I, *Myron A. Matzkin*
36. ABC da Criação de Periquitos, *Cyril H. Rogers*
37. O Livro dos Gatos, *Bärbel Gerber e Horst Bielfeld*
38. Manual do Super 8 – II, *Myron A. Matzkin*
39. ABC do Mergulho Desportivo, *Walter Mattes*
40. Circuitos Integrados/Aplicações Práticas, *F. Bergtold*
41. A Apicultura, *H. R. C. Riches*
42. ABC do Cultivo das Plantas, *H. G. Witham Fogg*
43. ABC da Criação de Pombos, *Kai R. Dahl*
44. Construção de Caixas Acústicas de Alta Fidelidade, *R. Brault*
45. Raças de Canários, *Klaus Speicher*
46. Jogos de Cartas, *Graciano Dolma*
47. Cocker Spaniels, *H. S. Lloyd*
48. ABC da Caça, *Fabián Abril*
49. Aprenda Televisão, *Gordon J. King*
50. Iniciação à Pesca, *Juan Nadal*

51. Basquetebol, *Marius Norregard*
52. Cães de Caça, *Santiago Pons*
53. Aprenda Electrónica, *T. L. Squires e C. M. Deason*
54. A Avicultura, *Jim Worthington*
55. A Produção de Coelhos, *P. Surdeau e R. Henaff*
56. ABC dos Computadores, *T. F. Fry*
57. Natação para Crianças, *John Idorn*
58. O Boxer, *Anni Mortensen*
59. Voleibol, *Ole Hansen e Per-Göran Persson*
60. Iniciação à Vela, *Donald Law*
61. ABC da Filatelia, *Jacqueline Caurat*
62. A Pesca à Beira-Mar, *J.-M. Boëlle e B. Doyen*
63. Enxerto de Árvores de Fruto, *Alejo Rigau*
64. A Cultura do Morangueiro, *Luís Alsina Grau*
65. Emissores-Receptores (Walkies-Talkies), *P. Duranton*
66. Iniciação à Fotoelectrónica, *Heinz Richter*
67. Doces e Conservas de Fruta, *Robin Howe*
68. A Criação de Hamsters, *C. F. Snow*
69. A Criação de Porcos, *Roy Genders*
70. Calendário do Horticultor, *Luís Alsina Grau*
71. Jogos Electrónicos, *F. G. Rayer*
72. Cultivo de Cogumelos e Trufas, *Alejo Rigau*
73. Aprenda Televisão a Cores, *Gordon J. King*
74. Gravação em Fita Magnética, *Ian R. Sinclair*
75. Poda de Ávores e Arbustos, *Roy Genders*
76. Como Treinar o Seu Cão, *E. Fitch Daglish*
77. Instrumentos de Medida e Verificação, *Heinrich Stöckle*
78. A Criação de Caracóis, *Matías Josa*
79. Rádio – Fundamentos e Técnicas, *Gordon J. King*
80. Como Fazer Gelados, *Sylvie Thiébault*
81. Iniciação à Jardinagem, *Noel Clarasó*
82. A Congelação dos Alimentos, *Suzanne Lapointe*
83. Windsurf – Prancha à Vela, *Ernstfried Prade*
84. Raças de Cães, *O. Hasselfeldt*
85. Rummy e Canasta, *Claus D. Grupp*
86. A Encadernação, *Annie Persuy*
87. Aprenda Electricidade, *Heinz Richter*
88. Taxidermia, Embalsamamento de Aves e Mamíferos, *Harry Hjortaa*
89. Jogging – Correr para Manter a Forma, *Werner Sonntag*
90. ABC da Cozinha Chinesa, *Sonya Richmond*
91. Jogos T. V., *C. Tavernier*
92. Amplificadores de Som, *Richard Zierl*
93. O Livro do Poker, *Claus D. Grupp*
94. Aprenda a Desenhar, *Rose-Marie de Prémont e Nicole Philippi*
95. O Minitrampolim na Escola, *Sonja Helmer Jensen e Klaus Dano*
96. Jogos de Luzes e Efeitos Sonoros para Guitarras, *B. Fighiera*
97. O Cultivo do Tomate, *Louis N. Flawn*
98. Pilhas Solares, *F. Juster*
99. A Criação Doméstica de Coelhos, *C. F. Snow*
100. Iniciação ao Futebol, *Wieland Männle* e *Heinz Arnold*

101. Horóscopos Chineses, *Georg Haddenbach*
102. Guia Prático de Marcenaria, *Charles H. Hayward*
103. Andebol, *Fritz* e *Peter Hattig*
104. Dispositivos Anti-Roubo, *H. Schreiber*
105. Perus, Pintadas e Codornizes, *Jerome Sauze*
106. Crepes – Doces e Salgados, *Florence Arzel*
107. Aperitivos e Entradas, *Myrette Tiano*
108. Ténis de Mesa, *Leslie Woollard*
109. Aprenda Surf, *R. Abbott e M. Baker*
110. Futebol — Técnica e Táctica, *Kurt Lavall*
111. A Vaca Leiteira, *Colin T. Whittemore*
112. O Cubo Mágico, *Josef Trajber*
113. O Perdigueiro Português, *José M. Correia*
114. Pizzas e Massas à Italiana, *Marieanne Rank*
115. O Cubo Para Quem Já o Faz, *Josef Trajber*
116. A Pirâmide Mágica, A Torre, O Barril do Diabo, *M. Mrowka-W. J. Weber*
117. Gansos e Patos, *Marie Mourthe*
118. Iniciação ao Kung Fu, *A. P. Harrington*
119. Electrónica e Fotografia, *Hanns-Peter Siebert*
120. O Livro da Fortuna, *Douglas Hill*
121. Construção de um Alimentador de Corrente, *Waldemar Baitingu*
122. Hóquei em Patins, *Francisco Velasco*
123. Técnicas de Tiro, *Anton Kovacic*
124. Aprenda a Tricotar, *Uta Mix*
125. ABC da Patinagem, *Christa-Maria e Richard Kerler*
126. A Pesca e os seus Segredos, *Armand Deschamps*
127. O Osciloscópio, *R. Rateau*
128. Guia Prático da Banda do Cidadão, *T. M. Normand*
129. Sumos e Batidos, *Manfred Donderski*
130. Introdução à Programação de Microcomputadores, *Peter C. Sanderson*
131. Aprenda Croché, *Uta Mix*
132. ABC do Microprocessador, *P. Mélusson*

P. MÉLUSSON

ABC DO MICROPROCESSADOR

EDITORIAL PRESENÇA

Título original:
INICIATION À LA MICROINFORMATIQUE:
LE MICROPROCESSEUR

Tradução de Conceição Jardim e Lúcio Nogueira
Fotografia da capa gentilmente cedida pela *LANDRY, Engenheiros Consultores, Lda. — LISBOA*

EDITORIAL PRESENÇA, LDA.
Rua Augusto Gil, 35-A — 1000 LISBOA

1
INTRODUÇÃO

Da Máquina de Calcular ao Computador
Do Computador ao Microcomputador

A ciência informática é actualmente, nas suas aplicações essenciais, o domínio das *Calculadoras* e dos *Computadores*.

Uma rápida observação da evolução passada desta ciência permitirá situar melhor o seu contexto de desenvolvimento. Encontra-se evidentemente ligada à evolução de certas técnicas, que vamos apresentar por ordem cronológica:

1. A ÉPOCA DA MECÂNICA PURA

Já anteriormente à era cristã, cerca de 500 anos antes de Jesus Cristo, os chineses inventaram o primeiro instrumento para realização de cálculos: o *ábaco*. Este instrumento permitiu-lhes contar rapidamente e com rigor. O ábaco ainda hoje é empregado na nossa época em certos locais do planeta. Serve de suporte material ao homem para a realização de um cálculo. Para o utilizar judiciosamente o homem deve empregar a sua força muscular e a sua inteligência.

Já não acontece o mesmo quando em 1642 *Blaise Pascal inventa a primeira máquina de calcular* que contém o seu próprio mecanismo de funcionamento.

No mesmo século *Jacquard* inventa o *tear programado por cartões perfurados*. Esta ideia de programação é já um prelúdio do funcionamento de certas unidades periféricas actualmente utilizadas nos computadores.

2. A ÉPOCA DOS MECANISMOS ELÉCTRICOS

Em 1924 o engenheiro norueguês *Bull* deposita uma patente de um conjunto electro-mecânico, e em 1936 os americanos utilizam uma máquina de calcular cujo automatismo é assegurado por *relés electro-mecânicos*.

3. A ÉPOCA DA ELECTRÓNICA

Esta época pode ser dividida em três fases sucessivas, em função dos componentes existentes:

a) **Fase das válvulas.** Iniciar-se-á em 1950, com o nascimento do *Univac 1*. É o primeiro computador comercial. Distingue-se do calculador por possuir um jogo de instruções que asseguram a possibilidade de utilizar numerosos programas. A elaboração de programas permite, com efeito, à máquina resolver problemas de tipos diferentes.

b) **Fase dos semicondutores** (díodos e transístores): Inicia-se por volta de 1960, data em que surgem no mercado os primeiros computadores completamente transistorizados. Incluem geralmente entre 5000 e 10 000 transístores e um número equivalente de díodos.

c) **Fase dos circuitos integrados.**

Trata-se em particular dos circuitos designados por **LSI,** ou seja, «large scale integration». Podem conter, cada um deles, entre 15 000 e 20 000 junções difundidas numa pastilha de salício com algumas dezenas de milímetros quadrados de superfície.

Graças a estes circuitos vai-se poder passar do domínio da informática para o da microinformática. Surge primeiramente a era da *máquina de calcular de algibeira,* e em seguida a actual, ainda nos seus inícios mas já evoluída, do *microcomputador*.

A alma deste microcomputador é o *microprocessador*. É o órgão encarregue de resolver as operações aritméticas e as funções lógicas de acordo com informações tratadas de *maneira sequencial*.

Em 1971 surge o primeiro microprocessador no mercado. Em 1973 aparece o primeiro microcomputador equipado com um microprocessador: o *Micral,* da firma francesa **R2E.**

O computador é uma máquina de grandes dimensões

cujo preço de custo é bastante elevado. É-lhe no entanto possível reduzir com facilidade a quantidades e a qualidades os mais diversos problemas, devendo ainda, para ser rentável, funcionar regularmente em pleno rendimento e evitar os tempos mortos. Por estas razões, só pode ser usado por firmas comerciais, industriais ou financeiras de grandes dimensões.

Nas firmas menos importantes, podem-se instalar terminais. Estes terminais encontram-se ligados ao computador central através de linhas convenientemente montadas, alugadas por essas firmas ao possuidor do computador central. Neste caso os terminais trabalham em *tempo partilhado* ou, como se diz em inglês, em regime de *time sharing*.

Os *terminais* tomam, segundo uma ordem de prioridades estabelecida em função da urgência e da qualidade das suas necessidades, uma parte do tempo do computador central.

No entanto, é ainda necessário, admitir aqui que um tal sistema de exploração exige no seu conjunto uma organização massiva e cara.

O microcomputador é pelo contrário, quando comparado ao computador de grandes dimensões, um modelo de preço de custo claramente inferior, graças, em particular, à flexibilidade da organização das suas estruturas, que se podem adaptar com facilidade, depois de um estudo sério, às necessidades específicas de cada empresa.

As suas pequenas dimensões e a sua faculdade de adaptação economicamente muito vantajosas, baseadas na simplicidade dos seus circuitos, devem permitir nos próximos anos um desenvolvimento sem precedentes em todos os domínios da electrónica, tanto no dos circuitos lógicos como no âmbito profissional das *telecomunicações,* dos *automatismos industriais,* da *televisão,* dos *electrodomésticos* e em muitos outros sectores a que ao longo dos tempos serão aplicados.

2

O CÉREBRO HUMANO E O COMPUTADOR: CÉREBRO ROBOT

O computador é um cérebro robot que apresenta analogias com o cérebro humano.

A fim de os definir, é portanto necessário imaginar o que é o cérebro humano e estabelecer paralelismo entre este e o cérebro robot de um computador.

É o que iremos tentar fazer.

No plano material, é um órgão. É a parte *hardware* do cérebro ou, por outras palavras, a parte *concreta* e palpável do cérebro.

Devemos agora tentar compreender a maneira como, neste órgão, neste «hardware», são encaminhadas as ideias, se elabora o raciocínio que conduz ao comando do nosso corpo.

Toda esta elaboração das ideias constitui o *software* do cérebro, ou seja a sua parte *abstracta,* não palpável.

Assim, para o *técnico de informática:*

O concreto, ou seja o circuito, tudo o que se pode ver, tocar, é o **hardware;**

O abstracto, ou seja a parte imaterial da ciência, a matéria cinzenta, é o **software.**

O pensamento humano, para se forjar e exprimir tem necessidade de dois centros constitutivos distintos:

O primeiro centro é a *memória.* É o lugar onde todas as informações adquiridas são classificadas em «pequenos compartimentos» perfeitamente bem ordenados.

A capacidade total da memória é «variável» mas não «infinita».

Além disso, certas informações, ou *dados,* ou ainda, em inglês, *data,* devem ser armazenados definitivamente, sem nunca serem limpos da memória. Estes «dados» serão cuidadosamente «empilhados» numa memória **ROM**(Read Only Memory). Trata-se de uma memória apenas de *leitura, indestrutível.*

Esta memória fixa é por vezes designada por *memória morta.*

Pelo contrário, outras informações, que não constituem dados principais, mas apenas resultados da composição ordenada daqueles, não têm necessidade de ficarem eternamente guardadas; podem até só ser necessárias uma vez. Deve portanto ser possível:

1. Servir-nos delas:

2. Apagá-las de modo a deixar livres os compartimentos necessários à acumulação de novos dados.

Todas estas informações fugitivas ou temporárias serão *escritas* e depois *apagadas* nas **RAM** (Random Access Memory).

Por vezes, estas memórias são também chamadas, por oposição às memórias mortas, *memórias vivas.*

Encontram-se no cérebro robot, sendo memórias de onde se pode extrair os dados para os ler, apagando-os em seguida a fim de deixar espaço livre que será em seguida utilizado para armazenar novas informações.

A leitura de um dado extraído ou a escrita de um dado introduzido realiza-se graças à função *Read/Write* do computador.

O segundo centro do nosso cérebro é o do *raciocínio.*

É nele que se elabora o tratamento das ideias. Este centro é capaz de efectuar:

— Um raciocínio *lógico* (resolve os problemas relativos às *funções lógicas*);

— um raciocínio de cálculo que os informáticos designam por *algoritmo* (um processo de cálculo). Resolve os problemas relativos às *operações aritméticas.*

Este centro é designado no microcomputador, ou seja no nosso cérebro robot, por *microprocessador.*

As abreviaturas usadas para o designar, são:

A **CPU** (Central Processing Unit), ou seja, a unidade central de tratamento do computador, ou ainda a **MPU** (microprocessing Unit), ou seja, unidade de tratamento do microcomputador.

O centro do próprio microprocessador, onde se executa verdadeiramente a resolução dos problemas lógicos e aritméticos, designa-se pelo termo **ALU** (Arithmetic and Logic Unit).

Para efectuar um raciocínio, por exemplo a operação aritmética (4 + 5), o centro de raciocínio (o microprocessador do nosso cérebro robot) deve procurar nas memórias as informações 4 e 5, e finalmente a *instrução* (+). Para tal é necessário que existam fibras condutoras entre o MPU e os locais das memórias onde estão armazenados os diferentes dados. Estas fibras são designadas **BUS**, ou também *feixes de linhas.*

1. O *Adress Bus* e o *Data Bus*

O Adress Bus (bus de endereços) é o feixe de linhas que liga o MPU aos endereços onde estão contidos na memória os dados procurados.

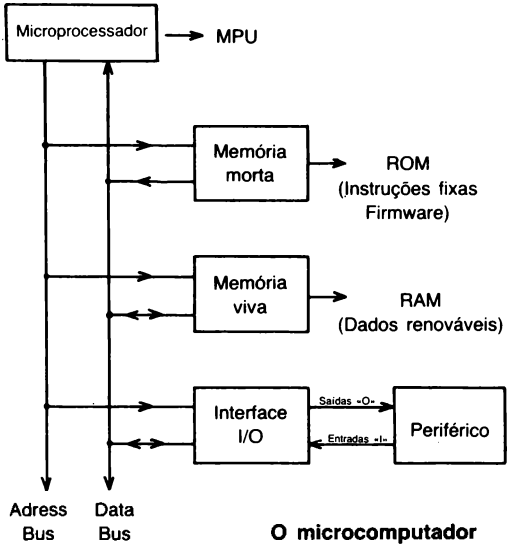


Figura 1

O Adress Bus é geralmente *unilateral*, ou seja, os dados só se movimentam nas suas linhas num único sentido:

MPU —› MEMÓRIA

O Data Bus (bus de dados) é geralmente *bilateral*, ou seja, pode funcionar em ambos os sentidos: **MEMÓRIA —› —› MPU** quando a informação foi procurada pelo MPU no seu endereço, ou ainda no sentido **MPU ‹— MEMÓRIA** quando acaba de ser obtido na unidade central de tratamento o resultado de uma operação e se pretende guardá-lo num endereço livre de uma memória RAM.

Na nossa operação, 4 e 5 são os «data», ou dados aritméticos. O «algoritmo» ou «processo de cálculo» é definido pelos sinais + e =. Estes sinais constituem a «instrução» do cálculo a efectuar. A organização do trabalho do computador é feita por *instruções*. O conjunto das instruções de um microprocessador pode ser designado por *jogo de instruções*.

O conjunto das instruções necessárias para resolver uma função chama-se *programa*.

Certos programas são complexos. A fim de os simplificar, pode-se dividi-los em *sub-rotinas* típicas. Aquele que concebe um programa é designado por *programador*. Todas as operações indicadas num programa são realizadas sequencialmente segundo uma ordem bem estabelecida.

A execução das instruções é sintonizada temporalmente e controlada por *sinais de relógio* ou *clock* que podem ser obtidos a partir de um «quartzo».

Os órgãos de comando ou de controlo das operações podem ser *activados* por sinais a eles conduzidos por um «bus» designado por *bus de comando* ou *de controlo*.

Toda esta organização de um microcomputador tem como suporte o *sistema binário* que se presta facilmente à utilização de um sinal eléctrico que trabalha no regime de *tudo ou nada*.

As funções lógicas e os cálculos podem ser expressas neste sistema, que apenas utiliza os algarismos 1 e 0 («tudo») ou «nada») para designar todo o conteúdo de um dado programa.

A informação elementar do sistema binário é designada pela palavra *bit* (Binary Digit).

Uma palavra é composta por um certo número de bits. Quanto mais bits puder comportar maior é o seu rigor.

Na inscrição de uma palavra na memória, o bit da direita é o **LSB** (Less Significant Bit), ou seja, o *bit menos significativo.*

O bit mais à esquerda é o **MSB** (Most Significant Bit), ou seja, o *bit mais significativo.*

Os microprocessadores comportam em geral palavras de 4 a 8 ou 16 bits. Actualmente, a maior parte destas máquinas são estruturadas com palavras de 8 bits.

Uma palavra de 8 bits é designada por *byte* ou ainda *octeto.*

Finalmente, nenhum cérebro robot pode viver apenas em circuito fechado, tal como de resto o cérebro humano.

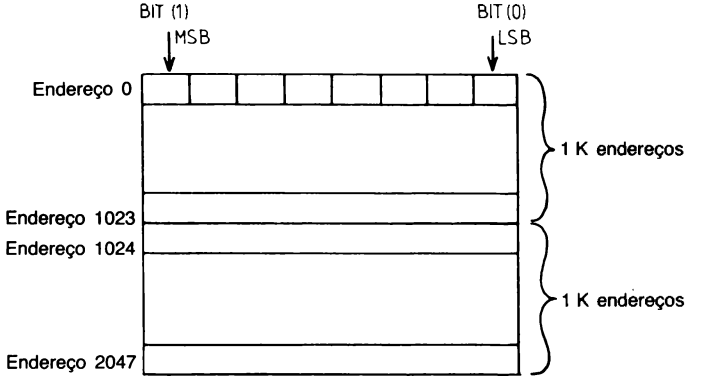


Figura 2 — Representação de uma memória de 2 K palavras de 8 bits.

Com efeito, este último necessita de trocar as suas ideias com os seus semelhantes. Relativamente a cada um de nós, os nossos vizinhos constituem «periféricos». Quando alguém nos põe uma questão, esta surge-nos como o resultado do seu próprio pensamento. Este não pode ser completamente igual ao nosso, e para podermos compreender e relacionar o

seu tratamento lógico ao nosso, necessitamos de uma *interface* entre o nosso centro de raciocínio, ou seja o nosso CPU, e o do periférico, a qual adapta o nosso sistema de raciocínio ao do nosso interlocutor.

Esta interface pode ter vários nomes:

É o **I/O** (input/output, entrada/saída) de:

Input = entrada dos dados provenientes de um periférico.

Output = saída dos dados para um periférico.

É ainda o **PIA** (Peripheral Interface Adapter)

Recebemos normalmente de um periférico um pedido de dados ou um pedido de cálculo ou de «lógica».

Se se trata de um simples pedido de dados, o nosso cérebro robot pode ter a possibilidade de colocar em posição de entrada — **Three state control** (*TSC*) — o CPU, ou seja, pôr a entrada em estado de impedância infinita. Pelo contrário, o PIA liga-se directamente à memória de um microcomputador que inclui os dados exigidos. Diz-se então que o PIA trabalha em **DMA** (Direct Memory Access), ou seja, em acesso directo à memória.

Se se trata de um pedido de cálculo ou de lógica, o PIA é ligado directamente ao nosso MPU.

No momento em que o nosso MPU recebe um pedido proveniente do periférico através do PIA e está já a executar um programa, podem ser consideradas duas soluções:

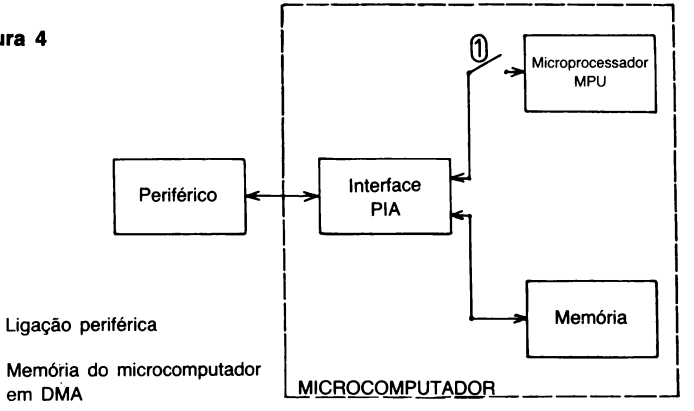
Solução 1. O problema que está a ser resolvido é mais importante do que o pedido exterior. Continua portanto a ser executado normalmente até ser terminado, e o pedido exterior fica à espera durante algum tempo.

Solução 2. O pedido proveniente do exterior é mais urgente do que a resolução do programa actualmente tratado no MPU.

Neste caso, produz-se uma *interrupção*. O programa que está ser resolvido pára. O ponto onde se chegou pode ser guardado na memória a fim de ser conservado numa memória especial designada **LIFO** (last in, first out), isto é, a última informação a entrar é a primeira a sair. O pedido exterior é então tratado, sendo o programa anterior retomado no ponto onde estava quando o pedido exterior acaba de ser satisfeito.

E já que comparámos o homem à máquina, resumamos em algumas palavras esta analogia:

Figura 4



Ligação periférica

Memória do microcomputador em DMA

① → em TSC (Impedância infinita)

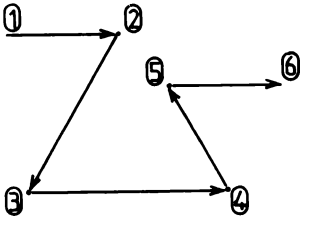


Figura 3

1 — programa em tratamento;
2 — intervenção de um fenómeno exterior exigindo uma interrupção;
3 — o programa é interrompido;
4 — a interrupção é tratada;
5 — volta-se ao programa 1.
6 — o programa 1 é retomado, terminando-se o seu tratamento.

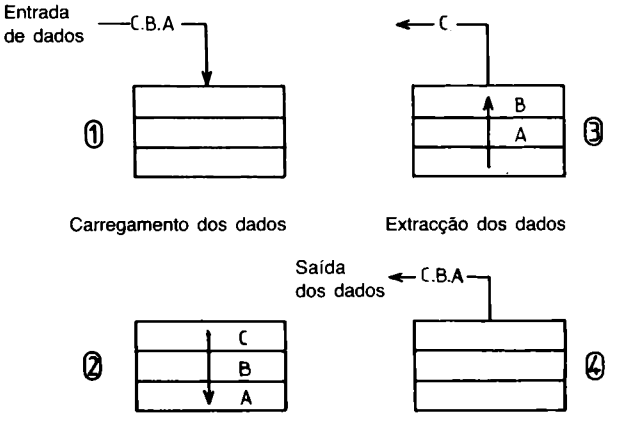


Figura 5 — A última informação introduzida é a primeira extraída, pois os dados são «empilhados» arbitrariamente por cima e esvaziados igualmente por cima.
Diz-se assim que uma memória LIFO é uma «pilha», ou ainda um registo de empilhamento.

Suponhamos que se pretende realizar a soma de vários números:

O órgão periférico do microcomputador neste sistema pode ser assimilado à folha de papel onde está inscrita a lista dos números a adicionar.

A folha de papel é com efeito a *fonte* dos dados a tratar. Quando o resultado for escrito por baixo, essa folha será igualmente o local de *destino*.

Os olhos que observam os dados a tratar e a mão que escreve o resultado constituem o *sistema de interface* **(I/O).**

Os olhos conduzem os números à *memória*.

As memórias, quer se trate da do homem ou da do computador, desempenham o mesmo papel. Contêm ambas o *algoritmo* da soma.

A porção do cérebro que realiza a soma é o seu centro de raciocínio; o **CPU** do computador realiza a mesma função.

E no entanto, existe entre o homem e a máquina duas diferenças essenciais:

1. A máquina tem a possibilidade de *resolver sem cansaço e a grandes velocidades tarefas repetitivas,* o que o homem não pode fazer.
2. *O homem tem a ideia de criação* graças à sua intuição e à sua imaginação. A máquina encontra-se completamente desprovida desta ideia, o que obriga o homem a ditar-lhe um programa em todos os detalhes. Assim, a máquina encontra-se ainda a um nível bastante inferior ao do homem.

3
COMPUTADOR — CALCULADORA MICROPROCESSADOR
O QUE É VOCÊ?

Um computador é uma máquina capaz de tomar decisões e realizar operações aritméticas ou lógicas disponíveis na sua saída de acordo com os programas apresentados à sua entrada.

Podem-se distinguir três tipos diferentes de computadores:

1. OS COMPUTADORES ANALÓGICOS

Nestes computadores, a informação é apresentada e tratada *sob forma analógica.*

Exemplo de uma forma analógica:

Pretende-se resolver a função: $y = ax + b$.

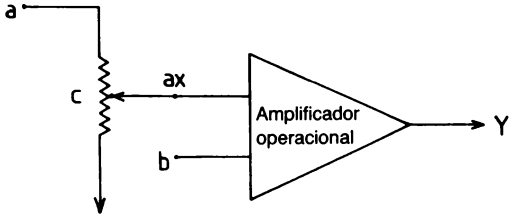


Figura 6

O esquema da figura 6 mostra o circuito (hardware) de computador que permite resolver a função numa *única sequência.* A variável é x, e a amplitude eléctrica do seu sinal é proporcional à determinação da posição do contacto do potenciómetro (ponto C).

Pode-se portanto afirmar que existe uma proporcionalidade entre o valor calculado da função y e o da amplitude do sinal eléctrico que lhe corresponde.

2. OS COMPUTADORES DIGITAIS

Nestes computadores, a informação é apresentada e tratada numa *forma digital* (sequência de níveis eléctricos B (baixo) e H (alto) definidos pelos números binários 0 e 1).

Nos parágrafos seguintes serão dados exemplos de tratamento em forma binária.

3. COMPUTADORES HÍBRIDOS

Nestes computadores encontram-se combinados os dois tipos de informações eléctricas (analógico e digital).

4. COMPUTADORES E CALCULADORAS DIGITAIS

Um computador, tal como uma calculadora, de tipo «digital», é capaz de armazenar e tratar em forma binária as informações.

Interessa portanto definir mais claramente a diferença que existe entre a calculadora e o computador.

Uma calculadora realiza apenas a função que lhe foi atribuída, de forma repetitiva, ao serem montados os circuitos que possui. É um sistema hardware.

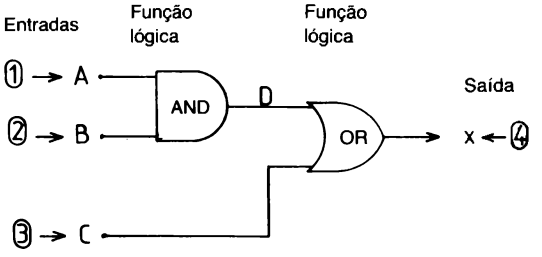
As suas ligações internas, associando um certo número de circuitos digitais elementares, *portas* e *flip-flops*, constitui o que se pode designar por uma *lógica por fios condutores.*

Esta lógica obtida por simples ligações eléctricas é realizada geralmente num *circuito impresso.*

Exemplo deste tipo de lógica: pretende-se resolver numa calculadora digital a função Y = A.B + C.

O esquema da figura 7 mostra o circuito lógico que permite resolver esta função *numa única sequência.*

Se se apresentam nas entradas 1, 2 e 3, num *modo para-*



lelo, ou seja, *simultaneamente,* as informações A, B e C, obter-se-á na saída 4 o valor *Y* procurado.

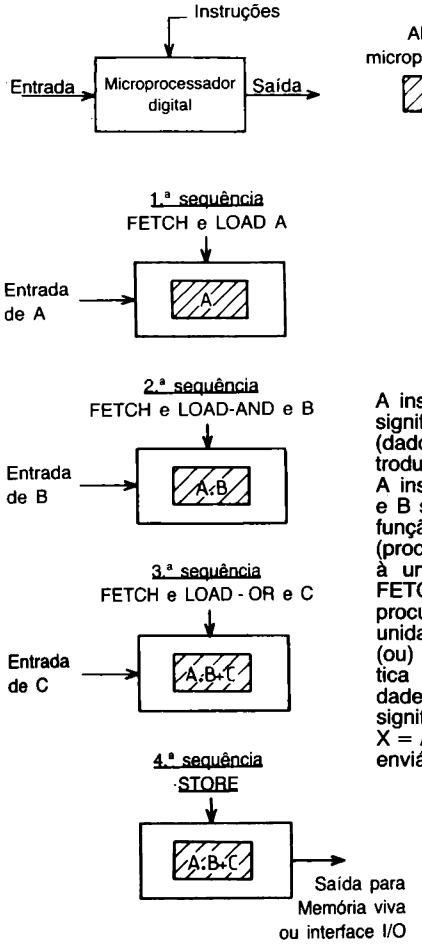
Um computador pode realizar qualquer tarefa ditada por uma série de instruções (sistema software).

É esta a diferença principal entre ele e uma calculadora. Por outro lado, a sua utilização torna-se mais complicada pelo facto de, para programar uma sequência de instruções, se tornar anteriormente necessário conhecer a linguagem assimilada pelo computador.

Exemplo de sequências de instruções programadas:

Retomemos a nossa função X = A.B + C, e procuremos resolvê-la utilizando um computador.

O esquema sinóptico das sequências de instruções da figura 8 mostra-nos como é resolvido este problema.



A instrução «FETCH» e «LOAD» A significa: «procurar» e «introduzir» A (dado existente na memória viva, introduzido na unidade aritmética). A instrução FETCH e LOAD «And» e B significa: procurar e introduzir a função lógica «and» (e) e o dado B (procurados na memória e passados à unidade aritmética). A instrução FETCH e LOAD «Or» e C significa procurar na memória e introduzir na unidade aritmética a função «OR» (ou) e o dado C. A unidade aritmética já calculou portanto a quantidade A.B + C. A instrução «Store» significa «armazenar» o resultado X = A.B + C na memória viva ou enviá-lo às interfaces I/O.

Figura 8

Através do esquema da figura 9 é possível resumir de uma maneira muito simples a diferença essencial de funcionamento entre a «lógica por ligações eléctricas» e um computador.

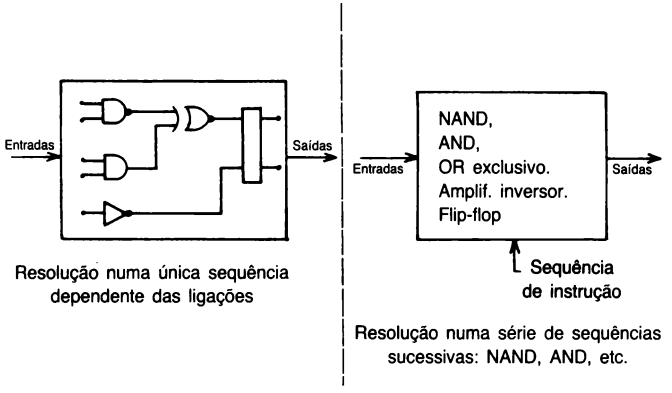


Figura 9 — À esquerda, exemplo de lógica hardware. À direita, exemplo de lógica software.

No comércio encontram-se três tipos de computadores digitais, classificados em função das aplicações consideradas:

— os grandes computadores;
— os minicomputadores;
— os microcomputadores

Todos têm a mesma estrutura principal, e o que os diferencia entre si é essencialmente a sua dimensão, ou seja:

— a *sua capacidade de memória;*
— a *dimensão de palavra* que aceitam (número de bits de uma palavra).
— *o número e a qualidade* das instruções neles disponíveis.
— *a velocidade de execução* das instruções.

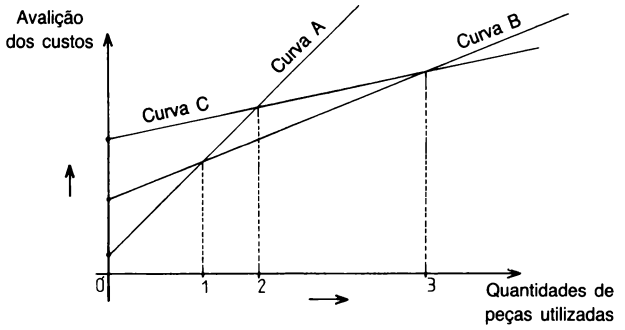


Figura 10

Distinguimos as abcissas:
Na zona 0, 1 — Preferência pela lógica por ligações eléctricas (A)
Na zona 1, 3 — Preferência pelo microprocessador (B)
Na zona acima de 3 — Preferência pelo Custom Design (C)
Deve-se notar que na zona 1, 2, se de facto é aconselhado o microprocessador, pode-se no entanto optar por uma lógica por ligações, mais barata que o custom design.

Estas diferentes potencialidades conduzem a grandes variações de preços e de campo de aplicações entre as três categorias.

Iremos considerar sobretudo, neste livro, os microprocessadores digitais, que são aqui objecto de um pequeno curso de iniciação.

Falemos portanto do microprocessador, alma do microcomputador.

É importante fornecer agora uma definição mais rigorosa que a anteriormente dada.

A França propôs à Comissão Electrónica Internacional a seguinte definição, bastante completa:

«Um microprocessador de circuito integrado é um circuito integrado numérico capaz de, após identificação de uma sequência de operações, efectuar operações de tratamento da informação».

Um microprocessador:

1. Aceita informações de dados codificados para tratamento e/ou armazenamento.
2. Efectua o conjunto das operações lógicas, aritméticas e de movimento sobre as informações de entrada e/ou sobre todas as informações guardadas em memória.
3. Restitui na saída as informações codificadas resultantes do tratamento efectuado.
4. Pode receber e/ou fornecer informações relativas ao seu funcionamento ou ao seu estado.

Para terminar este capítulo, definamos através de um gráfico uma comparação interessante dos custos entre:

Curva A — Uma lógica por ligações eléctricas, com elementos «standard»: *operadores, básculas, registos simples,* etc.

Curva B — Um microprocessador

Curva C — Uma lógica particular, estudada especialmente para um cliente e tendo em conta as suas necessidades específicas. Trata-se de um *Custom Design.*

Esta comparação, realizada na figura 10, entre os vários preços de custo é realizada em função do número de peças a negociar.

Tudo isto leva-nos a pensar que, antes de tomar uma decisão sobre o emprego de um destes três tipos de equipamento, se torna necessário um estudo prévio de rentabilidade.

Na figura 11 apresenta-se um quarto sinóptico que nos permite realizar, em função de critérios relacionados com o nosso tipo de utilização, uma escolha judiciosa entre a lógica por ligações e o microprocessador.

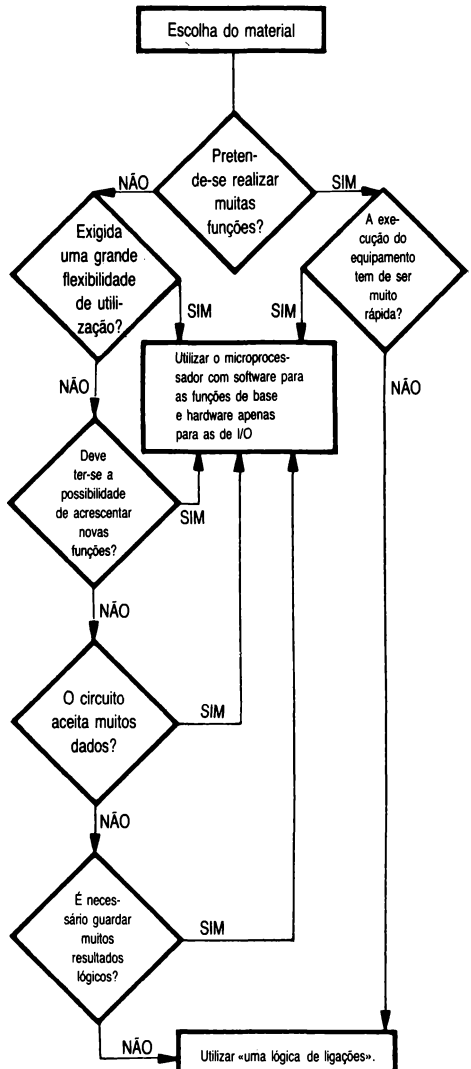

Escolha do material
Preten-de-se realizar muitas funções?
NÃO
SIM
Exigida uma grande flexibilidade de utili-zação?
A exe-cução do equipamento tem de ser muito rápida?
SIM
SIM
NÃO
NÃO
Utilizar o micropro-cessador com software para as funções de base e hardware apenas para as de I/O
Deve ter-se a possibilidade de acrescentar novas funções?
SIM
NÃO
O circuito aceita muitos dados?
SIM
NÃO
É neces-sário guardar muitos resultados lógicos?
SIM
NÃO
Utilizar «uma lógica de ligações».

Figura 11

4

AS LINGUAGENS DOS COMPUTADORES OS APARELHOS DE DESCODIFICAÇÃO DAS LINGUAGENS EM «BINÁRIO PURO»

Repitamos aqui uma vez mais que o computador só conhece a linguagem *binária pura*, também designada por *binária natural*, que se traduz por uma sucessão de zeros e de uns, utilizados tanto para receber pedidos como para efectuar operações aritméticas ou funções lógicas.

Se se impusesse ao programador a realização das instruções a dar ao computador directamente em binário natural, certamente que ao fim de alinhar alguns milhares de zeros e uns se enganaria e o trabalho se tornaria bastante aborrecido.

Imaginaram-se portanto *tipos de linguagens* que traduzem estas sucessões de zeros e uns de uma maneira mais sugestiva e fácil de utilizar pelo programador.

Estas linguagens são introduzidas numa máquina, a qual as traduzirá em binário natural a fim de ser digerido pelo computador.

O programa elaborado pelo programador é designado por *programa-fonte*.

Este programa é descodificado em binário natural pela máquina, constituindo ao entrar no computador o chamado *programa-objecto*.

O quadro seguinte mostra-nos algumas das linguagens de programação, cujas definições e diferenças essenciais iremos definir.

Neste quadro, em 1, os programas-fonte relativamente curtos são *codificados* em **octal**, em **hexadecimal** ou em **BCD** (decimais codificados em binário). Explicaremos o seu funcionamento no capítulo seguinte, dedicado ao *sistema de cálculo binário*.

QUADRO DAS PRINCIPAIS LINGUAGENS DE PROGRAMAÇÃO

<table>
<tr><td rowspan="2">Programa</td><td colspan="2" rowspan="2">Codificação ou Linguagem</td><td colspan="2">Aparelho tradutor em linguagem máquina</td></tr>
<tr><td>Interno ao computador</td><td>Externo ao computador</td></tr>
<tr><td rowspan="3">①
Curto</td><td colspan="2">Octal</td><td colspan="2" rowspan="3"></td></tr>
<tr><td colspan="2">Hexadecimal</td></tr>
<tr><td colspan="2">B.C.D.</td></tr>
<tr><td rowspan="5">②
Comprido</td><td colspan="2">Mnemónica ou simbólica</td><td>Assembler residente</td><td>Cross--assembler</td></tr>
<tr><td rowspan="4">Linguagem evoluída</td><td>Algol</td><td colspan="2" rowspan="4">Compilador</td></tr>
<tr><td>Fortran</td></tr>
<tr><td>Cobol</td></tr>
<tr><td>PL1-APL</td></tr>
<tr><td>Linguagem interpretada</td><td>Basic</td><td colspan="2">Intérprete</td></tr>
</table>

Em 2, os programas-fonte relativamente compridos são tratados, para não se tornarem fastidiosos, em *linguagem mnemónica* ou *simbólica*, em *linguagem evoluída* ou ainda em *linguagem interpretada*.

A máquina encarregue de definir o programa-objecto a partir do programa-fonte é designada, no caso de uma linguagem mnemónica, *assembler*, e o programa especial encarregue da tradução, *programa assembler*.

Quando a tradução em binário natural é executada directamente pelo computador ao qual é destinado o programa, o assembler é muitas vezes designado por *assembler residente*.

Quando a tradução em binário natural (código-máquina) é executada por um outro computador mais potente que aquele a que se destina o programa, é designado «cross--assembler».

O computador assim utilizado deve, neste caso, respeitar as regras de tradução aceites pelo computador destinatário.

Deve-se notar que existem *pseudo-instruções*, ou seja, *ordens* que não são directamente explorados pelo computador, mas que podem no entanto provocar uma sequência especial no assembler, destinada a operações de organização no computador, como por exemplo a reserva de zonas na memória, a definição de endereços, etc.

Finalmente, existe uma grande variedade de assemblers, como seja:

- O *macro assembler*, no qual cada instrução da linguagem fonte define uma operação mais completa decomposta numa série de sequências de instruções.

O *assembler translativo* («relocatable assembler»), que trata os endereços relativos.

O *assembler absoluto*, encarregue de calcular os endereços absolutos ou directos das informações nas memórias.

Quando se trata de uma linguagem evoluída, a máquina encarregue de a traduzir é designada *compilador*. A tradução é então designada por compilação.

No nosso quadro citámos algumas linguagens evoluídas a título de exemplo. Vejamos a utilização específica de cada uma delas.

Linguagens de tipo universal: PL1 e APL

Linguagens de tipo científico: Algol e Fortran

Linguagens orientadas para a resolução de probemas de gestão: Cobol.

A linguagem mnemónica, tal como as codificações «octal», hexadecimal, BCD são *linguagens elementares*, por oposição à linguagem evoluída.

Assim, para efectuar uma operação simples, será necessária em linguagem mnemónica toda uma série de sequências de instruções que obrigam já a um conhecimento íntimo do funcionamento do computador, enquanto que no caso da linguagem evoluída bastará escrever o que se pretende utilizando os termos da linguagem em causa.

Um microcomputador equipado com um *microprocessador* só pode actualmente utilizar uma linguagem mnemónica.

No capítulo 3 dá-se um exemplo, sendo **FETCH** e **LOAD** exemplos de instruções simbólicas.

A linguagem empregue obriga a dispender mais tempo na programação do que no caso das linguagens evoluídas, mas necessita de menos espaço em memória.

Dever-se-á então economizar no tempo de programação e empregar uma linguagem evoluída?

Ou deve-se economizar nas dimensões das memórias e utilizar uma linguagem assembler?

É óbvio que o segundo caso se torna mais interessante na medida em que os microprocessadores tinham um baixo preço de custo e sejam aplicados em utilizações específicas que obrigam a empregar apenas um programa ou pequeno número deles que possam ser considerados simples.

A *linguagem interpretada Basic* traduz no *intérprete* as instruções do programa-fonte uma a uma, procedendo imediatamente à sua execução passo a passo.

O tempo de execução do programa é portanto obrigatoriamente maior do que no caso de uma linguagem assembler ou compilada.

Por outro lado, este método convém pefeitamente ao *ensino da programação*, pois é então possível obter resultados assim que se executa cada etapa de instrução.

5
O CÁLCULO BINÁRIO
AS CODIFICAÇÕES: OCTAL, HEXADECIMAL E BCD

Para efectuar as operações aritméticas, é indispensável definir em primeiro lugar a maneira de escrever os números qualquer que seja a base do sistema de numeração adoptado.

Esta maneira de escrever os números resume-se numa *fórmula* que indicaremos depois de a ter demonstrado tomando como exemplo um número do sistema de base 10 (*sistema decimal*).

Considere-se o número, com três algarismos, 825.

Pode ser escrito $(8 \times 10^2) + (2 \times 10^1) + (5 \times 10^0)$.

Ou seja, 800 + 20 + 5.

8 — algarismo das centenas
2 — algarismo das dezenas.
5 — algarismo das unidades.

Sendo a base de 10, a fórmula geral decomposta, é:

$N = a \times 10^2 + b \times 10^1 + c \times 10^0$

Substituindo agora os coeficientes *a* (das centenas), *b* (das dezenas) e *c* (das unidades) por um coeficiente geral α dotado de um índice (n) que representa a ordem mais elevada no número, e a base 10 pela letra B, obtemos a seguinte fórmula genérica:

$$N = \alpha_n B^{n-1} + \alpha_{n-1} B^{n-2} \ldots + \alpha_1 B^0$$

Esta fórmula pode então ser aplicada a todos os números (N), qualquer que seja a base (B) do sistema de numeração escolhido.

Na prática de cálculo corrente, adoptou-se o *sistema decimal* na maior parte dos países do mundo.

Em informática, por razões de comodidade de tradução do cálculo em sinais eléctricos trabalhando no sistema *tudo ou nada*, adoptou-se o sistema binário, que se exprime em *bits* (0 ou 1).

Como se utilizam muitas vezes, particularmente em microinformática, números com quatro algarismos binários, com oito algarismos binários (8 bits = 1 byte) até de 16 algarismos binários, servimo-nos dos sistemas:

— BCD (Binary Coded Decimal)

— Octal (de base 8)

— Hexadecimal (de base 16).

Os dois quadros seguintes mostram-nos a correspondência entre estes sistemas de numeração muito úteis em informática.

Observando estes dois quadros pode-se concluir:

— que é necessário um grupo de 3 bits para traduzir em binário todos os algarismos de um número (0 a 7) lido no sistema octal;

— que é necessário um grupo de 4 bits para descrever em binário todos os algarismos de um número (0 a F) lido no sistema hexadecimal. Neste sistema representam-se os valores decimais 10 a 15 pelas primeiras letras do alfabeto: A, B, C, D, E, F.

— Que é necessário um grupo de quatro bits para traduzir em binário cada ordem (unidades, dezenas, centenas, etc.) de um número codificado em BCD.

Daremos aliás um exemplo de codificação binária, octal e hexadecimal no final do capítulo.

Vejamos como se deve passar de um sistema para outro.

Em primeiro lugar vejamos como se deve passar um número escrito no sistema binário para o número correspondente escrito no sistema decimal.

Para efectuar uma tal operação, basta recordar a fórmula precedente (N), que acabamos de desenvolver nestes quadros.

Quadro das correspondências entre o sistema decimal e os sistemas octal, hexadecimal e binário natural

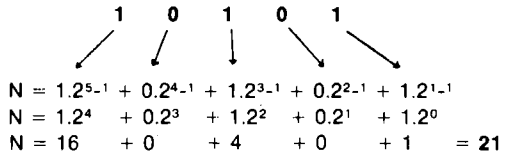
Valor decimal	Codificação binária	Codificação octal			Codificação hexadecimal	
		Valor octal	Codificação em grupo binário		Valor hexadec.	Codificação em grupo binário
			gr. 2	gr. 1		
0	0	0	000	000	0	0000
1	1	1	000	001	1	0001
2	10	2	000	010	2	0010
3	11	3	000	011	3	0011
4	100	4	000	100	4	0100
5	101	5	000	101	5	0101
6	110	6	000	110	6	0110
7	111	7	000	111	7	0111
8	1000	10	001	000	8	1000
9	1001	11	001	001	9	1001
10	1010	12	001	010	A	1010
11	1011	13	001	011	B	1011
12	1100	14	001	100	C	1100
13	1101	15	001	101	D	1101
14	1110	16	001	110	E	1110
15	1111	17	001	111	F	1111
16	10000	20	010	000	10	10000

Quadro da correspondência entre o sistema decimal e os sistemas binário natural e BCD

Valor decimal	Codificação binária natural	Sistema BCD Codificação em grupos binários	
		Dezenas	Unidades
0	0	0000	0000
1	1	0000	0001
2	10	0000	0010
3	11	0000	0011
4	100	0000	0100
5	101	0000	0101
6	110	0000	0110
7	111	0000	0111
8	1000	0000	1000
9	1001	0000	1001
10	1010	0001	0000
11	1011	0001	0001

Considere-se por exemplo o número binário *10101*, que deve ser escrito 10101_2 (número 10101 *na base 2*).

Desenvolve-se a fórmula (N) da seguinte maneira:

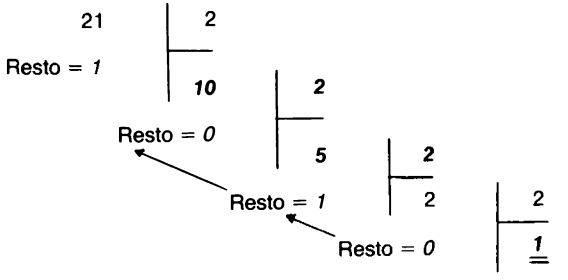


21 é portanto o número decimal correspondente ao número binário de 5 bits 10101_2.

Consideremos agora a passagem de um número escrito no sistema decimal a um número correspondente escrito no sistema binário.

A conversão pode efectuar-se da seguinte maneira.

Conversão de um número: Sistema decimal/sistema binário



Para escrever o número 21_{10} em sistema binário, sobe-se no sentido da seta considerando o último quociente (1) como unidade binária mais significativa do número que é em seguida constituído com os restos das sucessivas divisões.

Obtém-se portanto $21_{10} = 10101_2$.

Até agora discutimos apenas os *números inteiros*. Vejamos agora como se deve passar *os números fraccionários* de um sistema para outro.

A fórmula que permite definir a parte fraccionária de um número é:

$$N = \alpha_1 B^{-1} + \alpha_2 B^{-2} \ldots + \alpha_n B^{-n}$$

Recordemos que B^{-1} é a representação de $\frac{1}{B^1}$

B^{-2} é a representação de $\frac{1}{B^2}$

B^{-n} é a representação de $\frac{1}{B^n}$

Vamos agora poder passar de um número escrito no sistema binário para o número correspondente do sistema decimal.

Suponhamos que se pretende converter 0.101_2 para o sistema decimal.

Segundo a fórmula teremos:

$$\begin{aligned} 0{,}101 &= 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} \\ &= 1/2^1 + 0 + 1/2^3 \\ &= 0{,}5 + 0 + 0{,}125 \\ &= 0{,}625_{10} \end{aligned}$$

Note-se que o número $10101{,}101_2 = 21{,}625_{10}$

Para realizar a conversão inversa de decimal fraccionário para o sistema binário, efectuam-se as seguintes multiplicações sucessivas:

Pretende converter $0{,}625_{10}$ em binários naturais.

$0{,}625 \times 2 = 1{,}250 \longrightarrow$ obtém-se $1 \longrightarrow$ resto $0{,}250$
$0{,}250 \times 2 = 0{,}500 \longrightarrow$ obtém-se $0 \longrightarrow$ resto $0{,}500$
$0{,}500 \times 2 = 1{,}000 \longrightarrow$ obtém-se $1 \longrightarrow$ resto 0

Os resultados sucessivos, considerando como valor mais à esquerda do número o resultado da primeira multiplicação efectuada, constituem o número fraccionário procurado.

Neste caso, $0{,}625_{10} = 0{,}101_2$

Um número algébrico pode ser negativo ou positivo. O seu valor absoluto, em números binários, deve portanto ser precedido de um BIT de sinal. Existem várias maneiras de codificar um sinal.

Normalmente emprega-se um 0 para indicar o sinal + e um 1 para indicar o sinal —.

Vamos agora passar em revista a maneira como se deve actuar para realizar as quatro operações principais no sistema binário.

1. A ADIÇÃO

O processo de adição binária baseia-se nas regras seguintes:

$0 + 0 = 0$
$0 + 1 = 1$
$1 + 0 = 1$
$1 + 1 = 0$, e vai 1.

Exemplo:
$$\begin{array}{r} 1101_2 \\ +\ 0101_2 \\ \hline 10010_2 \end{array}$$

No nosso exemplo adicionámos dois números de 4 bits. O resultado surge sob a forma de um número de 5 bits.

Se o tratamento é realizado com palavras de 4 bits, existe aqui um quinto (correspondente ao «e vai um») que pode normalmente ser registado pelo computador num flip-flop especial, previsto para este efeito, designado pelos americanos de *carry* ou *link*.

2. A SUBTRACÇÃO

Existem várias maneiras de a efectuar. Estudaremos aqui a que nos parece mais normal em informática e que consiste em trabalhar *por complementaridade*.

Deve-se somar ao primeiro número binário dado o complemento para 2, designado por *complemento verdadeiro*, do segundo número a subtrair.

O complemento para 2 de um número obtém-se acrescentando 1_2 ao *complemento para 1* ou *complemento restrito* desse número.

O *complemento para 1* de um número obtém-se transformando todos os zeros em uns e os uns em zeros no número dado.

Assim, pretende-se subtrair 0101_2 de 1010_2.

O complemento de 0101_2 para 1 é 1010_2.

O complemento para 2 de 0101_2 é 1011_2.
O resultado da subtracção é portanto.

$$\begin{array}{r} 1010_2 \\ +\ 1011_2 \\ \hline = 10101_2 \end{array},$$

Elimina-se agora o último algarismo da esquerda do resultado desta operação, que no caso considerado é 1. O resultado da subtracção binária é 0101_2 , o que se pode verificar fazendo as contas em decimal:

$$\begin{array}{l} 1010_2 = 10_{10} \\ 0101_2 = \ 5_{10} \end{array} \qquad 10_{10} - 5_{10} = 5_{10}$$

Quando o algarismo da esquerda retirado no cálculo em binário é 1, como acontece neste caso, o resultado da subtracção é um número de *sinal positivo.*

Se não existe este algarismo à esquerda do número, ou seja, quando existe um zero, o resultado é um número de *sinal negativo.*

3. A MULTIPLICAÇÃO

Estudaremos duas maneiras simples de proceder:

a) A multiplicação por *adições sucessivas.*

Considere-se 12 × 3, na base 10, que em binário será 1100 × 0011:

$$\begin{array}{r} 1100 \\ +\ 1100 \\ +\ 1100 \\ \hline 100100 \end{array}$$

Adicionam-se três vezes o multiplicando, isto é, efectua-se uma soma contendo o multiplicando tantas vezes quanto o número de unidades existentes no multiplicador.

b) A multiplicação por *deslocamento e soma.*

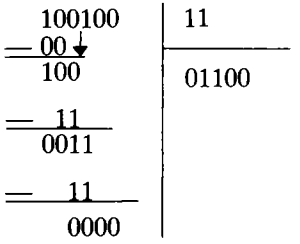
Executa-se a multiplicação binária da mesma maneira que se procederia no caso de uma multiplicação decimal.

Exemplo:

$$\begin{array}{r} 1100 \\ \times\ \ 11 \\ \hline 1100 \\ 1100 \\ \hline = 100100_2 \end{array}$$

4. A DIVISÃO

A divisão é efectuada da mesma maneira que a divisão decimal:



Acabamos de efectuar as quatro operações aritméticas essenciais em «sistema binário natural». É a única codificação ou *programa-objecto* ou ainda *código-máquina* que o computador ou o microcomputador conhece.

O programador pode por outro lado escrever um *programa-fonte* mais simples em *código octal* ou *código hexadecimal*. Damos em seguida um exemplo:

A instrução «CLEAR 64» dada ao minicomputador «PDP11» leva a zero todo o conteúdo da posição de endereço 64.

Esta instrução é codificada em duas palavras de 16 bits da seguinte maneira (em binário natural):

CLEAR — 0 000 101 000 011 111

64 — 0 000 000 001 000 000

Em codificação **octal,** traduzir-se-á cada grupo de 3 bits a partir da direita, o que dará:

CLEAR 0 000 101 000 011 111

0 0 5 0 3 7 = 005037_8

64 0 000 000 001 000 000

0 0 0 0 1 0 = 000100_8

Em código **hexadecimal**, traduzir-se-á cada grupo de 4 bits a partir da direita, o que dará:

CLEAR	0000	1010	0001	1111	
	0	A	1	F	$= 0A1F_{16}$
64	0000	0000	0100	0000	
	0	0	4	0	$= 0040_{16}$

O comprimento de palavra que pode ser tratado num computador pode por vezes ser insuficiente para o rigor pretendido.

Quando se ultrapassa o número de bits de *precisão simples*, dispõe-se do recurso a uma segunda *palavra* para tratar a mesma informação. Empregam-se duas palavras para a informação, ao que corresponde à *precisão dupla*, por oposição à *precisão simples* utilizado quando cada informação é contida numa única palavra.

Quando se diz que um microprocessador é de 8 bits, isto significa que ele recebe simultaneamente, ou seja em paralelo, nas suas 8 entradas, os 8 bits de cada palavra que compõe um *programa-objecto*. Neste caso cada palavra pode portanto ter $2^8 = 256$ *combinações possíveis de uns e de zeros,* o que imediatamente nos dá uma ideia da precisão ou *riqueza da linguagem* deste computador.

Finalmente, para terminar este capítulo dedicado ao cálculo no sistema binário, consideremos um número binário qualquer e procuremos traduzi-lo nos diferentes sistemas que acabamos de estudar. O futuro programador pode encarar este divertimento como um exercício.

Considere-se o número binário de 8 bits 10010111_2.

Se se trata de um *número binário natural*; equivale a 151_{10}.

Se se trata de um *número com sinal* (*complemento para 1*), é igual a -104_{10}.

Se se trata de um *número com sinal* (*complemento para 2*), é igual a -105_{10}.

Se se trata de um *número* BCD (2 grupos de 4 bits), é igual a 97_{10}.

Se se trata de um *número octal* (por grupos de 3 bits), é igual 227_8.

Se se trata de um *número hexadecimal* (por grupos de 4 bits), é igual a 97_{16}.

Finalmente, nada impede que este número binário com 8 bits seja a tradução de uma instrução de um programa-fonte codificado em linguagem mnemónica e simbólica.

Exemplo: Este número binário corresponde no microprocessador MC6800 da Motorola à instrução em linguagem mnemónica:

STAA,

que significa guardar no acumulador A. O termo acumulador será objecto de uma explicação no capítulo consagrado ao microprocessador.

6
AS FUNÇÕES LÓGICAS

O jogo de instruções de um microprocessador comporta geralmente, a fim de resolver os problemas que lhe são postos, instruções de funções lógicas.

Muitas vezes trata-se de simples *operadores*, como as *portas* E, OU inclusivo ou exclusivo, NAND, NOR, etc.

Além disso, é interessante ter uma ideia sobre a própria concepção dos circuitos de *memorização*, como as *básculas bi-estáveis* e os *registos* que participam na organização de um computador ou um microcomputador e estruturam em parte o seu *processador* ou *microprocessador*.

É por esta razão que iremos consagrar este capítulo aos *circuitos lógicos* e às *respectivas funções*.

Os circuitos lógicos necessários às funções correspondentes comportam *bornes de entrada* e de *saída*.

O técnico aplica tensões aos bornes de entrada e obtém outras nos bornes de saída, que são função das combinações das tensões aplicadas e dos circuitos empregues.

Todas estas tensões podem apenas tomar dois valores bem definidos.

1. O *nível alto* — símbolo H, correspondente à maior tensão.

2. O *nível baixo* — símbolo B, correspondente à tensão mais fraca.

Entre estes dois níveis, o técnico despreza as variações físicas das tensões.

No seu raciocínio, o técnico apoia-se em duas convenções simbólicas opostas:

1.ª CONVENÇÃO LÓGICA

Trata-se da *lógica positiva.*

Uma *afirmação* traduz-se pelo «1» do sistema binário, e define a presença do nível «H».

Uma *negação* traduz-se pelo «0» do sistema binário e define a presença do nível «B».

2.ª CONVENÇÃO LÓGICA

Trata-se da *lógica negativa.*

Uma afirmação traduz-se pelo «0» do sistema binário e define a presença do nível «H».

Uma negação traduz-se pelo «l» do sistema binário e define a presença do nível lógico «B».

Por razões mnemónicas bem evidentes, é hoje habitual raciocinar de preferência em «lógica positiva».

A fim de ilustrar as nossas afirmações, tomemos a título de exemplo a função lógica NÃO (normalmente designada pela palavra inglesa **NOT**).

É definida de tal modo que, ao aplicar-se à entrada um sinal H, se obtém na saída um sinal B, ou quando aplicando à entrada um sinal B se obtém na saída um nível H.

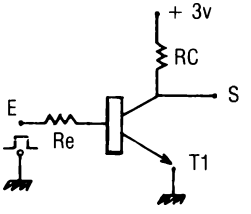


Figura 12

A fim de obter este resultado, imaginemos (figura 12) um transístor que fazemos trabalhar alternadamente em *regime saturado* e em *regime bloqueado*. Este transístor é montado de acordo com esta figura tendo ligada em série à sua alimentação de colector uma resistência «Rc». Por outro lado, «Re» impede que a junção emissor-base fique em curto-circuito.

1. Aplica-se um impulso positivo de + 3 V em (E). Este impulso em (E) corresponde ao nível H, igual a 1 em lógica positiva.

O transístor (T1) está saturado. No terminal S do circuito obtém-se uma tensão de + 0,3 V, igual à tensão de saturação de T1. Esta tensão de + 0,3 V relativamente à massa determina o nível B, igual ao «O» em lógica positiva.

2. Aplica-se agora uma tensão nula em «E», ou seja a tensão da massa de referência. Esta representa o nível B, correspondente ao valor lógico 0 no caso da lógica posisitva.

O transístor T1 fica bloqueado. No terminal S obtém-se uma tensão de + 3 V igual à tensão de alimentação do transístor. Esta tensão representa o nível H, correspondente ao nível lógico 1 no caso da lógica positiva.

Poderíamos raciocinar da mesma maneira no caso da lógica negativa, mas os uns e os zeros lógicos seriam invertidos, ou seja, a cada nível H corresponderia um 0 lógico e a cada nível B corresponderia o 1 lógico.

O fabricante tem o hábito de resumir as características de níveis lógicos dos seus circuitos naquilo que se designa por *Tabelas de verdade*. A título de exemplo, apresentamos na figura 13 a tabela de verdade da função NOT.

Figura 13 — Tabela de verdade da função NOT

Níveis eléctricos		Tabela de verdade			
		Lógica positiva		Lógica negativa	
E	S	E	S	E	S
H	B	1	0	0	1
B	H	0	1	1	0

Existem dois grandes tipos de funções lógicas:

1. A classe das *funções lógicas combinatórias*. Nesta classe, o sinal recolhido no borne «S» de um circuito só depende exclusivamente da combinação dos sinais aplicados no mesmo instante nos bornes de entrada (E) deste circuito.

Veremos que a *álgebra de Boole* é particularmente cómoda para todos os circuitos de lógica combinatória.

2. A classe das *funções lógicas sequenciais*.

O sinal recolhido no borne S de um circuito já não depende exclusivamente da combinação de sinais aplicada no mesmo instante aos bornes de entrada E, mas também do estado anterior em que o circuito se encontrava quando estes sinais foram aplicados.

Trata-se portanto de funções que possuem uma qualidade de *memorização* dos estados anteriores no instante considerado.

Têm em conta a *sequência* completa dos dados anteriores.

A álgebra de Boole já não é então utilizável, raciocinando-se a partir de *diagramas de tempo*, de *tabelas de verdade* ou ainda de *tabelas de fases*, cujas colunas permitem distinguir os níveis nos bornes de saída antes do instante em que são aplicados os sinais de entrada.

AS FUNÇÕES LÓGICAS COMBINATÓRIAS

Estas funções podem ser divididas em três grandes categorias hierarquizadas segundo o respectivo grau de complexidade.

A — As *funções elementares*:	**NOT (NÃO)** **OR (OU)** **AND (E)**
B — As *funções de funções de primeiro grau:*	**NOR (NOT-OR)** **NAND (NOT-AND)**
C — As *funções de funções complexas*	**OR** exclusivo **AND-OR-NOT** Circuitos de soma e de CARRY

AS FUNÇÕES LÓGICAS SEQUENCIAIS

Estas funções podem ser divididas em duas grandes ca-

tegorias hierarquizadas conforme o respectivo grau de complexidade.

A — As *básculas bi-estáveis*, elementos simples de memorização com duas portas estáveis.	Básculas simples — *tipo* **RS** Básculas sincronizáveis: **R_T S_T e J_T K_T** **D_Te Master — Slave**
B — Os *circuitos sequenciais*	*Registos* *Contadores de impulsos* *e divisores de frequência*

Iremos examinar primeiramente as funções elementares da lógica combinatória.

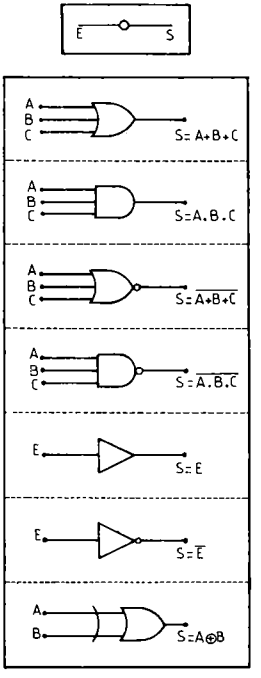


Figura 14 — Portas que realizam as funções (de cima para baixo) NOT, OR, AND, NOR, NAND, «BUFFER», «BUFFER-INVERSOR», OR Exclusivo:

1. FUNÇÃO **NOT**

Já demos a definição anteriormente. Esta função é também designada por *função complementar* ou *função inversa*.

Em álgebra booleana a função escrever-se-á

$$S = \bar{E},$$

pronunciando-se S = E *barra*, ou ainda S = E *complementado*.

A figura 14 representa o simbolismo esquemático utilizado para esta função (em cima).

2. FUNÇÃO **OR**

O circuito utilizado inclui vários terminais de entrada A, B, C, etc., e um único terminal de saída S.

Em lógica positiva, uma afirmação na saída (1 lógico) significa que vai aplicado um sinal de lógica 1 em qualquer dos terminais de entrada do circuito.

Em álgebra booleana, a equação da função OR escreve-se:

$$S = A + B + C$$

e lê-se

$$S = A \text{ ou } B \text{ ou } C$$

O sinal + indica o termo *ou* (em inglês, *or*). Esta operação é chamada soma lógica.

Na figura 14 representa-se o símbolo esquemático empregue para designar esta função.

No caso presente, representámos um «operador» ou «porta» de três entradas, cuja *tabela de verdade* indicamos na figura 15.

Função OR			
A	B	C	S
1	1	1	1
1	1	0	1
1	0	1	1
0	1	1	1
1	0	0	1
0	1	0	1
0	0	1	1
0	0	0	0

Figura 15 — Tabela de verdade da função OR executada por uma porta com três entradas.

3. FUNÇÃO **AND**

O circuito utilizado inclui diversos terminais de entrada A, B, C, etc., e uma única saída S.

Em lógica positiva só será recolhida uma afirmação na saída (1 lógico), quando todos os terminais de entrada se encontram a esse nível lógico.

Em álgebra booleana, a equação da função AND é escrita do seguinte modo:

$$S = A \times B \times C$$

e lê-se

$$S = A \text{ e } B \text{ e } C$$

O sinal × indica o termo *e* (*and*). Esta operação é designada por produto lógico.

Na figura 14 representa-se o símbolo esquemático usado para definir a função AND.

Neste caso considerámos uma porta AND de três entradas, e damos na figura 16 a tabela de verdade desta função.

Função AND			
A	B	C	S
1	1	1	1
1	1	0	0
1	0	1	0
0	1	1	0
1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	0

Figura 16 — Tabela de verdade da função AND no caso de uma porta com três entradas.

Examinemos seguidamente «as funções de funções de primeiro grau» da lógica combinatória.

1. *Função* **NOR (NOT-OR)**

2. *Função* **NAND (NOT-AND)**

Os circuitos utilizados têm uma implantação dos terminais de entrada e de saída igual à dos circuitos usados nas funções OR e AND.

As suas tabelas de verdade são obtidas a partir das fornecidas para as funções OR e AND, por um lado, e da correspondente à função NOT, por outro.

Dizendo por outras palavras, basta substituir nas tabelas de verdade das funções OR e AND todos os valores da coluna S pelos seus valores *complementares*, ou seja, substituir os uns por zeros e os zeros por uns.

Em álgebra booleana a equação da função NOR é escrita do seguinte modo:

$$S = \overline{A} + \overline{B} + \overline{C}$$

e lê-se:

$$Sx = \overline{A} \text{ ou } \overline{B} \text{ ou } \overline{C}$$

Do mesmo modo, a equação da função NAND escreve-se:

$$S = \overline{A} \times \overline{B} = \overline{C}$$

e lê-se:

$$S = \overline{A} \text{ e } \overline{B} \text{ e } \overline{C}$$

Na figura 14 representa-se o símbolo esquemático utilizado para definir uma destas funções.

Estes símbolos esquemáticos recordam portanto, como se pode observar, a sequência das duas operações (OR seguida de NOT e AND seguida de NOT).

Poder-se-ia ainda citar além das funções elementares e das funções de funções elementares, os circuitos encarregues apenas de amplificar os sinais utilizados para as comunicações em linhas que apresentam perdas fortes. Estes circuitos são designados por circuitos **BUFFER** (no caso dos amplificadores simples) ou ainda circuitos **BUFFER-INVERSOR** no caso dos amplificadores inversores de sinais.

Na figura 14 indica-se uma vez mais o símbolo utilizado para representar cada uma destas funções.

Finalmente, para terminar este pequeno curso sobre

funções elementares e funções de funções do primeiro grau, daremos na figura 17 o esquema de uma realização prática. Trata-se de um circuito lógico integrado da família **TTL 74** (tecnologia bipolar: lógica completamente transistorizada) designado por *7400*, que compreende uma *porta NAND quádrupla de duas entradas* montada numa cápsula **DIL** (Dual in Line) cujas ligações são em número de 14 e se encontram dispostas em duas fiadas paralelas (daí o nome Dual-in-line). A cápsula é em plástico e tem a referência normalizada **TO116.**

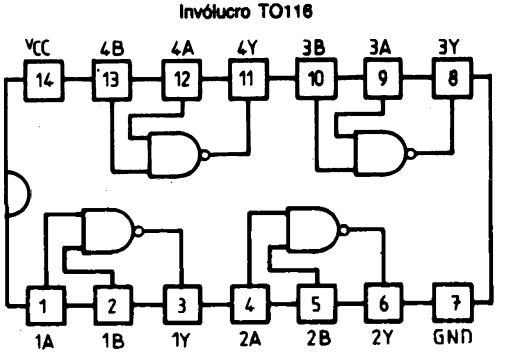


Figura 17 — Circuito TO116

Examinemos, para terminar este estudo sobre as funções da lógica combinatória, as funções de funções complexas.

1. FUNÇÃO **OR EXCLUSIVO**

A função OR, que já estudámos, é também designada por *função OR inclusivo*, o que significa que numa porta com duas entradas A e B a saída tem o valor 1 se ambas as entradas, ou pelo menos uma delas, apresentarem o valor 1.

Na função *OR exclusivo*, a saída encontra-se ao nível lógico 1 se pelo menos uma das entradas se encontra a esse nível, mas passa ao nível lógico 0 se for aplicado a *todas* as entradas o valor lógico 1.

Na figura 18 podemos observar a tabela de verdade da função OR exclusivo.

Função OR exclusivo		
A	B	S
1	1	0
0	1	1
1	0	1
0	0	0

Figura 18 — Função OR exclusivo no caso de uma porta com apenas duas entradas.

Lendo esta tabela verificamos que quando

$$S = 1,$$

teremos

$$A \neq B;$$

quando

$$S = O,$$

teremos

$$A = B.$$

Imagina-se já «uma primeira aplicação» da função «Or exclusivo». Esta função pode servir para *comparar sinais lógicos.*

O operador OR exclusivo permite verificar os **estados** *de coincidência* ou **não** *coincidência* dos sinais lógicos.

Uma segunda aplicação que iremos descrever adiante é a de elemento constitutivo na concepção de um *somador binário.*

Na figura 14 representa-se uma vez mais de maneira simbólica o esquema de um circuito OR exclusivo.

A função escreve-se:

$$S = A \oplus B$$

e lê-se

S = A ou B exclusivo.

O sinal + rodeado por um círculo indica a função escrita OR exclusivo. Em álgebra booleana demonstra-se que:

$$S = A \oplus B = A \times B + B \times A$$

2. FUNÇÃO **AND-OR-NOT**

Esta função de funções é geralmente obtida utilizando um circuito constituído pela combinação de operadores AND, OR e NOT.

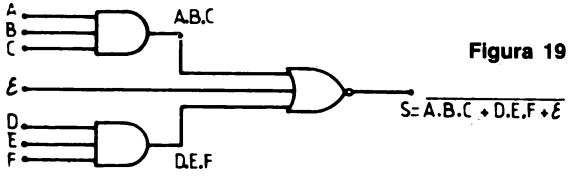


Figura 19

A título de exemplo, apresentamos na figura 19 um esquema possível de realização desta função.

3. FUNÇÃO DE SOMA BINÁRIA E «CARRY»

Trata-se neste caso de uma verdadeira lógica por ligações, incluindo dois circuitos *semi-somadores* formados por um operador OR exclusivo e uma porta AND. Não daremos aqui nenhuma explicação sobre o funcionamento deste circuito, pois essa explicação ultrapassa o âmbito da nossa exposição. Digamos apenas que é necessário somar um número binário A_n a outro número B_n (ver a figura 20).

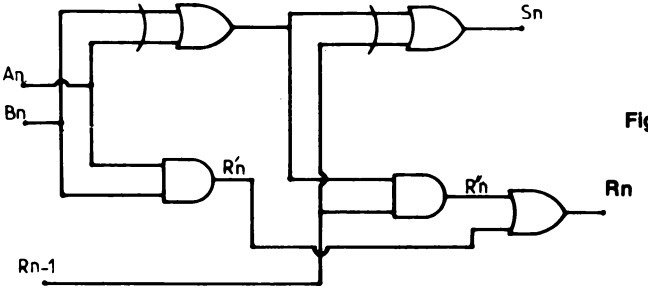


Figura 20

A adição aos números precedentes A_{n-1} e B_{n-1} deu lugar a uma quantidade de ordem superior ao primeiro algarismo da esquerda (do tipo «e vai um»), R_{n-1}, que deve ser igualmente adicionada a $A_n + B_n$. O resultado da adição é S_n, e pode incluir igualmente um algarismo de ordem superior R_n.

Iremos abordar agora as *funções lógicas sequenciais*. Estudaremos apenas as básculas ou *flip-flops* e os *registos*, a fim de adquirirmos uma ideia elementar da estrutura das *memórias*.

Começaremos esta exposição pelas *básculas bi-estáveis*. Examinemos sucessivamente as básculas simples **RS** e em seguida as básculas sincronizáveis: $\mathbf{R_TS_T}$ - $\mathbf{J_TK_T}$ - $\mathbf{D_T}$ e as do tipo piloto-pilotada (master-slave).

1. BÁSCULAS «RS»

As mais simples são constituídas por duas portas NOR e NAND (ver a figura 21).

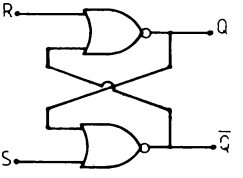


Figura 21

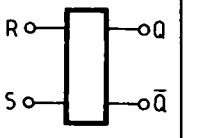
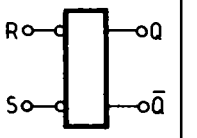
N.º de estados	R	S	Q	$\overline{Q}$	Observações sobre as saídas	Esquema simbólico
1	$\mathcal{H}$	$\mathcal{H}$			1. Estado indeterminado se o estado anterior era 2	
2	$\mathcal{B}$	$\mathcal{B}$				
3	$\mathcal{B}$	$\mathcal{H}$	$\mathcal{B}$	$\mathcal{H}$	2. Estado determinado mas sem variação se o estado anterior era 3 ou 4	
4	$\mathcal{H}$	$\mathcal{B}$	$\mathcal{H}$	$\mathcal{B}$		
3	$\mathcal{B}$	$\mathcal{H}$	$\mathcal{H}$	$\mathcal{B}$		
4	$\mathcal{H}$	$\mathcal{B}$	$\mathcal{B}$	$\mathcal{H}$		

Figura 22

Estas básculas, também designadas por *flip-flop,* são do tipo RS (*Reset* e *Set* — remarcar e marcar). A sua representação simbólica é semelhante a um rectângulo como se indica na figura 22, na qual é igualmente incluída a respectiva tabela de verdade.

Uma báscula encontra-se no estado 1 ou *estado activo* quando

$Q = 1$ (nível H em lógica positiva)
$\overline{Q} = 0$

Encontra-se no estado 0 ou *estado inactivo* quando

$Q = 0$ (nível B em lógica positiva)
$\overline{Q} = 1$

2. BÁSCULAS SINCRONIZÁVEIS

Primeiramente consideramos as básculas R_TS_T e J_TK_T

A figura 23 fornece as respectivas representações simbólicas.

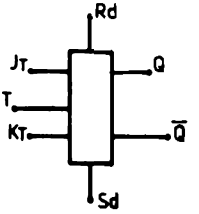


Figura 23

J_T e K_T (ou R_T e S_T) são os terminais de entrada em que se aplicam os estados lógicos que devem definir o estado futuro da báscula.

T ou C_p é o terminal onde se aplica o sinal de sincronização que dá a *ordem decisiva de mudança de estado.* Este sinal é designado por *sinal de relógio.* T significa *trigger* e C_p *clock pulse.*

Q e Q são os terminais de saída. Os níveis obtidos em Q e Q são portanto sempre complementares.

R_d e S_d são *terminais* que servem para impor o estado da báscula instantaneamente e independentemente de T.

Para estabelecer a tabela de fases que indica o funcionamento de uma báscula sincronizável, quer seja a R_TS_T ou a J_TK_T, é necessário considerar o tempo t_n após a aplicação da última ordem de mudança de estado e o tempo t_{n+1} depois da execução desta ordem.

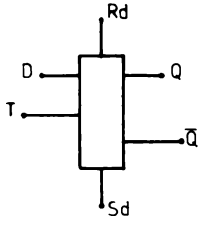
Tipo de básculas	RT_{tn}	ST_{tn}	Q_{tn+1}	$\overline{Q}_{tn+1}$
Báscula R_TS_T	1	1	Estados indeterminados	
	1	0	1	0
	0	1	0	1
	0	0	Q_{tn}	$\overline{Q}_{tn}$
Báscula J_TK_T	1	1	Q_{tn}	Q_{tn}
	1	0	1	0
	0	1	0	1
	0	0	Q_{tn}	$\overline{Q}_{tn}$

Figura 24 — *Tabela de fases das básculas R_TS_T e J_TK_T*

Observando as tabelas de fase S (figura 24), verificamos que a diferença entre as básculas R_TS_T e J_TK_T reside no facto de *a indeterminação dos estados de saída ser eliminada na báscula J_TK_T quando se aplicam às duas entradas dois níveis 1 em lógica positiva.* Além disto, constata-se neste caso uma mudança dos níveis das saídas nas báscula J_TK_T. Não há mudança de estado no caso de entradas ao nível lógico 0, em qualquer dos dois tipos de básculas.

Examinemos em seguida as básculas do tipo D_T.

Trata-se de uma báscula sincronizável simplificada, pois possui apenas uma entrada.



Dtn	Qtn+1	$\overline{Q}$tn+1
0	0	1
1	1	0

Figura 25 — À direita, a tabela de fases da báscula D_T.

Representamos na figura 25 o esquema simbólico desta báscula e a sua tabela de fases.

Vejamos finalmente a báscula do tipo *master-slave*.

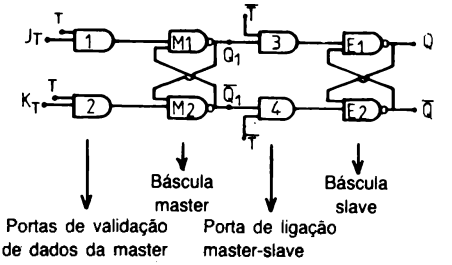


Figura 26

O esquema da figura 26 mostra-nos a configuração sofisticada deste tipo de báscula.

— Os circuitos M_1 e M_2 definem a *báscula «master»*.

— Os circuitos E_1 e E_2 definem a *báscula «slave»*.

— As portas 1, 2, 3 e 4 são consideradas *abertas* quando transmitem à saída a informação presente nas respectivas entradas. No caso contrário, são consideradas *fechadas*.

Se $T = B$ e $\overline{T} = H$, as portas 3 e 4 encontram-se abertas. A báscula master impõe então os níveis $\bar{Q}_1$ e Q_1 à báscula slave.

As portas 1 e 2 encontram-se então fechadas. A báscula master encontra-se *bloqueada* e não recebe as instruções dadas em J_T e K_T.

Se $T = H$ e $\overline{T} = B$, as portas 3 e 4 encontram-se fechadas. A báscula slave fica isolada da master, mantendo-se no estado que lhe foi anteriormente imposto por esta.

As portas 1 e 2 encontram-se abertas. As informações existentes em J_T e K_T impõem à báscula master o seu novo estado.

Depois de examinar as básculas simples RS e as básculas sincronizáveis iremos abordar o estudo dos *registos* que fazem parte do conjunto de *circuitos sequenciais de funções elaboradas*.

Os registos são órgãos de memória que podem ser divididos em 3 partes distintas:

a) Os órgãos de entrada, validados através de uma *ordem* de inscrição (ou *de escrita*). A validação (por exemplo um 1 lógico) autoriza a escrita de uma informação na memória.

b) *Os órgãos de memorização.* Cada báscula pode por exemplo conservar um bit (mantém-se num dado estado num determinado momento).

c) *Os órgãos de saída, validados* graças a uma *ordem de leitura.* A validação (por exemplo um 1 lógico) autoriza a leitura de uma informação armazenada em cada báscula correspondente da memória.

Podem distinguir-se, conforme a respectiva estrutura de ligações, quatro tipos de registos.

a) Os registos de escrita e de leitura dos bits *em paralelo.*

b) Os registos de escrita e leitura dos bits *em série.*

c) Os registos de escrita dos bits *em série* e de leitura *em paralelo.*

d) Os registos de escrita dos bits *em paralelo* e de leitura *em série.*

A título de exemplo para compreender melhor o que acabamos de dizer sobre os registos, damos nas figuras 27 e 28 dois esquemas de realizações simples de registos de diferentes estruturas de ligações.

1. ESQUEMA DE UM REGISTO DE ESCRITA E LEITURA DOS BITS EM SÉRIE (ver figura 27)

Neste registo as básculas B_1, B_2, e B_3 encontram-se ligadas em série. Em qualquer momento podem-se colocar as básculas ao nível 0 antes de proceder a uma nova *inscrição* utilizando uma *ordem de passagem do registo para 0*, que é aplicada nos terminais S_{D1}, S_{D2} e S_{D3} de cada uma das básculas do registo.

Na entrada em série envia-se então o 1.º bit de informação, assim como a ordem de escrita. Esta é imposta à báscula B_1 no *momento do disparo* do sinal de relógio T_n.

No momento T_{n+1}, o bit é *enviado* para a báscula B_2 e se for dada nova ordem de escrita pode-se igualmente enviar

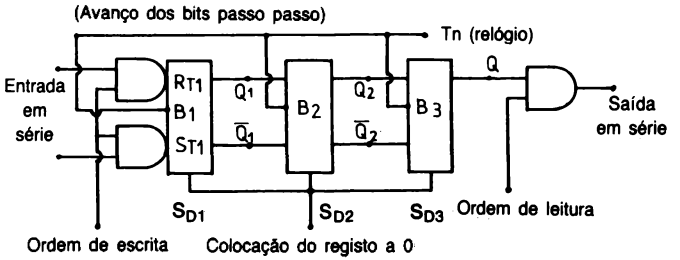


Figura 27 — Registo de escrita e leitura dos bits em série

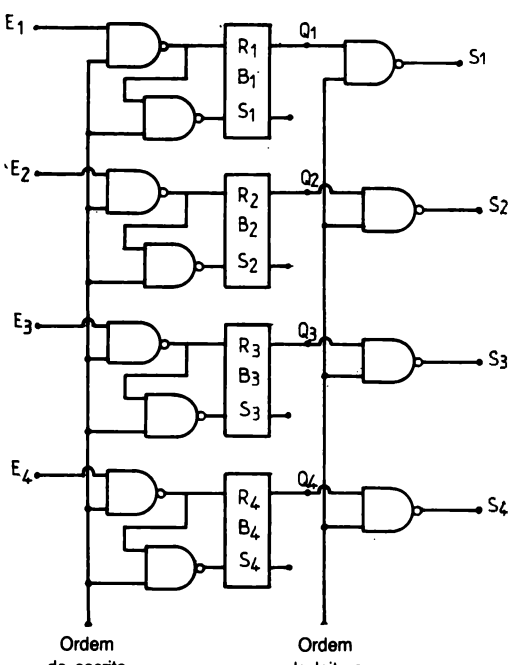


Figura 28 — Registo de escrita e leitura dos bits em paralelo

para B_1 um 2.º bit de informação. No momento T_{n+2} desloca-se novamente o registo.

— O 1.º bit é então *armazenado* em B_3 (MSB)
— O 2.º bit passa para B_2 e em B_1 surge o 3.º bit (LSB)

Graças a este deslocamento dos bits *um a um* consegue-se formar uma palavra de 3 bits.

Se se dá então ao registo uma ordem de leitura, poder-se-á ler igualmente na saída, sucessivamente, a série dos 3 bits desta palavra, nos momentos T_{n+3}, T_{n+4} e T_{n+5}, sendo esta leitura assegurada pelos disparos sucessivos dos sinais de relógio.

2. ESQUEMA DE UM REGISTO DE ESCRITA E LEITURA DOS BITS EM PARALELO (ver a figura 28)

Trata-se neste caso de um registo-memória para palavras de 4 bits de escrita e leitura paralela. A ordem de escrita e a ordem de leitura actuam portanto simultaneamente sobre os quatro bits que compõem cada palavra desta memória.

É fácil compreender que um registo de memória é lido ou escrito mais rapidamente quando os bits que compõem as palavras têm um acesso em paralelo, sendo então escritos ou lidos todos os bits simultaneamente, ou seja, num mesmo momento t_n.

Examinámos assim os principais registos dos circuitos sequenciais de funções elaboradas. Citámos entre estas funções elaboradas circuitos diferentes dos registos, mas não os estudaremos aqui pois não é necessário conhecê-los no nosso curso de iniciação sobre microcomputadores. Foram aliás editadas muitas obras sobre este assunto, e convidamos os leitores curiosos a consultarem-nas.

7

A TECNOLOGIA DOS MICROPROCESSADORES

Como já vimos, o microprocessador pôde nascer graças ao desenvolvimento das tecnologias de difusão aplicadas aos circuitos integrados **LSI** (Large scale integration).

As tecnologias melhor adaptadas a estes tipos de circuitos são sem dúvida alguma as tecnologias **MOS (N-MOS** ou **P-MOS)** e **C-MOS.**

Contam-se actualmente no mercado mais de 25 microprocessadores utilizando estes métodos de fabrico.

Estas tecnologias permitem a qualquer microprocessador executar somas em tempos da ordem de um a dois microsegundos.

A tecnologia N-MOS é um pouco mais rápida do que a P-MOS, dado que os portadores móveis N são menos pesados do que os portadores P.

Os consumos de potência em funcionamento são bastante reduzidos e a superfície ocupada por um transístor é ínfima, o que dá a possibilidade de integrar, sem um aquecimento proibitivo, todas as funções de um microprocessador numa única pastilha de silício com uma superfície de alguns milímetros quadrados.

Anteriormente foram realizados alguns ensaios com uma tecnologia **TTL/S** (lógica completamente transistorizada e díodos Schottky), que apresenta a vantagem de uma grande rapidez de execução das operações, podendo uma soma ser executada em menos de 1/10 de microssegundo.

No entanto, esta tecnologia consome mais energia em funcionamento, sendo em geral necessário montar vários circuitos integrados (pelo menos 2) se se deseja definir todas as funções essenciais de um microprocessador.

Por esta razão a tecnologia TTL/S foi praticamente abandonada no fabrico de microprocessadores.

A fim de obter velocidades de execução elevadas, a Motorola pensou em desenvolver um microprocessador em tecnologia **ECL** (Emitter Coupled Logic). Trata-se de uma lógica de «nível H» não saturado.

Finalmente, estão ainda para surgir no mercado alguns microprocessadores que recorrem a uma nova tecnologia designada **I^2L** (Integrated Injection Logic). A rapidez de execução é neste caso média, mas a energia consumida é muito fraca e o processo de fabrico é simples (é necessário utilizar um número muito reduzido de máscaras no decurso da difusão de impurezas).

A maior parte dos microprocessadores actuais são de 8 bits, o que significa que aceitam palavras formadas por 8 bits em paralelo ou *bytes*.

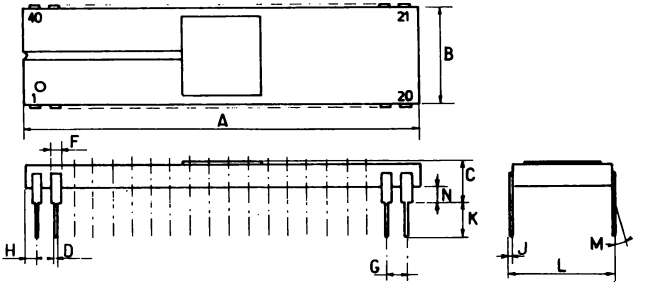
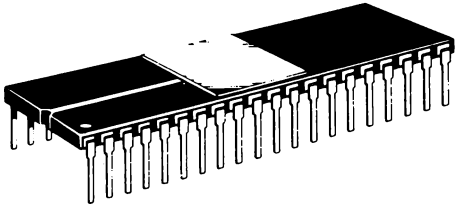


Figura 29

	MILÍMETROS	
DIM	MIN	MAX
A	50.29	51.31
B	14.86	15.62
C	2.54	4.19
D	0.38	0.53
F	0.76	1.40
G	2.54 BSC	
H	0.76	1.78
J	0.20	0.33
K	2.54	4.19
L	14.60	15.37
M	–	10^0
N	0.51	1.52

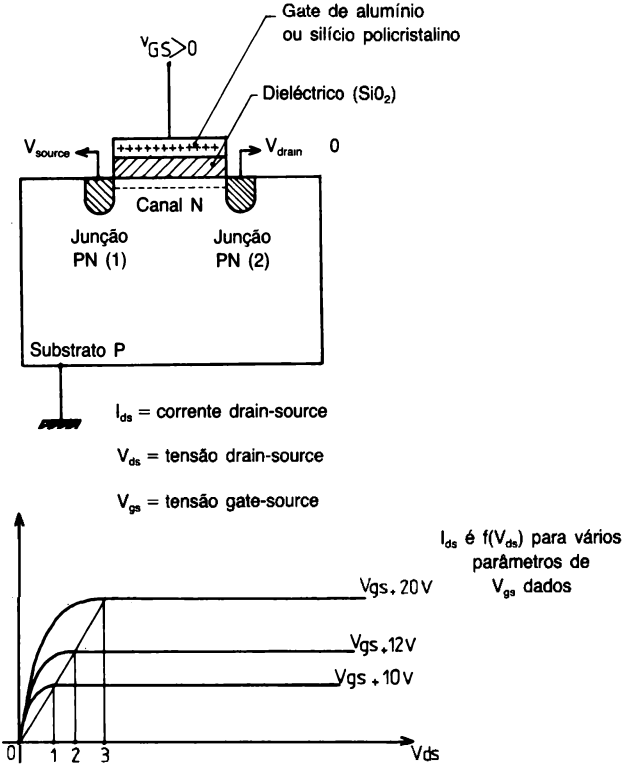


Em número mais reduzido existem também modelos de 4 bits, de 23 bits e de 16 bits. A *capacidade de memória* é actualmente na maior parte dos modelos de 64 K bytes.

Encontram-se encapsulados numa caixa DIL (Dual in line) tendo conforme os casos entre *16 e 40 terminais de acesso*.

Na figura 29 pode-se observar a fotografia de um deles.

Procuraremos neste capítulo explicar a física das tecnologias MOS (N-MOS e P-MOS) e C-MOS, assim como dar uma ideia das tecnologias TTL/S, ECL e I^2L. A título documental, indicaremos finalmente num quadro, e em função da tecnologia utilizada, os microprocessadores actualmente comercializados.



PROCESSO DE FABRICO DE UM TRANSÍSTOR **MOS**

Dispõe-se de um *substracto de silício monocristalino com impurezas de tipo P.* Fabrica-se então um transístor **MOS** *de canal* **N**. Coloca-se sobre o material do substracto um *dieléctrico* constituído por óxido de silício (SiO_2).

Por cima do óxido deposita-se um contacto que formará a *gate* (porta). É constituída por alumínio ou silício policristalino.

O conjunto formado pelo substracto P, o dieléctrico e a gate compõe um *condensador.*

Se se aplica à armadura «gate» uma tensão $V_{GS} > 0$ *(tensão gate-source)*, ou seja, maior do que a tensão de massa do substracto com impurezas do tipo P, deve-se observar:

— No substracto P, à frente do dieléctrico (SiO_2), *cargas de electrões N.*

— Na gate, à frente do dieléctrico, *cargas de buracos positivos P.* Estamos portanto perante um condensador (c) carregado.

Se de um lado e do outro deste condensador, se difundirem *2 junções idênticas PN_1 e PN_2*, e se se designa a junção PN_1 por *source* e PN_2 por *drain*, polarizando positivamente o drain relativamente ao substracto e à source produzir-se-á uma corrente de drain I_{DS} que dependerá principalmente da tensão VGS aplicada à gate do transístor MOS. É isto que é traduzido pela curva da figura 30 da corrente de drain I_{DS} em função de V_{DS}, considerando V_{GS} (tensão de gate) como parâmetro. Com efeito, quanto mais positiva for V_{GS}, maior é a carga do condensador MOS e mais facilmente a tensão positiva de drain pode captar electrões do canal N.

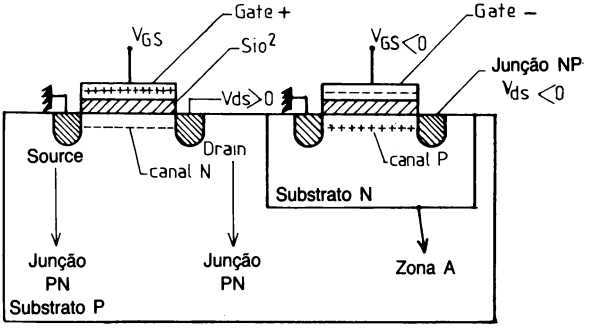
Pode-se igualmente partir de um *substracto N.* A carga entre a source e o drain será então constituída por buracos positivos. V_{GS} e V_{DS} relativamente ao substracto e à source serão alimentados por tensões negativas que constituem, relativamente ao dieléctrico de SiO_2 montado sobre a gate, uma reserva importante de electrões.

Os buracos positivos circularão, desta vez, no canal P entre a source e o drain.

Teremos assim formado um transístor **MOS** *de canal P.*

PROCESSO DE FABRICO DE 2 TRANSÍSTORES **MOS** COMPLEMENTARES **(C-MOS)**

A figura mostra-nos que se trata de difundir num substracto P uma zona A suficiente com impurezas N, com uma densidade de concentração superior à da difusão P original necessária para formar em seguida um transístor de canal N no substracto P e um transístor de canal P no novo substracto N constituído na zona A (ver a figura 31).



APLICAÇÕES PRÁTICAS DESTAS TECNOLOGIAS **MOS** E **CMOS** NOS SISTEMAS DE MICROPROCESSADORES

1. *Utilização dos transístores MOS nas memórias.*

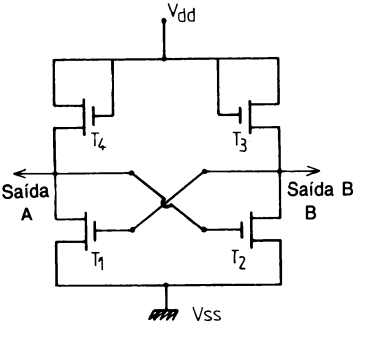


Figura 32

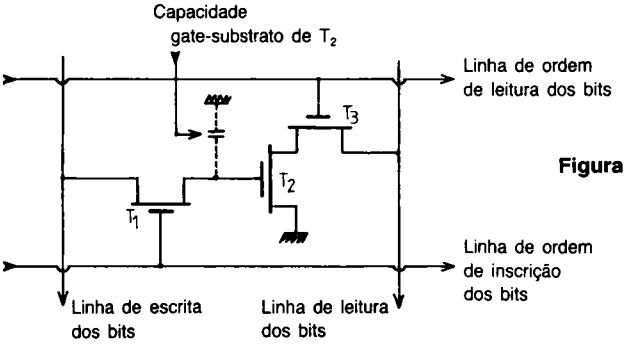
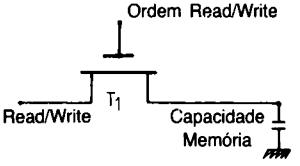


Figura 33



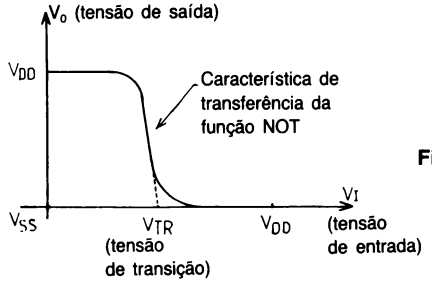


Figura 36

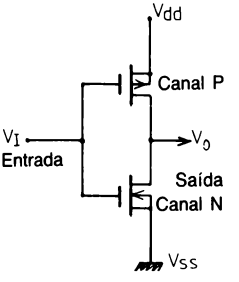


Figura 35 — Esquema eléctrico de um inversor C-MOS.

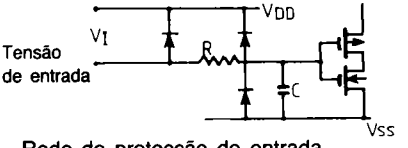


Figura 37 — Rede de protecção de entrada de díodo duplo.

Primeiramente vejamos o caso da memória **RAM**-*estática*. Representamos aqui um *flip-flop* (MOS). T_1 e T_2 são carregados por T_3 e T_4, cujas gate e source se encontram curto-circuitadas e funcionam como *resistência de carga*. Uma báscula assim montada tem dois possíveis *estados estáveis* (nível alto e nível baixo). O flip-flop pode conservar *indefinidamente* uma qualquer informação (0 ou 1) desde que não seja interrompida a sua alimentação. É por esta razão que ele define um *ponto memória* de uma RAM designada por estática.

Em seguida pode-se utilizar igualmente a tecnologia MOS no caso das memórias **RAM** *dinâmicas*. Desta vez utiliza-se o efeito de capacidade gate-substracto dos transístores MOS como elemento de memorização de um estado (0 ou 1). No entanto, a carga do condensador degrada-se em 5 ou 10 milisegundos, sendo portanto necessário *refrescar* a memória, enviando-lhe periodicamente, por exemplo de 2 em 2 milisegundos, uma carga que compensa a corrente de fuga (perdas de capacidade gate-substracto).

Define-se assim uma memória RAM dinâmica cuja vantagem consiste em utilizar apenas 3 transístores em vez de 4 para definir um *ponto memória* idêntico ao de um flip-flop (ver figura 33).

Nas memórias de grande capacidade, por exemplo nas RAM dinâmicas de 16K bits, conseguem-se ainda melhores resultados utilizando apenas um único transístor MOS e reforçando a capacidade do transístor MOS (T_1 da figura 34) com capacidades suplementares. Este método permite ganhar ainda espaço, tornando deste modo possível incluir um maior número de junções numa mesma pastilha de silício.

2. Utilização dos transístores C-MOS na função lógica NOT

A figura 35 mostra-nos o princípio de ligação de um *inversor* **CMOS**, que efectua portanto a função lógica NOT. A tensão de alimentação V_{DD} na entrada V_1 coloca em condução o transístor de canal N e bloqueia o transístor de canal P, de tal modo que a tensão de saída é igual a V_{SS}. Do mesmo modo, uma tensão de entrada igual a V_{SS} torna condutor o transístor de canal P e bloqueia o transístor de canal N, de tal modo que a tensão de saída é igual a V_{DD}. O circuito comporta-se portanto como o de um inversor, sendo a variação da tensão de saída igual à da tensão de entrada. Nesta montagem com *alimentação de tensão em série*, e em função

do nível aplicado à entrada V_1, um dos transístores fica pronto a conduzir enquanto o outro se mantém bloqueado, exceptuando a sua corrente de fuga.

A montagem só conduz verdadeiramente durante o período de mudança dos níveis de entrada, sendo portanto o consumo de energia de um tal circuito extremamente fraco.

A *característica de transferência* (figura 36) de um inversor C-MOS, apresentada nesta figura, indica a variação do nível de saída V_0 em função da tensão de alimentação aplicada V_i.

V_{TR} é a *tensão de transição* para a qual se realiza a transferência de nível de tensão no circuito.

Protecção dos circuitos de entrada (figura 37). Os circuitos C-MOS possuem *uma elevada impedância de entrada* (10^{15} ohms), e durante a sua manipulação uma carga estática de fraca energia pode bastar para criar na entrada uma alta tensão capaz de destruir um terminal não protegido. É portanto necessário montar em todos os circuitos integrados C-MOS um *circuito de protecção de entrada* tal como o indicado. Uma tensão de entrada parasita é atenuada pelo filtro RC e os díodos de protecção eliminam as cristas das amplitudes.

FUNCIONAMENTO DO **TTL/S**

Vimos já que a vantagem essencial desta tecnologia no fabrico de microprocessadores poderia ser a sua rapidez de execução das operações. Infelizmente o seu consumo leva certos construtores, como a Motorola, a preferirem, para as utilizações específicas rápidas, o desenvolvimento de um microprocessador em *tecnologia* «**ECL**».

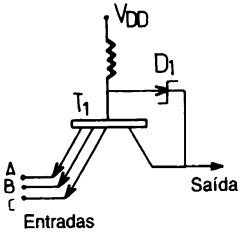


Figura 38

O esquema básico de um circuito integrado TTL utiliza na entrada um transístor de **emissores múltiplos**. Neste dispositivo, a corrente que passa através da base de T_1 nunca é interrompida. É constantemente dirigida numa das duas direcções seguintes: para os emissores ou para o colector de T_1. A carga difundida na base de T_1 encontra-se portanto sempre disponível no próprio momento das comutações, o que aumenta sensivelmente a rapidez de funcionamento. No entanto, quando o transístor está em condução, trabalhando em saturação, e deve passar ao estado bloqueado, a intensa corrente que atravessa o espaço colector-base de T_1 antes do bloqueio aumenta o tempo de comutação.

Monta-se então um *díodo Schottky* entre a base e o colector de T_1. O díodo Schottky apresenta uma *fraca corrente directa* e uma *forte corrente inversa*, pelo que ao ser montado em shunt permite um aumento da velocidade de comutação de T_1.

CARACTERÍSTICAS E EMPREGO DOS TRANSÍSTORES DE EFEITO DE CAMPO (ver figura 39).

Os transístores MOS são transístores de efeito de campo *de enriquecimento*(«enhancement mode», em inglês), o que no caso de um exemplar de canal N se traduz pelo facto de para $V_{GS} = 0$ I_{DS} ser também igual a zero, assim como de I_{DS} aumentar (curva B) quando V_{GS} se torna cada vez mais positiva.

Os transístores de efeito de campo de *junção* (*FET* – – Field Effet Transistor) não têm, ao contrário dos transístores MOS, a *gate isolada*, sendo portanto do tipo de *empobrecimento* («depletion mode», em inglês), o que se traduz no caso de um canal N pelo facto de I_{DS} ser máxima para $V_{GS} = 0$ e de I_{DS} diminuir à medida que V_{GS} se torna negativa (curva A).

Com estes dois tipos de transístores podem-se fabricar circuitos que podem servir para o sistema de organização de um microprocessador. Assim, a Motorola introduz um PIA, o MC6820A (modo de empobrecimento), enquanto que o seu PIA clássico MC6820 (MOS canal N) é de enriquecimento.

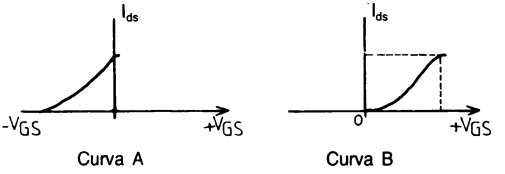


Figura 39

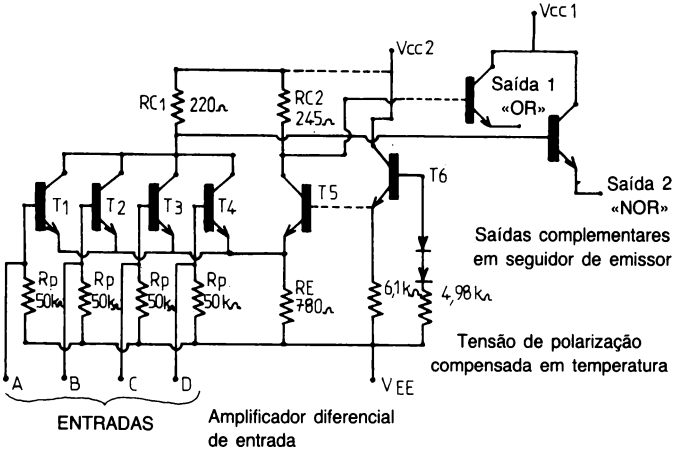


Figura 40 — Configuração geral de um circuito ECL.

CONFIGURAÇÃO GERAL DE UM CIRCUITO **ECL**

Representámos na figura 40, a título de exemplo, uma lógica ECL. Trata-se de duas portas complementares **(OR** e **NOR)**.

Esta figura mostra-nos as entradas (T_1, T_2, T_3, T_4) montadas em paralelo e em amplificador diferencial com um transístor (T_5) cuja base é polarizada por uma tensão de referência obtida no emissor de T_6. Esta tensão de referência é obtida por uma montagem clássica de compensação da variação da tensão em função das variações de temperatura.

Trata-se portanto de uma lógica de *níveis diferenciais não saturados*, o que elimina o tempo de deslocamento dos portadores num transístor quando se passa de um nível alto para um nível baixo, ou inversamente o *tempo de orientação dos portadores* quando se passa de um nível baixo para um nível alto.

Permite-se deste modo a produção de uma lógica muito rápida, que é portanto utilizada em sistemas que exigem uma *grande velocidade de execução das ordens*. É este o caso especificamente de certos microprocessadores e das respectivas memórias.

As outras características interessantes desta tecnologia são:

1. Uma alta impedância de entrada e um alto ganho de tensão.
2. Uma baixa impedância de saída (seguidor de emissor), de onde decorre a possibilidade de obter um *fan out* importante (ou seja a possibilidade de alimentar em paralelo um grande número de entradas de circuitos lógicos compatíveis).

TECNOLOGIA I^2L

I^2L significa Integration Injection Logic, ou lógica integrada de injecção (ver a figura 41). Trata-se de uma *lógica bipolar* que inclui um circuito de entrada formado por um transístor PNP montado em base à massa seguido de um transístor NPN de *colectores múltiplos*. O transístor de entrada trabalha por *injecção de corrente* no emissor do transístor PNP.

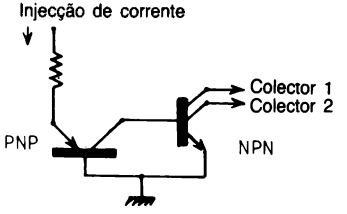


Figura 41 — Tecnologia I^2L

Já indicámos sumariamente no início deste capítulo as qualidades da tecnologia I^2L, pelo que não voltaremos a repeti-las.

Existe já um microprocessador fabricado utilizando esta tecnologia.

TECNOLOGIA SOS

Para terminar esta enumeração das tecnologias actualmente empregues nos microprocessadores e nos seus sistemas de memórias agregadas, convém citar ainda uma nova tecnologia designada SOS, que significa *Silicon on Sapphire*.

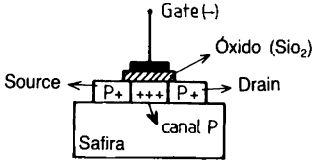


Figura 42 — Tecnologia SOS

A diferença entre a MOS-C-MOS por um lado e a SOS por outro consiste em nesta última tecnologia se utilizar já não um substracto semicondutor de silício com impurezas, mas sim um substracto isolante: a *safira*, como se mostra na figura 42, que nos apresenta um transístor **SOS** *de canal P.*

Esta tecnologia permite ao microprocessador apresentar velocidades de execução semelhantes às das TTL/S sem no entanto consumir mais energia do que a MOS ou a C-MOS.

Está actualmente a ser desenvolvido um microprocessador que utiliza esta tecnologia.

Para terminar este capítulo sobre as tecnologias de fabrico dos microprocessadores e dos seus sistemas agregados (em particular as memórias adicionais e os circuitos de interface) damos em seguida uma lista, que certamente aumentará bastante nos próximos anos, dos microprocessadores comercializados, que são aqui classificados em função da tecnologia que empregam. Alguns destes microprocessadores foram estudados por um dado fabricante e ulteriormente adoptados por outros.

É esse o caso do microprocessador **MOTOROLA MC6800**, adoptado nos Estados Unidos pela **AMI**, no Japão pela **Hitachi** e na Europa pela **SESCOSEM.**

QUADRO DOS MICROPROCESSADORES

	TECNOLOGIA **N-MOS**
Palavras de 8 bits	M6800 — MOTOROLA
	EA902 — Electronic Arrays
	F8 — FAIRCHILD
	8080 — INTEL
	CMP 8 — NATIONAL S/C
	2650 — SIGNETICS
	TMS8080 — TEXAS
	CP1611 — WESTERN
Palavras de 16 bits	CP1600 —GENERAL INSTRUMENT
Palavras de 12 bits	TLCS12 — TOSHIBA
	TECNOLOGIA **P-MOS**
Palavras de 4 bits	PPS25 — FAIRCHILD
	MCS4004 — INTEL
	MCS040 — INTEL
	IMP4 — NATIONAL S/C
	PPS4 — ROCKWELL
	TMS 1000 — TEXAS
Palavras de 8 bits	MC58/8008 — INTEL
	8008/1 — INTEL
	5065 — MOSTEK
	IMP8 — NATIONAL S/C
	PPS8 — ROCKWELL
Palavras de 16 bits	PACE — NATIONAL S/C
	TECNOLOGIA **C-MOS**
Palavras de 8 bits	COSMAC — RCA
Palavras de 12 bits	IM100 — INTERSIL
	TECNOLOGIA **ECL**
Palavras de 4 bits	M10800 — MOTOROLA

	TECNOLOGIA I_2L
Palavras de 4 bits	SB4040 — TEXAS
	TECNOLOGIA **SOS**
Palavras de 8 bits	Em desenvolvimento pela Solid State Scientific

8
ORGANIZAÇÃO DO MICROPROCESSADOR

Atingimos aqui o coração de um microcomputador, ou seja, o microprocessador, no qual as informações aritméticas ou lógicas são tratadas por ordem de um programador, segundo sequências de instruções adequadas; o jogo de instruções é específico para cada microcomputador.

Vamos explicar neste capítulo a organização geral de um microprocessador (esquema sinóptico da figura 43), e para melhor compreender a sua constituição e o seu funcionamento apresentamos em seguida, a título de exemplo, um microprocessador bem conhecido no mercado, o MC6800 da Motorola.

Todos os microprocessadores incluem geralmente:

1. Um *contador de programa* (PC — Program Counter);
2. Um *registo de instruções* (IR — Instruction Register);
3. Um *acumulador* (ACC);
4. A *unidade aritmética e lógica* (ALU)
5. Um *registo de estados* (CCR — Condition Code Register);
6. *Feixes de linhas* (BUS) que ligam o microprocessador aos elementos do sistema (memórias e circuitos de interface);
7. Um conjunto constituído pela *unidade de controlo* e um *descodificador de instruções* (IS — Instruction Decoder).

1. O CONTADOR DE PROGRAMA (PC)

Trata-se de um simples registo cuja informação é a de uma palavra binária de n bits (por exemplo, 8 bits). O PC está encarregue de provocar, sequência a sequência, o desen-

volvimento por ordem cronológica das instruções de um programa que deve ser executado, indicando continuamente os endereços da memória onde se encontram as instruções.

O PC trabalha da seguinte maneira:

Indica primeiramente o endereço da 1.ª instrução do programa a efectuar.

É em seguida *aumentado de 1* (o que significa que se acrescenta uma unidade binária à sua *palavra* binária anterior) ao mesmo tempo que se realiza a execução da primeira instrução. Quando esta se encontra terminada o PC indica então o endereço da segunda instrução do programa, sendo novamente incrementado de uma unidade binária. Os ciclos do programa a realizar prosseguem assim até estarem terminados.

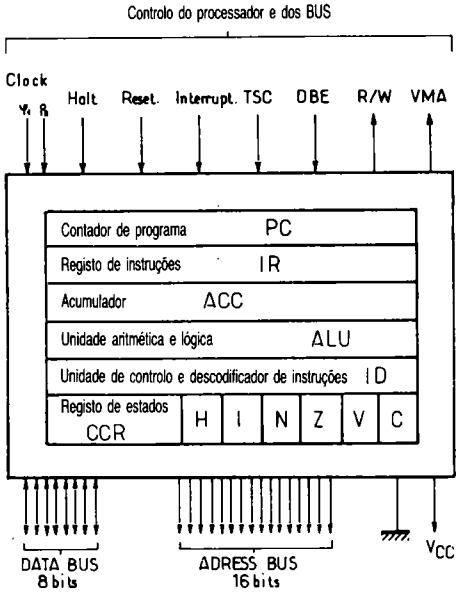


Figura 43 — Este esquema apresenta a organização de um microprocessador clássico (organização interna + comunicações externas).
Representámos aqui um microprocessador de palavras de 8 bits.
O Adress Bus contém 16 bits de endereço, o que permite uma capacidade de endereçamento na memória interna do microprocessador de 64K bytes (2^{16} palavras de 8 bits).

No entanto, em certos casos, para simplificar a resolução de um programa, pode ser necessário executar instruções já armazenadas numa outra porção do programa. O programa modifica então o conteúdo do PC através de instruções especiais.

2. O REGISTO DE INSTRUÇÕES **(IR)**

Depois da instrução ter sido lida na memória deve ser *colocada* durante todo o tempo em que é executada no *registo de instruções*. Esta instrução possui geralmente duas partes distintas:

a) O *código de operação* que indica a natureza da operação a efectuar (por exempo, *somar, passar para, incrementar,* etc.).

b) Um *endereço relativo do operando*, ou seja do dado considerado, a fim de poder utilizá-lo depois de o procurar no local da memória onde se encontra.

3. O ACUMULADOR **(ACC)**

O acumulador é um registo que efectua um trabalho complementar ao do registo de instruções. Com efeito, serve para colocar os dados a que se refere a instrução que já foi guardada no registo de instruções.

4. A UNIDADE ARITMÉTICA E LÓGICA **(ALU)**

Trata-se da unidade encarregue de resolver as operações aritméticas e lógicas. Possui pelo menos duas entradas:

— O registo de instruções deve ser ligado à primeira entrada (transmite à ALU a instrução a executar).

— O acumulador deve ser ligado à 2.ª entrada (transmite à ALU os dados a que a instrução se refere).

No próprio interior da ALU existem registos que asseguram as operações aritméticas e lógicas.

Conforme o caso, as *ordens de comando* servirão para *activar* ou não estes diferentes registos.

Finalmente, a ALU deve pelo menos possuir uma saída que permita recolher os resultados das operações efectuadas.

Deve-se notar que esta saída, conforme as necessidades, pode ser ligada através do *Data Bus* a uma memória exterior

ao microprocessador, mas que faça parte do seu sistema, ou ainda a um *circuito de interface* (**I/0** ou **PIA**) situado entre o microprocessador e um periférico.

Finalmente, essa saída pode igualmente ser ligada, se for necessário, ao acumulador.

5. O REGISTO DE ESTADO (CCR)

Como o seu nome indica, este registo permite dar a *conhecer o estado,* ou seja, indicar o conteúdo (nível lógico) num dado momento de uma parte bem definida do microprocessador.

Este registo comporta um certo número de *flip-flops* tendo cada um deles significado particular.

Os americanos chamam a estes flip-flops *flags*, ou seja, *bandeiras* tomadas no sentido de *indicadores.*

Passamos em revista os principais indicadores que existem normalmente num microprocessador e vejamos como nos podemos utilizar deles.

a) O *carry flip-flop «C»*, também chamado *registo de ligação «L»* (de «link»). Ver figura 44.

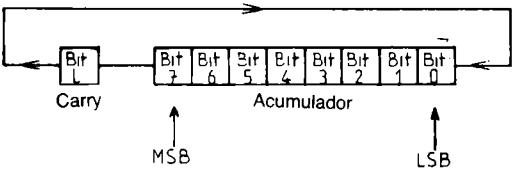


Figura 44 — O flip-flop carry («C») ou Link («L») integra-se ao nível do acumulador.

Quando é realizada uma *operação de rotação*, este flip-flop é utilizado para servir de *ligação* entre o **MSB** e o **LSB** de um número contido no acumulador.

Pode também ser considerado, no exemplo apresentado na figura 45, como um bit de *extensão no acumulador*, quando a soma de dois números de 8 bits dá lugar ao aparecimento de um bit de ordem superior. Trata-se portanto, em

suma, de um 9.° bit que deste modo não será perdido, sendo pelo contrário tomado em consideração numa soma de números de 16 bits efectuados em dois tempos num acumulador de 8 bits. O 9.° bit de A conservado no registo «link» pode então ser somado ao LSB de B a fim de fornecer o resultado correcto da soma de dois números C e D de 16 bits obtido num microcomputador de 8 bits trabalhando aqui em dupla precisão.

Soma simples A num microcomputador de 8 bits	Número de 16 bits	Soma de números de 16 bits trabalhando num sistema de 8 bits em dupla precisão
10010101 +10011010 =[1]00101111 9.° bit de ordem superior	C + D	B: 01010000 + 10100110 = 11110110 A: 10010101 + 10011010 = 00101111 + [1] ←
	= E	= 11110111 00101111 MSB LSB MSB LSB

Figura 45 — Quadro

b) *Indicador negativo «N»*. Flip-flop colocado em 1 quando o conteúdo do acumulador se torna negativo (*bit de sinal*). Ver sobre este assunto o capítulo sobre o cálculo binário.

c) O *Indicador zero (Z)* encontra-se normalmente em 0. Passa a 1 sempre que o conteúdo do ACC se torna nulo (ou seja, quando contém 8 bits 0).

Exemplo de aplicações do bit (C) ou (L) e do Bit (Z).

Pretende-se adicionar os números binários de 8 bits:

Primeiro: Lê-se na memória 11001011
Segundo: Lê-se no ACC 00110101
Resultado da soma: 1 00000000

Se se examina este caso de edição:

O bit (C) ou (L) torna-se igual a 1, o que indica a existência de um algarismo de ordem superior, ou 9° bit.

O bit (Z) indica, passando a 1, que o ACC se tornou nulo.

D) O *indicador Overflow* **(O)** ou **(OVF)** é colocado em 1 quando uma operação dá lugar a um excesso. Pode ser chamado o *bit de excesso*.

e) O *indicador interrupt (I)*. Permite aceitar ou recusar temporariamente um pedido de interrupção.

f) O *indicador half-carry (H)*. Não é utilizado em todos os microprocessadores. É em particular empregue no MC6800 da Motorola. Permite efectuar uma soma aritmética de 2 números dados no sistema BCB sem que seja necessário convertê-los um a um em binário natural, o que simplifica as sequências de instruções (ver quanto a isto o exemplo da figura 46).

Figura 46 — Quadro

Número	Soma de dois números decimais	Posição no micro	Transposição dos 2 números no sistema BCD
A	15	ACC	0001.0101
+ B	+ 28	Memória	+ 0010.1000
= C	= 43		0011 1101

Os algarismos binários apresentados à direita, em baixo, constituem o resultado da adição efectuada pelo ALU em binário, incorrecto no caso de uma adição em BCD.

Se agora graças a uma instrução especial se acrescenta simplesmente o algarismo + 6 em decimal codificado em binário, ou seja, 0110, obtém-se:

A + B		0011.1101
(somar em binário	+ 6 ⟶	+ 0000.0110
= C	= 43 ⟶	0100.0011

Desta vez o algarismo em sistema BCD 0100.0011 corresponde de facto em decimal ao número 43.

A correcção é realizada no microprocessador Motorola por meio de uma instrução particular **(DAA)**, mas provoca igualmente um transporte no bit 4. É a existência deste transporte que afecta precisamente o indicador **(H)**. Este é então colocado em 1.

O registo de estado, que contém todos os indicadores que acabamos de estudar, pode apresentar-se esquematicamente da maneira indicada na figura 47.

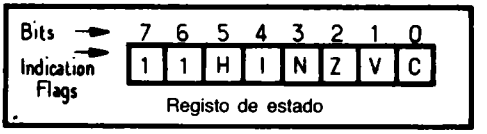


Figura 47

6. OS FEIXES DE LINHAS «**BUS**»

Já falámos deles anteriormente. Consultar quanto a isto o capítulo 2, sobre o cérebro humano e o computador ou cérebro robot. Recordemos portanto sumariamente o facto de encontrarmos em geral estes «bus» em quatro utilizações diferentes.

— Os *Data Bus*, ou bus de dados (ligações bilaterais).

— Os *Adress Bus*, ou bus de endereços das memórias (ligações unilaterais).

— Os *Control Bus*, ou bus que liga a unidade de controlo do microprocessador aos outros elementos do microcomputador.

— Os *Processor Control Bus* ou bus que conduz os sinais de comando e controlo do microprocessador.

Estudaremos mais em detalhe todos estes feixes de ligações quando analisarmos a aplicação do microprocessador MC6800.

7. UNIDADE DE CONTROLO E DESCODIFICADOR DE INSTRUÇÕES **(ID)**

O comando de certas operações de funcionamento efectuadas pelo microprocessador é devido a sinais fornecidos à sua unidade de controlo ou de descodificação de instruções.

O microprocessador pode igualmente, no sentido inverso, e através da sua unidade de comando, dirigir aos outros elementos do microcomputador (memórias, circuitos de entreface), sinais de disparo de operações.

Alguns destes sinais aplicam-se apenas a um dado microprocessador, e outros pelo contrário existem praticamente em todos os modelos.

São estes últimos que iremos examinar em primeiro lugar.

a) O *comando de relógio* (comando de *clock*). Este comando assegura uma *cadência síncrona* da execução das diversas sequências de instruções. O rigor destes relógios é geralmente definido pela frequência dos sinais de um oscilador electrónico piloto que utiliza um quartzo. Os sinais de relógio podem ser *desfasados*, como se mostra na figura 48, representando sinais de duas fases Q_1 *e* Q_2 (*sinais bifasados*). Estes sinais Q_1 e Q_2 assegurarão o cadenciar do funcionamento do microprocessador. Poder-se-ia conceber uma *cadência assíncrona* dos microprocessadores, sendo então as instruções executadas a seguir uma às outras, qualquer que fosse o tempo necessário para a realização de cada uma delas. No entanto, deve-se notar que isto introduz problemas de sincronização com os elementos externos ao microprocessador, por vezes bem difíceis de resolver. É pena que isto aconteça, pois a execução de uma cadência assíncrona das instruções é evidentemente mais rápida do que a de uma cadência síncrona que deve obrigatoriamente perder tempos mortos entre cada sequência de operações.

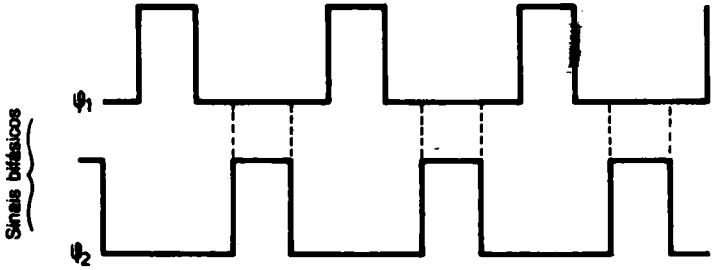


Figura 48 — Ciclo de relógio de duas fases: Q_1 e Q^2

b) O *sinal de paragem de um programa (halt* ou *stop).* Este sinal é produzido no final de um programa ou a seguir a um pedido de interrupção de um programa.

c) O *sinal de início da leitura de instruções (reset).* Este sinal é utilizado para pôr a funcionar o microprocessador ou ainda para retomar o seu funcionamento a seguir, por exemplo, a uma avaria da alimentação geral. Podem ser-lhe atribuídos outros nomes como por exemplo **START**, no primeiro caso, e **RESTART**, no segundo.

d) O *sinal de pedido de interrupção do programa* (**IRQ**, *Interrupt Request).* Este sinal pode ser emitido quando um periférico faz a um processador um pedido urgente.

O microprocessador parará a execução do seu programa a fim de satisfazer este pedido exterior, depois do que poderá retomar o programa na sequência que se segue àquela onde parou.

e) O *sinal de comando de 3.º estado* (**TSC** – *Three state control*). Já nos referimos a este sinal no segundo capítulo. Pedimos ao leitor que o consulte.

f) O *sinal de funcionamento do Data Bus (DBE – Data bus enable*). Este sinal estabelece a ligação entre a unidade de controlo e de comando do microprocessador e as diferentes memórias do sistema onde estão armazenados dados.

Já examinámos os principais sinais que actuam sobre a unidade de controlo de um microprocessador.

Resta-nos agora observar os principais sinais saídos desta unidade de controlo e que comandam os circuitos do microcomputador externos ao microprocessador (memórias e circuitos de interface com periféricos).

g) O *sinal Read/Write* (**R/W**). É o sinal que permite ler ou escrever um dado na memória, sendo a ligação entre o microprocessador e esta memória assegurada pelo «data bus». Suponhamos por exemplo que para um nível lógico H nos encontramos na posição *read* (leitura). Neste caso, leremos o dado existente em memória, que circulará no data bus no sentido memória —› microprocessador a fim de ser colocado no acumulador do microprocessador; pelo contrário, para um nível lógico B, estaremos então na posição *write*, isto é deveremos introduzir na memória o dado que acaba de ser elaborado na **ALU** do microprocessador. Os sinais circulam então no data bus no sentido contrário: microprocessador —› memória.

b) O *sinal de validação memória —› endereço* **(VMA)**. Este sinal valida um endereço no adress bus que pode activar uma RAM ou ainda interfaces com periféricos, por exemplo um PIA.

8. O MICROPROCESSADOR **MC6800**

Agora que acabámos de definir sumariamente a organização geral de um microprocessador, iremos descrever, para nos familiarizar com eles, um modelo que existe no mercado: MC6800 da Motorola. Este estudo permitir-nos-á sobretudo ter uma ideia de como o podemos utilizar.

A exposição incluirá os seguintes parágrafos:

— 8.1 – Generalidades
— 8.2 – Ligações dos terminais
— 8.3 – Configuração sinóptica
— 8.4 – Aspectos particulares do MC6800.
— 8.5 – Jogo de instruções – A linguagem mnemónica
— 8.6 – *Modos de endereçamento*
— 8.7 – Quadros de manipulação das instruções.
  a) no acumulador e na memória
  b) no *registo de índice* e no stack
  c) instruções de *saltos e ramificações*
  d) no *registo de estado*.

8.1 — GENERALIDADES

O microprocessador MC6800 da Motorola é um microprocessador monolítico de 8 bits, empregando a tecnologia MOS de canal N.

Porta de silício. É encapsulada num envólucro DIL que possui 40 terminais de saída.

O MC6800 é a unidade central, que associada aos outros circuitos integrados (elementos de memória ou de interfaces com os periféricos) da família MC6800 determinam sistemas de microcomputadores. A escolha e a composição dos circuitos da família MC6800 podem ser realizadas em função das necessidades específicas de utilização.

Os produtos da família M6800, por exemplo o MC6800, são compatíveis com os produtos da família TTL. São alimentados por uma única tensão de +5V. Não há necessidade de Buffers externos TTL ligados aos circuitos Bus de interface.

O MC6800, dotado de um adress Bus de 16 bits, tem uma capacidade de endereçamento de memória de 65K bytes. O endereçamento em memória pode ser efectuado segundo 7 **modos** diferentes. Mais adiante explicaremos como se organiza cada um destes modos.

Vejamos a designação destes:

Modos de endereçamento:

— *Acumulador — Accumulator adressing* (ACCX)

— *Imediato — Immediate adressing*
— *Directo — Direct Adressing*
— *Extenso — Extended Adressing*
— *Indexado — Indexed Adressing*
— *Implícito — Implied Adressing*
— *Relativo — Relative Adressing*

O «Data bus» do MC6800 é bidireccional, dotado de uma lógica de três estados: níveis H e B e impedância infinita.

O MC6800 inclui seis registos internos:

— Dois acumuladores: *A e B* + ALU
— Um *registo de índice*
— Um registo de salvaguarda dos dados designado por *Stack Pointer*, um registo de estado, um registo de instruções e o «program-counter».

Explicaremos mais adiante o funcionamento dos registos especiais do MC6800, ou seja, do *Stack Pointer* e do registo de indexação.

Existem 72 instruções em linguagem mnemónica.

O sinal de relógio é bifasado (Q_1 e Q_2) e é obtido a partir de um quartzo de 1 MHz.

Pode trabalhar com um periférico em DMA.

8.2 — DISTRIBUIÇÃO DE TERMINAIS DO MC6800

Ver a figura 49.

8.3 — CONFIGURAÇÃO SINÓPTICA DO MC6800

Ver a figura 50.

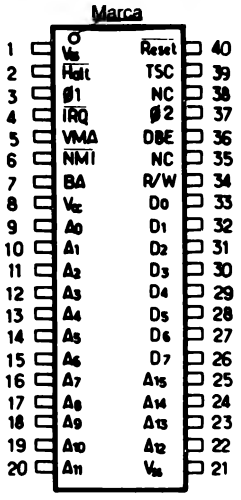


Figura 49 — Disposição dos terminais do MC6800

Primeiramente, quando a instrução apresenta uma barra (exemplo, RESET), isto significa que a validação é efectiva para um sinal B = 0 em lógica positiva. Quando a instrução não tem barra (exemplo: TSC), à validação é efectiva para um sinal H =1 em lógica positiva.

A_0 a A_{15} — linhas do feixe de Adress Bus.

D_0 a D_7 — linhas do feixe de Data Bus

V_{SS} — massa, tensão de referência (terminais 1 e 21)

V_{CC} — tensão de alimentação positiva (terminal 8).

Instruções especiais **(MC6800):**

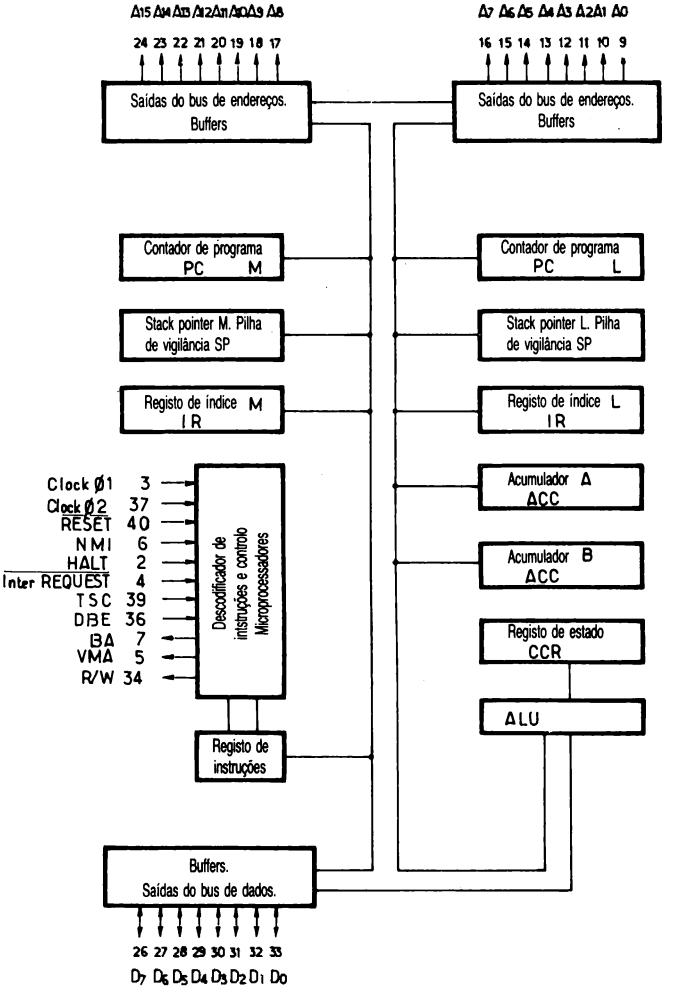
BA — *Bus available* (bus disponível).

IRQ — Pedido de interrupção. O programa em curso é terminado antes de se aceitar o pedido de um periférico.

NMI — Interrupção Não Mascarável (Non Maskable Interrupt). O programa em curso é interrompido no final da sequência de instruções que está a ser tratada, sendo então atribuída a prioridade ao pedido de um periférico. Quando o programa do periférico está esgotado, é retomado o programa anterior até terminar.

Figura 50 — Configuração sinóptica do MC6800. O microprocessador MC6800 funciona a 8 bits.

O endereçamento realiza-se sobre 16 bits a fim de aumentar a capacidade da memória, trabalhando o microprocessador em dupla precisão, isto é, é endereçado sobre duas palavras de 8 bits. O **PC**, o **SP** e o **IR** trabalham com 16 bits de endereçamento, e são representados no esquema em 2 partes de 8 bits: a parte L (Less – menor ordem) e M (Most – maior ordem). Pelo contrário tanto o acumulador A como o acumulador B trabalham apenas com palavras de 8 bits, assim como o BUS das saídas bilaterais de dados (↕) . As saídas de «Bus Adress» unilaterais (↕) estão divididas em 2 partes de 8 bits.


A15 A14 A13 A12 A11 A10 A9 A8
24 23 22 21 20 19 18 17
A7 A6 A5 A4 A3 A2 A1 A0
16 15 14 13 12 11 10 9
Saídas do bus de endereços. Buffers
Saídas do bus de endereços. Buffers
Contador de programa PC M
Contador de programa PC L
Stack pointer M. Pilha de vigilância SP
Stack pointer L. Pilha de vigilância SP
Registo de índice M IR
Registo de índice L IR
Acumulador A ACC
Acumulador B ACC
Registo de estado CCR
ALU
Clock Ø1 3
Clock Ø2 37
RESET 40
NMI 6
HALT 2
Inter REQUEST 4
TSC 39
DBE 36
BA 7
VMA 5
R/W 34
Descodificador de instruções e controlo Microprocessadores
Registo de instruções
Buffers. Saídas do bus de dados.
26 27 28 29 30 31 32 33
D7 D6 D5 D4 D3 D2 D1 D0

8.4 — ASPECTOS PARTICULARES DO MC6800

As particularidades do MC6800 que iremos examinar são as seguintes:

a) Existência de 2 acumuladores **(ACCA** e **ACCB)** em vez de um só.

b) Existência de 2 registos especiais, um designado *Stack Pointer* e outro *Registo de índice* **(IR)**.

Exame dos dois acumuladores:

Permitem:

a) Uma economia em tempo de execução;

b) Uma economia em posições de memória (menos instruções e dados a representar).

Com efeito, pode-se realizar primeiramente o armazenamento de um resultado parcial programado num acumulador (por exemplo, o número 4 em ACCB), enquanto se efectua um cálculo no outro acumulador (por exemplo 2 + 5 no acumulador ACCA).

Finalmente, pode-se somar 4 a 7 transferindo o conteúdo de ACCB para ACCA).

O cálculo assim efectuado é evidentemente menos demorado do que no caso de ser necessário procurar o algarismo 4 numa memória exterior ao CPU, economizando-se ainda uma posição de memória.

Exame do Stack Pointer (SP):

O «Stack Pointer» é uma pilha LIFO (ver explicação no capítulo 2).

A sua importância manifesta-se quando se tratam as interrupções ou sub-rotinas.

Deve-se com efeito salvaguardar na memória uma parte de um programa em curso, durante o tempo em que se desenvolve um outro programa resultante de um pedido de interrupção prioritário de um periférico **(NMI)** ou ainda o tempo durante o qual se desenvolve um pedido de subprograma ou *sub-rotina* comandada por uma instrução de salto **(JUMP)**.

O tratamento de um Stack Pointer faz intervir duas operações:

A operação de *Pushing*, durante a qual o acumulador é esvaziado na memória do Stack Pointer num endereço definido.

A operação de *Popping* durante a qual, pelo contrário, o dado armazenado na memória do SP é esvaziada para o acumulador.

Na figura 51 fornece-se um exemplo e uma explicação do funcionamento do *stack*.

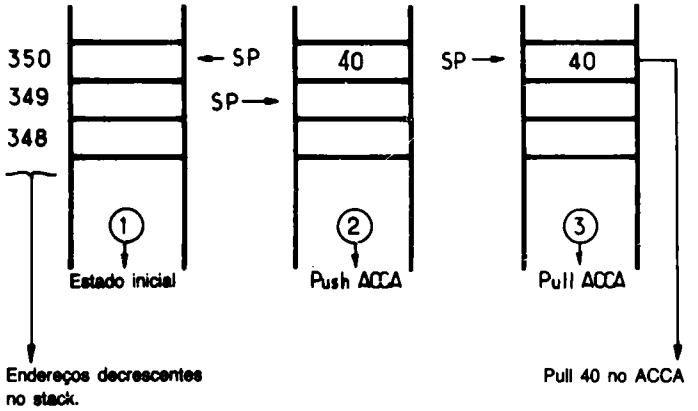


Figura 51 — ① — Antes de começar uma operação de *pushing*: *inicializar* o SP no endereço disponível de partida (exemplo: 350).
② — Operação de *pushing*. O conteúdo do ACCA (exemplo 40) é carregado **(LOAD)** no endereço 350 e o SP é *diminuído* de 1. Indica portanto o endereço 349 (350-1).
③ — Operação de *popping*. O SP é *aumentado* de 1. Indica assim o endereço 350 e esvazia o seu conteúdo (40) passando-o para o ACCA (349 + 1).
④ — Na operação de *pushing*, o SP indica o endereço 349, pelo que se poderia introduzir igualmente neste o conteúdo eventual do ACCB e em seguida *aumentar* novamente de um. Depois de terminada a operação de *pushing* e de os acumuladores estarem esvaziados (conteúdo igual a zero), o microprocessador pode efectuar depois de um *interrupt* o pedido programado num periférico ou então, depois de uma instrução *jump* (salto), um pedido de *subrotina*. Depois de estes pedidos serem satisfeitos, pode-se passar à operação de *popping* a fim de recuperar o programa inicial e continuá-lo normalmente até estar terminado.

EXAME DO REGISTO DE ÍNDICE

Permite executar as instruções de acordo com o modo de endereçamento indexado (veremos o funcionamento deste modo no parágrafo reservado aos diferentes modos de endereçamento).

8.5 — JOGO DE INSTRUÇÕES DO MC6800 E LINGUAGEM MNEMÓNICA NELE USADA

1. ABA – Add Accumulators.
2. ADC – Add with carry.
3. ADD – And.
4. AND – Logical And.
5. ASL – Arithmetic Shift Left.
6. ASR – Arithmetic Shift Right.
7. BCC – Branch if carry clear.
8. BCS – Branch if carry set.
9. BEQ – Branch if equal to zero.
10. BGE – Branch if greater or equal to zero.
11. BGT – Branch if greater than zero.
12. BHI – Branch if higher.
13. BIT – Bit test.
14. BLE – Branch if less or equal.
15. BLS – Branch if lower or same.
16. BLT – Branch if less than zero.
17. BMI – Branch if minus.
18. BNE – Branch if not equal to zero.
19. BPL – Branch if plus.
20. BRA – Branch always.
21. BSR – Branch to subroutine.
22. BVC – Branch if overflow clear.
23. BVS – Branch if overflow set.
24. CBA – Compare accumulators.
25. CLC – Clear carry.
26. CLI – Clear interrupt Mask.
27. CLR – Clear.
28. CLV – Clear overflow.
29. CMP – Compare.
30. COM – Complement.
31. CPX – Compare Index Register.
32. DAA – Decimal Adjust.
33. DEC – Decrement.
34. DES – Decrement Stack Pointer.
35. DEX – Decrement Index Register.
36. EOR – Exclusive OR.
37. INC – Increment.
38. INS – Increment Stack Pointer.
39. INX – Increment Index Register.
40. JMP – Jump.
41. JSR – Jump to subroutine.
42. LDA – Load Accumulator.
43. LDS – Load Stack Pointer.
44. LDX – Load Index Register.
45. LSR – Logical Shift Right.
46. NEG – Negate.
47. NOP – No operation.
48. ORA – Inclusive OR Accumulator.
49. PSH – Push Data.
50. PUL – Pull data.
51. ROL – Rotate Left.
52. ROR – Rotate right.
53. RTI – Return from interrupt.
54. RTS – Return from subroutine.
55. SBA – Subtract Accumulators.
56. SBC – Subtract with carry.
57. SEC – Set carry.
58. SEI – Set interrupt Mask.
59. SEV – Set Overflow.
60. STA – Store Accumulator.
61. STS – Store Stack Register.
62. STX – Store index register.
63. SUB – Subtract.
64. SWI – Software Interrupt.
65. TAB – Transfer Accumulators.
66. TAP – Transfer accumulators to condition code Register.
67. TBA – Transfer accumulators.
68. TPA – Transfer condition code register to accumulator.
69. TST – Test.
70. TSX – Transfer Stack Pointer to index register.
71. TXS – Transfer Index Register to Stack Pointer.
72. WAI – Wait for interrupt.

Reconhecemos aqui a maior parte das instruções que já mencionámos nos capítulos anteriores. Assim, as instruções 1 a 3 são instruções de *soma*, e por exemplo a instrução 2 deve ter em conta o indicador *Carry «C»*.

As instruções 55, 56 e 63 servem para efectuar *subtracções*.

As instruções 4, 36 e 48 realizam operações de *funções lógicas*

As instruções 7 a 23 servem para executar *ramificações*.

As instruções 25 a 28 servem para *apagar (clear)* os bits colocados nos diversos registos mencionados no microprocessador.

As instruções 33 a 35 servem para *diminuir* o registo especificado.

As instruções 37 a 39 servem para *incrementar* o registo especificado.

As instruções 42 a 44 servem para *carregar (load)* o registo especificado.

As instruções 60 a 62 vão permitir *guardar (store)* o registo especificado.

As instruções 65 a 71 servem para efectuar *operações de transferência* de um registo para outro.

Etc.

Resta ainda uma dúvida: como utilizar de uma maneira prática todas estas instruções, ou por outras palavras, como programar aquilo de que necessitamos a fim de que o computador nos possa responder.

Se se deseja conhecer todos os detalhes relativos à programação é necessário consultar o manual de programação específico de cada microprocessador, por exemplo o do MC6800 que possui mais de 150 páginas.

No entanto, no âmbito desta exposição, iremos agora examinar os diferentes *modos de endereçamento*; além disto, para termos uma ideia mais completa da programação, reproduziremos tabelas de *manipulação das instruções*.

8.6. — OS DIFERENTES MODOS DE ENDEREÇAMENTO

Pudemos verificar que o registo de instruções comporta duas partes:

	(Dual Operand)	ACCX	Immediate	Direct	Extended	Indexed	Implied	Relative		(Dual Operand)	ACCX	Immediate	Direct	Extended	Indexed	Implied
ABA		•	•	•	•	•	2	•	INC		2	•	•	6	7	•
ADC	x	•	2	3	4	5	•	•	INS		•	•	•	•	•	4
ADD	x	•	2	3	4	5	•	•	INX		•	•	•	•	•	4
AND	x	•	2	3	4	5	•	•	JMP		•	•	•	3	4	•
ASL		2	•	•	6	7	•	•	JSR		•	•	•	9	8	•
ASR		2	•	•	6	7	•	•	LDA	x	•	2	3	4	5	•
BCC		•	•	•	•	•	4	•	LDS		•	3	4	5	6	•
BCS		•	•	•	•	•	4	•	LDX		•	3	4	5	6	•
BEA		•	•	•	•	•	4	•	LSR		2	•	•	6	7	•
BGE		•	•	•	•	•	4	•	NEG		2	•	•	6	7	•
BGT		•	•	•	•	•	4	•	NOP		•	•	•	•	•	2
BHI		•	•	•	•	•	4	•	ORA	x	•	2	3	4	5	•
BIT	x	•	2	3	4	5	•	•	PSH		•	•	•	•	•	4
BLE		•	•	•	•	•	4	•	PUL		•	•	•	•	•	4
BLS		•	•	•	•	•	4	•	ROL		2	•	•	6	7	•
BLT		•	•	•	•	•	4	•	ROR		2	•	•	6	7	•
BMI		•	•	•	•	•	4	•	RTI		•	•	•	•	•	10
BNE		•	•	•	•	•	4	•	RTS		•	•	•	•	•	5
BPL		•	•	•	•	•	4	•	SBA		•	•	•	•	•	2
BRA		•	•	•	•	•	4	•	SBC	x	•	2	3	4	5	•
BSR		•	•	•	•	•	8	•	SEC		•	•	•	•	•	2
BVC		•	•	•	•	•	4	•	SEI		•	•	•	•	•	2
BVS		•	•	•	•	•	4	•	SEV		•	•	•	•	•	2
CBA		•	•	•	•	•	2	•	STA	x	•	•	4	5	6	•
CLC		•	•	•	•	•	2	•	STS		•	•	5	6	7	•
CLI		•	•	•	•	•	2	•	STX		•	•	5	6	7	•
CLR		2	•	•	6	7	•	•	SUB	x	•	2	3	4	5	•
CLV		•	•	•	•	•	2	•	SWI		•	•	•	•	•	12
CMP	x	•	2	3	4	5	•	•	TAB		•	•	•	•	•	2
COM		2	•	•	6	7	•	•	TAP		•	•	•	•	•	2
CPX		•	3	4	5	6	•	•	TBA		•	•	•	•	•	2
DAA		•	•	•	•	•	2	•	TPA		•	•	•	•	•	2
DEC		2	•	•	6	7	•	•	TST		2	•	•	6	7	•
DES		•	•	•	•	•	4	•	TSX		•	•	•	•	•	4
DEX		•	•	•	•	•	4	•	TSX		•	•	•	•	•	4
EOR	x	•	2	3	4	5	•	•	WAI		•	•	•	•	•	9

Figura 52 — *Tabela das instruções, do seu modo de endereçamento e do seu tempo de execução em ciclos-máquina.*

a) A instrução propriamente dita designada *código operação* **(OP)** escrita em linguagem mnemónica e b), o *endereço relativo ao operando* que permite procurar o dado referido na instrução. O quadro da figura 52 dá-nos para cada instrução as suas possibilidades de endereçamento e o tempo de execução em *ciclos máquina*. O sinal de relógio é sincronizado por um quartzo de 1 MHz, podendo-se dizer que um ciclo máquina é igual a um micro-segundo.

Endereçamento acumulador **(ACCX)**. O código operação inclui três letras, o endereço do operando **(X)**, é por exemplo **A** para designar o acumulador A ou **B** para designar o acumulador B. O Assembler traduz o endereço em código máquina ou código objecto num único «byte» (palavra de 8 bits). As instruções endereçadas na coluna ACCX podem ser igualmente utilizadas no modo *extended* ou no modo *indexed*.

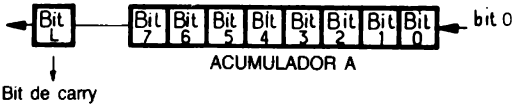


Figura 53 — *Execução da instrução **ASLA**. Exemplo: introdução de 0 no bit 0 e deslocamento do número binário de um bit para a esquerda.*
— O bit 0 toma o lugar do bit 1;
— O bit 1 toma o lugar do bit 2; etc., sendo o bit 7 conservado no bit «L».

Na figura 53 pode-se observar um exemplo deste funcionamento.

Considere-se a operação **ASLA** (instrução de *deslocamento para esquerda*) dos bits de um número aritmético no acumulador A.

A instrução-fonte ASLA é traduzida em linguagem máquina por uma palavra binária de 8 bits (1 byte) e a execução compreende no total 2 ciclos máquina.

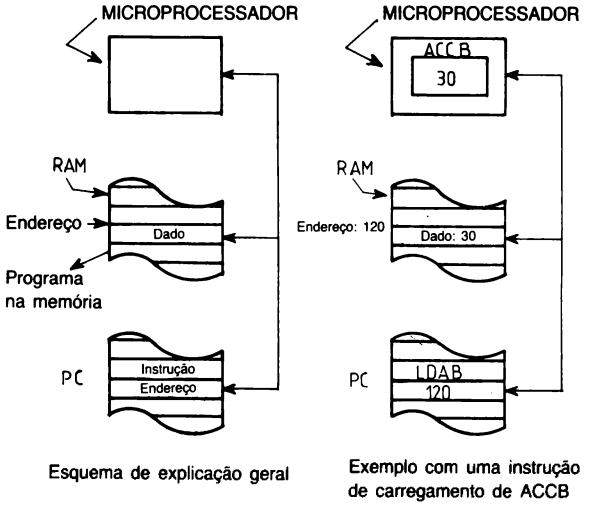
Durante o 1.° ciclo, são activados apenas o adress bus e o data bus. O dado existente em memória é passado para o acumulador A.

Durante o 2.° ciclo, a máquina executa a operação exigida pela instrução e **«VMA»** é activado (nível H).

Endereçamento imediato (#). O sinal que indica este modo de endereçamento em linguagem fonte, segundo o có-

digo operação, é # . As instruções traduzidas em linguagem objecto ocupam 2 ou 3 bytes.

Neste endereçamento, o operando não se encontra inscrito nas memória, sendo o programador, que, por exemplo, o introduz através das teclas da consola do seu periférico, dirigindo-o assim para o microprocessador através das respectivas interfaces.



Esquema de explicação geral

Exemplo com uma instrução de carregamento de ACCB

Figura 54 — Para uma boa compreensão dos modos de endereçamento do MC6800 damos aqui um esquema explicativo do modo de endereçamento directo de um dado.

O endereçamento realiza-se sobre um byte. Está portanto compreendido entre o endereço 0 e o endereço 255.

O PC indica o endereço 120. O dado existente neste endereço numa RAM é 30 (NB traduzido em código binário CAD em linguagem objecto).

O dado é introduzido no acumulador B.

O operando está contido no 2.° byte, excepto no caso das instruções **LSD**, **LDX** e **CPX**, que exigem a inscrição do operando no segundo e no terceiro bytes.

Endereçamento directo ou endereçamento absoluto.

As instruções traduzidas em código máquina comportam dois bytes. Neste endereçamento, a instrução contém o *endereço efectivo* da memória. O endereço está situado no segundo byte da instrução. O endereçamento directo permite procurar directamente as 256 palavras (2^8 possibilidades) menores da memória, ou seja, as que se encontram contidas nos endereços 0 a 255. Na maior parte da configurações de microcomputador, estes dados do programa encontram-se guardados numa **RAM**.

Endereçamento extended

Tal como no endereçamento directo, a instrução contém o endereço efectivo da memória. As instruções traduzidas em código máquina comportam 3 bytes, sendo o 3.° aquele que determina a *palavra de menor ordem do endereço do operando,* e o 2.° a *palavra de maior peso deste endereço.*

O assembler selecciona o modo de endereçamento extensivo sempre que o endereço é superior à posição 255. Quando o endereço da memória é inferior a 256, o assembler selecciona o modo de endereçamento directo.

Endereçamento indexado

Inclui uma instrução em linguagem máquina de 2 bytes. Obtém-se graças a um registo especial designado por *registo de índice.* O carácter **X** é empregue para designar o registo de índice, sendo este *modo de endereçamento escolhido* pelo assembler sempre que este carácter lhe é transmitido no código operação (OP).

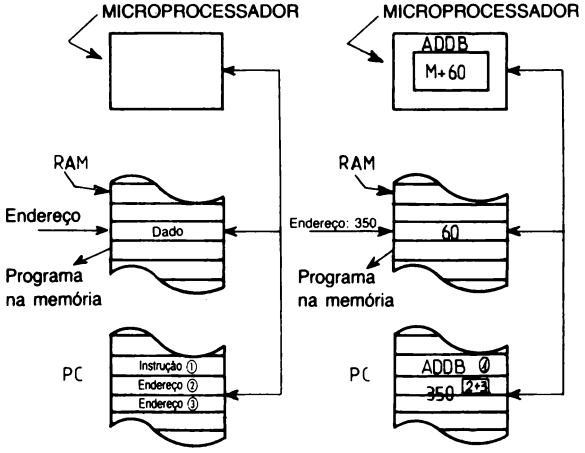
Exemplo:

INX — significa aumentar o registo de índice
LDX — carregar o registo de índice
STX — armazenar no registo de índice, etc.

O registo de índice é um registo de *16 bits* que facilita o acesso a dados sequenciais armazenados na memória. No modo de endereçamento indexado, obtém-se o endereço efectivo do operando realizando a soma **B** +**X** sendo *B* o endereço fornecido pela instrução e *X* o conteúdo do registo

de índice. O resultado conservado temporariamente durante a execução da sequência de instrução, num *registo de endereços* (**AR** — Adress Register), define o endereço efectivo escolhido na memória.

Depois de cada sequência o registo de índice é *aumentado*, sendo portanto possível endereçar todas as palavras de uma tabela completa com uma mesma instrução, no caso de estes serem programados na memória por uma ordem cronológica com um endereçamento crescente.



Esquema de explicação geral

Exemplo com uma instrução

Figura 55 — Para uma boa compreensão dos modos de endereçamento do MC6800, damos aqui um esquema explicativo do modo de endereçamento *extended* de um dado.

O endereçamento realiza-se com dois bytes 2 e 3. A instrução é traduzida pelo byte 1. Este substitui o endereçamento directo sempre que o endereço é igual ou superior a 256 na memória RAM.

M está contido no acumulador, juntamente com um número vindo do endereço 350 da RAM (número 60). O PC indica o endereço 350 da RAM (ligação por Adress Bus).

O NB 60 é lido na RAM e vai carregar o acumulador B (ligação RAM —› MPU através do data bus).

A instrução soma de 60 com o NB M é executada a seguir na ALU.

Endereçamento implícito

A instrução traduz-se em código máquina num byte. Neste modo de endereçamento «implícito», o código de operação da instrução (OP) basta para indicar o endereço do operando. Por exemplo: o código operação **ABA** (soma dos acumuladores) exige dois operandos que serão localizados no acumulador A e no acumulador B do microprocessador. O código operação determina igualmente que o resultado da execução da instrução seja guardado no acumulador A.

Existem *25 instruções* cujo código operação indica imediatamente o endereço, ou seja, o registo onde se encontra o operando (acumulador, registo de estado, stack pointer, etc).

Atenção! Certos códigos operação de uma instrução como **LDS** ou **LDX** definem igualmente o endereço do registo-destino; a instrução inclui no entanto um outro modo de endereçamento, pois o operando deve primeiramente ser procurado numa memória diferente do registo de destino. O código da instrução em linguagem máquina já não se lê então num byte mas em 2 ou 3, sendo um dos bytes reservado ainda para o código operação da instrução, e os outros para o endereço do operando.

Endereçamento relativo

A instrução deste modo de endereçamento relativo inclui dois bytes. O modo de endereçamento relativo aplica-se aqui às operações de *ramificação* (*Branch*). Sabe-se que estas ramificações permitem por exemplo a execução de instruções ou subrotinas **(BSR)** já armazenadas num certo número de endereços situados numa parte da memória diferente daquela onde se desenrola o programa em curso. É portanto necessário modificar a indicação sucessiva de endereços do programa no PC (controlador de programa). Imediatamente antes de se iniciar esta modificação, o PC indica um endereço do programa tratado. Este endereço vai ser designado *endereço efectivo* do novo programa, sendo acrescentado ao endereço relativo da instrução de «branch». Esta operação é realizada da seguinte maneira:

O endereço contido no 2.° byte da instrução de «branch» é somado ao do byte de menor ordem do contador de programa, incrementando-se em seguida duas vezes o PC. O conteúdo do carry «C» é então somado ao byte de maior ordem do PC.

A regra que se aplica ao modo de endereçamento relativo consiste em o endereço de destino da instrução de «branch» (endereço efectivo) dever encontrar-se numa gama de – 126 a + 129 bytes da instrução presente, ou seja:

$$(PC + 2) - 128 \leq D \leq (PC + 2) + 127$$

O modo de endereçamento relativo só se aplica às operações de «branch», pelo que o código operação de 3 letras da instrução basta ao assembler, sem que seja necessário ao programador indicar este modo de endereçamento através de qualquer sinal.

Operando duplo ou duplo endereçamento do operando

11 das instruções dadas na figura 52 permitem um duplo endereçamento (indicadas nesta figura por cruzes; veja-se igualmente os quadros das figuras 56 e 57). O primeiro endereçamento corresponde ao acumulador A, e o segundo ao acumulador B. O carácter que indica A ou B é escrito atrás dos 3 caracteres do código operação da instrução.

8.7. — QUADROS DE MANIPULAÇÃO DAS INSTRUÇÕES

Para terminar este capítulo sobre o microprocessador, vamos apresentar em seguida os quadros de manipulação das instruções nos diversos registos do microprocessador MC6800, juntamente com a legenda correspondente a cada um deles.

No que se refere aos algarismos rodeados de um círculo nos flip-flops do registo de estado, são por vezes necessários alguns detalhes para a sua maior compreeensão. Aconselhamos os nossos leitores a consultarem o livro de programação manual do MC6800 editado pela Motorola.

Figura 56 – Primeiro quadro das instruções sobre acumuladores e memórias

		Modos de endereçamento												Operações lógicas e aritméticas	Registo de estado					
		IMMED			DIRECT			INDEX			EXTND			IMPLIED						
OPERATIONS	MNEMONIC	OP	~	#	OP	~	#	OP	~	#	OP	~	#	OP	~	#		5 H	4 I	3 N	2 Z	1 V	0 C
Add	ADDA	8B	2	2	9B	3	2	AB	5	2	BB	4	3				A + M → A	↕	•	↕	↕	↕	↕
	ADDB	CB	2	2	DB	3	2	EB	5	2	FB	4	3				B + M → B	↕	•	↕	↕	↕	↕
Add Acmltrs	ABA													1B	2	1	A + B → A	↕	•	↕	↕	↕	↕
Add with Carry	ADCA	89	2	2	99	3	2	A9	5	2	B9	4	3				A + M + C → A	↕	•	↕	↕	↕	↕
	ADCB	C9	2	2	D9	3	2	E9	5	2	F9	4	3				B + M + C → B	↕	•	↕	↕	↕	↕
And	ANDA	84	2	2	94	3	2	A4	5	2	B4	4	3				A · M → A	•	•	↕	↕	R	•
	ANDB	C4	2	2	D4	3	2	E4	5	2	F4	4	3				B · M → B	•	•	↕	↕	R	•
Bit Test	BITA	85	2	2	95	3	2	A5	5	2	B5	4	3				A · M	•	•	↕	↕	R	•
	BITB	C5	2	2	D5	3	2	E5	5	2	F5	4	3				B · M	•	•	↕	↕	R	•
Clear	CLR							6F	7	2	7F	6	3				00 → M	•	•	R	S	R	R
	CLRA													4F	2	1	00 → A	•	•	R	S	R	R
	CLRB													5F	2	1	00 → B	•	•	R	S	R	R
Compare	CMPA	81	2	2	91	3	2	A1	5	2	B1	4	3				A − M	•	•	↕	↕	↕	↕
	CMPB	C1	2	2	D1	3	2	E1	5	2	F1	4	3				B − M	•	•	↕	↕	↕	↕
Compare Acmltrs	CBA													11	2	1	A − B	•	•	↕	↕	↕	↕
Complement, 1's	COM							63	7	2	73	6	3				$\overline{M}$ → M	•	•	↕	↕	R	S
	COMA													43	2	1	$\overline{A}$ → A	•	•	↕	↕	R	S
	COMB													53	2	1	$\overline{B}$ → B	•	•	↕	↕	R	S
Complement, 2's (Negate)	NEG							60	7	2	70	6	3				00 − M → M	•	•	↕	↕	①	②
	NEGA													40	2	1	00 − A → A	•	•	↕	↕	①	②
	NEGB													50	2	1	00 − B → B	•	•	↕	↕	①	②
Decimal Adjust, A	DAA													19	2	1	Converts Binary Add. of BCD Characters into BCD Format	•	•	↕	↕	↕	③
Decrement	DEC							6A	7	2	7A	6	3				M − 1 → M	•	•	↕	↕	4	•
	DECA													4A	2	1	A − 1 → A	•	•	↕	↕	4	•
	DECB													5A	2	1	B − 1 → B	•	•	↕	↕	4	•
Exclusive OR	EORA	88	2	2	98	3	2	A8	5	2	B8	4	3				A⊕ M → A	•	•	↕	↕	R	•
	EORB	C8	2	2	D8	3	2	E8	5	2	F8	4	3				B⊕M → B	•	•	↕	↕	R	•
Increment	INC							6C	7	2	7C	6	3				M + 1 → M	•	•	↕	↕	⑤	•
	INCA													4C	2	1	A + 1 → A	•	•	↕	↕	⑤	•
	INCB													5C	2	1	B + 1 → B	•	•	↕	↕	⑤	•
Load Acmltr	LDAA	86	2	2	96	3	2	A6	5	2	B6	4	3				M → A	•	•	↕	↕	R	•
	LDAB	C6	2	2	D6	3	2	E6	5	2	F6	4	3				M → B	•	•	↕	↕	R	•
Or Inclusive	ORA	8A	2	2	9A	3	2	AA	5	2	BA	4	3				A+M → A	•	•	↕	↕	R	•
	ORB	CA	2	2	DA	3	2	EA	5	2	FA	4	3				B+M → B	•	•	↕	↕	R	•

Figura 57 – Segundo quadro de instruções. Acumuladores e memórias

		Modos de endereçamento												Operações lógicas e aritméticas	Registo de estado					
		IMMED			DIRECT			INDEX			EXTND			IMPLIED				5	4	3	2	1	0
OPERATIONS	MNEMONIC	OP	~	#	OP	~	#	OP	~	#	OP	~	#	OP	~	#		H	I	N	Z	V	C
Push Data	PSHA													36	4	1	A → M_{SP}, SP – 1 → SP	•	•	•	•	•	•
	PSHB													37	4	1	B → M_{SP}, SP – 1 → SP	•	•	•	•	•	•
Pull Data	PULA													32	4	1	SP + 1 → SP, M_{SP} → A	•	•	•	•	•	•
	PULB													33	4	1	SP + 1 → SP, M_{SP} → B	•	•	•	•	•	•
Rotate Left	ROL							69	7	2	79	6	3				M	•	•	↕	↕	⑥	↕
	ROLA													49	2	1	A	•	•	↕	↕	⑥	↕
	ROLB													59	2	1	B	•	•	↕	↕	⑥	↕
Rotate Right	ROR							66	7	2	76	6	3				M	•	•	↕	↕	⑥	↕
	RORA													46	2	1	A	•	•	↕	↕	⑥	↕
	RORB													56	2	1	B	•	•	↕	↕	⑥	↕
Shift Left, Arithmetic	ASL							68	7	2	78	6	3				M	•	•	↕	↕	⑥	↕
	ASLA													48	2	1	A	•	•	↕	↕	⑥	↕
	ASLB													58	2	1	B	•	•	↕	↕	⑥	↕
Shift Right, Arithmetic	ASR							67	7	2	77	6	3				M	•	•	↕	↕	⑥	↕
	ASRA													47	2	1	A	•	•	↕	↕	⑥	↕
	ASRB													57	2	1	B	•	•	↕	↕	⑥	↕
Shift Right, Logic	LSR							64	7	2	74	6	3				M	•	•	R	↕	⑥	↕
	LSRA													44	2	1	A	•	•	R	↕	⑥	↕
	LSRB													54	2	1	B	•	•	R	↕	⑥	↕
Store Acmltr.	STAA				97	4	2	A7	6	2	B7	5	3				A → M	•	•	↕	↕	R	•
	STAB				D7	4	2	E7	6	2	F7	5	3				B → M	•	•	↕	↕	R	•
Subtract	SUBA	80	2	2	90	3	2	A0	5	2	B0	4	3				A – M → A	•	•	↕	↕	↕	↕
	SUBB	C0	2	2	D0	3	2	E0	5	2	F0	4	3				B – M → B	•	•	↕	↕	↕	↕
Subtract Acmltrs.	SBA													10	2	1	A – B → A	•	•	↕	↕	↕	↕
Subtr. with Carry	SBCA	82	2	2	92	3	2	A2	5	2	B2	4	3				A – M – C → A	•	•	↕	↕	↕	↕
	SBCB	C2	2	2	D2	3	2	E2	5	2	F2	4	3				B – M – C → B	•	•	↕	↕	↕	↕
Transfer Acmltrs	TAB													16	2	1	A → B	•	•	↕	↕	R	•
	TBA													17	2	1	B → A	•	•	↕	↕	R	•
Test, Zero or Minus	TST							6D	7	2	7D	6	3				M – 00	•	•	↕	↕	R	R
	TSTA													4D	2	1	A – 00	•	•	↕	↕	R	R
	TSTB													5D	2	1	B – 00	•	•	↕	↕	R	R
																		H	I	N	Z	V	C

Figura 58 – Quadro de manipulação das instruções destinadas ao registo de índice e ao «stack pointer»

		Modos de endereçamento														Operações lógicas e aritméticas	Registo de estado					
		IMMED			DIRECT			INDEX			EXTND			IMPLIED				5	4	3	2	1	0
POINTER OPERATIONS	MNEMONIC	OP	~	#	OP	~	#	OP	~	#	OP	~	#	OP	~	#		H	I	N	Z	V	C
Compare Index Reg	CPX	8C	3	3	9C	4	2	AC	6	2	BC	5	3				$X_H - M, X_L - (M + 1)$	•	•	⇕	⇕	④	•
Decrement Index Reg	DEX													09	4	1	$X - 1 \rightarrow X$	•	•	•	⇕	•	•
Decrement Stack Pntr	DES													34	4	1	$SP - 1 \rightarrow SP$	•	•	•	•	•	•
Increment Index Reg	INX													08	4	1	$X + 1 \rightarrow X$	•	•	•	⇕	•	•
Increment Stack Pntr	INS													31	4	1	$SP + 1 \rightarrow SP$	•	•	•	•	•	•
Load Index Reg	LDX	CE	3	3	DE	4	2	EE	6	2	FE	5	3				$M \rightarrow X_H, (M + 1) \rightarrow X_L$	•	•	⑨	⇕	R	•
Load Stack Pntr	LDS	8E	3	3	9E	4	2	AE	6	2	BE	5	3				$M \rightarrow SP_H, (M + 1) \rightarrow SP_L$	•	•	⑨	⇕	R	•
Store Index Reg	STX				DF	5	2	EF	7	2	FF	6	3				$X_H \rightarrow M, X_L \rightarrow (M + 1)$	•	•	⑨	⇕	R	•
Store Stack Pntr	STS				9F	5	2	AF	7	2	BF	6	3				$SP_H \rightarrow M, SP_L \rightarrow (M + 1)$	•	•	⑨	⇕	R	•
Indx Reg → Stack Pntr	TXS													35	4	1	$X - 1 \rightarrow SP$	•	•	•	•	•	•
Stack Pntr → Indx Reg	TSX													30	4	1	$SP + 1 \rightarrow X$	•	•	•	•	•	•

Legenda das figuras:

OP — Código operação em hexadecimal (o assembler compreende o código hexadecimal e a linguagem mnemónica).
~ — Número de ciclos do MPU.
 # — Número de octets do programa de instruções.
+ — Sinal aritmético da soma.
— — Sinal aritmético menos.
• — Sinal do produto lógico em álgebra booleana (AND).
+ — Or inclusivo.
⊕ — Or exclusivo.
M — Complemento de M.
→ — Transferência para
0 — Bit 0
00 — Byte 0
MSP — Local da memória indicado pelo stack pointer. As instruções do modo de endereçamento acumulador estão incluídas na coluna de endereçamento implícito.
Símbolos utilizados no registo de estado (CCR, Condition Code Register):
H (Half carry) — ver parágrafo 5 do presente capítulo
I (interrupt Mask Bit) — Idem
N (bit de sinal) — Idem
Z (Passagem do acumulador para zero) — Idem
V (overflow) — Idem
C (carry flip-flop) — Idem
R — Reset (remarcar), ver parágrafo 7
S — Set (marcar), ver parágrafo 7
⇕ — Se estiver correcto testar e carregar em memória, senão apagar
• — A instrução pode testar, mas não modificar o bit do flip-flop (ponto de memória definido pelo registo de estado).

Figura 59 — *Quadro das instruções de ramificação e saltos («branch» e «jump»)*

		Modos de endereçamento												Testes de ligações	Registo de estado					
		RELATIVE			INDEX			EXTND			IMPLIED				5	4	3	2	1	0
OPERATIONS	MNEMONIC	OP	~	#	OP	~	#	OP	~	#	OP	~	#		H	I	N	Z	V	C
Branch Always	BRA	20	4	2										None	•	•	•	•	•	•
Branch If Carry Clear	BCC	24	4	2										C = 0	•	•	•	•	•	•
Branch If Carry Set	BCS	25	4	2										C = 1	•	•	•	•	•	•
Branch If = Zero	BEQ	27	4	2										Z = 1	•	•	•	•	•	•
Branch If ≥ Zero	BGE	2C	4	2										N ⊕ V = 0	•	•	•	•	•	•
Branch If > Zero	BGT	2E	4	2										Z + (N ⊕ V) = 0	•	•	•	•	•	•
Branch If Higher	BHI	22	4	2										C + Z = 0	•	•	•	•	•	•
Branch If ≤ Zero	BLE	2F	4	2										Z + (N ⊕ V) = 1	•	•	•	•	•	•
Branch If Lower Or Same	BLS	23	4	2										C + Z = 1	•	•	•	•	•	•
Branch If < Zero	BLT	2D	4	2										N ⊕ V = 1	•	•	•	•	•	•
Branch If Minus	BMI	28	4	2										N = 1	•	•	•	•	•	•
Branch If Not Equal Zero	BNE	26	4	2										Z = 0	•	•	•	•	•	•
Branch If Overflow Clear	BVC	28	4	2										V = 0	•	•	•	•	•	•
Branch If Overflow Set	BVS	29	4	2										V = 1	•	•	•	•	•	•
Branch If Plus	BPL	2A	4	2										N = 0	•	•	•	•	•	•
Branch To Subroutine	BSR	8D	8	2											•	•	•	•	•	•
Jump	JMP				6E	4	2	7E	3	3					•	•	•	•	•	•
Jump To Subroutine	JSR				AD	8	2	BD	9	3					•	•	•	•	•	•
No Operation	NOP										02	2	1	Advances Prog. Cntr. Only	•	•	•	•	•	•
Return From Interrupt	RTI										3B	10	1		———			⑩		———
Return From Subroutine	RTS										39	5	1		•	•	•	•	•	•
Software Interrupt	SWI										3F	12	1		•	•	•	•	•	•
Wait for Interrupt	WAI										3E	9	1		•	⑪	•	•	•	•

Figura 60 — *Quadro de manipulações das instruções destinadas ao registo de estado*

		Modo de endereçamento			Operação booleana	Registo de estado					
		IMPLIED				5	4	3	2	1	0
OPERATIONS	MNEMONIC	OP	~	#		H	I	N	Z	V	C
Clear Carry	CLC	0C	2	1	0 → C	•	•	•	•	•	R
Clear Interrupt Mask	CLI	0E	2	1	0 → I	•	R	•	•	•	•
Clear Overflow	CLV	0A	2	1	0 → V	•	•	•	•	R	•
Set Carry	SEC	0D	2	1	1 → C	•	•	•	•	•	S
Set Interrupt Mask	SEI	0F	2	1	1 → I	•	S	•	•	•	•
Set Overflow	SEV	0B	2	1	1 → V	•	•	•	•	S	•
A → CCR	TAP	06	2	1	A → CCR	⑫					
CCR → A	TPA	07	2	1	CCR → A	•	•	•	•	•	•

Nota — Nos registos de estado citados nos vários quadros de manipulação das instruções: Algarismos rodeados de círculos, de 1 a 9 — 0 bit do indicador especificado na coluna correspondente do registo de estado é verificado (ver MC6800 — Programming Manual).
⑩ — Carga do stack pointer para o registo de estado (operações especiais). Ver MC6800 — programming Manual).
*⑪ — O bit é colocado quando surge um pedido de interrupção. Se já estava anteriormente, será indispensável o sinal NMI para romper o estado de espera **(WAI)**.*
⑫ — A indicar conforme o conteúdo do acumulador A.
⑬ — Os bits 6 e 7 do registo de estado encontram-se permanentemente ao nível lógico 1 (ver a figura 61).

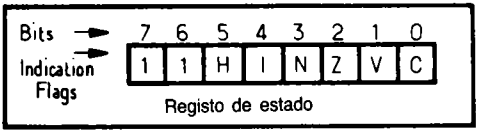


Figura 61

9
AS MEMÓRIAS

Servem para guardar os dados necessários à programação de instruções e operandos que serão tratados na unidade central do microprocessador, ou seja, na sua ALU.

São utilizadas a dois níveis:

a) AO NÍVEL DOS PERIFÉRICOS

Aqui, são necessárias para o estabelecimento de um programa a partir de um periférico que pode encontrar-se longe do microcomputador interrogado.

Trata-se normalmente de memórias de grande capacidade.

Nesta categoria, distinguem-se:

— As memórias de *cartões perfurados*.
— As memórias de *fitas perfuradas*.
— As memórias de *discos (floppy-disk)*.
— As memórias de *fitas magnéticas*.

Trata-se de memórias relativamente lentas pois para atingir uma dada informação é necessário correr todas as informações guardadas anteriormente à que se deseja.

Neste caso diz-se que se trata de uma *memória de acesso sequencial*.

Todas estas memórias têm o nome geral de *memórias de massa*.

b) AO NÍVEL DA UNIDADE CENTRAL

Originalmente empregavam-se *memórias de ferrites*.

Devido ao seu fabrico mecânico bastante complicado e ao seu fraco nível de tensão utilizável (alguns milivolts), estas foram progressivamente substituídas por memórias de circuitos integrados.

Na maior parte destes últimos circuitos, cada dado encontra-se num endereço conhecido que pode ser atingido directamente. Trata-se portanto de memórias de acesso mais rápido do que as anteriores. Diz-se ainda que são memórias de *acesso aleatório*.

Existem numerosos tipos destas chamadas *memórias centrais*.

Já falámos delas em capítulos anteriores, em particular de um modo genérico no capítulo 2 e de maneira mais detalhada no capítulo 7.

Interessando-nos mais neste livro a microinformática, falaremos mais das memórias centrais de circuitos integrados (LSI). Em particular, abordaremos:

1. As características e as definições dos parâmetros eléctricos das memórias de circuitos integrados.
2. As diversas categorias e constituição de memórias de circuitos integrados.
3. As diversas memórias, a título de exemplo, da família Motorola M6800.

1. CARACTERÍSTICAS E DEFINIÇÕES DOS PARÂMETROS ELÉCTRICOS DAS MEMÓRIAS DE CIRCUITOS INTEGRADOS

As memórias de circuitos integrados são *voláteis*.

Diz-se que uma memória é volátil quando perde a sua informação ao verificar um corte na sua tensão de alimentação. Pode dizer-se pelo contrário que uma memória em fita magnética não é volátil.

Uma memória é *destrutiva* quando a operação de leitura destrói a informação.

Certas memórias de circuitos integrados são destrutivas. É o caso das *memórias vivas* (**RAM** — Random Access Memory).

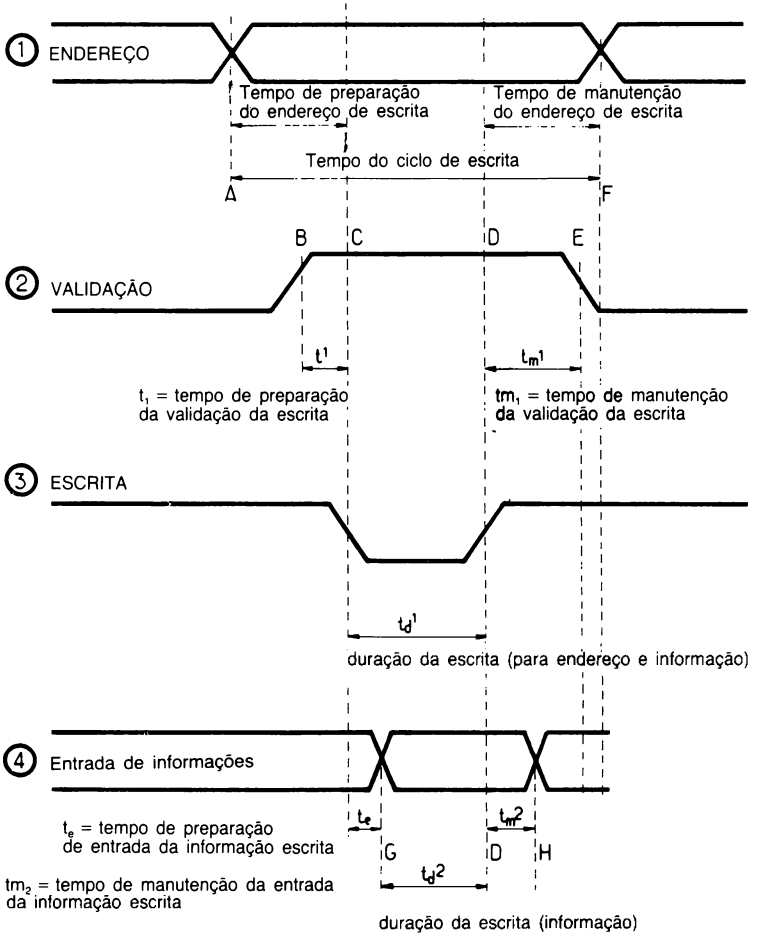
Quando a CPU necessita de um dado situado na memória, procura-a através do seu Bus de endereços no endereço em que se encontra. Depois de o atingir, lê a informação.

Então:

— Ou se trata de uma memória morta e, depois da leitura, o dado é *conservado* no seu endereço.

— Ou se trata de uma memória viva e o dado é *destruído* depois de ser lido. Isto permite obter uma nova posição livre, que pode ser preenchida por um novo dado calcu-

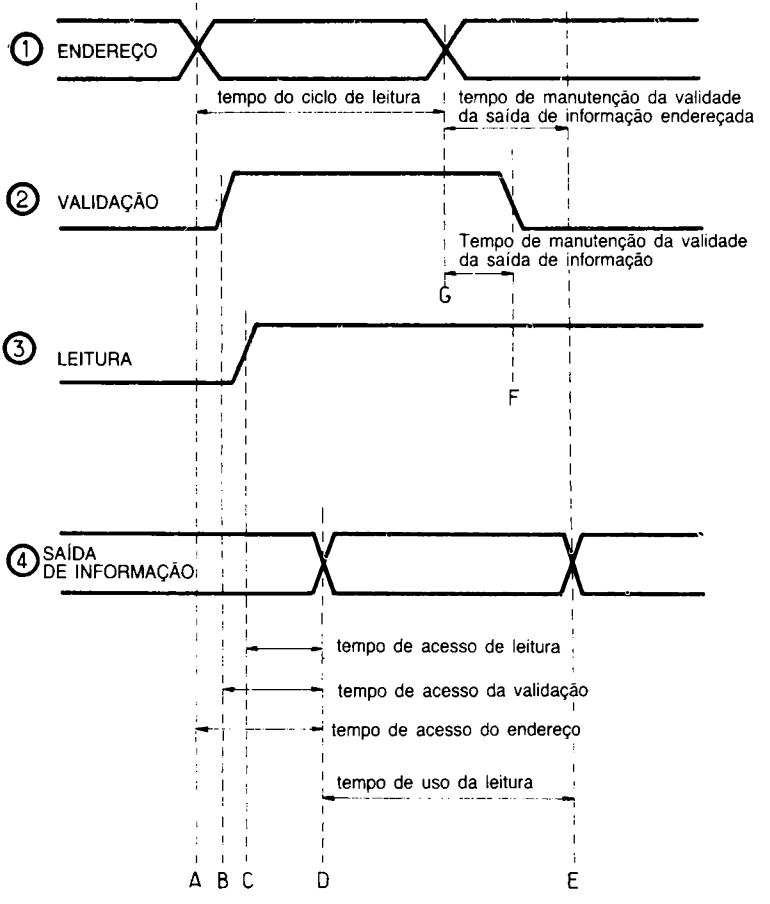
Figura 62 — Ciclo de escrita-memória



lado pela ALU. A *transferência* microprocessador —› memória realiza-se através do bus de dados (Data bus).

As características de tempos de escrita e leitura nas memórias são bastante complexas, pelo que apresentamos as figuras 62 e 63 para auxiliar a compreendê-las.

Figura 63 — Ciclo de leitura-memória



O CICLO DE ESCRITA-MEMÓRIA (figura 62)

O ciclo ① numa memória sincronizada é definido pelo cadenciar do sinal de relógio **Q**. A sua duração é portanto função da frequência do quartzo do oscilador (exemplo: um quartzo de 1 MHz irá determinar um ciclo de um microsegundo).

Em ②, trata-se do sinal **VMA** (validação do Adress Bus para a memória).

Em ③, trata-se do *tempo real de escrita dos bits* de endereçamento em primeiro lugar, seguido das informações na memória.

Em ④, trata-se da *introdução da informação* através do data bus. Esta introdução vai portanto ser realizada depois da escrita do endereço. A escrita do endereço efectua-se durante o *tempo de preparação da entrada de informação (te)*.

Por outro lado, a figura mostra que cada escrita de endereço e informação se concebe em três tempos.

Assim, para a escrita de uma informação (dado) dispõe-se de:

a) Um *tempo de preparação (te)* para a escrita;
b) Um *tempo de escrita* (td_2), propriamente dito;
c) Um *tempo de manutenção* (tm_2) de escrita.

Finalmente, o tempo de *validação* da escrita na memória inscreve-se no interior do tempo de *ciclo de escrita*, ou seja, no interior dos tempos de preparação por um lado e de manutenção da escrita por outro.

O CICLO DE LEITURA MEMÓRIA (figura 63)

Encontramos as mesmas configurações de sinais já encontradas na figura precedente sobre ①, ② e ③.

③ Pode ser definido pelo sinal *read/write* em posição *read*, vindo do microprocessador; na figura 62, este sinal deveria ser comutado para a posição *write*.

④ No que se refere a este quadro sinóptico, encontra-se ainda, para além da figura 62, as definições dos diferentes tempos de acesso.

Tempo de acesso-leitura. Trata-se do tempo compreendido entre o instante de disponibilidade do sinal de leitura (C) e o do início da leitura da informação no endereço desejado (D) (tempo CD).

Tempo de acesso-validação. É o tempo compreendido entre o instante de disponibilidade do sinal de validação (B) e o do início da leitura da informação no endereço desejado (D) (tempo BD).

Tempo de acesso de endereço. É o tempo compreendido entre o instante em que é descoberto o endereço de um dado procurado na memória (A) e o instante do início de leitura da sua informação (D) (tempo AD).

As outras características interessantes de uma memória são:

A sua capacidade: é expressa pelo número total de palavras (bytes) de «n» bits que a memória pode armazenar.

A sua cadência de transferência. Mede-se em frequência, ou seja, em «n» bits por segundo, aquilo que a memória pode aceitar (tanto em leitura como em escrita).

2. AS DIVERSAS CATEGORIAS E A CONSTITUIÇÃO DE MEMÓRIAS DE CIRCUITOS INTEGRADOS

Estas categorias de memórias de circuitos integrados são numerosas. Citaremos apenas aquelas que nos parecem mais importantes entre todos os tipos tecnológicos que florescem em cada novo período de tempo.

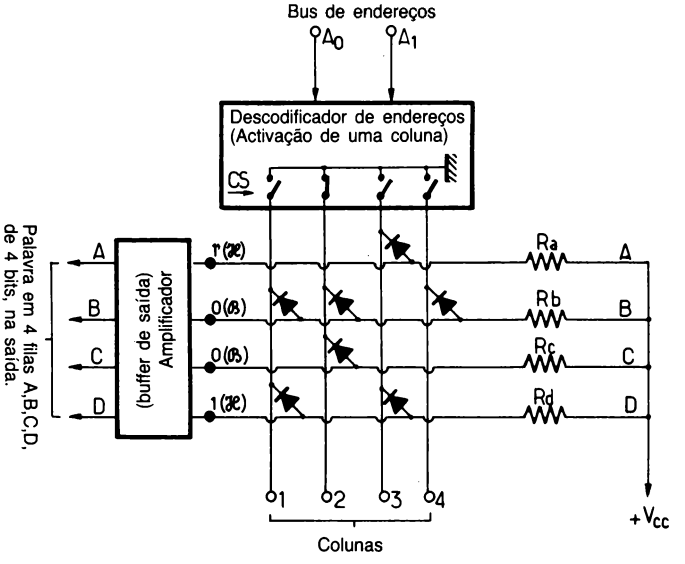
As **RAM** (Random Access Memory). Memórias vivas de acesso aleatório. Podem ser lidas, escritas ou apagadas. Existem **RAM** *estáticas e* **RAM** *dinâmicas*. Convidamos o leitor a consultar o capítulo 7, onde estas memórias já foram descritas.

As **ROM** (Read Only Memory). Memórias mortas não-destrutivas. Trata-se de memórias reservadas exclusivamente à leitura. Têm um acesso aleatório. Podem-se distinguir as **ROM** *de díodos* e as **ROM** *de transístores* bipolares ou MOS. Estas memórias são consideradas *fixas,* escritas previamente pelo fabricante sob pedido do cliente. Assim, no exemplo, da ROM de díodos que apresentamos na figura 64, a *máscara de difusão* tapou 9 pontos programados pelo cliente entre os 16 pontos possíveis de memória. Ao realizar a difusão, só serão obtidas 16-9 = 7 junções díodas. As junções díodas difundidas permitem ligar a tensão de alimentação + V_{CC} à massa, escrevendo na fila correspondente o bit 0 (nível B) desde que uma das colunas seja activada por um sinal proveniente de uma das entradas de acti-

vação do *descodificador de endereços*, designadas respectivamente por CS0, CS1, CS2 ou CS3 (CS = *Chip select*). Na coluna activada onde não existe uma junção díoda difundida numa ou mais filas A, B, C ou D, estas apresentam um bit de valor 1 (nível H).

É evidente que o custo inicial de uma ROM fabricada num exemplar único é muito elevado e que para diminuir o preço de fabrico é necessário ter necessidade de uma grande quantidade de ROM do mesmo modelo. O preço da maior parte destas só se torna rentável a partir de um mínimo de 100 a 150 peças; para quantidades menores pode-se recorrer a uma **PROM**.

Figura 64 — *Realização de uma **ROM** de díodos com 4 palavras de 4 bits.*



O bus de endereços comporta duas linhas A_0 e A_1 que podem definir na memória os quatro endereços binários seguintes:

	A_0		A_1	
1.° endereço	0	-	0	Coluna 1 —› 1010
2.° endereço	0	-	1	Coluna 2 —› 1001
3.° endereço	1	-	0	Coluna 3 —› 0110
4.° endereço	1	-	1	Coluna 4 —› 1011

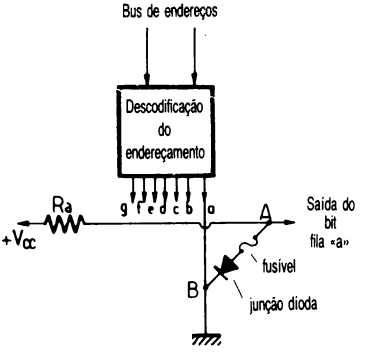
endereços de dois bits em fila ABCD

Cada um destes endereços corresponde a uma das colunas 1, 2, 3 ou 4.

Os endereços indicados são descodificados por um descodificador de endereços que activa uma das colunas (ordem de leitura dada pelo *chip select* (CS). O exemplo apresentado mostra como, depois de CS ser activado, o endereço da coluna 2 permite ler nas filas A, B, C e D a palavra de 4 bits de acesso em paralelo 1001.

As PROM(Program ROM) são **ROM** *programáveis* (evidentemente pelo utilizador). São adquiridas virgens pelo cliente. A tecnologia do seu fabrico é tal que o utilizador pode escrever nelas o que entender de acordo com as suas necessidades de tratamento de dados. Existem vários tipos de fabrico. Distinguem-se em particular as **PROM** *de fusível* e as **PROM** *de destruição da junção*. Em ambos os casos todas as junções díodas entre colunas de filas são difundidas pelo fabricante.

Figura 65 — *Exemplo de PROM de fusível.*



Se se faz passar uma corrente forte entre a fila *a* e a coluna *a*, ou seja, entre os pontos A e B, o fusível rebenta; este corte escreve no ponto de cruzamento entre A e B o bit 1 (nível H).
Se não se faz passar corrente é escrito permanentemente nessa posição o bit 0 (nível B).

Nas «junções PROM de fusível» (ver figura 65) o díodo é montado em série com um fusível de alumínio ou níquel-crómio.

Basta ao utilizador fazer passar uma corrente forte na coluna e na fila que passam pelo ponto escolhido para levar o fusível a rebentar, cortando assim o contacto ao díodo difundido. Pode-se então ler nesse ponto um bit 1 (nível H).

Nas «PROM de destruição da junção» (ver a figura 66), aplicar-se-á uma tensão inversa suficiente para destruir o díodo (2), criando assim em seu lugar um curto-circuito no ponto correspondente da memória. Pode-se assim ler nele um bit 0 (nível B).

Se é certo que as PROM são mais baratas unitariamente do que as ROM, tornam-se porém muito mais caras para grandes quantidades.

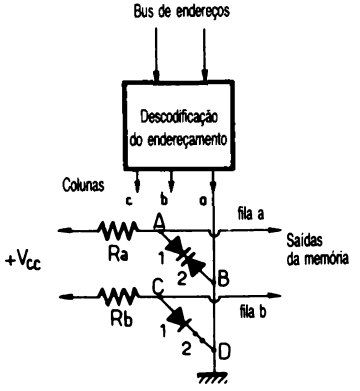


Figura 66 — *Exemplo de PROM de destruição da junção. Entre A e B, existem duas junções que mantêm na saída da fila* a *o bit (nível H) devido ao facto de o díodo estar montado ao contrário.*
Entre C e D, se se aplicar uma tensão inversa suficiente, destrói-se o díodo 2, que se comporta então como um curto-circuito. Na saída da fila b *lê-se então o bit 0 (nível B).*

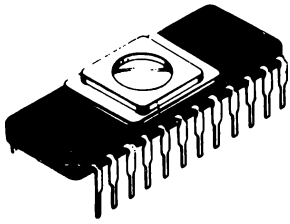


Figura 67 — *Exemplo de* ***AROM***

Observa-se na parte superior do invólucro, ao centro, a janela de quartzo.

Existem actualmente memórias PROM que podem ser apagadas e, em seguida, novamente escritas. São chamadas **REPROM**. Estas memórias são em geral muito difíceis de apagar e gravar novamente, pelo que foram de facto quase completamente abandonadas em proveito das **EPROM** e das **AROM.**

EPROM (Erasable PROM), e **AROM** (*Alterable* **ROM**) são memórias de técnica bastante semelhante.

Estas memórias de circuitos integrados comportam no seu invólucro uma pequena janela de abertura em quartzo a fim de serem sensíveis aos raios ultravioletas (UV).

Uma exposição média de 10 minutos em condições de radiação bem definidas basta para apagar toda a programação.

Os circuitos encontram-se então prontos para serem novamente programados.

3. ESTUDO DAS MEMÓRIAS QUE FAZEM PARTE DA FAMÍLIA **MOTOROLA** — M6800

1. *Memória RAM* **MC6810A** (ver figura 68).

Trata-se de uma **RAM** *estática* de tecnologia MOS canal N com gate de silício. É uma memória de *128 palavras de 8 bits.*

É compatível com todos os produtos da família M6800 e da família TTL (alimentação simples de + 5 V).

É concebida para uma utilização directa, sem buffers, com um sistema de interface de linhas BUS.

BUS de dados — D0 a D7 (palavras de 8 bits, acesso em paralelo).

BUS de endereços — A0 a A6 (7 linhas, ou seja, 128 palavras).

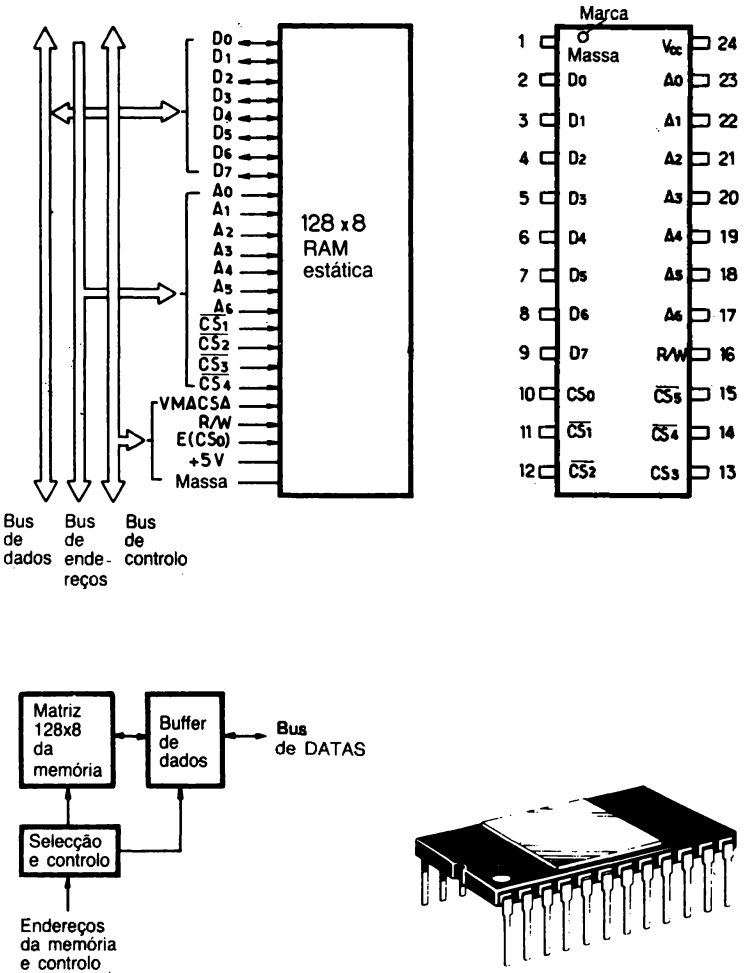


Figura 68 — Memória RAM — MCM6810A. Em cima, à esquerda: Ligações da memória ao bus de interface do microprocessador. Em cima, à direita: Distribuição dos terminais do invólucro. Em baixo, à esquerda: Diagrama de blocos do acesso à memória. Em baixo, à direita: invólucro DIL de 24 ligações.

Quatro «chip select» ($\overline{CS1}$ — $\overline{CS2}$ — $\overline{CS4}$ e $\overline{CS5}$) que activam o descodificador de endereços.

BUS de controlo: CS3 (validação endereço —› memória). R/W (read-write)

(E)CS0 — Validação dos dados

Alimentação + 5 V e massa.

— O ciclo de leitura memória é de 450 nanosegundos, mínimo.

— O tempo de acesso-endereço é de 450 nanosegundos, máximo.

— O ciclo de escrita memória é de 450 nanosegundos, mínimo.

— A duração de escrita da informação é no mínimo de 190 ns.

— O tempo de manutenção do endereço é de 10 ns, mínimo.

2. *Memória ROM*: **MCM6830A** (ver figura 69)

Trata-se de uma **ROM** *estática* de tecnologia de canal N, gate de silício. É uma memória de *1024 palavras de 8 bits.* É compatível com os produtos da família do microcomputador M6800 e também da família TTL. É concebida para ser utilizada directamente, sem buffer, com um sistema de interface de linhas BUS, ou seja:

— BUS de dados: D0 a D7 (palavras de 8 bits).

— BUS de endereços: A0 a A9 (10 linhas, ou seja, 1024 palavras: 2^{10}) mais três «chip select» de descodificação de endereços: CS1, CS2 e CS3.

— BUS de controlo: E (CS0) — validação de dados mais alimentação de + 5 V e massa.

Os níveis de activação de entrada dos «chip select» e do conteúdo da memória são definidos pelo cliente. As saídas de informações são de três estados (TSC).

O ciclo de leitura memória é de 500 ns no mínimo.

O tempo de acesso de endereço é de 500 ns no máximo.

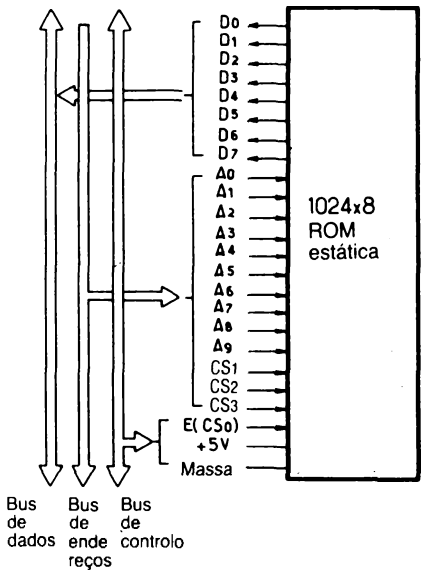
A título indicativo, apresentamos para terminar a lista das memórias da família M6800 de que o utilizador pode dispor.

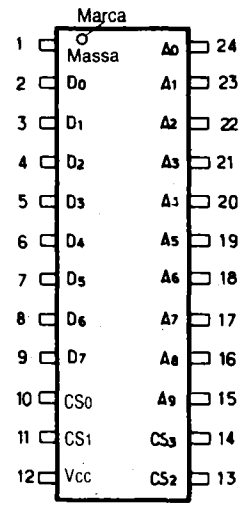
MCM6810 A —RAM estática 128 × 8.
MCM6830 e **MCM6830 A — ROM 1024** × 8.
MCM6832 — ROM 2048 × 8.
MCM68317 — ROM 2048 × 8

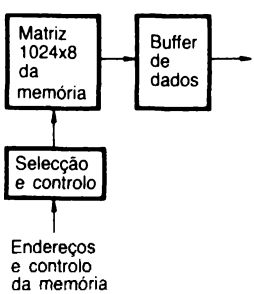
MCM6604 e **MCM6605A** — RAM dinâmica 4096 × 1.

Invólucro DIL 16 terminais

Invólucro DIL 22 terminais







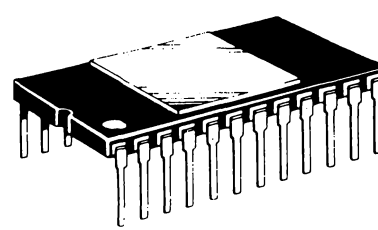


Figura 69 — Memória ROM — MCM6830 A.

Em cima, à esquerda: ligação da memória ao bus de interface do microprocessador. Em cima, à direita: Distribuição dos terminais. Em baixo, à esquerda: Diagrama de blocos do acesso à memória. Em baixo, à direita: Invólucro DIL de 24 terminais.

10
OS CIRCUITOS E OS SISTEMAS DE INTERFACE ENTRE O MICROPROCESSADOR E AS MEMÓRIAS E A UNIDADE CENTRAL E OS PERIFÉRICOS

Os circuitos de interface num microcomputador realizam a ligação entre, por um lado, o microprocessador e as memórias da unidade central e, por outro lado, os periféricos.

Os circuitos de interface constituem o sistema **I/O** (input-output) ou o sistema de *entradas-saídas* das informações do periférico para o microcomputador e vice-versa.

Porquê um tal sistema? Porque existem por natureza *incompatibilidades* de funcionamento entre um periférico e o seu microcomputador, e porque para que ambos *conversem* entre si, é necessário montar um circuito de interface capaz de adaptar o funcionamento de um deles ao do outro. Estas *adaptações* devem muitas vezes efectuar-se a três níveis:

1.° NÍVEL

Uma *adaptação de tempo*. Com efeito, um microcomputador possui velocidades de trabalho que devido à concepção do seu hardware puramente electrónico, são em geral superiores às de um periférico cuja aparelhagem comporta muitas vezes um certo número de elementos de funcionamento mecânico ou electromecânico.

2.° NÍVEL

Uma *adaptação de lógica*. A lógica do periférico pode ser com efeito diferente da do microcomputador.

3.º NÍVEL

Uma *adaptação de formato dos dados*.. O microcomputador recebe palavras em *acesso paralelo,* enquanto que o periférico transmite de um modo geral, dado o seu sistema de suporte de programação, todos os dados *em série.*

O sistema de interface vai permitir a *tradução série-paralelo* das informações.

Vejamos agora de que maneira se pode construir o sistema de comando de um automatismo industrial.

O funcionamento deste automatismo é verificado com o auxílio de *sensores*. Estes registam os dados relativos ao funcionamento do automatismo (ou seja, as sequências cronológicas de desenvolvimento das operações, medições quantitativas e qualitativas destas sequências). Em seguida transmitem-nas a um periférico de computador (esta transmissão realizar-se-á através de um circuito *conversor analógico-digital*, por exemplo). O periférico vai então interrogar o microcomputador com o objectivo de controlar os dados que recebeu e *compará-los* com o programa previamente registado numa memória da unidade central. Depois de esta operação ter sido efectuada, e em caso de necessidade, é então transmitida ao periférico uma operação de pedido de modificação quantitativa, e este actuará em consequência disso sobre os circuitos de comando do automatismo.

No decurso destas operações, verifica-se portanto uma troca de informações:

1) do automatismo para o periférico;
2) do periférico para o circuito de interface I/O;
3) do circuito de interface I/O para o microprocessador e as suas memórias centrais.

Em seguida, no retorno, verifica-se novamente uma troca de informações em sentido inverso. É portanto interessante conhecer agora a maneira como se pode realizar nesta cadeia de transmissão a troca das informações entre o microcomputador e o seu periférico.

O interface I/O deve possuir, a fim de assegurar esta troca, um *registo de estados* que controle o periférico.

Por exemplo, num dado momento, o periférico pode encontrar-se num estado de envio de dados para o microcomputador ou de recepção de resultados. Pode estar, por exem-

plo, a rebobinar uma fita magnética que faça parte da sua memória.

O interface I/O deve igualmente possuir um *registo de informações* que permita a troca dos dados.

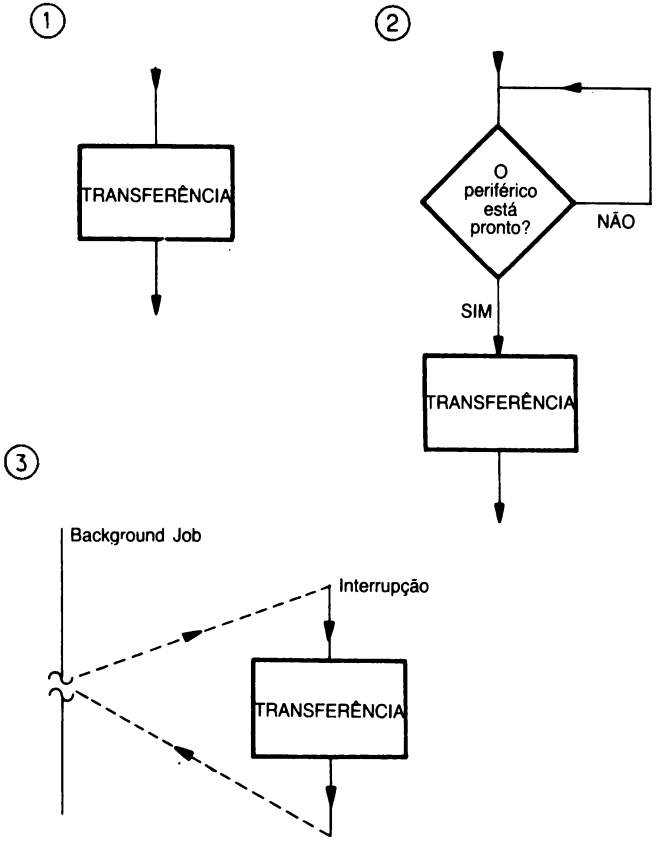


Figura 70 — *Esquemas de apresentação das transferências de entrada-saída.*

1 – Transferência incondicional. 2 – Transferência condicional. 3 – Transferência sob interrupção. Nota: o «background job» designa o trabalho inicial em curso no microcomputador.

Esta troca pode ser realizada em quatro situações diferentes a saber:

1. A *transferência incondicional*;
2. A *transferência condicional*;
3. A *transferência sob interrupção*;
4. A *transferência com interrupções múltiplas*.

Estes diferentes tipos de transferência são ilustrados na figura 70.

1. A TRANSFERÊNCIA INCONDICIONAL

Não se verifica o registo de estado pois transferem-se directamente os dados para o periférico. É então necessário conhecer perfeitamente os seus tempos de disponibilidade, o que nem sempre é evidente.

2. A TRANSFERÊNCIA CONDICIONAL

Verifica-se o registo de estado a fim de saber se o periférico se encontra disponível. A transferência de dados realiza-se assim que a resposta é positiva.

3. A TRANSFERÊNCIA SOB INTERRUPÇÃO

É, evidentemente, o meio mais seguro e mais rápido. Este problema já foi de resto tratado no capítulo 2 (ver a figura 3) e 8 (parágrafo 7 d), assim como quando se referiu as instruções **IRQ** e **NMI** do MC6800.

4. A TRANSFERÊNCIA COM INTERRUPÇÕES MÚLTIPLAS

Pode ser este o caso quando é ligado um número bastante grande de periféricos ao mesmo microcomputador.

a) «Prioridade do pedido». Se dois periféricos 1 e 2 fazem simultaneamente um pedido de interrupção, o facto de *verificar* o registo de estado de 1 em seguida de 2 permite assegurar uma *prioridade de interrupção* para 1.

b) «Prioridade da rotina de serviço». 1 irá portanto efectuar a sua *rotina de serviço*. O bit *I* do microprocessador encontra-se em «1» e *mascara* os pedidos dos outros periféricos.

No entanto, se a rotina de serviço comporta uma instrução que coloca o bit «I» (interrupt) do registo em «0», a ro-

tina de serviço pode ser por sua vez interrompida por um pedido do periférico 2, que passará então a dispor de uma prioridade de serviço superior à de 1.

Finalmente, para terminar este problema da transferência de dados, recordemos o caso particular do trabalho em **DMA** (acesso directo à memória da unidade central). Deve-se rever o que foi dito no capítulo 2, figura 4. Este método permite executar a transferência de dados entre memórias da unidade central e o periférico.

O microprocessador encontra-se mascarado durante esta operação devido à impedância infinita aplicada à sua entrada (**TSC** — Three State Control).

EXEMPLOS DE CIRCUITOS DE INTERFACE

Vamos abordar agora exemplos de circuitos de interface.

Primeiramente, o **PIA**, que possui uma ligação com o periférico realizada por um BUS de dados formado por 8 linhas (palavras de 8 bits de *acesso em paralelo*).

Em seguida, o **ACIA**, que possui uma ligação com o periférico assegurada por uma linha de informações de *acesso em série*. Assim, neste segundo exemplo, observamos que o circuito de interface adapta o formato de dados. Permite uma *transição série-paralelo* das informações a fim de que o periférico possa «conversar» com o **MPU**.

1. **PIA** (Peripheral Interface Adapter).

Trata-se de um circuito de interface integrado (representado na figura 71), que faz parte dos produtos da família M6800 da Motorola. Apresenta a grande vantagem de poder colocar directamente em comunicação o microcomputador com dois periféricos exteriores.

O PIA — **MC6820** — é um circuito integrado MOS de canal N, gate de silício. É apresentado num invólucro DIL de 40 terminais de saída.

Possui duas partes distintas:

A 1.ª parte, que assegura a ligação directa sem buffer externo aos BUS do MPU, e a 2.ª parte que assegura a ligação directa sem buffer externo aos 2 BUS de dados dos 2 periféricos ligados.

1.ª parte

O PIA compreende:

1. 8 linhas bidireccionais **(D0 a D7)** de informações (palavras de 8 bits de acesso em paralelo). Estabelecem a ligação com o BUS de dados do MPU.

2. As linhas do BUS de controlo, que são:

a) *A linha* **R/W** (Read-Write). O sinal é produzido pelo MPU a fim de controlar a direcção de transferência dos dados no BUS de dados. Um nível lógico «B» na linha R/W do PIA valida a entrada dos buffers e o dado é transferido do MPU para o PIA (MPU —› PIA). Um nível lógico «H» na linha R/W do PIA prepara este para uma transferência de dados para o BUS de dados do MPU (PIA —› MPU).

b) *A linha* **«E»** — linha de «timing» (tempo de execução da transferência dos dados). Recebe o sinal de relógio lançado pelo MPU (clock Q_2).

c) *A linha* **$\overline{\text{RESET}}$**. O nível «B» de activação da linha é empregue para colocar em zero todos os bits dos registos do PIA. Serve para fazer *reinicializar* o PIA depois de um corte de alimentação.

d) **$\overline{\text{IRQA}}$ e $\overline{\text{IRQB}}$**. O nível «B» de activação serve para efectuar um pedido de interrupção do periférico A ou B, quer directamente ao MPU quer através de um circuito de interrupção prioritária.

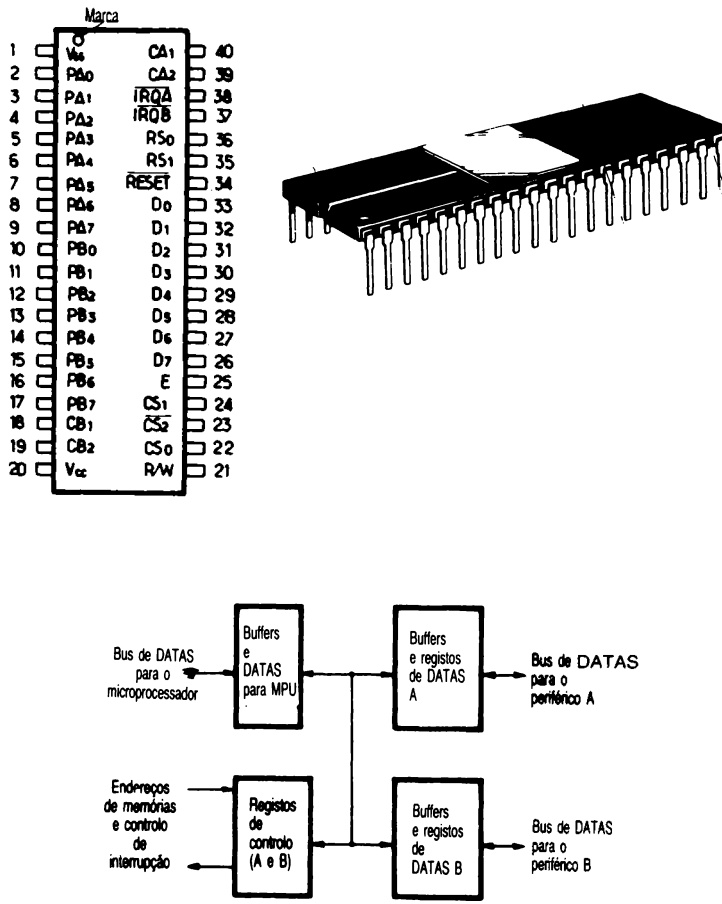


Figura 71 — O circuito de interface das ligações do PIA. À direita, em cima, Invólucro DIL de 40 terminais. Em baixo, diagrama de blocos do PIA.

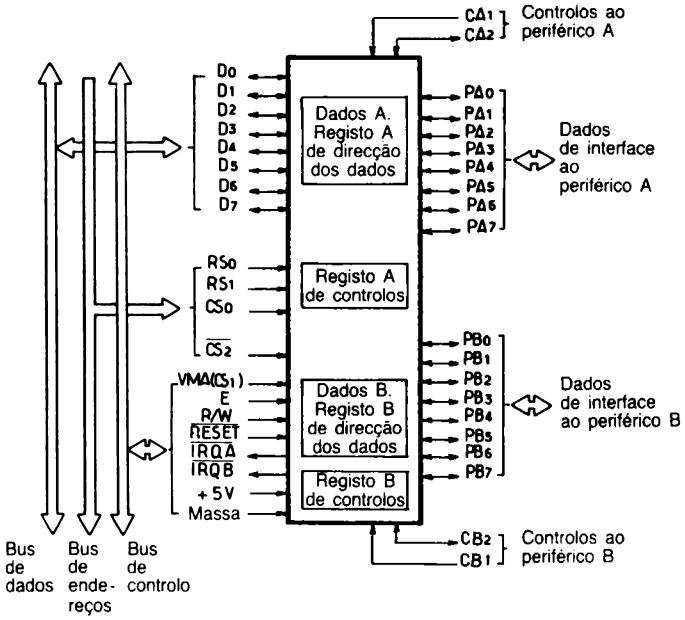


Figura 71 A — O circuito de interface PIA — MC6820

e) **CSO, CS1 e $\overline{\text{CS2}}$**. Trata-se dos três *«chip select»* que devem ser activados a fim de estabelecer a ligação entre o BUS de dados e o PIA, efectuando-se as transferências sempre sob o controlo das linhas E e R/W.

f) **RSO** e **RS1**. Trata-se dos sinais de controlo encarregados de seleccionar o registo (A ou B) em que se deve realizar a escrita ou a leitura das informações.

2.ª parte

Esta parte inclui:

A secção A e a secção B do PIA.

A secção A do PIA está ligada ao periférico A.

A secção B do PIA está ligada ao periférico B.

«A secção A» liga o periférico A ao circuito de interface através das linhas **PAO** a **PA7**, portadoras de palavras de 8 bits de acesso em paralelo.

Sendo cada uma destas linhas bilateral, o seu sentido é programado pelo «registo A de direcção de dados.»

Se o indicador (flag) deste registo se encontra em «1», estas linhas comportam-se como saídas (PIA —› periférico).

Se o indicador (flag) deste registo se encontra em «0», estas linhas comportam-se como linhas de entrada (periférico —› PIA).

Todas estas linhas admitem níveis compatíveis com os da família dos circuitos TTL.

«A secção B» liga o periférico B ao circuito de interface graças às linhas de dados **PB0** a **PB7**. Tudo o que acabamos de dizer sobre a secção A e o periférico A se aplica à secção B e ao periférico a que se encontra ligada. No entanto, pode-se acrescentar que as linhas **PB0** a **PB7** podem igualmente admitir uma lógica de três estados (TSC), o que lhes permite possuir no sentido periférico —› PIA *entradas de alta impedância.*

No sentido PIA —› periférico, podem ainda ser empregues como *fonte de corrente* de 1 mA sob 1,5 V e comandar assim, em caso de necessidade, directamente a base de um transístor de comutação.

As linhas de entradas **CA1** *e* **CB1**, são linhas de entradas que dirigem um pedido de interrupção feito pelos periféricos A e B aos *registos de controlo* correspondentes A e B.

A *linha de controlo* **CA2** do periférico A, bilateral, serve para definir uma interrupção de entrada ou ainda para controlar a saída para o periférico.

A função desta linha é programada pelo «registo de controlo A».

A *linha de controlo* **CB2** do periférico B é um linha bilateral equivalente à linha CA2. Tudo o que se disse sobre esta aplica-se igualmente a CB2, relativamente ao periférico B.

No entanto, deve-se acrescentrar que no sentido periférico —› PIA a sua entrada é de *alta impedância* e, no sentido contrário esta linha pode servir de *fonte de corrente* de 1 mA sob 1,5 V, podendo comandar directamente, em caso de necessidade, a base de um transístor de comutação.

As duas linhas CA2 e CB2 apresentam características de compatibilidade com as da família TTL.

2. O ACIA (Asynchronous Communications Interface Adapter)

Este ACIA da Motorola constitui um outro tipo de interface, diferente do PIA (ver a figura 72). É um circuito in-

tegrado MOS de canal N e gate de silício, encapsulado num invólucro DIL de 24 terminais.

Transmite as informações ao MPU — MC6800 graças a palavras de 8 bits de acesso em paralelo e através de um BUS de dados que possui as linhas **D0** a **D7**.

As informações recebidas do periférico não se encontram no *formato de acesso em paralelo*, mas sim no de *acesso em série*. A cadência destes dados é *assíncrona*.

A fim de que o MPU receba correctamente as informações do periférico verifica-se portanto uma tradução «série-paralelo» do formato dos dados no interior do ACIA. Inversamente, quando o MCU responde ao periférico, verifica-se uma *tradução «paralelo-série»* dos dados no interior do ACIA.

O ACIA pode comunicar entre o periférico e o MPU através de um **MODEM**.

O MODEM é um material especial que permite conduzir a grande velocidade e à distância de vários quilómetros, na rede de telefones, dados provenientes de um periférico (recebidos pelo MPU) ou provenientes de um MPU (transmitidos por este para um periférico). A ligação é efectuada segundo a representação sinóptica da figura 73.

O MODEM, ao receber informações provenientes de um periférico, desmodula a corrente portadora a fim de transmitir ao ACIA as informações desmoduladas do periférico e, quando transmite informações do MPU para o ACIA, modula-se a fim de as enviar à rede telefónica.

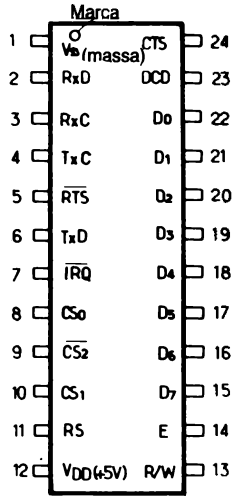
O ACIA, montado como tampão, *traduz os formatos dos dados* recebidos do periférico (tradução série-paralelo) ou dirigidos para este (tradução paralelo-série).

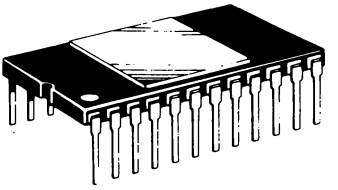
Vamos agora descrever rapidamente o ACIA.

No ACIA, distinguem-se dois registos:

Um *registo receptor de dados* que recebe na linha **(R x D)** as informações de «acesso em série» provenientes do periférico e um *registo de dados* que transmite na linha **(T x D)** as informações de «acesso em série» para o periférico.

Estes dois registos são controlados por um *registo de estado* e um *registo de controlo*, eles próprios activados exteriormente através de linhas de controlo. Estas *linhas de controlo* recebem sinais apropriados provenientes do MPU ou dirigidos a este.





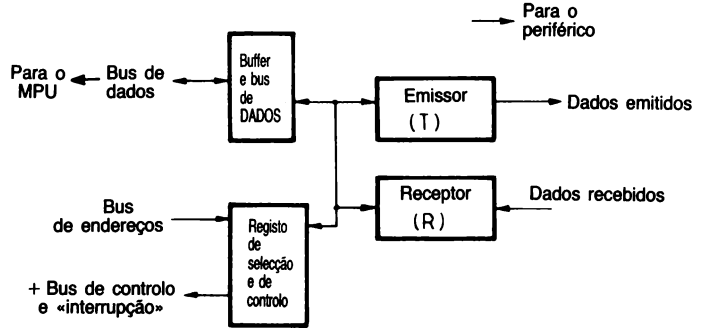


Figura 72 — O circuito de interface ACIA — MC6850

À esquerda, em cima, determinação das ligações do ACIA. À direita, em cima, Invólucro DIL de 24 terminais. Em baixo: Diagrama de blocos do ACIA.

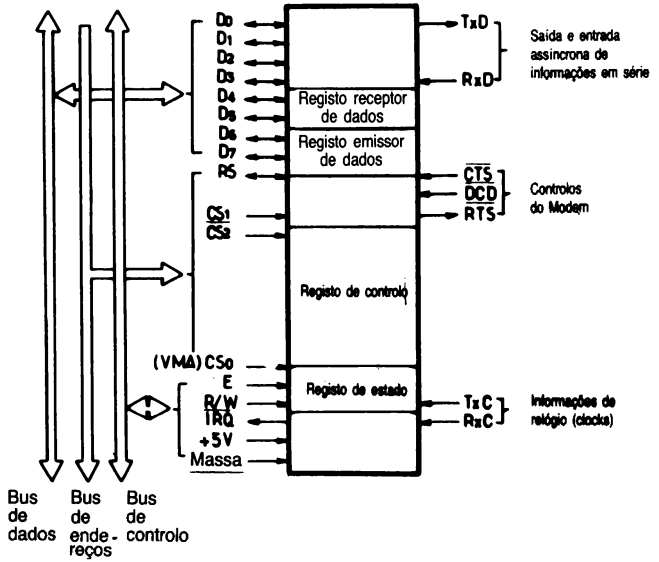


Figura 72 A — O circuito de interface ACIA — MC6850

São emitidos sinais especiais para o controlo do MODEM exterior ao ACIA **(CTS – DCD – RTS)**, assim como sinais de sincronização de relógio que são dirigidos para o ACIA **(T x C** e **R x C)**. Para mais detalhes é bom consultar as edições da Motorola.

OS AMPLIFICADORES DE **BUS**: EMISSOR-RECEPTOR

Em certos casos, a ligação entre um BUS de linhas bilaterais (exemplo: bus de dados) e o microprocessador deve realizar-se através de um circuito de interface vulgarmente designado *transceiver*, quando o bus de dados liga ao microprocessador um sistema muito carregado de circuitos de memórias e de circuitos de interfaces I/O. Neste caso o BUS em questão é muitas vezes designado, juntamente com o transceiver, um **BUS**-*extender*.

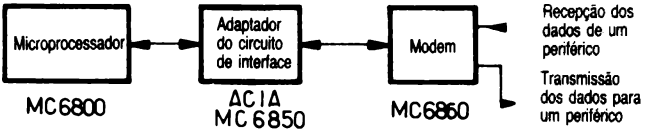


Figura 73 — Quadro sinóptico de uma ligação microprocessador periférico. MODEM significa «**Mod**ulador — **De**smodulador».

Trata-se na realidade de um conjunto de dois amplificadores montados invertidos, que amplificam em corrente os sinais provenientes do MPU em direcção às memórias (sinal de escrita «emissor»), e das memórias para o MPU (sinal de leitura «receptor»). O esquema básico desta montagem é dado na figura 74.

Vê-se a maneira como o microprocessador actua para «ler» um dado vindo da memória e apresentado na entrada (1) pelo BUS de dados.

O **strobe «1»** ou **enable E_1** autoriza o amplificador ▷ a aceitar a entrada do dado e a amplificá-lo em corrente, enquanto que o *strobe 2* ou *enable 2* impede o acesso de qualquer outro dado. Para tal, o amplificador ◁ vê a sua entrada colocada em regime de alta impedância. Do mesmo modo, para escrever um dado vindo do MPU para a memória,o strobe 2 ou enable 2 autoriza o amplificador ◁ a aceitar a passagem do dado, amplificando-o simultaneamente em cor-

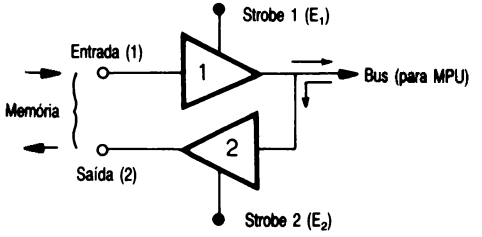


Figura 74 — *Quadro sinóptico de um circuito interface «transceiver». Exemplo de montagem: o Quad Transceiver da Motorola, MC8T26.*

rente, enquanto que o strobe 1 (E_1) impede o acesso de um novo dado colocando a entrada do amplificador ▷ em regime de alta impedância (TSC).

OBTENÇÃO DE PERIFÉRICOS ECONÓMICOS

Neste capítulo estudaram-se os circuitos de interface que permitem o diálogo entre um microcomputador e os seus periféricos.

Recordemos que se hoje é possível encontrar no mercado um conjunto microcomputador capaz de realizar um programa mínimo e possuindo além do microprocessador uma RAM, uma ROM e um circuito I/O, além de um circuito de relógio, por um preço bastante reduzido, um *telétipo* ou qualquer outro periférico do género custa pelo menos cinco vezes mais caro.

No entanto, as *unidades de disquete (floppy-disk)* recentes, com pequenas dimensões e preço relativamente baixo, permitem finalmente utilizar periféricos em diversos arranjos ligados aos microcomputadores. A disquete (ver figura 75) tem 18 cm de diâmetro e encontra-se coberta de uma face de *óxido magnético*, sendo protegido por uma capa em cartão. As outras características destes discos podem variar um pouco. A título de informação daremos as seguintes características aproximadas:

— Número de pistas: 77
— Densidade de escrita: 48 pistas por polegada.
— Capacidade em megabits: 3,1
— Velocidade de transferência em kilobits por segundo: 250
— Velocidade de rotação dos discos: 360 voltas por minuto.

Está actualmente em estudo o emprego de pequenas «cassetes» magnéticas nos periféricos de microcomputadores. A capacidade destas cassettes é da ordem dos 300 000 caracteres por fita com uma velocidade de acesso de 120 caracteres por segundo.

A *transmissão de caracteres* (algarismos e letras maiúsculas e minúsculas, além de certos sinais) realiza-se através de uma *codificação de interface*.

Existem dois códigos principais:

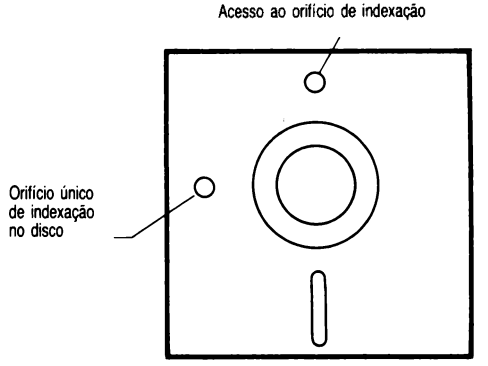


Figura 75 — *Representação de uma disquete.* Citemos a título de exemplo alguns tipos de disquetes actualmente vendidos em França.

Calcomp 140	*Memorex 651*
Control Data 9400	*Memorex 652*
DRI74	*Potter 4740*
IBM 3740	*Sagem D53*
Logabax 45	

Está actualmente em estudo o emprego de pequenas «cassetes» magnéticas nos periféricos de microcomputadores. A capacidade destas cassettes é da ordem dos 300 000 caracteres por fita com uma velocidade de acesso de 120 caracteres por segundo.

A *transmissão de caracteres* (algarismos e letras maiúsculas e minúsculas, além de certos sinais) realiza-se através de uma *codificação de interface.*

Existem dois códigos principais:

O *código* **ASCII** (American Standard Code for Information Interchange), e o *Código* **EBCDIC** (Extended Binary Coded Decimal Interchange Code), extensão do sistema BCD.

Finalmente, constroem-se actualmente microprocessadores com os respectivos circuitos de interface montados directamente nos periféricos, realizando-se assim aquilo que é chamado *terminais «pesados»,* ou ainda *terminais «inteligentes».*

11

A PROGRAMAÇÃO
COMO CONSTRUIR UM MICROCOMPUTADOR PARA UMA DADA UTILIZAÇÃO

É necessário ter presente o seguinte facto, uma verdade que deve ser aceite como um postulado com o qual não é possível transigir: o microcomputador é uma máquina *estúpida e disciplinada.*

Esta expressão mostra bem o *carácter mental* da máquina, e permite-nos compreender melhor a maneira como a devemos utilizar para obtermos os resultados que nos interessam.

Com efeito, se se pretende que o microcomputador

— Guarde informações

— Realize cálculos

e

— Tome decisões, o programador deve efectuar um programa que contenha uma dada sequência de instruções.

Cada uma destas instruções tem como objectivo prever e ditar passo a passo todos os detalhes de funcionamento e de controlo a que a máquina deve corresponder.

A programação é portanto muito importante e obriga a uma boa compreensão da organização do microcomputador e bom conhecimento da sua linguagem de expressão.

Obriga ainda o programador a aplicar muita ordem e método na elaboração do seu programa.

Para que assim seja, recomenda-se ao programador que realize o seu trabalho partindo do seguinte *ciclo cronológico*:

1.1 — PROCURAR DEFINIR O PROBLEMA A RESOLVER

Esta procura deve ser tão minuciosa quanto possível. Assim, para gerir um controlo de automatismo, é necessário

definir as diferentes sequências a controlar, ou seja, a sua ordem de execução, o seu tempo de duração, a sua qualidade, etc.

1.2 — ESCOLHER O MÉTODO MELHOR ADAPTADO À MÁQUINA A FIM DE OBTER A RESOLUÇÃO DO PROBLEMA CONSIDERADO

Os métodos a utilizar para resolver um problema são por vezes bastante numerosos. Convém então procurar aquele que melhor se adapta às qualidades do microprocessador usado. Uma representação gráfica por vezes designada por *flow-chart* pode ajudar bastante a definir ideias. Examinaremos adiante algumas modalidades de execução.

1.3 —APLICAR O MÉTODO ADAPTADO RECORRENDO A UMA ESCOLHA JUDICIOSA DE UMA ORDEM LÓGICA DE SEQUÊNCIAS DE INSTRUÇÕES A PROGRAMAR

É aquilo que se designa pela expressão *codificação do problema*. Esta operação conduz-nos a armazenar na memória central do microcomputador o programa de execução do problema que deve ser resolvido pela máquina. Neste estádio, poder-se-ia pensar que a programação fica deste modo terminada e que o homem conseguia já vencer «a estupidez» da máquina.

Infelizmente, não é isto que acontece. É ainda necessário:

1.4. —VERIFICAR O PROGRAMA

Para se ter a certeza de que o homem foi suficientemente «inteligente» na compreensão do efeito de cada instrução no interior do conjunto do programa realizado. Esta verificação pode ser obtida a partir de um *programa de verificação* designado pelos americanos *debug program*.

Agora que já temos uma ideia geral da concepção, pelo programador, de um ciclo teórico cronológico de definição de um programa, vamos estudar a maneira como devemos procurar realizá-lo de uma maneira prática: para tal iremos sucessivamente tentar:

2.1. — Dar um ideia da representação gráfica dos *símbolos de um organigrama* (flow-chart).

2.2 — Descrever a maneira como se pode realizar a *introdução de um programa por etapas graduais* num microcomputador que acaba de ser instalado e posto em serviço.

2.3 — Partindo destes conhecimentos, resolveremos um problema prático simples.

2.4 — Finalmente, para terminar, mencionaremos alguns dos *utensílios utilizados*, a título de exemplo, pela Motorola para se servir de uma maneira eficaz do seu microprocessador MC6800.

2.1. — OS SÍMBOLOS PRÁTICOS DE ESTABELECIMENTO DE UM ORGANIGRAMA (FLOW-CHART).

Estes são apresentados na figura 76.

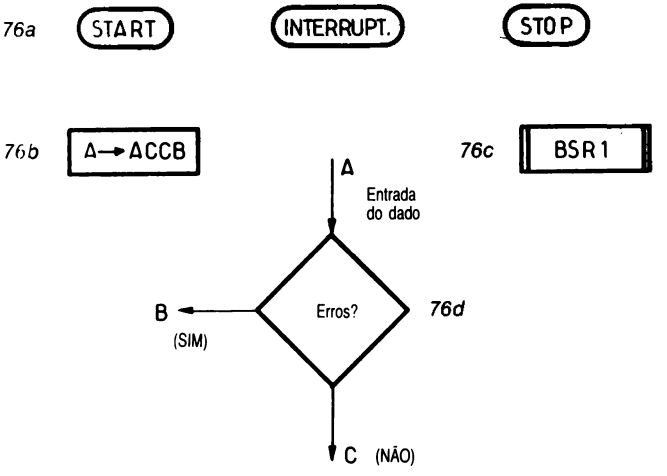


Figura 76 — *Representação dos diferentes símbolos de um organigrama.*

Em 76 a, encontraremos uma oval que representa conforme o que nela está escrito o seguinte:

Um *início*, uma *interrupção* ou um *final de programa*.

Em 76 b, o rectângulo representa, conforme o que nele está escrito, *a tarefa a desenvolver ao longo do programa* (neste caso, deve-se transferir o resultado A para o acumulador B, ou seja, deve-se carregar A no acumulador B).

Em 76 c, mostra-se a maneira de representar no programa em curso o *desenvolvimento de um subprograma exterior* (neste caso, por exemplo, ligação à subrotina 1).

Em 76 d, o losango simboliza um *teste*. Neste caso, vê-se no interior do losango a indicação de um *teste de erros*. O dado entra em A. É submetido ao teste de erros. Se há de facto um erro, o dado sai em B, onde passará por uma sequência de *correcção de erros* antes de entrar novamente em A e sair finalmente em C.

Se não existe nenhum erro, o dado sai directamente em C e o programa continua a desenvolver-se normalmente.

2.2 INTRODUÇÃO DE UM PROGRAMA POR ETAPAS GRADUAIS

Pode-se distinguir duas grandes categorias de programas diferentes: os *programas utilitários* que preparam o programa de aplicação que se necessita do computador, e os *programas de aplicações*.

a) «Os programas utilitários» comportam uma sequência de *inicialização* que compreende por sua vez duas etapas diferentes.

1.ª etapa. No painel frontal de um microcomputador toda uma série de chaves (interruptores) definem os bits de uma palavra, assim como funções dadas directamente em linguagem máquina (binário natural). Actuando assim sobre estas chaves, inicia-se a primeira etapa que consiste em abrir o diálogo com o periférico. É este o papel do *bootstrap* que executa o *programa de arranque*.

2.ª etapa. Este bootstrap é agora seguido de um *programa de carregamento* ou *carregador absoluto*. Desta vez o programa é introduzido através do teclado da tele-impressora do periférico. Prepara o programa de aplicações fixando os endereços reais na memória, verificando os formatos de endereços e de dados, detectando os erros, etc. A fim de evitar

a necessidade de o operador executar toda a etapa de inicialização, tanto o *bootstrap* como o carregador absoluto podem ser armazenados numa memória (ROM) da unidade central do microcomputador. Depois de este trabalho se encontrar realizado, podem-se iniciar os *programas de serviço e de aplicações.*

b) «Os programas de serviço e de aplicações»

Nestes programas de aplicações pode-se distinguir a edição do programa-fonte e a edição do programa-objecto.

«Programa fonte de aplicações»: efectuando este programa em linguagem mnemónica. Pode ser rigorosamente ajustado com o auxílio de *programas de serviço*, entre os quais se pode mencionar o *editor de ligações* e o *editor de textos.*

Editor de ligações (linkage editor). Tem como objectivo reunir pela ordem escolhida todos os módulos-programa separados necessários para a elaboração de um programa completo de aplicação.

O *editor de textos* corrige e monta os programas-fonte. Executa por exemplo uma ordem de anulação de um carácter errado numa fita, ou ainda, estando o programa na memória, pode suprimir uma sua instrução, deslocá-la, substituí-la por outra, etc. Serve em suma para fazer a «montagem» do programa fonte definitivo. Quando o editor de texto é parte integrante do sistema do microcomputador, é designado por *editor residente.*

Programa-objecto de aplicação. Sob pedido do programador, o assembler irá codificar a linguagem-fonte em linguagem máquina (binária natural). Seguir-se-á a etapa do programa objecto. Depois de terminada esta etapa, e após eventuais testes de erros e de correcção, se houver necessidade deles, o programa deve ser *parado* no microcomputador. A figura 77 resume este parágrafo 2.2, relativo à introdução de um programa, por etapas graduais.

2.3 — RESOLUÇÃO DE UM PROBLEMA PRÁTICO SIMPLES

Ao retomar o nosso estudo teórico do início do capítulo a fim de o aplicar de uma maneira concreta, iremos sucessivamente:

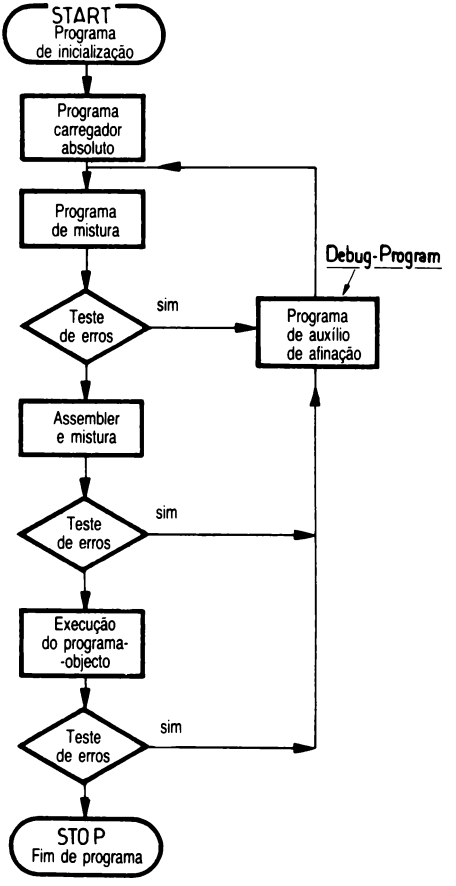


Figura 77 — *As diferentes etapas de introdução de um programa*

a) Procurar a definição de um problema de resolução simples;

b) Definir a adopção de um método que conduza ao estabelecimento de um organigrama de base (*Flow-chart*);

c) Aplicar finalmente este método à *codificação do problema.*

Exemplo de problema a resolver. Elaboramos a nossa solução com auxílio de um jogo de instruções imaginado.

a) Definição: Pretende-se resolver a operação 4 – – 3 + 2. Isto significa que é necessário subtrair 3 de 4 e somar em seguida 2 a este primeiro resultado parcial. Finalmente a informação resultante deve ser carregada na memória da unidade central.

b) Escolha do método a adoptar: Iremos escolher um modo de *endereçamento directo* (supondo que os dados 4, 3 e 2 são endereçados para uma memória RAM do sistema do microcomputador).

Para efectuar a subtracção 4 – 3 iremos servir-nos do método dito de *subtracção por complementação* que já expusémos no decurso do capítulo 5, relativo ao cálculo binário.

Iremos procurar sucessivamente:

O *complemento para 1* do número 3_{10} escrito em binário, e em seguida *incrementar* (+ 1) este complemento a fim de obter o *complemento para 2* do número 3_{10}.

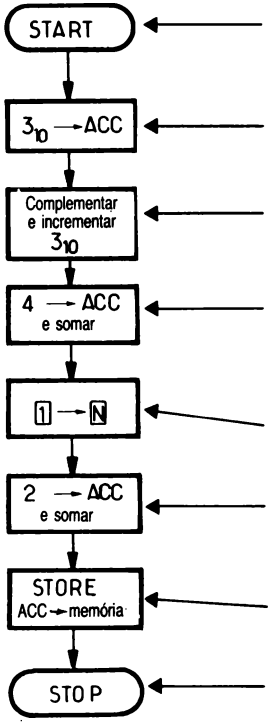
Bastar-nos-á acrescentar então a este complemento para 2 os números 4_{10} e 2_{10} para obter o resultado desejado e *carregá-lo na* memória da unidade central.

Pode-se portanto construir agora a *flow-chart* da figura 78, que será neste caso explicitada através de um texto de acompanhamento que a tornará mais clara.

Um organigrama como este, construído com caixas em sucessão, constitui um *esquema de programação em linhas rectas* (straight-line programming).

c) Codificação do problema:

Pretende-se codificar as sequências de instruções (ver quadro da figura 79) a fim de resolver a equação 4 – 3 + 2 segundo um modo de endereçamento directo.



Anúncio de início de programa (inicialização)

Carregar ($3_{10} = 011_2$) no acumulador do microprocessador.

Complemento para 1 de $3_{10} = 100_2$ (011_2) (complementar).
Complemento para 2 de $3_{10} = 100_2$ (incrementar).

Procurar ($4_{10} = 100_2$) na memória e transferir para o acumulador, a fim de somar ao complemento para 2 de 3_{10}, ou seja: $100_2 + 101_2 = 1001$ (o primeiro dígito do resultado é o sinal positivo).

Pôr 1 em N, que é o bit de sinal (Flag) do registo de estado.

Procurar ($2_{10} = 010_2$) na memória e transferir para o acumulador a fim de somar ao resultado anterior, ou seja, $010_2 + 001_2 = 011_2$.

ou $011_2 = 3_{10}$, *o que verifica a operação: $4_{10} - 3_{10} + 2_{10} = 3_{10}$, que será carregado na memória.*

Anúncio de fim de programa.

Figura 78 — *Organigrama do programa para resolução da operação (4 – 3 +2).*

MODO DE ENDEREÇAMENTO DIRECTO

Endereço	Instrução mnemónica	Tradução da instrução	Operação aritmética ou booleana
150	***CLR***	Passar a zero (clear)	ACC = 0 (ACC = acumulador)
151	***ADD***	Somar (159)	3 → ACC
152	***COM***	Complementar	ACC = $\overline{3}_{10}$
153	***INC***	Incrementar	ACC = $\overline{3}_{10}$ + 1
154	***ADD***	Somar (158)	4_{10} → ACC (+)
155	***ADD***	Somar (160)	2_{10} → ACC (+)
156	***STR***	Carregar o resultado (161)	ACC → Memória
157	***STP***	Paragem	
158		4_{10}	
159		3_{10}	
160		2_{10}	
161		Resultado $(4 - 3 + 2)_{10}$	
162			

Figura 79 — *Quadro da sequência de instruções que devem resolver o programa de operações 4 – 3 + 2, segundo um modo de endereçamento directo.*

2.4. UTENSÍLIOS DE TRABALHO

Considera-se a instalação básica apresentada no esquema simplificado da figura 80:

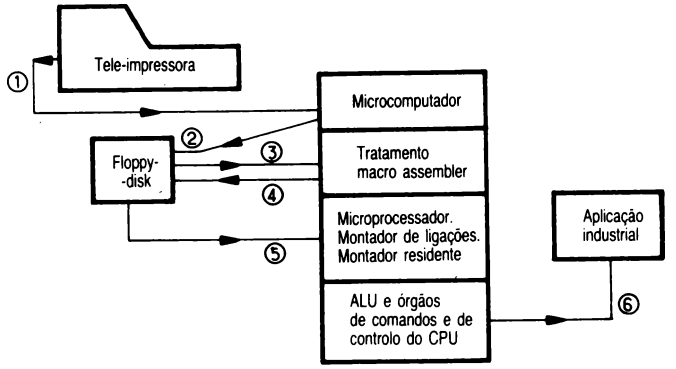


Figura 80

① — O teclado da tele-impressora vai servir ao programador para compor o programa fonte (em linguagem mnemónica).

② — O programa fonte será então colocado em memória no floppy-disk (utilizado como extensão da memória central do microcomputador).

③ — O macro-assembler do microcomputador comporá então o programa-objecto a partir do programa fonte gravado no floppy-disk.

④ — Pode-se em seguida colocar na memória este programa-objecto (em linguagem binária pura) que se encontra no floppy-disk.

⑤ — Carrega-se finalmente este programa-objecto no microcomputador assim como outros *módulos objecto*, utilizando o *editor de ligações* e ainda o *programa do editor residente,* o que assegura o comando e o controlo em ⑥ da aplicação industrial pretendida.

Depois destas considerações gerais sobre os utensílios de trabalho é bom entrarmos no domínio da prática. É o que faremos descrevendo um dos utensílios mais interessantes

postos à disposição do técnico. Trata-se do «*exorciser*» da Motorola. Este *exorciser*, utilizado juntamente com um floppy-disk (*exordisk* da Motorola) ou ainda um leitor de teletipo (*exortape* da Motorola) vai poder conceber e testar um protótipo construído com o conjunto MPU (M6800).

O *exorciser* simples **(M68SDT)** possui:

«a) Um *módulo MPU* no qual está incorporado o circuito integrado MC6800, tal como o seu sistema de relógio. Este módulo é designado **MEX6800**.

b) Um *módulo de verificação de sistemas, Debug module*, que dá ao *exorciser* a possibilidade de avaliar e proceder à correcção do programa do utilizador.

Este módulo contém entre outras duas memórias RAM (MCM6810) e 3 memórias ROM (MCM6830). Nestas últimas está inscrito um programa designado **EXBUG**, que é um *programa de simulação*.

A simulação permite calcular, além das consequências de cada instrução no microcomputador, o tempo real total de execução do programa desejado. O *simulador* é o aparelho que define uma simulação de programa.

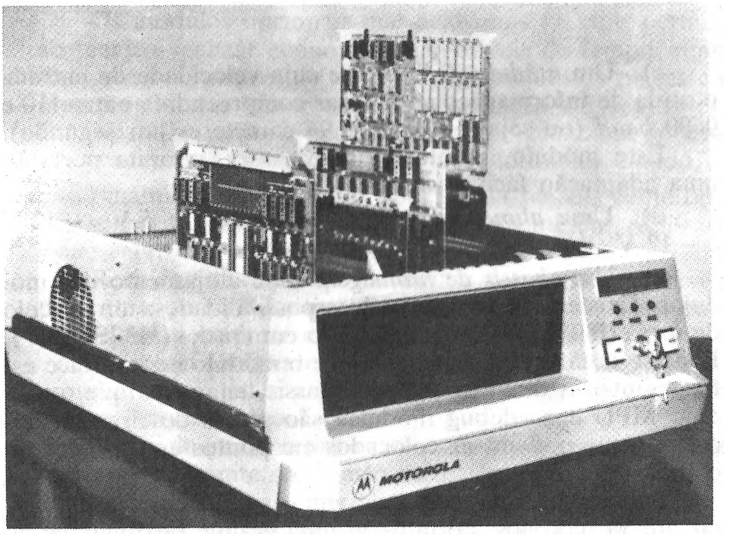


Figura 81 — O *exorciser* e placas de circuitos impressos.

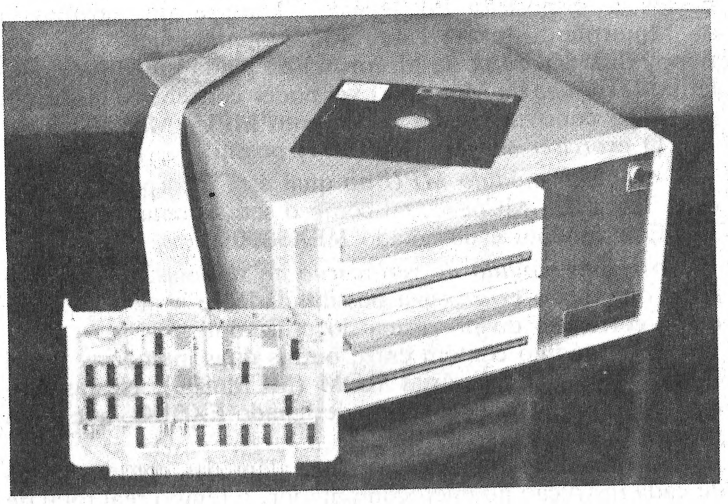


Figura 82 — Floppy Disk com placa de interface para o *exorciser*

c) Um *módulo de interface* cuja velocidade de entrada e saída de informações pode estar compreendida entre 110 e 9600 *baud* (ou seja, entre 11 e 96 caracteres por segundo).

Este módulo, montado no *exorciser*, permite portanto uma adaptação fácil a um dado terminal.

d) Uma *alimentação* com três fontes: + 5 V, + 12 V e – 12 V.

e) Um *chassis de montagem* e de alojamento dos módulos. Existem na realidade duas possibilidades: um modelo de mesa **(M68SDT-T)** e um modelo em «rack» **(M68SDT-R)**. Pode-se notar que a alimentação e o módulo de interface estão montados directamente no chassis, enquanto que o módulo MPU e o «debug module» são placas de circuitos impressos que podem ser colocados em pontos apropriados do chassis.

Este módulo de base pode, em caso de necessidade, alojar até 14 módulos, permitindo uma grande flexibilidade de emprego.

Com efeito, é possível definir com o *exorciser*, graças a esta concepção, sistemas muito variados de configurações capazes de corresponder a quaisquer necessidades dos utilizadores.

Os principais tipos de módulos que podem ser incluídos no *exorciser* são:

— O **MEX6812-1** (módulo de RAM estática de 2 K bits)

— O **MEX6815-1** (módulo memória RAM dinâmica de 8 K bits).

— O **MEX6820** (módulo de entradas-saídas I/O).

— O **MEX681C** (trata-se de um cabo de ligação entre o MEX6820 e um periférico).

— O **MEX68WW**. Trata-se de um sistema universal de arranque mecânico e eléctrico dos módulos citados no *exorciser*, permitindo ao cliente simular a sua própria concepção dos circuitos.

— O **MEX68XT** *(módulo de extensão)*. Trata-se de uma placa impressa com ligações ordenadas que permite ligar exteriormente ao *exorciser* outras configurações, o que tem como consequência dar ao aparelho uma nova extensão.

Os módulos opcionais que acabamos de citar permitem ao técnico realizar economias apreciáveis de tempo, pois é-lhe possível, através de simples substituições de placas e comutações destas obter uma representação do seu sistema. Este aparelho permite assim uma grande economia de tempo relativamente à montagem de um protótipo de estudo.

Finalmente, o *exorciser* pode ainda possuir como opção um *software residente*. Este software inclui um *editor residente* e um *assembler residente*.

Recordemos aqui que o editor serve para escrever o programa fonte definitivo, já corrigido, e que o assembler serve para o traduzir em linguagem máquina.

A opção «software residente» M6800 fornece assim ao programador a possibilidade de desenvolver inteiramente o seu próprio software.

Utilizado juntamente com o seu *firmware exbug* lógico, o *exorciser* permite ao técnico de hardware, tal como ao programador de *software*, verificar e corrigir a configuração em estudo tanto no que se refere aos circuitos que devem ser utilizados como ainda no domínio do programa.

Com o *exorciser*, é assim possível verificar completamente e de maneira definitiva o sistema mais adaptado às reais necessidades do cliente.

O *exorciser* M6800 da Motorola é portanto, em conclusão, um excelente meio, muito económico, de desenvolver as aplicações da clientela para o microprocessador MC6800.

12
ÚLTIMAS NOVIDADES

Introdução

No momento em que esta edição é feita, parece-nos necessário apresentar um complemento às noções de base já expostas em edições anteriores a fim de documentar os leitores sobre as últimas novidades em matéria de técnica aplicada aos microprocessadores. Pensámos em particular na descrição de uma parte dos futuros microprocessadores em *tecnologia VLSI*, anunciados pela Motorola: o *MC68000* encapsulado num invólucro DIL (dual in line) tendo pelo menos 64 terminais, e por outro lado da descrição de um monitor de televisão montado numa *consola de visualização de caracteres alfabéticos e numéricos*, empregues muitas vezes como elemento periférico de um microcomputador ou de um microcomputador em serviços públicos como, por exemplo, aeroportos, estações, administrações, bancos, etc., do qual ainda não falámos nas edições anteriores.

O MC68000

1. Tecnolocgia do microprocessador

Este microprocessador pôde ser fabricado pela Motorola graças a uma nova família tecnológica, a **VLSI** (Very Large Scale Integration, ou seja, escala de integração muito grande). Relativamente à família **LSI**, é agora possível difundir, numa única pastilha de silício com apenas algumas dezenas de milímetros quadrados, *dez vezes mais junções activas*. A tecnologia deste novo microprocessador tomou o nome de **HMOS** (high MOS density, ou seja, MOS de grande densidade). Entre o H-MOS e o N-MOS podem-se estabelecer as seguintes comparações no que se refere às características:

a) A densidade de circuito, na pastilha de silício, é mais de duas vezes maior. No MC68000 uma célula elementar de alojamento de um bit ocupa uma superfície de *1852 microns quadrados*, enquanto que no MC6800 era de *4128 microns quadrados*.

b) O produto *duração de propagação* × *potência consumida* é quatro vezes melhor em tecnologia H-MOS:

Em **N-MOS** = *4 picojoules*;

Em **H-MOS** = *1 picojoule*.

2. Arquitectura interna e funcionamento do microprocessador

O microprocessador MC68000 pode explorar seis tipos de dados:

a) *O binário natural*

b) *O decimal codificado em palavras de 4 bits* (sistema BCD — ver quadro no capítulo V)

c) *Os caracteres ASCII* (ver final do capítulo X)

d) *As palavras* (bytes) *de 8 bits*

e) *As palavras de 16 bits* **(WORD)**

f) *As palavras de 32 bits* (trabalhando em extensão dupla — Long word).

O microprocessador MC68000 comporta um jogo de *61 instruções* distintas escritas em linguagem numérica de 3, 4 ou até 5 letras. Entre estas instruções, podem-se citar as seguintes resumidamente:

a) *Funções aritméticas com ou sem sinal*, incluindo em particular as quatro operações fundamentais (soma, subtracção, multiplicação e divisão), incluindo as instruções auxiliares necessárias para as determinar, como a complementação para 1, complementação para 2, deslocamento num registo, etc.

b) *Funções lógicas* tais como AND, OR inclusivo, OR exclusivo.

c) *Ligações, saltos para subrotinas, inicialização, paragem, carregamento dos diferentes registos do microprocessador, verificação dos indicadores do seu registo de estados, etc.*

d) Novas instruções, em particular **TRAP** e **TRAPV**. Devem ser utilizadas pelo programador para aplicações destinadas à detecção de erros ou ainda de correcção de rotinas.

Estas 61 instruções são repartidas em 5 modos principais de endereçamento (ver anteriormente).

Trata-se dos modos:

Endereçamento directo.
Endereçamento indirecto.
Endereçamento imediato.
Endereçamento absoluto.
Endereçamento relativo.

Para uma maior flexibilidade de emprego no MC68000, cada um destes cinco modos principais de endereçamento inclui particularidades distintas, o que nos leva a poder falar de facto em *14 sub-modos de endereçamento deste microprocessador.*

Deve-se notar finalmente que um *«trace mode»* assinalado por bit de teste no registo de estados dá ao programador a possibilidade de verificar o seu programa mesmo quando o está a elaborar, podendo fazê-lo instrução a instrução.

Vamos falar agora da arquitectura interna propriamente dita do microprocessador MC68000 (ver figuras 83 e 84).

O MC68000, como se mostra na figura 83, compreende *16 registos de 32 bits*, além do *contador ordinal de 24 bits* e do *registo de estados de 16 bits.*.

— **D0** *a* **D7** são os oito primeiros registos destinados ao armazenamento e à leitura dos dados sob a forma *de palavras de 8 bits* (bytes).

— Ou ainda *de palavras de 16 bits* (words).

— Ou ainda *de palavras de 32 bits* (long words).

— **A0** *a* **A7** são os oito segundos registos destinados ao endereçamento dos dados, ou ainda ao registo *stack pointer* (ver anteriormente). Podem ser usados com *palavras de 16 bits* ou *de 32 bits.*

Por outro lado, os 24 bits do contador ordinal fornecem um endereçamento na memória com uma capacidade de *16 megabytes* 16.777.216 palavras de 8 bits).

Examinemos finalmente (figura 82) o registo de estados. Inclui:

a) Oito níveis possíveis de interrupções (ou seja 2^3 bits), graças ao jogo dos seus *3 indicadores* (flags) I_0 I_1 I_2.

b) Os códigos de condições já conhecidos:
Overflow (V)
Zero (Z)
Negativo (N)
Carry (C)
Extend (X)

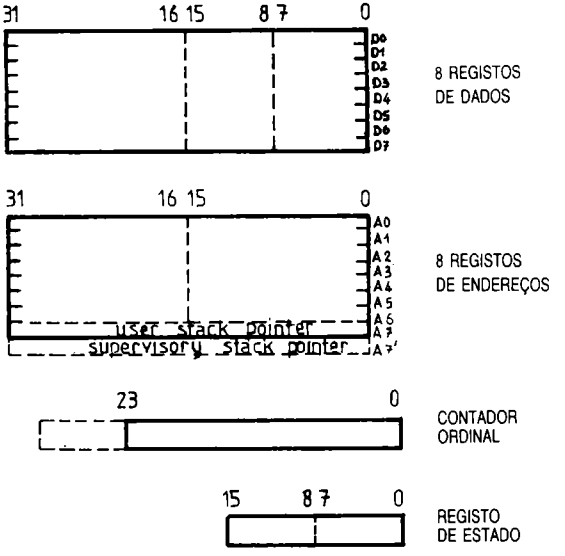


Figura 83 — Registos do MC68000

O indicador **X** é um bit de teste complementar do bit carry C, encarregado particularmente numa inscrição LINK de ligar o MSB e o LSB (ver a figura 44).

c) O novo bit de teste *Trace* **(T)**, cujo significado acabamos de ver.

d) O estado de controlo: *Supervisory* **(S)**.

O **Trace Mode** só pode ser pedido pelo programador quando o microprocessador se encontra no estado de controlo *(supervisory* **S**) e não no estado habitual de funcionamento designado **USER**.

e) Finalmente, a figura 84 mostra *posições disponíveis de indicadores* (*5, 6, 7, 11, 12* e *14*) no registo de estados, com vista a aumentar as possibilidades de exploração do MC68000 aquando de futuras extensões previstas para a família M68000.

O MC68000 é compatível com os periféricos do microprocessador MC6800, em particular com aqueles que já examinámos, ou seja, o **MC6821** (que é a nova versão do **PIA — MC6820**) e o **MC6850** (ver atrás).

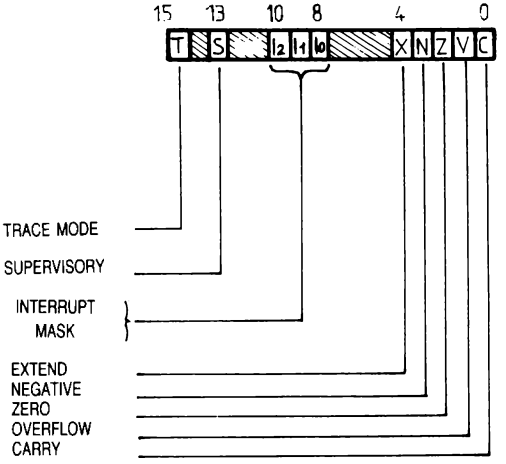


Figura 84 — Registo de estados.

Acrescentemos igualmente os **MC6843** e **MC6849**, que são novos circuitos de controlo dos *floppy-disk* (ver anteriormente) e o **MC6847**, que é um circuito gerador de caracteres video para apresentação num visor de televisão e de que falaremos mais detalhadamente na parte deste capítulo consagrada às consolas de visualização (monitores de video-frequência).

3. Distribuição dos terminais de saída do microprocessador MC68000 (ver a figura 85).

A leitura do esquema da figura 85 permite-nos distinguir:

— Um *Bus de endereços* unilateral de 23 linhas paralelas $\mathbf{A_1}$ a $\mathbf{A_{23}}$ e um *Bus de dados* de 16 linhas paralelas bilaterais $\mathbf{D_0}$ a $\mathbf{D_{15}}$. Dado que o bus de dados se encontra separado do bus de endereçamento, não se torna necessário definir um circuito de multiplexagem entre os endereços e os dados.

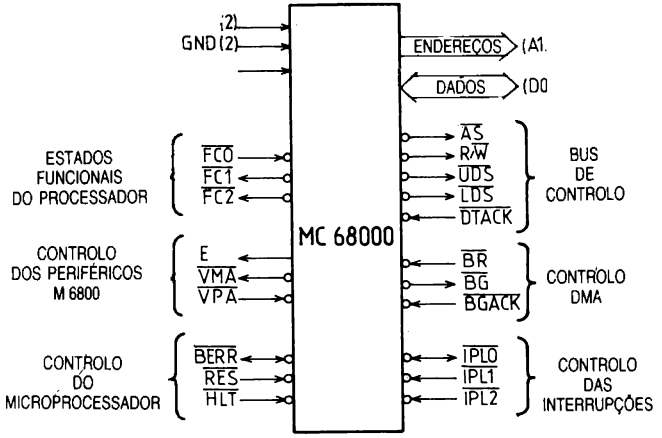


Figura 85

— 2 ligações reservadas à aplicação da tensão positiva de alimentação (+ V_{CC} = *5 volts*) e duas outras à tensão de referência (**GND** = *massa*). O valor da tensão de alimentação mostra-nos a *compatibilidade* com os componentes periféricos da família TTL.

— Um sinal de relógio (**CLK** = *clock*) de *8 MHz*. Pode ser obtido a partir de um oscilador de quartzo.

— Um bus de controlo de 5 linhas — 4 linhas são orientadas do microprocessador para as memórias ou interfaces. São:

a) A linha $\overline{\mathbf{AS}}$ (*Adress Strobe*), que permite afirmar que o endereço no bus de endereços se encontra validado.

b) A linha **R/W** (*read-write*), que permite distinguir um ciclo de leitura ou um ciclo de escrita de uma memória.

c) As duas linhas **UDS** (*Upper data strobe*) e **LDS** (*Low Data strobe*), que permitem por exemplo distinguir numa palavra de 16 bits o byte de ordem inferior **LDS** e o de ordem superior UDS.

A 5.ª linha, **DTACK** (*data transfer knowledge*) dá por exemplo ordem de transferir o dado para o microprocessador, num ciclo de leitura da memória.

— Três linhas de controlo das interrupções **IPL0 — — IPL1 — IPL2** que, num sistema binário, permitem com o auxílio dos bits 1 e 0 a possibilidade de obter 2^3, ou seja, 8, níveis diferentes de prioridade de pedidos de interrupções em curso.

— Três linhas que numa mesma ordem de ideias permitem através de 8 possíveis combinações binárias definir o estado funcional do microprocessador. A fim de permitir uma melhor compreensão deste assunto, apresenta-se o quadro da figura 86, onde são dadas as 8 determinações funcionais do microprocessador em função da composição binária de **FC2 — FC1 —** e **FC0**.

FC2	FC1	FC0	Código de função	Identificação
0	0	0	Other	①
0	0	1	User program	②
0	1	0	User data	③
0	1	1	Halt	④
1	0	0	Test mode	⑤
1	0	1	Supervisor program	⑥
1	1	0	Supervisor data	⑦
1	1	1	Interrupt knowledge	⑧

Figura 86 — Quadro

Comentários a este quadro. O microprocessador encontra-se no estado **(HALT)** quando está alojado:

a) O bit *0* no registo **$\overline{FC2}$**
b) O bit *1* no registo **$\overline{FC1}$**
c) O bit *1* no registo **$\overline{FC0}$** (ver ④)

No caso dos outros estados, determinados do mesmo modo, consultar o quadro. São os estados:

Estados de controlo ou *estado habitual de funcionamento* quer na realização normal de um programa quer quando se transfere um dado — ② ③ ⑥ ⑦.

O programador pode ainda tentar saber num dado momento o modo de funcionamento do microprocessador (*Test Mode*) ⑤.

É ainda possível conhecer um nível de interrupção ⑧.

Finalmente, quando os três registos se encontram em zero o microprocessador está num estado diferente **(OTHER)** ① dos que acabam de ser descritos.

— 3 linhas de *controlo em* **DMA** asseguram o pedido de um periférico de *funcionamento em acesso directo à memória,* sem necessidade de passar pelo microprocessador. Estas três linhas são utilizadas pela seguinte ordem:

$\overline{BR}$ (*Bus request*) é o pedido do periférico ao microprocessador para um acesso directo à memória.

$\overline{BG}$ (*Bus grant*) é a resposta afirmativa que o microprocessador pode dar ao periférico.

BGACK é a atribuição do Bus que permite a troca de dados entre o periférico e a memória.

— 3 linhas de controlo do microprocessador definem:

a) O comando de paragem do microprocessador (**$\overline{HLT}$** = *Halt*).

b) O período de inicialização ou passagem a zero do microprocessador (**RES** = *Restart*).

c) Determinação de um erro num bus (**$\overline{BERR}$** = *bus error*).

Finalmente, três linhas de controlo do microprocessador com periféricos da família M6800. São:

a) A indicação **E** = *enable* pelo microprocessador de um bus disponível.

b) A validação **$\overline{VMA}$** = *Valid memory adress* no mo-

mento da transferências de um endereço do microprocessador para a memória da família M6800.

c) A validação **VPA** = *Valid Peripheral adress* no momento da transferência de um endereço do periférico (componente da família MC800) para o microprocessador.

4. Funcionamento de um ciclo de leitura

Ver quadro da figura 87.

5. Funcionamento de um ciclo de escrita

Ver quadro da figura 88.

Estes dois quadros das figuras 87 e 88 dão-nos um exemplo bastante simples mas no entanto significativo do funcionamento lógico entre o microprocessador MC68000 e as suas memórias ou circuitos de interface associados a um ciclo de leitura ou de escrita.

Figura 87 — *Quadro*

Microprocessador	*Memória ou interface*
«1) Endereço de memória» 1-1 Colocar o endereço em A_1 a A_{23} 1-2 Colocar R/W em Read 1-3 Validar o bus de endereço AS 1-4 Validar os dados UDS e LDS	«2) Entrada dos lados» 2-1 Descodificar o endereço 2-2 Colocar os dados em D_0 a D_{15} 2-3 Dar ordem de transferência dos dados (DTACK)
«3) Aquisição dos dados» 3-1 Carregar os dados no registo interno adequado 3-2 Apagar o sinal de validação UDS e LDS 3-3 Apagar o sinal validação AS	«4) Final do ciclo de leitura» 4-1 Suprimir os dados em D_0 a D_{15} 4-2 Suprimir a ordem de transferência dos lados (DTACK)
«5) Pronto para início do ciclo seguinte.»	

Figura 88 — *Quadro*

«1) Aquisição de dados» 1-1 Colocar R/W em Write 1-2 Colocar os dados saídos do ALU em D_0 a D_{15} 1-3 Validar UDS e LDS	«2) Aceitação dos dados» 2-1 Carregar os dados D_0 a D_{15} definidos por UDS e LDS. 2-2 Validar a transferência de dados dos DTACK
«3) Terminar a transferência para a saída 3-1 Apagar a validação UDS e LDS 3-2 Suprimir os dados em D_0 a D_{15} 3-3 Colocar R/W em Read	«4) Final do ciclo de escrita» 4-1 Suprimir a validação DTACK
«5) Pronto para arranque do ciclo seguinte»	

Os monitores de televisão M68MDM1 e M68MDM9

1. Generalidades

Estes monitores de videofrequências são inteiramente transistorizados e servem para apresentar, no respectivo «écran», caracteres alfabéticos e numéricos. Os modelos M68MDM funcionam com uma entrada «*Video-composta*» ou com entradas separadas para:

— Um sinal de *video-frequência*
— Um sinal de *sincronização de campo*
— Um sinal de *sincronização de linhas*

Neste último caso os níveis dos sinais de sincronização são compatíveis com os níveis da lógica **TTL** (Ver figura 90).

Os tubos de raios catódicos empregues (**CRT** = Cathode ray tube) são de cinco polegadas **(M68MDM1)** ou de 9 polegadas **(M68MDM9).** Esta numeração em polegadas indica o comprimento da diagonal dos écrans. Recordemos que uma polegada á igual a 2,54 cm.

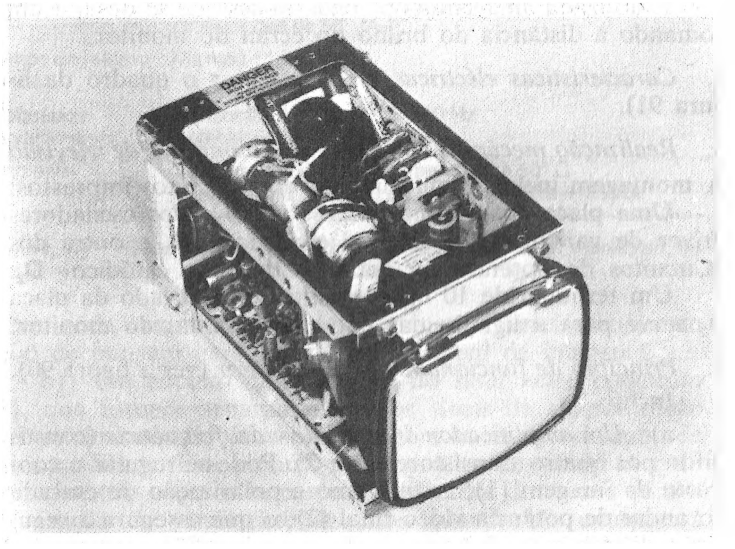


Figura 89 — O monitor de televisão M68MDM1

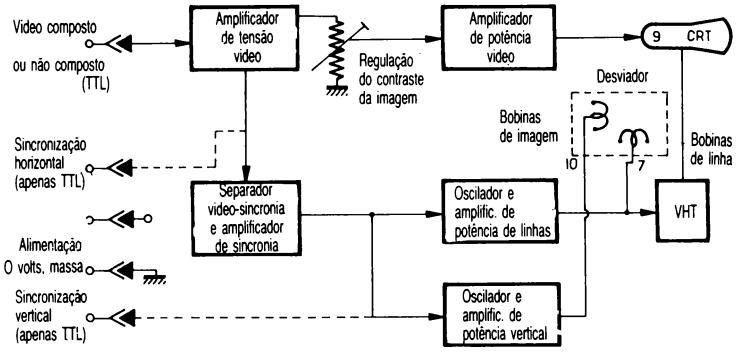


Figura 90 — Esquema sinóptico dos monitores de televisão.

Os tubos são *auto-protegidos,* ou seja, anti-implosão e do tipo de desvio magnético.

O chassis do monitor não inclui a alimentação.

É portanto necessária uma alimentação contínua de 12 volts/650 mA exterior ao chassis.

É utilizada uma saída opcional no caso de se desejar um comando à distância do brilho do écran do monitor.

2. *Características eléctricas essenciais* (ver o quadro da figura 91).

3. *Realização mecânica do chassis dos monitores de televisão*

A montagem inclui essencialmente dois circuitos impressos:

Uma placa dos «sinais eficazes video e dos osciladores driver de varrimento horizontal e vertical» $\mathbf{S_1}$ e outra dos «Circuitos de potência de deflexão do tubo catódico» $\mathbf{D_2}$.

Um terminal de 10 ligações situado num lado da placa $\mathbf{D_2}$ serve para a ligação das entradas e saídas do monitor.

4. *Princípio de funcionamento do monitor* (ver a figura 90).

Inclui:

a) *Um amplificador do sinal de videofrequência* (constituído por quatro transístores (1 e 2). Pode-se regular o contraste da imagem (1), assim como a polarização de entrada no andar de potência video final (2), o que assegura a regulação da corrente de repouso deste andar. São asseguradas protecções ao nível da limitação da corrente do feixe de elec-

Tubo de imagem **CRT**		*Ângulo de desvio «diagonal 55°»* *Fósforo P4*
Alimentação em **CC**		**12 V – 650 mA**
Sinais de entrada	Caso do sinal video composto	Entre 0,5 e 2,5 volts crista a crista Sincronização negativa Z_e = 75 ohms ou 12 kohms
	Caso das sincronizações horizontal e vertical Níveis TTL	2,5 V a 5 V de crista Sincronização positiva Z_e = 75 a 250 ohms na ligação video Z_e > 2 kohms para as sincronizações horizontal e vertical
Definição imagem	650 pontos no centro da imagem 500 pontos nos lados do «écran»	
Banda passante video	10 Hz a 12 MHz a – 3 dB	
Linearidade	Inferior a 2%, normas EIA	
VHT	9,5kvolts com uma corrente de feixe de electrões de 50 A	
Tempo de retorno de linha	Máximo 11 μs	
Frequência de varrimento	Horizontal: 15 750 Hz ± 500 Hz Vertical: 50 Hz a 60 Hz.	
Ambiente	Temperatura de funcionamento: 0° a 50° C Temperatura de armazenamento: – 40° a + 65° C.	

Figura 91 — *Quadro das especificações eléctricas dos monitores de televisão.*

trões e da realimentação dos «arcings» intereléctrodos do tubo de imagem para os andares do sinal de imagem.

b) *Um circuito de separação do sinal video composto* (3), que fornece separadamente os sinais de sincronização dos varrimentos de linha e de imagem ou campo do tubo de raios catódicos. Este circuito pode igualmente servir, no caso do video não composto, como amplificador dos sinais de sincronização do varrimento de linha quando são externos à placa de ligações e compatíveis com a tecnologia TTL (ver quadro da figura 91, níveis de compatibilidade).

c) *Um circuito oscilador mantido na frequência de varrimento de linha* (4) graças à tensão de detecção de erro de fase entre os «tops» (impulsos) de sincronização e os impulsos de retorno de linha.

d) *Um circuito de potência de varrimento de linha* (5), encarregue de alimentar as bobinas de deflexão de linha (7), os eléctrodos de aceleração e de concentração do tubo de raios catódicos (**CRT**) (9) e de gerar a tensão extra alta (VHT) (8). Deve-se notar ainda a possibilidade de regular neste circuito a largura do varrimento de linha.

e) *Um oscilador vertical e o seu amplificador de potência* (6) encarregue de alimentar as bobinas de reflexão vertical (10). Deve-se notar neste circuito a possibilidade de regular correctamente a linearidade e a amplitude do varrimento de imagem vertical.

O gerador de caracteres video Motorola MC6847

O gerador de caracteres Motorola (**VDG** — *Video Display Generator*) **MC6847** é um circuito integrado que serve como *interface* entre um micro-sistema que inclui por um lado os microprocessadores MC68000 e MC6800 associados às suas diversas memórias compatíveis, e por outro um monitor de videofrequências montado numa consola de visualização periférica. *O* **VDG** lê numa memória do micro-sistema um dado que traduzirá em sinal video composto. Este sinal é injectado na entrada de um monitor video que o *escreverá no écran do seu tubo de raios catódicos sob a forma de caracteres alfabéticos e numéricos* (ver a figura 92).

O MC6847 é um circuito integrado **N-MOS** encapsulado num invólucro **DIL** *de 40 terminais*, semelhante ao invólucro representado na figura 71.

Cada página de texto emitida pode incluir *16 linhas de 32 caracteres alfanuméricos por linha.*

Por encomenda especial pode-se obter uma ROM de instruções interna ao gerador de caracteres com ligações programáveis.

O MC6847 compreende dois modelos. Um possui um varrimento de linha não entrelaçado e o outro, o MC6847Y, com um varrimento de linha entrelaçado.

Este circuito integrado produz igualmente sinais de diferença de cor R-Y e B-Y.

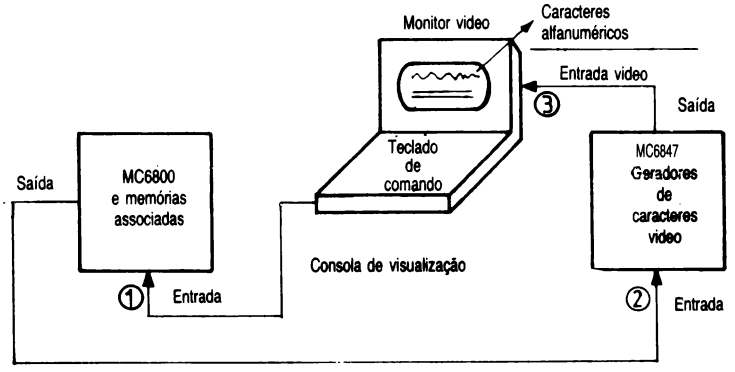


Figura 92

É compatível com o circuito integrado modulador Motorola MC1372. O conjunto MC6847 e MC1372 pode então ser ligado à tomada de entrada de antena de um receptor de televisão a preto e branco ou a cores, apresentando no seu écran sinais *alfanuméricos* programados a partir de um micro-sistema M6800. É a partir de um tal conjunto que se podem realizar os novos *jogos* de *televisão*.

O modulador MC1372 recebe:

1) O sinal de luminância video.

2) Os sinais de diferença de cor R-Y e B-Y na saída do MC6847.

Por outro lado, no MC1372 figuram

1) Um circuito que produz uma *portadora de alta frequência* situada na banda 1 de televisão (canais 3 ou 4) e um circuito em que esta portadora é modulada pelo sinal de luminância.

2) Um circuito que produz uma *subportadora de crominância* pilotada por um quartzo exterior ao MC1372 e um circuito em que esta subportadora é modulada pelos sinais de diferença de cor R-Y e B-Y.

3) Um *circuito misturador* que junta a portadora de alta frequência e a subportadora cromática depois de ambas serem moduladas.

A saída do circuito misturador do MC1372 pode em seguida ser ligada à *tomada de antena de um receptor clássico de televisão a cores* ou *a preto e branco* sintonizado para a banda 1 de televisão (canais 3 ou 4).

Na figura 93 representa-se o diagrama de blocos da montagem de conjunto.

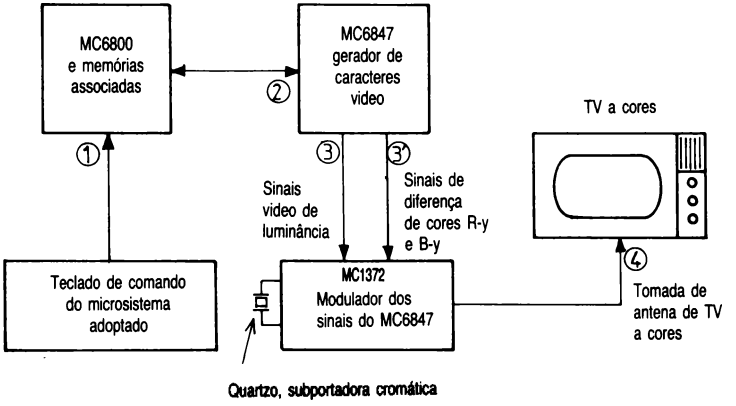


Figura 93

CONCLUSÃO

Terminamos assim este capítulo, que por sua vez fecha este livro de iniciação à microinformática.

O autor tem consciência de ter escrito um simples livro de iniciação nesta matéria. Gostaria de poder ter interessado o leitor pela descoberta deste mundo novo, a qual sem dúvida vos permitirá compreender melhor os progressos desta nova geração da electrónica nos domínios já existentes: informática, telecomunicações, radar, transmissões HF, automatismos industriais, televisão, transmissão radiofónica, etc.

O autor espera que o livro tenha criado nos seus leitores a curiosidade e o interesse necessários para estudar este assunto em profundidade, transformando-os em verdadeiros especialistas neste novo domínio prometedor de um vasto campo de trabalho.

Se tal aconteceu, o autor pensa ter cumprido com êxito a sua ambição: atrair, com os seus modestos meios, a jovem geração de técnicos para uma nova fonte de riqueza intelectual e científica, que abre novas vias para a compreensão entre os homens para além das fronteiras geográficas e linguísticas da sociedade humana.

O autor

ÍNDICE

Este livro acabou de se imprimir
em 1983
para a
EDITORIAL PRESENÇA, LDA.
na
Empresa Gráfica Feirense, L.da
Vila da Feira

A alma do microcomputador é o microprocessador. Recebendo os dados que lhe são fornecidos, o microprocessador apresenta-os, resolvendo-os de uma forma sequencial lógica, sob a forma de resultado. No intuito de melhor compreender o mundo da microinformática, o autor apresenta nesta obra as bases para o conhecimento desta peça essencial do computador, fornecendo esquemas e quadros elucidativos, facilmente acessíveis a todos os interessados nesta nova técnica, definitivamente implantada no mundo.

EDITORIAL PRESENÇA / MARTINS FONTES