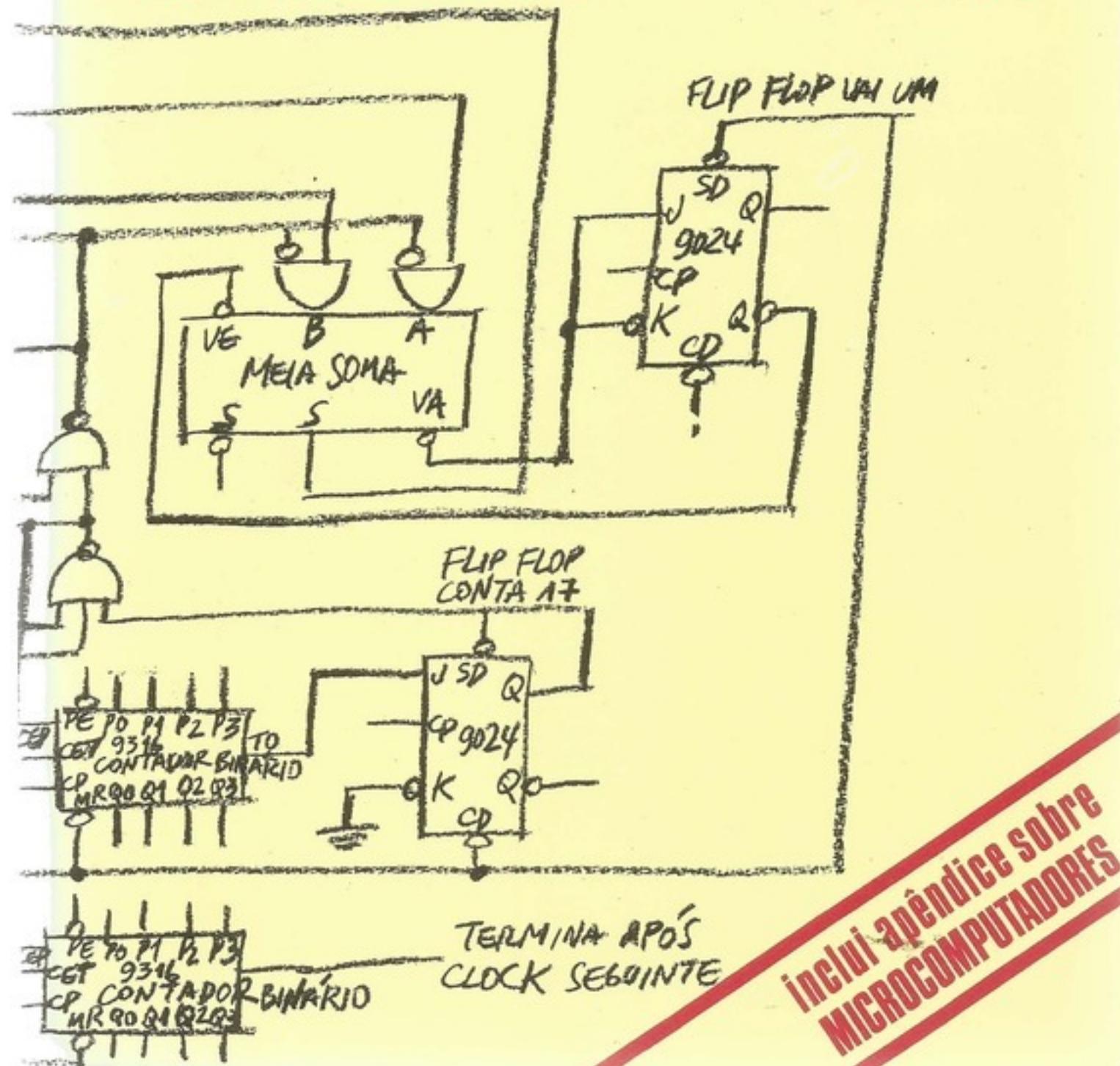


Glen George Langdon Jr.
Edson Fregni

PROJETO DE COMPUTADORES DIGITAIS

2. edição



PROJETO DE COMPUTADORES
DIGITAIS

FICHA CATALOGRÁFICA

L263p Langdon, Glen George, 1937—
Projeto de computadores digitais [por] Glen George
Langdon Júnior [e] Edson Fregni. São Paulo, Edgard
Blücher, 1974. 1977 2.^a edição.
ilust.
Bibliografia.
1. Computadores eletrônicos digitais — Projeto e cons-
trução I. Fregni, Edson, 1947— II. Título.

74-0646

17. CDD-621.381958
18. -621.3819582

Índices para catálogo sistemático:

1. Computadores digitais : Projeto : Engenharia eletrônica 621.381958 (17.)
621.3819582 (18.)
2. Projeto de computadores digitais : Engenharia eletrônica 621.381958 (17.)
621.3819582 (18.)

GLEN GEORGE LANGDON JUNIOR

*Professor visitante na Escola Politécnica da Universidade de
São Paulo. Pesquisador da IBM em San Jose, California, EUA*

EDSON FREGNI

*Mestre em Engenharia Elétrica; Professor da
Escola Politécnica da Universidade de São Paulo
Diretor técnico de SCOPUS TECNOLOGIA, São Paulo*

PROJETO DE COMPUTADORES DIGITAIS

2.ª edição



EDITORAS EDGARD BLÜCHER LTDA.

CONT

©1974 Editora Edgard Blücher Ltda.

1.^a Reimpressão da 2.^a edição 1977

2.^a Reimpressão da 2.^a edição 1979

*É proibida a reprodução total ou parcial
por quaisquer meios
sem autorização escrita da editora*

EDITORIA EDGARD BLÜCHER LTDA.

01000 CAIXA POSTAL 5450
END. TELEGRAFICO: BLUCHERLIVRO
SÃO PAULO — SP — BRASIL

Impresso no Brasil Printed in Brazil

CONTEÚDO

<i>Prefácio</i>	XV
Prefácio dos autores	XVI
1 <i>Introdução e conhecimentos básicos</i>	1
1.1 Organização de sistemas de computação	1
1.2 Álgebra booleana	5
a. Postulados e definições	5
b. Blocos lógicos	6
c. Teorema de álgebra booleana	7
1.3 Circuitos digitais	9
a. Introdução	9
b. Descrição de um circuito combinatório	10
c. Análise de circuitos combinatórios	12
d. Síntese de circuitos combinatórios	13
e. Determinação da soma mínima	15
f. Circuitos seqüenciais	19
g. Descrição de um circuito seqüencial	20
1.4 Tecnologia: transistores e diodos	23
a. Circuitos lógicos com diodos	23
b. O transistor como chave	24
c. Transistores em circuitos lógicos	26
1.5 Tecnologia: circuitos integrados digitais	28
a. Introdução	28
b. Família <i>RTL</i> (<i>resistor transistor logic</i>)	28
c. Família <i>DTL</i> (<i>diode transistor logic</i>)	29
d. Família <i>TTL</i> (<i>transistor transistor logic</i>)	29
e. Família <i>HTL</i> (<i>high threshold logic</i>)	30
f. Família <i>ECL</i> (<i>emitter coupled logic</i>)	30
g. Circuitos <i>MOS-FET</i>	31
<i>Exercícios</i>	33
<i>Bibliografia</i>	34
2 <i>Códigos, números e aritmética</i>	36
2.1 Códigos	36
2.2 Números	37
a. Sistemas	37
b. Números inteiros e fracionários	39

3.6	Fluxo de sinal
a.	Transistor
b.	Transistor
c.	Transistor
d.	Fluxo de sinal
Exercícios	
3.7	O circuito de saída
a.	Energia
b.	A tensão
Exercícios	
3.8	Circuitos de saída
a.	Oscilador
b.	"Limpador"
c.	Entrada
d.	Driver
e.	Fornecedores
f.	Outros
g.	Circuito de saída
3.9	Sumário
Bibliografia	
4	Memória e armazenamento
4.1	Introdução
4.2	O hardware
4.3	Memória programável
4.4	Memória de dados
4.5	Memória massiva
a.	Registers
b.	Memória
4.6	Sistemas de memória
4.7	O sistema de memória
a.	Motivação
b.	O sistema
c.	Localização
d.	A performance
e.	A visibilidade
f.	A administração
g.	Assumptions
4.8	Armazenamento
a.	Componentes
b.	A técnica
c.	O sistema
d.	Exemplos
4.9	Memória virtual
a.	Motivação
b.	Transição
c.	O mecanismo
d.	O mecanismo
e.	Tabelas de associação
f.	Tabelas de associação
g.	Implementação
h.	Experiência

c.	Números negativos	39
d.	Números binários em complemento de um	39
e.	Números em complemento de dois	40
f.	Ponto fixo e ponto flutuante	41
2.3	Aritmética binária	42
a.	Números positivos	42
b.	A técnica da complementação	42
c.	Aritmética usando complemento de um do subtraendo	43
d.	Aritmética usando complemento de dois	44
e.	Aritmética com números em sinal e amplitude	47
f.	Variações	48
2.4	Aritmética decimal	50
a.	Representação dos negativos	51
b.	Exemplos de aritmética decimal	51
c.	Aritmética em BCD usando representação sinal e amplitude	51
2.5	Conversão entre números binários e decimais	53
a.	Conversão binária a decimal	55
b.	Conversão decimal a binária	56
2.6	Sumário	58
<i>Exercícios</i>		59
<i>Bibliografia</i>		59
		60
3	Elementos do projeto lógico	61
3.1	Introdução	61
3.2	Circuitos combinatórios	61
a.	Decodificadores	61
b.	Circuitos de paridade	62
c.	Circuitos de prioridade	63
Exercícios		64
3.3	Flip-flop	64
a.	Flip-flop RS	66
b.	Flip-flop tipo PH	66
c.	Flip-flop tipo D, sensível à borda	67
d.	Flip-flop tipo JK	68
e.	Flip-flop master-slave	71
f.	Outros tipos de flip-flops	73
Exercícios		74
3.4	Registradores	75
a.	Registrador deslocador	78
b.	Registrador-contador	79
c.	Contadores binários	85
d.	Contadores não-binários	85
Exercícios		90
3.5	Unidades aritméticas	93
a.	Somadores	98
b.	Substratores	98
c.	Complementação	107
d.	Funções lógicas	109
e.	Unidades aritméticas	110
f.	Exemplo detalhado de uma unidade aritmética	111
Exercícios		114

39	3.6 Fluxo de sinais	115
39	a. Transferência de dados para registradores	115
40	b. Transferência em paralelo	118
41	c. Transferências por vias	121
42	d. Fluxo de dados	122
42	Exercícios	123
43	3.7 O <i>timing</i> e o ciclo da máquina	124
44	a. Exemplo	124
47	b. A determinação do ciclo da máquina	125
48	Exercício	126
50	3.8 Circuitos especiais	126
51	a. Osciladores	127
51	b. "Limpa ao ligar"	129
51	c. Entradas do sistema	129
53	d. <i>Driver</i>	129
55	e. Fontes de alimentação	130
56	f. Outros circuitos	131
58	g. Circuitos de interface com contatos mecânicos	132
59	3.9 Sumário	136
59	<i>Bibliografia</i>	137
60	4 Memória e armazenamento	138
61	4.1 Introdução	138
61	4.2 O <i>bandwidth</i> e suas implicações	138
61	4.3 Memória primária	140
61	4.4 Memória de núcleo de ferrite	141
62	4.5 Memória monolítica	147
63	a. Bipolar	148
64	b. Memória <i>MOS</i>	149
64	4.6 Sistemas de memória primária	152
66	4.7 O sistema <i>cache</i>	154
66	a. Motivação	154
67	b. O sistema <i>cache-backing</i>	154
68	c. Localidade de referência	154
71	d. A <i>performance</i> do esquema	155
73	e. A viabilidade do <i>cache</i>	156
74	f. A administração do <i>cache</i>	157
75	g. Assuntos relacionados ao <i>cache</i>	158
78	4.8 Armazenamento secundário	159
79	a. Composição de arquivos	159
85	b. A técnica de <i>blocking</i>	160
85	c. O sistema <i>IOCS</i>	160
90	d. Exemplo	161
93	4.9 Memória virtual	162
98	a. Motivação	162
98	b. Transcodificação dinâmica de endereço	162
107	c. O mecanismo de transcodificação	163
109	d. O tamanho dos blocos ou páginas	164
110	e. Tabelas de paginação de um nível	164
110	f. Tabelas de paginação de dois níveis	165
111	g. Implementação de relocação dinâmica no <i>S/360</i> modelo 67	167
114	h. Experiência com memória virtual	167

4.10	Sumário	167	
	<i>Exercícios</i>	168	b. Relações
	<i>Bibliografia</i>	168	c. Formatos
5	<i>Exemplo de um minicomputador</i>	170	d. O sistema
5.1	Introdução	170	6.3 Especificações
5.2	Esboço da arquitetura	170	a. Imediatas
5.3	Instruções principais	172	b. Diretas
	a. Instruções longas	172	c. Indiretas
	a.1. Grupo principal	172	d. Indiretas
	a.2. Grupo de entrada/saída	173	e. Relacionadas
	a.3. Grupo das imediatas	173	f. Altamente
	b. Instruções curtas	175	g. Combinatórias
	b.1. Microgrupo 1 (<i>MICG1</i>)	176	6.4 A posição
	b.2. Microgrupo 2a (<i>MICG2a</i>)	176	a. Instruções
	b.3. Microgrupo 2b (<i>MICG2b</i>)	177	b. Instruções
5.4	Fluxo de dados	178	c. Descrição
	a. Análise dos circuitos	179	d. Descrição
	a.1. Memória	181	e. Superestrutura
	a.2. Registradores	181	f. Outras
	a.3. Unidade aritmética	185	6.5 O PDP-8
	a.4. Portas de seleção	185	a. Introdução
	a.5. Ciclo da máquina	188	b. As instruções
	b. Implementação dos circuitos	189	c. Estudando
	b.1. <i>FR1-1 e FR2-2</i>	190	d. Comunicação
	b.2. <i>FS1-3 e FS2-4</i>	190	6.6 Arquitetura
	b.3. <i>FAC-5</i>	190	a. Mecânica
	b.4. <i>FM1-6</i>	191	b. As dimensões
	b.5. <i>FCM-7</i>	191	c. Estrutura
	b.6. <i>FIN-8</i>	191	d. Componentes
5.5	Fases e microoperações	192	e. Software
	a. Fases	193	6.7 A arquitetura
	b. Microoperações	193	a. Processador
	c. Exemplo	194	b. Formato de
5.6	Unidade de controle	194	c. Formato
	a. Considerações gerais	195	6.8 Funções das
	b. Exemplo	195	a. Introdução
	c. Decodificador	199	b. Processador
	d. Controle de estado	199	c. Relógio
5.7	Interrupção	199	d. O PDP-8
5.8	Modos de operação especiais	204	e. Comunicação
	a. Grupo 1	205	f. O sistema
	b. Grupo 2	205	g. A interface
5.9	Sumário	206	6.9 Sumário
5.10	Pranchas	207	<i>Exercícios</i>
	<i>Exercícios</i>	209	<i>Bibliografia</i>
			<i>Leituras complementares</i>
6	<i>Arquitetura da UCP</i>	226	
6.1	Introdução	226	7. Entrada/saída
6.2	Formatos de informação	226	7.1 Introdução
	a. Tamanho de palavra	226	7.2 Dispositivos
			a. Funções
			b. A comunicação
			c. A ligação
			d. Os sistemas
			e. O terminal

167	b. Relacionamento operando/instrução	226
168	c. Formato dos operandos	227
168	d. O formato das instruções	228
170	6.3 Especificação do operando	230
170	a. Imediato ou implicado	230
170	b. Direto	231
170	c. Indireto	231
172	d. Indexado	231
172	e. Relativo	233
172	f. Abreviado	233
173	g. Combinações	233
175	6.4 A potência do conjunto de instruções	233
176	a. Instruções aritméticas	234
176	b. Instruções referentes à memória	235
177	c. Desvios	235
178	d. Deslocamento e operações booleanas	236
179	e. Supervisão e entrada/saída	237
181	f. Outras instruções	237
181	6.5 O PDP-8	237
181	a. Introdução	237
185	b. As instruções	238
185	c. Estados principais e interrupção	241
188	d. Comentário	242
189	6.6 Arquitetura da terceira geração e os sistemas IBM S/360 e IBM S/370	242
190	a. Motivação	242
190	b. As direções da arquitetura	242
190	6.7 A arquitetura do IBM S/360	243
191	a. Ponto de vista do programador	244
191	b. Formato dos dados	244
192	c. Formatos da instrução e do endereçamento	245
193	6.8 Feições do S/360 e S/370 para multiprogramação	248
193	a. Introdução e o sistema <i>Stretch</i>	248
194	b. Proteção da memória	248
194	c. Relocação e transcodificação dinâmica do endereço	249
195	d. O PSW e o mecanismo de interrupção	249
195	e. Comentário sobre o PSW	250
199	f. O estado "supervisor"	251
199	g. A instrução "test and set" (TS)	251
199	6.9 Sumário	251
204	<i>Exercícios</i>	252
205	<i>Bibliografia</i>	252
205	<i>Leituras complementares</i>	253
206		
207	7. Entrada/saída	254
209		
225	7.1 Introdução	254
225	7.2 Dispositivos e suas interfaces	255
226	a. Funções dos periféricos	255
226	b. A essência do controle dos periféricos	255
226	c. A ligação entre a UCP e o dispositivo	255
226	d. Os sinais da interface	257
226	e. O timing da via de E/S	258

e.1. Caso sincrono	258
e.2. Caso assíncrono	259
f. Características tecnológicas da via	260
7.3 A interface do programa e o dispositivo	260
a. Programado	261
b. Acesso direto à memória (<i>DMA</i>)	262
c. Canal	263
d. Processador periférico (<i>CDC 6600</i>)	263
7.4 Interrupção	263
a. Classes e tratamento geral de interrupção	263
b. Interrupção de nível simples	264
c. Interrupção de níveis múltiplos	265
d. <i>Hardware</i> generalizado para cada nível	265
e. Técnicas para efetuar a interrupção	267
f. Técnicas para descobrir a causa da interrupção	269
g. Interrupção para casos de falta de energia	269
7.5 Interrupção no <i>HP 2116B</i>	270
a. Assuntos gerais	270
b. Os níveis e a seqüência da interrupção	271
c. A rede de prioridade	271
d. Os <i>flip-flops control</i> e <i>flag</i>	271
e. O projeto da parte da interrupção do cartão de interface	272
f. O subsistema de interrupção da <i>UCP</i>	274
g. Carta de <i>timing</i> da interrupção	275
7.6 A arquitetura de E/S no <i>S/360</i>	277
a. Assuntos gerais	277
b. Os compromissos da implementação	277
c. O formato e os tipos de comando	278
d. A seqüência típica das operações	279
e. Unidades de controle	280
f. As características dos canais	280
g. A interface padronizada	281
7.7 O painel	282
a. Função	282
b. Contatos mecânicos	283
c. A leitura e a escrita da memória principal	283
d. A carga do programa inicial (<i>IPL</i>)	283
e. Controles "roda", "partida" e "pare"	283
7.8 Sumário e conclusões	284
<i>Exercícios</i>	284
<i>Bibliografia</i>	284
8 Unidades de controle	285
8.1 Conceitos gerais	285
a. Possíveis classificações das unidades de controle	287
8.2 Controle fixo	288
a. Registrador do código da instrução	288
b. Registrador de endereço de instrução	288
c. Decodificador	289
d. Gerador de sinais de controle	289
e. Relógio central	289
f. Controle do modo de operação	289

8.3 Unidade de

- a. Memória
- b. Paralelo
- c. Cadeia
- d. Resumos

8.4 Comparação

- a. Facilidade
- b. Flexibilidade
- c. Memória
- d. Velocidade
- e. Economia

Bibliografia

Apêndice

Índice

258	8.3 Unidade de controle microprogramada	290
259	a. Monofásicos e polifásicos	291
260	b. Paralelo e série	292
260	c. Codificado e não-codificado	292
261	d. Resumo	292
262	8.4 Comparação entre as duas técnicas de controle	293
263	a. Facilidade de projeto	293
263	b. Flexibilidade	293
263	c. Manutenção	293
263	d. Velocidade	293
264	e. Emulação	293
265	<i>Bibliografia</i>	294
265	<i>Apêndice Os microcomputadores</i>	295
267	<i>Índice</i>	353
269		
269		
270		
270		
271		
271		
271		
271		
272		
274		
275		
277		
277		
277		
278		
278		
279		
280		
280		
281		
282		
282		
283		
283		
283		
283		
284		
284		
284		
285		
285		
287		
288		
288		
288		
289		
289		
289		
289		

PREFÁCIO

As atividades relacionadas com projeto, montagem e produção de computadores, iniciadas há alguns anos no Brasil, estão sendo intensificadas rapidamente. Diversas iniciativas importantes nessa nova área foram tomadas, ou serão tomadas a curto prazo, tanto em universidades como na indústria, em grandes empresas estatais e outros órgãos do Governo. Como consequência, o país defronta-se com a necessidade de um número crescente de especialistas em sistemas digitais, e começa a formá-los. A publicação deste livro, ajuda significativa para a solução desse problema, é, portanto, muito oportuna.

É também auspicioso que essa tarefa tenha sido encetada sob a responsabilidade de profissionais altamente credenciados como o doutor Glen G. Langdon Jr. e o engenheiro Edson Fregni.

O doutor Langdon obteve com brilhantismo o título de doutor na University of Syracuse, em 1967, após seu mestrado na University of Pittsburgh e sua graduação na Washington University. Toda sua atividade acadêmica, suas inúmeras publicações e sua intensa atividade profissional têm sido concentradas na mesma área enfocada neste livro. Em várias oportunidades, o doutor Langdon ministrou cursos de pós-graduação na Escola Politécnica da Universidade de São Paulo, onde tem contribuído, com rara dedicação, para o desenvolvimento do Laboratório de Sistemas Digitais do Departamento de Engenharia de Eletricidade.

O engenheiro Fregni graduou-se pela Escola Politécnica da Universidade de São Paulo, em 1972, onde obteve o grau de Mestre em Engenharia. Frequentou vários cursos e desenvolveu atividades de pesquisa no Digital Systems Laboratory da Stanford University, e está engajado num programa de doutoramento. Pertence ao corpo docente da Escola Politécnica, e participa das atividades do Laboratório de Sistemas Digitais. Os trabalhos realizados pelo engenheiro Fregni, neste seu inicio de vida profissional, prenunciam uma brilhante carreira.

Nessas condições, foi com prazer que recebi a incumbência de escrever esta página introdutória, seguro de que o primeiro livro do doutor Langdon e do engenheiro Fregni será de grande valia para os interessados nos fascinantes problemas dos computadores e de todos os sistemas digitais.

Antonio Hélio Guerra Vieira

PREFÁCIO DOS AUTORES

A primeira versão deste texto surgiu como "notas de aulas" do curso de pós-graduação "Projeto de sistemas digitais", ministrado pelo professor Langdon, na Escola Politécnica da Universidade de São Paulo (EPUSP), no primeiro semestre de 1971. Tal curso era dirigido principalmente aos engenheiros e professores do Laboratório de Sistemas Digitais (LSD) da EPUSP com planos para projetarem um minicomputador. No segundo semestre de 1971, o professor Langdon convidou o engenheiro Fregni, que assistira ao curso, para, juntos, reescreverem aquelas notas de aula. Surgiu então uma segunda versão, chamada "Estruturas de computadores", em sete capítulos (os primeiros sete capítulos deste livro). Ambos os autores, bem como outros professores da EPUSP, tiveram oportunidade de usar tal texto em vários cursos, tanto de graduação, como de pós-graduação. Muito os autores devem aos estudantes e professores da EPUSP, principalmente aos elementos do LSD, pelos erros encontrados e modificações sugeridas.

Acredita-se ser desnecessário justificar a existência de tal livro no Brasil. Aqui ele não é "mais um", pois pouca coisa se tem publicado em português nessa área. Espera-se que tal trabalho venha a incentivar o inicio de outras publicações correlatas. Na época da organização do seu primeiro curso na EPUSP, o professor Langdon não encontrou livro algum no idioma inglês que se adaptasse às necessidades, ou seja, que, além de detalhar o funcionamento das unidades básicas de um computador mostrasse como elas se interligam e integravam. Espera-se que o presente trabalho tenha conseguido reunir essas duas qualidades. Por isso, acreditam os autores que este livro seja válido e útil, não apenas pelo fato de ter sido escrito em português.

Este livro é dirigido principalmente aos estudantes de engenharia de computação ou ciência de computação, nos últimos anos de graduação ou em nível de pós-graduação. Conhecimentos de álgebra booleana e teoria da comutação (*switching theory*) são pré-requisitos, apesar de uma breve introdução a esses assuntos ser dada no Capítulo 1. Além disso, procurou-se redigir este texto de forma tal a facilitar o engenheiro autodidata que queira se atualizar no assunto, mas que não disponha de tempo livre para freqüentar algum curso de pós-graduação. Os autores testaram este manuscrito em um curso organizado por eles e pelo engenheiro Victor Penna da Rocha, da IBM do Brasil, e ministrado pelos professores do LSD aos engenheiros e técnicos da fábrica da IBM do Brasil, em Campinas (SP). Os estudantes, apesar de não possuirem muita experiência no assunto, estavam se preparando para a produção e ensaio da unidade central de processamento de um computador de porte médio (IBM S/370, modelo 145) e unidades de controle de fitas magnéticas. Constatou-se que este texto se adapta bem às circunstâncias onde o espírito totalmente acadêmico é posto em segundo plano, destacando-se os detalhes práticos do assunto.

Os primeiros capítulos apresentam as unidades básicas de um sistema isoladamente e os últimos analisam o problema de como interligar essas unidades, e como controlar os sinais no tempo (*timing*).

O Capítulo 1 oferece um resumo do que se entende serem os pré-requisitos para os capítulos seguintes. Esse capítulo é muito útil para o professor, nas primeiras aulas do curso, quando normalmente se faz uma revisão rápida dos pré-requisitos. Um problema básico nessa área é o da representação dos números internamente à máquina. Sabe-se que os computadores digitais eletrônicos usam grandezas elétricas, tais como tensões ou correntes, para representar as constantes e variáveis do problema que está sendo resolvido.

O Capítulo 2 é dedicado ao estudo e comparação das inúmeras maneiras de se representarem tais números e operar com tais representações. Um sistema eletrônico tão sofisticado quanto um computador digital seria tremendamente complicado para se entender e projetar se o problema fosse tratado ao nível dos transistores. Por isso, o que se faz é definir algumas unidades básicas, tais como registradores, unidades aritméticas etc., que são interligadas para formar o sistema completo. Tais unidades, que, por sua vez, são subdivididas em partes menores, os blocos lógicos (Capítulo 1), são estudadas no Capítulo 3.

O Capítulo 4 é reservado para o estudo das memórias. Dedica-se um capítulo a esse assunto, dada a importância dessa unidade também pelo fato de estar em franco desenvolvimento, acompanhando a evolução da tecnologia. Nesse capítulo discutem-se também alguns aspectos de como a organização do sistema pode melhorar a eficiência das unidades. Tome-se, por exemplo, a organização tipo "memória virtual", onde se faz uma memória grande e lenta, associada a uma pequena, mas veloz, comportar-se como uma memória grande e de alta velocidade.

No Capítulo 5, existe uma descrição detalhada de um minicomputador. A existência de tal material é justificada pela necessidade de um exemplo que ajude o estudante, a essa altura, a entender como as unidades, estudadas isoladamente, funcionam em um todo. Tal computador é o mesmo que foi projetado e montado após o primeiro curso do professor Langdon, mais tarde chamado de "Patinho Feio" pelos membros da equipe que o projetou. Pelo que se tem conhecimento, esse foi o primeiro computador digital brasileiro.

No Capítulo 6, estudam-se as diferentes maneiras de organizar as unidades básicas formando um sistema completo ("arquitetura"). Analisa-se nesse capítulo como tal organização evoluiu de geração para geração. Apresentam-se também alguns exemplos reais, tais como *PDP-8*, *IBM S/360* e *Burroughs 5000*.

O Capítulo 7 é dedicado ao estudo dos processos de entrada e saída. Também ai se manteve o que os autores procuraram fazer em todo o livro: apresentar inúmeras diferentes soluções para o mesmo problema, e sempre que possível citar um exemplo real de onde e por que se adotou tal solução. No final desse capítulo estuda-se o sistema de entrada e saída dos computadores *HP2116B* e *IBM S/360*.

O Capítulo 8 oferece um breve estudo das unidades de controle. Dá-se ênfase às diferenças entre as unidades de controle cujo funcionamento é definido por circuitos (*hardwired*) e as microprogramadas. Ao longo de todo o livro procurou-se, sempre que possível, apresentar alguns exercícios ao leitor. Recomendamos aos professores que adotarem este texto em algum curso que o usem como tarefa para os estudantes, pois tais exercícios complementam o material apresentado, e são peças importantes para o aprendizado.

Um problema enfrentado pelos autores durante a redação deste trabalho foi a necessidade de se escrever em *português* sobre um assunto aprendido pela maioria em inglês. É fato sabido que os técnicos evitam traduzir os nomes dos elementos ou conceitos introduzidos pela literatura em inglês. Isso é devido, principalmente, ao medo de não se fazerem entender, caso se aventurem a traduzi-los. Este trabalho, que se propõe ser um livro-texto, tem a grande responsabilidade de introduzir muitas pessoas a essa área, e se os autores não fossem cuidadosos e, por que não dizer, conservadores, correr-se-ia o risco de dificultar o prosseguimento do estudo dessas pessoas através das referências fornecidas, que, na sua grande maioria, são escritas no idioma inglês. Entretanto os termos que são normalmente traduzidos, e cuja tradução é única e fácil de se voltar ao inglês, foram aqui traduzidos sem

referência ao mas que já forneceu palavras que para que se p o fato de ser a "anglicanização" alguma novas

Para rem à Edição da Vieira, pelas que nossas pelas críticas Yoshiyuki Ito Filho, pelo encontro com Endicott Noble professor Luiz

referência ao nome original^[1]; outros termos cuja tradução não é ainda geralmente usada, mas que já começa a se popularizar, foram traduzidos com o correspondente termo em inglês fornecido entre parênteses^[2]. Foram escritas em inglês, sem se tentar uma tradução, as palavras que ou são usadas em inglês pela maioria^[3] ou cujos conceitos são muito novos para que se possa determinar qual a tradução adotada^[4]. Deve-se esclarecer, entretanto, que o fato de serem usados muitos termos em inglês não implica que os autores concordem com a "anglicanização" do nosso idioma. A uniformização de tais termos deveria ser tarefa de alguma revista técnica, que, infelizmente, inexiste em nosso idioma.

Para terminar, os autores gostariam de agradecer à Editora Edgard Blücher Ltda. e à Editora da USP, pelo apoio nesta tão pouco lucrativa tarefa; ao professor A. H. Guerra Vieira, pelas críticas e sugestões ao nosso trabalho bem como pelo apoio fundamental para que nossos planos em relação a este livro se tornassem realidade; a toda equipe do LSD, pelas críticas e sugestões, da qual gostaríamos de citar em particular o engenheiro Célio Yoshiyuki Ikeda, pelas intermináveis discussões; ao engenheiro Walter Rodrigues Ferreira Filho, pelo trabalho de coordenação durante a revisão final do texto, enquanto os autores se encontravam fora do país. Finalmente, à IBM (IBM do Brasil, IBM World Trade e IBM Endicott N.Y.) pelo apoio dado — sem o qual este livro não existiria — quando da visita do professor Langdon à EPUSP.

São Paulo, fevereiro de 1974

Edson Fregni
Glen George Langdon, Júnior

^[1]Núcleo magnético, instrução, memória etc.

^[2]"Vai-um antecipado" (*carry lookahead*), via (*bus*)

^[3]Hardware, flip-flop, fan-out etc.

^[4]Cache, schedules etc.

capítulo

INTROD

Os autores ficariam muito gratos se sugestões e eventuais falhas fossem comunicadas para

Professor Edson Fregni
Laboratório de Sistemas Digitais
Escola Politécnica da USP – DEE
Cidade Universitária
05508 – São Paulo – SP

1.1 ORGANIZAÇÃO

Em 1953, de calcular em dade aritmética de 20 dígitos e uma máquina pela inclusão de por falta de memória eram suficientes cobrir que, não em conta a necessidade de válvulas a visar o computador, e com relés pelo sistema eletrônico finalizado o cálculo de tabela feita por chaveamento, é fácil mudar o programa armazenado na memória pelo reléário.

É desse modo que foram implementados os primeiros computadores.

Atualmente, o computador é composto por um somador, registradores, controladores, etc. O conjunto formado por essas unidades constitui a unidade central de processamento, ou seja, a memória. Os

O programador pode programar conjuntamente com o operador de telex a execução de 8000 instruções diferentes. É possível o uso de computadores que executam instruções na velocidade das instruções comuns, que estão sendo executadas.

A separação das instruções não permite a execução de

capítulo 1

INTRODUÇÃO E CONHECIMENTOS BÁSICOS

1.1 ORGANIZAÇÃO DE SISTEMAS DE COMPUTAÇÃO

Em 1833, o matemático inglês Charles Babbage apresentou os planos de uma máquina de calcular (engenho analítico)⁽¹⁾ provida de memória (1 000 números de 50 dígitos), unidade aritmética (que somava dois números de 50 dígitos em 1 s e multiplicava dois números de 20 dígitos em 3 min⁽²⁾) e instruções. As instruções foram baseadas no “Jacquard loom”, uma máquina para controlar a tecelagem com fios coloridos, aperfeiçoados por Babbage pela inclusão de uma instrução do tipo desvio condicional. Essa máquina não foi realizada por falta de tecnologia, já que os dispositivos mecânicos disponíveis no século XIX não eram suficientes para implementar a organização projetada. Babbage foi o primeiro a descobrir que, no projeto de organização de uma máquina de computação, é necessário levar-se em conta a tecnologia (*hardware*) existente. Somente após o desenvolvimento dos relés e válvulas a vácuo que as idéias de Babbage foram redescobertas e realizadas. O primeiro computador, o Harvard Mark I, projetado por Aiken (de Harvard) em 1939 e implementado com relés pela IBM em 1944⁽³⁾, utilizou os princípios de Babbage. O primeiro computador eletrônico foi o Eniac, cujo projeto começou em 1943 (com a aplicação específica para o cálculo de tabelas de balística), ficando operacional em 1946. No Eniac a programação era feita por chaves e pela colocação de vários fios em soquetes (*plugboard wiring*). Não era fácil mudar o programa e, através dessa experiência, evoluiu-se para o conceito de *programa armazenado* (*stored program concept*), cuja idéia foi apresentada pela primeira vez em 1946 pelo relatório de Burks, Goldstine e von Neumann⁽⁴⁾.

É desse relatório que conhecemos a clássica “máquina tipo von Neumann”. As idéias foram implantadas no computador *IAS*, cuja organização aparece na Fig. 1-1.

Atualmente, a unidade aritmética é conhecida como *fluxo de dados*, constando de um somador, registradores centrais e as vias (“busses”) de interligação entre essas unidades. O conjunto formado pelos fluxo de dados, unidade de controle e memória é chamado de *unidade central de processamento (UCP)*. A unidade de controle recebe as instruções codificadas da memória. A unidade aritmética e a entrada/saída recebem e fornecem dados para a memória. Os dados são codificados em binário de 40 bits, lidos da memória em paralelo.

O programa consta de instruções, que são codificadas e armazenadas na memória, em conjunto com os dados. Cada instrução é dividida em duas partes: um campo de código de operação de 8 bits e um campo de endereço de 12 bits [Fig. 1-1(b)]. Essa máquina dispõe de instruções para modificar ou carregar o campo de endereço de outras instruções. Assim, é possível o uso da técnica de programação de *loop*. Os dois principais aperfeiçoamentos do computador *IAS* sobre o esquema de Babbage foram, então, (1) o armazenamento das instruções na memória, juntamente com os dados e (2) a possibilidade de se operarem as instruções como se fossem dados. É interessante notar que atualmente essas duas técnicas estão sendo criticadas.

A separação física das instruções dos dados aumenta a confiabilidade e a técnica de não permitir a modificações das instruções, facilita a descoberta e eliminação de erros

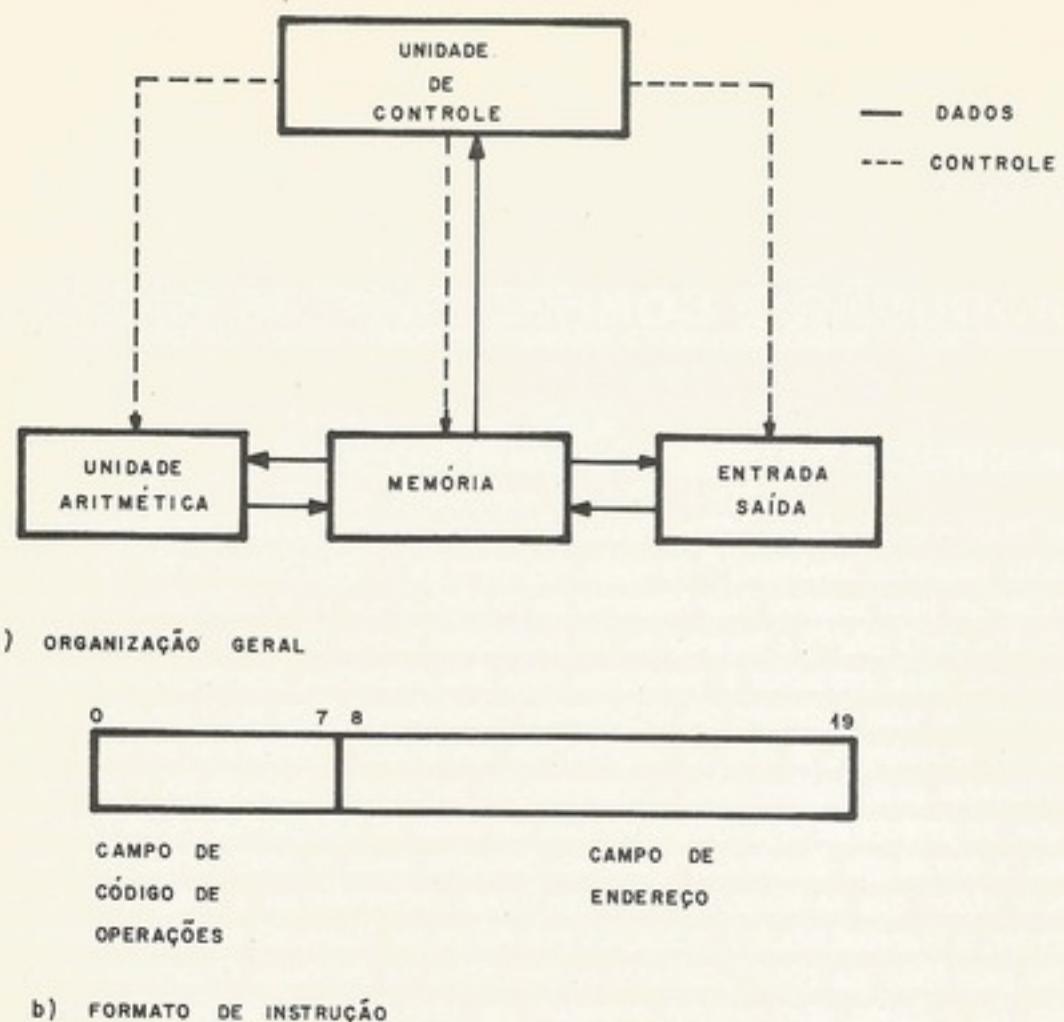


Figura 1-1. O computador *IAS*

(debugging) e a modificação e atualização dos programas. (Além do mais, existem dificuldades de simulação e emulação de programas que modificam a si mesmos).

No projeto do computador *IAS*, considerou-se a possibilidade de implementação das instruções em ponto flutuante, o que foi rejeitado por motivos econômicos. Para compensar, foram implantadas instruções de deslocamento que facilitaram a programação das sub-rotinas de ponto flutuante.

Um programa de instruções, ou simplesmente um *programa*, é o meio pelo qual o computador faz o processamento de dados. A unidade central só pode retirar as instruções da memória e executá-las exatamente como foi programado. Um programa mal feito é capaz de cometer muitos erros. (Muitos caracterizam o computador como o idiota mais veloz do mundo.)

As características do computador do ponto de vista do programador, são enquadradas dentro do que chamamos de *arquitetura* do sistema de computação. O conjunto de instruções, a organização e o endereçamento da memória, a interface e o método de entrada/saída, o painel (*console*) e o sistema de interrupção, são todos aspectos de arquitetura. A aplicação da tecnologia na implementação da arquitetura em *hardware* é o *projeto lógico* do sistema. O engenheiro, na implementação do sistema, tem de estudar a relação entre vários métodos, avaliando o custo e a *performance* de cada um (ou seja, estudar os compromissos ou *trade-offs*).

Um compromisso (*trade-off*) muito comum em sistemas de computação é o de “espaço-tempo”, onde o programador pode tornar seu programa mais veloz, ao custo de uma quantidade maior de memória principal. O mesmo compromisso ocorre em hardware, onde a *UCP*, por exemplo, pode processar instruções com mais velocidade ao custo de mais circuitos.

introdução e conclusão

Desde a época do desenvolvimento da tecnologia pelo transistores de silício, a indústria (usando endereços especializados para a mesma) fazem a sua parte. Também evoluíram os controladores fazem a sua parte. Para se indicar a geração de dados, "segundo

Na *primeira* memória de limbus eletrostático. Na *segunda* A memória primária usava temas simples, com pou-

Na segunda a lógicos. Transistor A memória principal endereço efetivo é desenvolveu-se o endereço da instrução que usam entrada para interrupção. O comandoário é feito com a auxiliando-se de ajuda o operador deve ser feita a memória. Os computadores calculam números decimais com velocidade numérica.

No começo os circuitos integrados eram fabricados numa só unidade de montagem, formando uma "família" de variadas performances (e com sistemas completamente integrados). Softwares que apóiam os programadores visores ou montadores da terceira geração permitem um programa para o processamento de um sistema, a UCP, e um diagrama de um sistema. As várias gerações não

A ideia de "tintas esbarradas" e arquitetura

Desde a época dos primeiros computadores, houve desenvolvimentos fantásticos. O desenvolvimento mais acentuado foi na tecnologia, onde a válvula a vácuo foi substituída pelo transistor discreto e este pelo circuito integrado. Acompanhando o desenvolvimento em tecnologia, a arquitetura dos sistemas tem recebido melhoramentos no endereçamento (usando endereços relativos, indiretos ou indexados), circuitos para ponto flutuante, canais especializados para entrada/saída independente e esquemas de *interrupção* de programas. Também evoluíram sistemas de programação chamados *software*, onde os programas monitores fazem a escalação dos serviços e regulam o fluxo de trabalho através do sistema. Para se indicarem as etapas desses desenvolvimentos, é comum o uso dos termos “primeira geração”, “segunda geração” e “terceira geração”.

Na *primeira geração* (1946-1956), a tecnologia era caracterizada pela válvula a vácuo, memória de linhas de atraso (tipicamente de mercúrio) ou de tubo de raios catódicos (*CRT*) eletrostático. No fim da primeira geração, foi introduzida a memória de núcleo de ferrite. A memória principal era muito pequena (da ordem de 1 024 a 4 096 palavras). Alguns sistemas usavam tambores magnéticos para memória principal. A entrada/saída era muito simples, com pouca concorrência com o processamento.

Na *segunda geração* (1956-1963), foi introduzido o transistor na tecnologia dos circuitos lógicos. Transistores, resistores e capacitores eram montados em placas de circuito impresso. A memória principal era de núcleo de ferrite (como na terceira geração). Para calcular o endereço efetivo das instruções, eram usados registradores de índice. Também nessa época, desenvolveu-se o conceito de endereçamento indireto, onde o conteúdo do campo de endereço da instrução é o endereço do operando (em vez do próprio operando). Muitos sistemas usam entrada/saída (*E/S*) com processamento simultâneo auxiliado por um esquema de *interrupção*. O *canal* controla a *E/S* independentemente da *UCP*. O armazenamento secundário é feito com tambores e fita magnética. O processamento é feito em “lotes”, desempenhando-se os serviços (*job* ou tarefa) seqüencialmente, com um programa monitor que ajuda o operador a evitar tempo ocioso da unidade central. Esse programa determina quando deve ser feita a montagem de fitas magnéticas, e também faz a contabilidade dos serviços. Os computadores dessa geração dividiam-se em duas categorias: “comercial” (mais lento, números decimais e nada para facilitar operações em ponto flutuante); e “científico” (mais veloz, números binários, com instruções que facilitavam ponto flutuante).

No começo da *terceira geração* (1964), a tecnologia era a de transistores montados em circuitos integrado, com uma “integração” de um ou dois circuitos lógicos por pastilha. Muitos fabricantes resolveram reunir aspectos do processamento comercial e científico numa só arquitetura. Além disso, essa mesma arquitetura (conjunto de instruções, endereçamento, formato de dados, esquema de *interrupção* e entrada/saída) foi implementada numa “família” de vários modelos [modelos de baixa *performance* (e custo) até modelos de alta *performance* (e custo)]. Dos sistemas de programação, evoluiu o *sistema operacional*, um sistema complexo de rotinas que oferece muitas opções e serviços aos usuários do sistema, e que cuida do andamento dos programas dos usuários. Esse sistema é *software* bem sofisticado. *Software* é uma palavra genérica usada para designar os programas do sistema que apóiam os programas de aplicação do usuário, tais como compiladores, montadores, supervisores ou monitores, editores, sub-rotinas de entrada/saída, e programas utilitários. Com a terceira geração, introduziu-se a técnica de *multiprogramação*, onde pode haver mais de um programa na memória principal ao mesmo tempo. Multiprogramação entrelaça o processamento de programas diferentes para melhor aproveitar os principais recursos do sistema, a *UCP*, a memória principal e o armazenamento em fita ou disco magnético. Um diagrama de um sistema de terceira geração aparece na Fig. 1-2 e as características principais das gerações são indicadas na Tab. 1-1.

A idéia de “geração” vem sendo amplamente criticada, pois a noção de gerações distintas esbarra em muitas exceções. O *CDC 6600*, um sistema de computador de grande porte e arquitetura típica da terceira geração, foi implementado em transistores discretos, o que

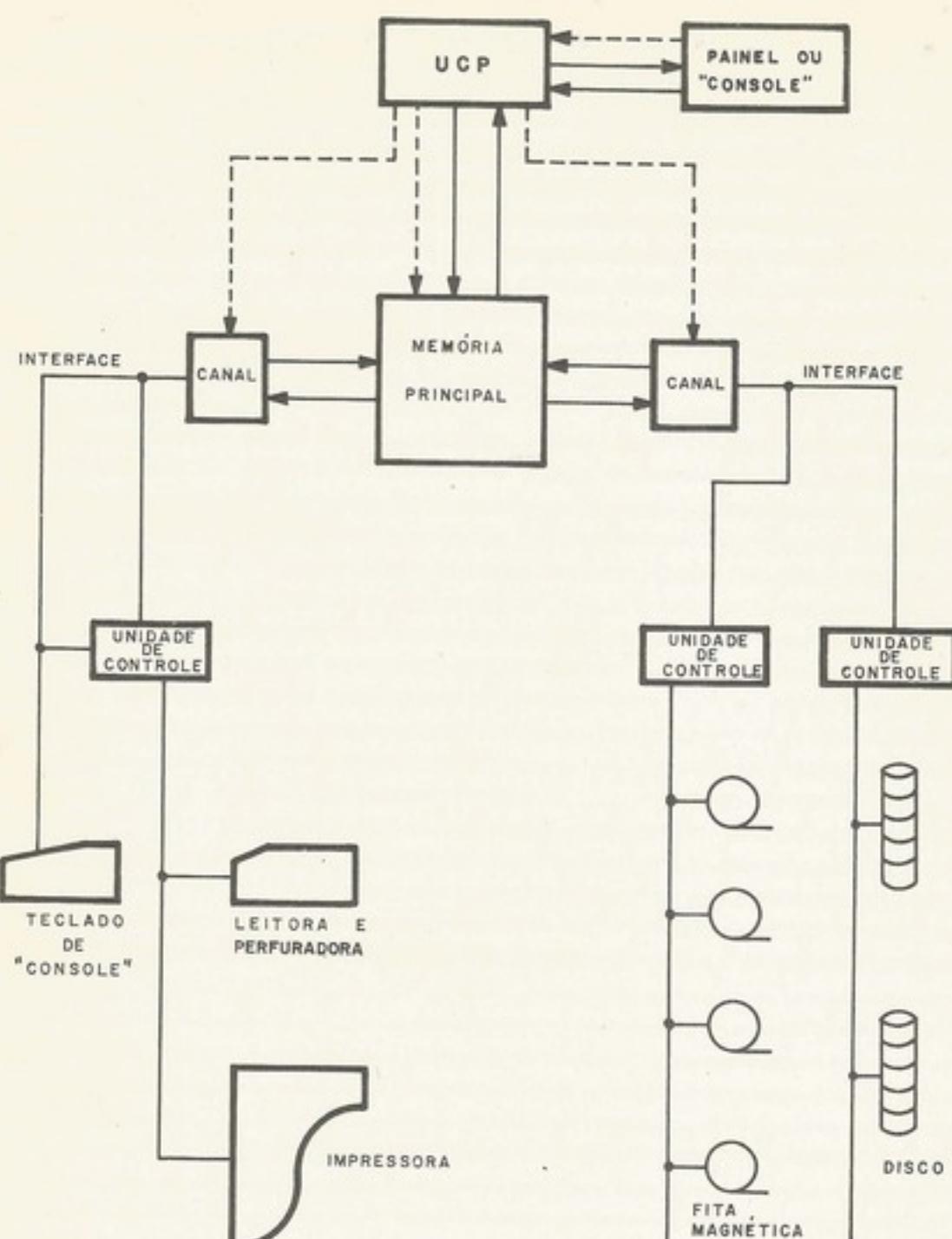


Figura 1-2. Um computador de terceira geração

é uma característica da segunda geração. E, ainda, muitos computadores implementados em circuitos integrados em grande escala, os minicomputadores, possuem algumas características da primeira geração.

Na primeira geração, os interesses eram mais concentrados na unidade central e as técnicas de projeto lógico tinham grande influência na organização do sistema. Naquela época o *hardware* era bem mais caro que a programação. Os engenheiros pensavam em termos de US\$10 por válvula ou *gate* lógico. Hoje em dia, o *software* e a programação são mais caros. Então aqueles que falam na arquitetura da quarta geração tendo em vista os novos compromissos pensam mais nas possibilidades de se aumentar o *hardware* para facilitar a programação do *software*. Também, em vez de se considerar o sistema como uma simples máquina de fazer somas e multiplicações, está surgindo a idéia da “informação” e seu fluxo entre os arquivos de armazenamento, a memória principal, e os registradores

Genérico
Primer...
Segund...
Terceir...

centrais da II...
em arquive...
temas de cre...
sistemas ma...
de informaç...
e

1.2 ÁLGEBRA
A álgebra
estudo da log...
e, em 1938, p...
telefônicas.

a. Postu...
A álgebra
recebem, con...
deve estar al...
No campo d...
quais associ...
“1” é chama...
O conjun...
booleana, em

O comp...
simbólos s...
e

Existem...
e OR (OU).

Operações
Usualmen...

Tabela 1-1. Características das gerações

Geração	Tecnologia	Processamento	Operação
Primeira	Válvulas a vácuo	Escalação com hora marcada	Por programador-operador
Segunda	Transistores discretos	Em lotes	Programa monitor simples para ajudar o operador
Terceira	Circuitos integrados	Multiprogramação	O operador segue as exigências de um sistema operacional complexo

centrais da UCP conforme a atividade. Hoje há grande interesse na organização de dados em arquivos (*file organization*) para tornar mais eficiente o acesso às informações. Os sistemas de computação já estão sendo caracterizados, em grande contraste com os primeiros sistemas usados para fazer cálculos rotineiros para tabelas matemáticas, como sistemas de informação.

1.2 ÁLGEBRA BOOLEANA

A álgebra booleana foi desenvolvida pelo matemático George Boole (1815-1864) para o estudo da lógica. Ela foi apresentada em 1854 no seu *An investigation of the laws of thought*⁽⁵⁾ e, em 1938, Claude E. Shannon⁽⁶⁾ adaptou a álgebra booleana para o projeto de redes telefônicas.

a. Postulados e definições

A álgebra booleana é definida sobre um conjunto com dois elementos. Esses elementos recebem, comumente, os nomes *verdadeiro* e *falso*, ou *alto* e *baixo*, ou “0” e “1”. O leitor deve estar alerta para o fato de não existir significado numérico algum nesses elementos. No campo dos sistemas digitais, esses dois valores são dois níveis de tensão pré-fixados nos quais associamos os nomes “0” e “1”. Uma variável que só pode assumir os valores “0” e “1” é chamada de variável booleana.

O conjunto $V = \{0,1\}$ é definido pelos seguintes postulados: seja x uma variável booleana; então

$$\begin{aligned} (P.1) \quad & \text{se } x \neq 1, \text{ então } x = 0; \\ (P.2) \quad & \text{se } x \neq 0, \text{ então } x = 1. \end{aligned}$$

O complemento de uma variável booleana x (que pode ser representado por um dos símbolos \bar{x} , x' ou $-x$) é definido pelos seguintes postulados:

$$\begin{aligned} (P.3) \quad & \text{se } x = 0, \text{ então } x' = 1; \\ (P.4) \quad & \text{se } x = 1, \text{ então } x' = 0. \end{aligned}$$

Existem duas operações básicas na álgebra booleana, que são chamadas de *AND* (*E*) e *OR* (*OU*):

Operação AND

Usualmente representada pelo símbolo \cdot (ou \wedge), é definida pelos seguintes postulados:

$$\begin{aligned} (P.5) \quad & 0 \cdot 0 = 0; \\ (P.6) \quad & 0 \cdot 1 = 0; \\ (P.7) \quad & 1 \cdot 0 = 0; \\ (P.8) \quad & 1 \cdot 1 = 1. \end{aligned}$$

Operação OR

Usualmente representada pelo símbolo + (ou \vee), é definida pelos seguintes postulados:

$$(P.9) \quad 0 + 0 = 0;$$

$$(P.10) \quad 0 + 1 = 1;$$

$$(P.11) \quad 1 + 0 = 1;$$

$$(P.12) \quad 1 + 1 = 1.$$

Uma outra operação booleana (ou binária) também importante é a *exclusive OR* (OU exclusivo), abreviadamente *XOR* (OU X). Ela é usualmente representada pelo símbolo \oplus (ou $\vee\!\vee$), e pode ser definida pelos seguintes postulados:

$$(P.13) \quad 0 \oplus 0 = 0;$$

$$(P.14) \quad 0 \oplus 1 = 1;$$

$$(P.15) \quad 1 \oplus 0 = 1;$$

$$(P.16) \quad 1 \oplus 1 = 0.$$

É fácil demonstrar que as três operações definidas são comutativas e associativas. Sejam A , B e C três variáveis booleanas. Então,

operação AND (passaremos a omitir o ponto nas expressões),

$$AB = BA,$$

$$(AB)C = A(BC) = ABC;$$

operação OR,

$$A + B = B + A;$$

$$(A + B) + C = A + (B + C) = A + B + C;$$

operação XOR,

$$A \oplus B = B \oplus A;$$

$$(A \oplus B) \oplus C = A \oplus (B \oplus C) = A \oplus B \oplus C.$$

b. Blocos lógicos

O nosso estudo de álgebra booleana objetiva exclusivamente sua aplicação aos sistemas digitais. Vamos então ilustrar esse estudo com comentários sobre os elementos lógicos digitais e, para que isso seja possível, vamos abrir um parêntese no nosso estudo para dar algumas definições necessárias.

Existem unidades funcionais básicas, que chamaremos de blocos lógicos, com os quais sintetizamos os circuitos digitais através de combinações. Neste mesmo capítulo faremos um breve estudo desses circuitos digitais. Apresentaremos 4 blocos lógicos básicos.

Inversor ou NOT

É o bloco que realiza a complementação (ou inversão) de uma variável booleana. A Fig. 1-3 mostra os vários símbolos desse bloco.

AND (E)

Tem várias entradas, e a saída é o resultado da operação AND das entradas (Fig. 1-4). Seu funcionamento pode ser caracterizado pela seguinte frase: "a saída do bloco AND será igual a '1' se, e apenas se, todas as entradas forem iguais a '1'".

OR (OU)

Tem várias entradas, e a saída é o resultado da operação OR das entradas (Fig. 1-5). Seu funcionamento pode ser caracterizado pela seguinte frase: "a saída do bloco OR será igual a '0' se, e apenas se, todas as entradas forem iguais a '0'".

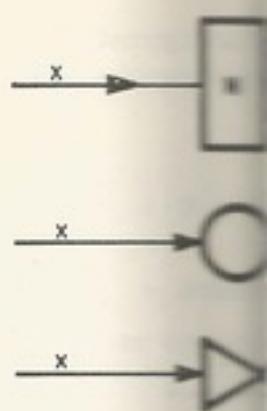
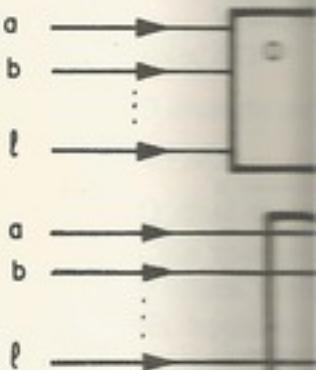
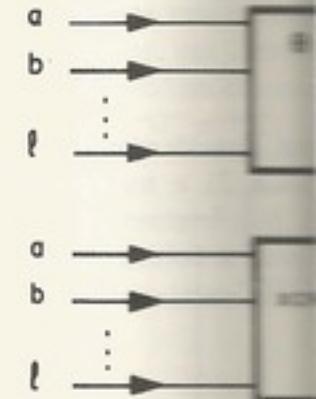


Figura 1-3. Símbolos lógico NOT



XOR (OU EXCLUSIVO)

Tem várias entradas, e a saída é o resultado da operação XOR das entradas. Seu funcionamento pode ser caracterizado pela seguinte frase: "a saída do bloco XOR será igual a '1' se, e apenas se, exatamente uma das entradas forem iguais a '1'".



c. Teoremas de De Morgan

Sejam A , B e C variáveis booleanas.

$$(T.1) \quad A \cdot 0 = 0;$$

$$(T.5) \quad A + 0 = A;$$

$$(T.9) \quad A \oplus 0 = A;$$

$$(T.13) \quad (A \bar{Y}) = A;$$

$$(T.14) \quad A \cdot B + A \cdot \bar{B} = A;$$

$$(T.15) \quad (A + B) \cdot (A + \bar{B}) = A;$$

introdução e conhecimentos básicos

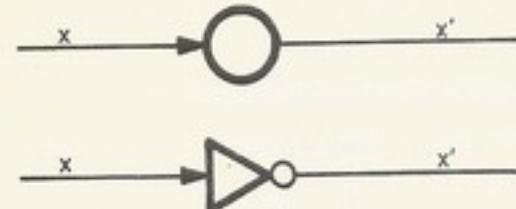


Figura 1-3. Símbolos mais comuns do bloco lógico NOT

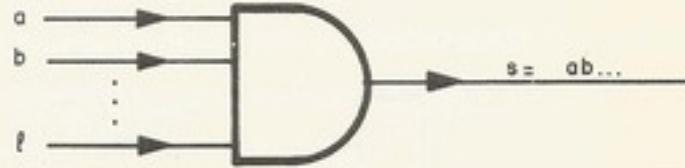
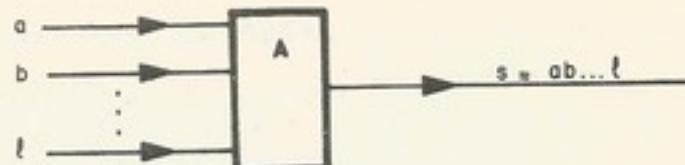
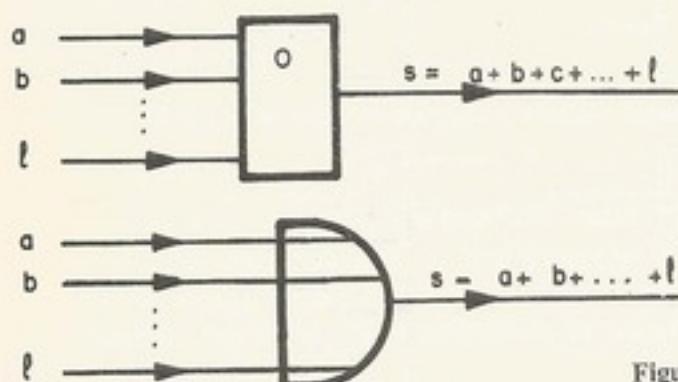


Figura 1-4. Símbolos mais comuns do bloco lógico AND



XOR (OUX)

Tem várias entradas e a saída é o resultado da operação XOR das entradas (Fig. 1-6). Seu funcionamento pode ser caracterizado pela seguinte frase: "a saída do bloco XOR será igual a '1' se, e apenas se, existir um número *ímpar* de entradas iguais a '1'".

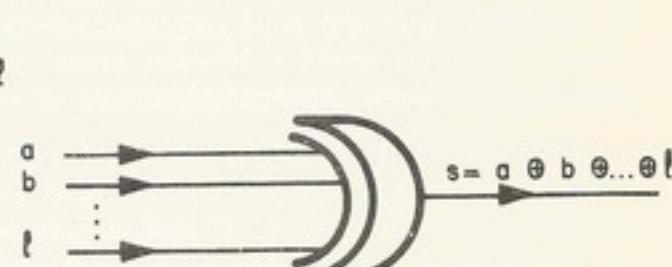
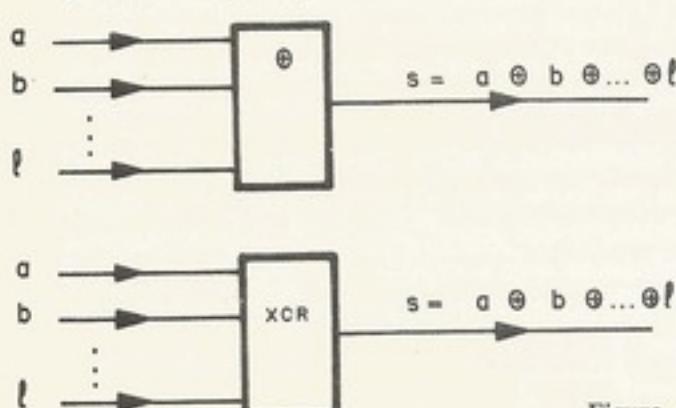


Figura 1-6. Símbolos mais comuns do bloco lógico XOR

c. Teorema de álgebra booleana

Sejam A, B e C três variáveis booleanas. É possível demonstrar as seguintes identidades:

- | | | | |
|--|---------------------------|--------------------------|---------------------------|
| (T.1) $A \cdot 0 = 0;$ | (T.2) $A \cdot 1 = A;$ | (T.3) $A \cdot A = A;$ | (T.4) $A \cdot A' = 0;$ |
| (T.5) $A + 0 = A;$ | (T.6) $A + 1 = 1;$ | (T.7) $A + A = A;$ | (T.8) $A + A' = 1;$ |
| (T.9) $A \oplus 0 = A;$ | (T.10) $A \oplus 1 = A';$ | (T.11) $A \oplus A = 0;$ | (T.12) $A \oplus A' = 1;$ |
| (T.13) $(A')' = A;$ | | | |
| (T.14) $A \cdot B + A \cdot C = A \cdot (B + C)$ | | | |
| (T.15) $(A + B) \cdot (A + C) = A + B \cdot C$ | | | |
- } (fatoração);

- (T.16) $A \cdot B + A \cdot B' = A$; (T.17) $A + A \cdot B = A$; (T.18) $A + A' \cdot B = A + B$;
 (T.19) $(A \cdot B)' = A' + B'$
 (T.20) $(A + B)' = A' \cdot B'$ (teoremas de De Morgan);
 (T.21) $A \cdot B + A' \cdot C + B \cdot C = A \cdot B + A' \cdot C$;
 (T.22) $A \oplus B = A \cdot B' + A' \cdot B$;
 (T.23) $(A \oplus B)' = A' \oplus B = A \oplus B' = A' \cdot B' + A \cdot B$.

Na Fig. 1-7 apresentamos o significado físico de algumas dessas identidades.

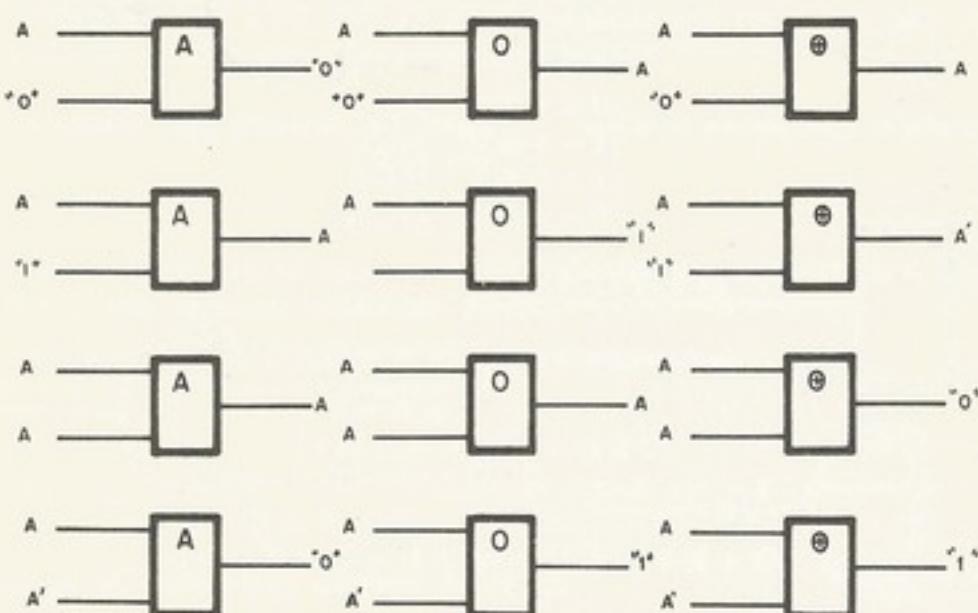


Figura 1-7. Significado físico das doze primeiras identidades

A demonstração dos teoremas pode ser feita pela sua verificação para todos os valores das variáveis, já que o conjunto universo tem apenas dois elementos. Algumas identidades podem ser demonstradas por dedução através da manipulação das expressões utilizando-se os teoremas anteriores e os postulados.

EXEMPLO. Prova por verificação

Seja a identidade (T.19), $(A \cdot B)' = A' + B'$. Montamos a tabela que segue, consideramos todas as combinações possíveis de valores das variáveis A e B (colunas 1 e 2), a seguir calculamos os valores de $(A \cdot B)'$ para cada combinação das entradas (coluna 3) e, finalmente, os valores de $A' + B'$ (coluna 4). Por fim verificamos que a coluna 3 é igual à coluna 4. Portanto a igualdade (T.19) é verdadeira para todos os valores possíveis das variáveis, o que mostra ser essa igualdade uma identidade.

1	2	3	4
A	B	$(A \cdot B)'$	$A' + B'$
0	0	1	1
0	1	1	1
1	0	1	1
1	1	0	0

EXEMPLO. Prova por dedução

Seja a identidade (T.16), $A \cdot B + A \cdot B' = A$. Basta evoluirmos, conforme segue:

$$(T.14) \quad (T.8) \quad (T.2)$$

$$A \cdot B + A \cdot B' = A \cdot (B + B') = A \cdot 1 = A.$$

Demonstração
Seja a identidade

$$A \cdot B + A \cdot B' = A$$

$$= A \cdot B + A \cdot B'$$

Como as operações

$$A \cdot B + A \cdot B'$$

Wood¹⁷ aplica
booleana.

DEFINIÇÃO

O dual de

TEOREMA

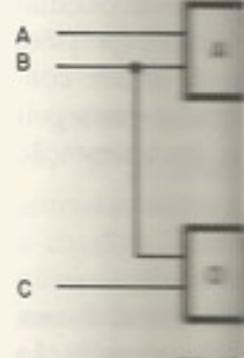
O dual de

Com esse co
(T.20) estariam
álgebra booleana

1.3 CIRCUITOS

a. Introdução

A grossa ma
circuitos de com
tamos um exem
sentar um circui
dependem das en
por "0" e "1".



Os circuitos
combinatórios
das entradas. Os
valores atentam

introdução e conhecimentos básicos

Demonstrações por manipulação da expressão, algumas vezes, são árduas e artificiosas. Seja a identidade

$$(T.21) \quad A \cdot B + A' \cdot C + B \cdot C = A \cdot B + A' \cdot C,$$

$$(T.2) \qquad \qquad \qquad (T.8)$$

$$A \cdot B + A' \cdot C + B \cdot C = A \cdot B + A' \cdot C + B \cdot C \cdot 1 = \\ (T.14)$$

$$= A \cdot B + A' \cdot C + B \cdot C \cdot (A + A') = A \cdot B + A' \cdot C + B \cdot C \cdot A + B \cdot C \cdot A'.$$

Como as operações *AND* e *OR* são associativas e comutativas, podemos escrever

$$(T.15)$$

$$A \cdot B + A' \cdot C + B \cdot C = A \cdot B + A \cdot B \cdot C + A' \cdot C + A' \cdot C \cdot B = A \cdot B + A' \cdot C.$$

Wood⁽⁷⁾ apresenta o conceito de dualidade, que é muito útil no estudo da álgebra booleana.

DEFINIÇÃO

O dual de uma expressão é obtido trocando-se as operações *AND* por *OR* e vice-versa.

TEOREMA

O dual de uma identidade é outra identidade.

Com esse conceito, se, por exemplo, demonstramos o teorema (T.19) a expressão dual (T.20) estará também provada. Para o leitor interessado num estudo mais detalhado da álgebra booleana, recomendamos as referências⁽⁷⁾ e⁽⁸⁾.

1.3 CIRCUITOS DIGITAIS

a. Introdução

A grosso modo, chamaremos de circuitos digitais (ou circuitos lógicos ou, ainda, circuitos de comutação) a combinação de blocos lógicos interligados. Na Fig. 1-8(a) apresentamos um exemplo de um circuito digital. Esquematicamente [Fig. 1-8(b)] podemos representar um circuito digital por um bloco com várias entradas e várias saídas. As saídas dependem das entradas e cada linha delas pode ter apenas dois valores, que representaremos por "0" e "1".

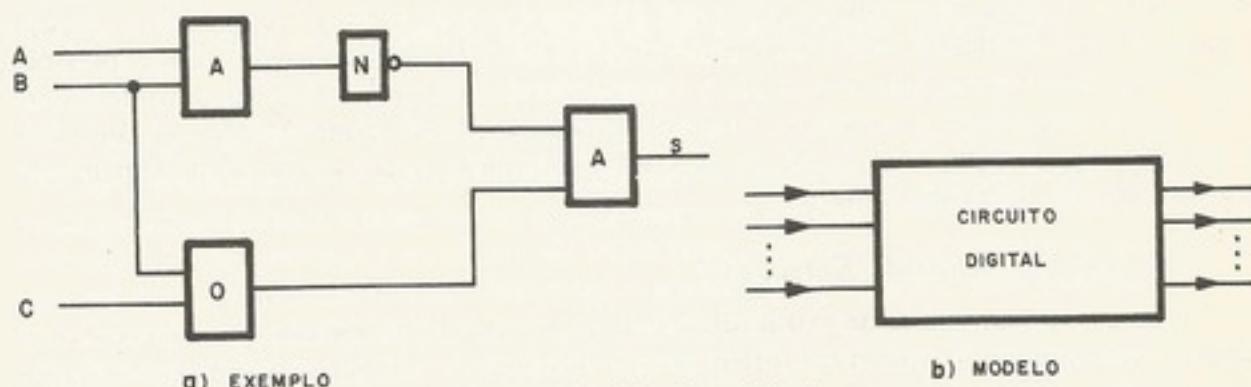


Figura 1-8. Circuitos digitais

Os circuitos lógicos são divididos em duas classes, *combinatórios* e *seqüenciais*. Circuitos combinatórios (ou combinacionais) são aqueles cuja saída depende apenas dos valores atuais das entradas. Circuitos seqüenciais são aqueles cuja saída depende dos valores atuais e dos valores anteriores da entrada. Têm uma espécie de memória interna.

b. Descrição de um circuito combinatório

Vamos estudar aqui apenas o caso de circuitos de apenas uma saída. O leitor pode generalizar para o caso de n saídas, sem muitos problemas.

Consideremos um circuito combinatório com 3 entradas (Fig. 1-9). Já que um circuito combinatório se caracteriza por ter a saída dependendo apenas das entradas atuais, podemos dizer que a saída (s) é dada por $s = F(a, b, c)$. Vamos estudar aqui as várias maneiras de se descrever a função $F(a, b, c)$, que define um circuito combinatório.

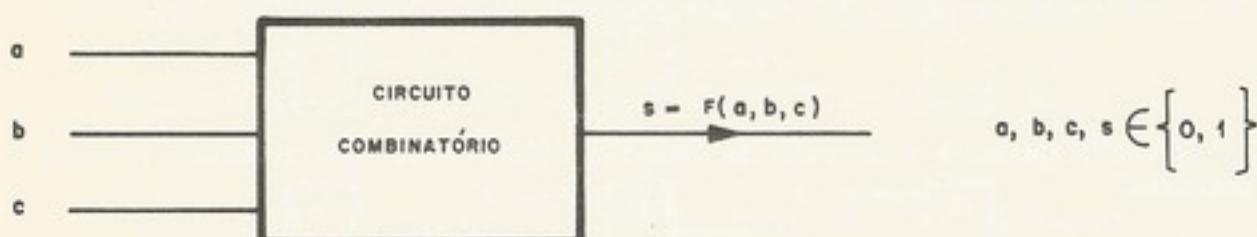


Figura 1-9. Esquema de um circuito combinatório

Apresentaremos quatro maneiras diferentes de se descrever um circuito combinatório, além de seu diagrama lógico (desenho detalhado do circuito mostrando os blocos e as interligações): tabela de combinações, expressão booleana, mapa de Karnaugh e transformada numérica. Conforme o objetivo da descrição, cada maneira tem suas vantagens, seja na documentação, seja nas várias formas de síntese ou, ainda, para ressaltar alguma particularidade importante dos circuitos.

Descrição pela tabela de combinações (tabela da verdade)

A função $s = F(a, b, c)$ pode ser dada por uma tabela, conforme se vê na Fig. 1-10(b). Nessa tabela, fazemos uma primeira coluna com todas as combinações dos valores possíveis das entradas. E, para cada combinação, temos a segunda coluna com o valor correspondente de saída (s). Essa descrição para funções com um grande número de variáveis tem o inconveniente de apresentar um excessivo número de linhas.

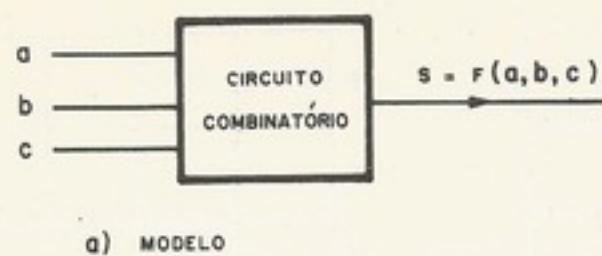
Descrição por uma expressão booleana

A função $s = F(a, b, c)$ pode ser apresentada por uma expressão booleana, conforme se vê na Fig. 1-10(c). Para cada combinação possível dos valores de a, b e c pode-se calcular o valor da correspondente saída. Essa maneira não é unívoca, isto é, existem várias expressões diferentes, equivalentes, que descrevem o mesmo circuito. Uma das técnicas de síntese consiste em se determinar a expressão booleana equivalente mais simples, para se conseguir o circuito mais barato. Essa forma de descrição é também muito usada na documentação e simulação de sistemas digitais.

Descrição pelo mapa de Karnaugh⁽⁹⁾

O mapa de Karnaugh, de extraordinária importância na síntese em dois níveis, é uma tabela com tantas células quantas forem as combinações das entradas. A cada combinação das entradas, corresponde uma célula, dentro da qual colocamos o correspondente valor da saída.

Na Fig. 1-11, apresentamos as tabelas para circuitos com duas, três e quatro entradas, para um maior número de entradas, a descrição pelo mapa perde as suas vantagens. A forma de montagem do mapa é feita de tal maneira que as células vizinhas correspondem a duas combinações de valores das entradas, com apenas uma das entradas tendo seu valor alterado. Na Fig. 1-10(d) há um exemplo do mapa na descrição de um circuito combinatório.



a	b	c	s
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

$$s = f(a, b, c) = \bar{a}c + ab$$

c) EXPRESSÃO BOOLEANA

b) TABELA DE COMBINAÇÕES

bc \ a	0	1
00	0	0
01	1	0
11	1	1
10	0	1

d) MAPA DE KARNAUGH

$$TN(s) = [1100 \quad 1010]_2$$

e) TRANSFORMADA NUMÉRICA

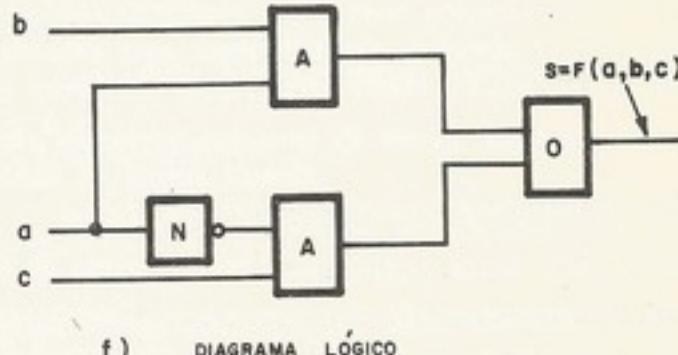


Figura 1-10. Formas de descrição de um mesmo circuito combinatório

Descrição pela transformada numérica⁽¹⁰⁾

Se considerarmos a tabela de combinação e supusermos que as combinações das entradas sejam sempre ordenadas na ordem crescente, apenas a coluna de saída descreverá o circuito. Se escrevermos essa coluna de baixo para cima como um número na base 2, esse número (transformada numérica) descreverá o circuito [Fig. 1-10(c)]. Essa descrição é recomendada quando se pretende analisar o circuito por meio de computadores.

Vamos estudar o seguinte problema: dado um circuito definido por uma certa descrição, determinar uma outra descrição.

As interconversões entre as descrições por tabelas da verdade, mapa de Karnaugh e transformada numérica são imediatas. Para convertermos a descrição através de expressão booleana para o mapa de Karnaugh ou tabela da verdade, basta calcularmos o valor da expressão para cada combinação dos valores das entradas.

A conversão mais delicada é a da tabela de combinações para a expressão booleana. É o que veremos a seguir.

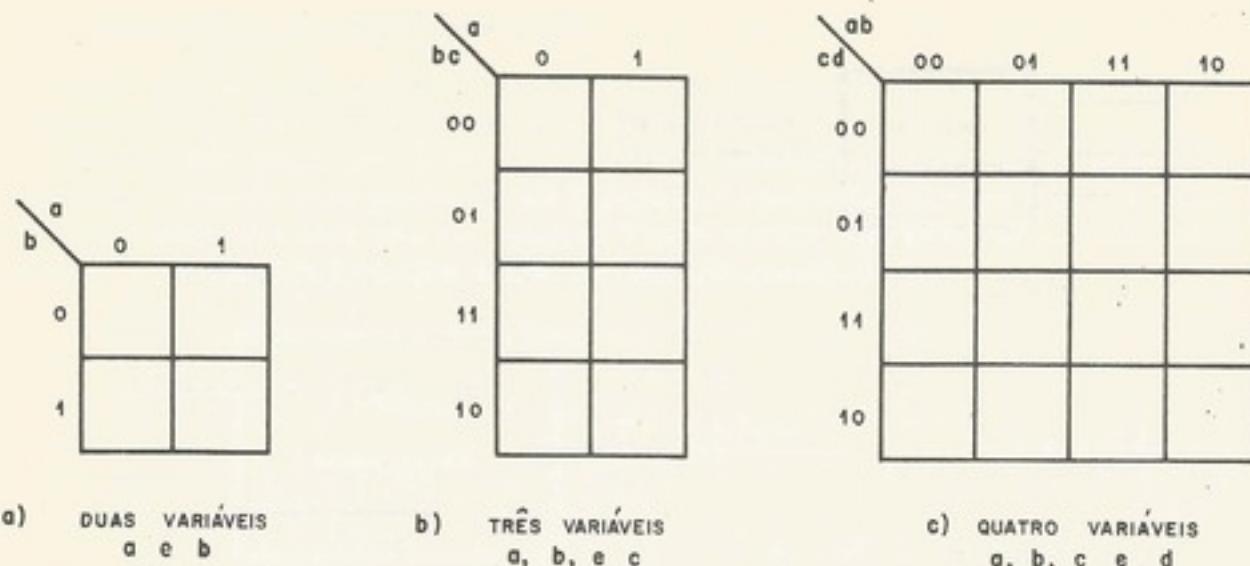


Figura 1-11. Mapas de Karnaugh

DEFINIÇÃO. Produto fundamental

Dada uma tabela da verdade, a cada combinação de valores das entradas, associamos um produto (*AND*) das variáveis de entrada, complementadas se o correspondente valor for "0" e não-complementadas se for "1". Esse produto é chamado de produto fundamental.

EXEMPLOS

entrada	a	b	c
0 1 1			

produto fundamental correspondente, $a' \cdot b \cdot c$;

entrada	a	b	c	d
1 0 0 1				

produto fundamental correspondente, $a \cdot b' \cdot c' \cdot d$.

DEFINIÇÃO. Soma canônica⁽⁹⁾

Dada uma tabela da verdade, a descrição equivalente por uma expressão booleana é dada pela soma dos produtos fundamentais correspondentes a cada combinação das entradas cuja saída seja "1".

EXEMPLO

Seja o circuito dado pela seguinte tabela da verdade:

a	b	c	s
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

A descrição pela expressão booleana é dada pela soma canônica, que é igual à soma de cinco produtos fundamentais:

$$s = a' \cdot b' \cdot c + a' \cdot b \cdot c' + a \cdot b' \cdot c + a \cdot b \cdot c' + a \cdot b \cdot c.$$

[Observação. Existe um método⁽⁹⁾ dual que nos leva a um produto de somas, cada soma correspondendo a uma linha com saída zero na tabela da verdade.]

c. Análise de circuitos combinatórios

Dado um circuito combinatório através de seu diagrama lógico, analisá-lo é encontrar a sua descrição por uma expressão booleana.

Isso é feito escrevendo as entradas do circuito. Ver...

d. Síntese de

O problema difícil. Como o a e⁽⁹⁾ da Bibliografia a tecnologia, bus...

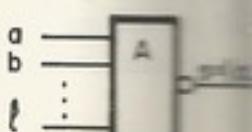
Custo (1). Embarque aquele que...

Custo (2). Embarque aquele que...

É claro que classe. Um bloco Antes de que são os mais tecnologia am...

NAND

É um bloco inversor na saída.



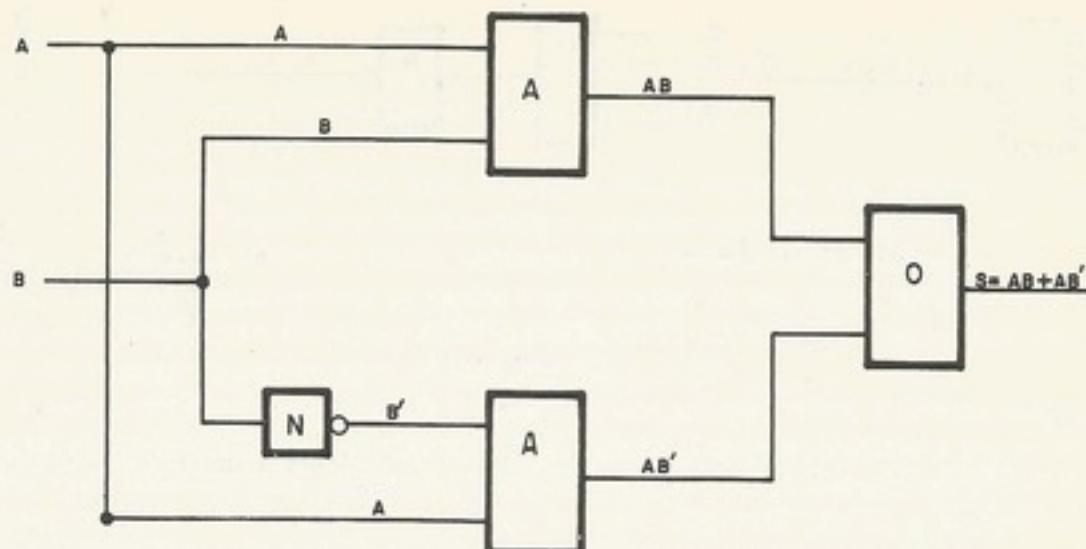


Figura 1-12. Exemplo de análise de um circuito combinatório

Isso é feito de um modo simples e metódico: basta partirmos das entradas e irmos escrevendo as expressões das saídas dos blocos lógicos até chegarmos à expressão de saída do circuito. Veja um exemplo na Fig. 1-12.

d. Síntese de circuitos combinatórios

O problema da síntese de circuitos combinatórios é um problema muito delicado e difícil. Como o abordaremos com superficialidade, recomendamos aos leitores os itens⁽⁷⁾ e⁽⁹⁾ da Bibliografia. Esse problema envolve uma variável muito delicada e imprevisível, a *tecnologia*, buscando um ótimo que envolve o conceito de *custo*. Vamos definir custo.

Custo (1). Entre dois circuitos com diferentes números de blocos lógicos, será mais barato aquele que tiver o menor número de blocos lógicos.

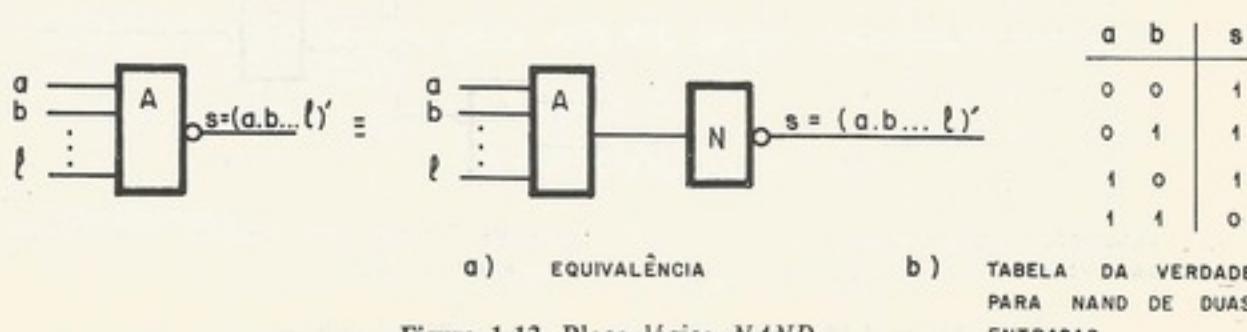
Custo (2). Entre dois circuitos de mesmo número de blocos lógicos, será mais barato aquele que tiver menor número total de entradas dos blocos.

É claro que esse conceito de custo exige a comparação de blocos lógicos de mesma classe. Um bloco *XOR* custa mais caro que um bloco *AND*.

Antes de tratarmos do problema da síntese, vamos definir dois novos blocos lógicos, que são os mais usados em circuitos lógicos, dadas as suas vantagens de custo e atraso na tecnologia atual: os blocos *NAND* e *NOR*.

NAND

É um bloco, cujo símbolo é visto na Fig. 1-13, equivalente a um bloco *AND* com um inversor na saída.

Figura 1-13. Bloco lógico *NAND*

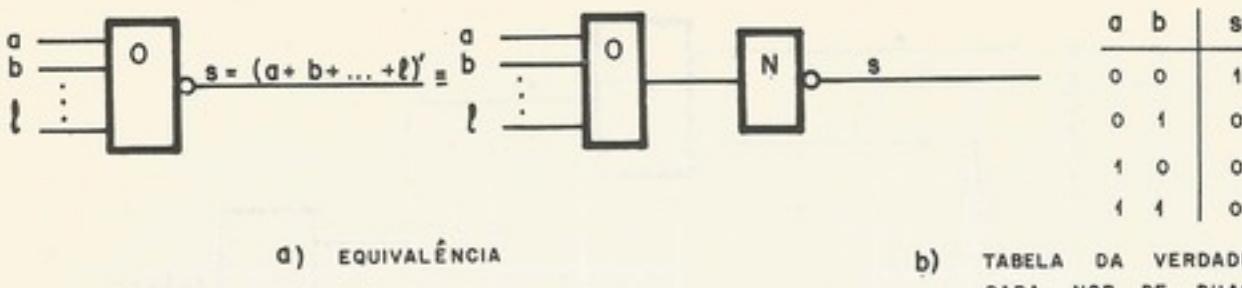


Figura 1-14. Bloco lógico NOR

NOR

É um bloco, cujo símbolo é visto na Fig. 1-14, equivalente a um bloco *OR* com um inversor na saída.

Uma técnica de síntese muito usada, pela sua simplicidade é aquela já citada: parte-se de sua expressão booleana e, utilizando-se os teoremas, busca-se uma expressão mais simples, que é sintetizada pelo processo inverso ao de análise.

EXEMPLO

Sintetizar o circuito dado pela tabela da verdade da Fig. 1-15(a).

Numa primeira etapa, escrevemos a sua expressão booleana:

$$s = abcd + abcd' + abc'd + abc'd' + ab'cd + ab'cd'.$$

A seguir, tentamos obter uma expressão mais simples cuja síntese deverá ser mais barata. Então, simplificando

$$\begin{aligned} s &= abc + abc' + ab'c, \\ s &= ab + ab'c, \\ s &= a(b + b'c), \\ s &= a(b + c). \end{aligned}$$

O circuito sintetizado com a expressão mais simples é visto na Fig. 1-15(b).

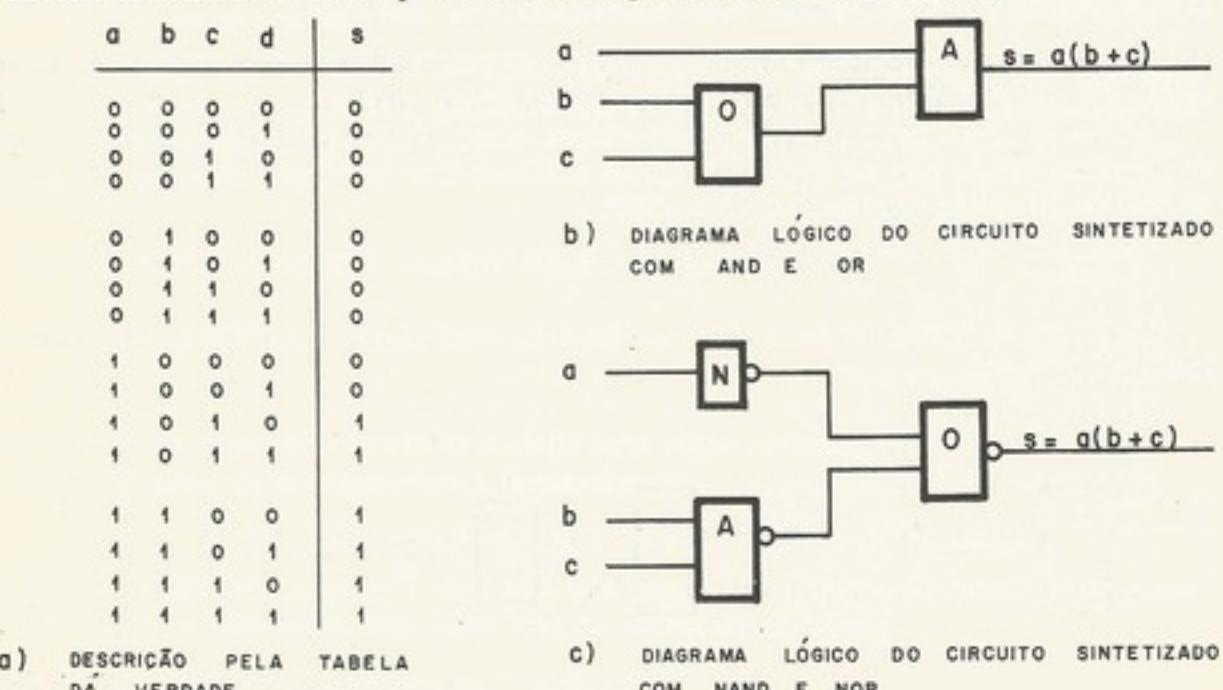


Figura 1-15. Circuito detetor de código BCD inválido

Esse método é conhecido como síntese por minimização ao mínimo caminho seguido. Na realidade, o resultado é sempre o mesmo.

Para a síntese de diferenças no final da transformação, podemos invertirmos a saída do circuito do exemplo.

Como o problema é uma tabela grande, a síntese por minimização é economicamente viável.

Um caso particular é o circuito combinacional com componentes ativos (NAND-NAND). Um circuito combinacional com componentes ativos reside na velocidade de operação de um circuito combinacional com componentes ativos (NAND-NAND).

Como o resultado da expressão é uma função barata, que é fácil de implementar, para encontrá-la.

e. Determinação da expressão booleana

Determinação da expressão booleana

Dado o circuito, determinar a expressão booleana

soma canônica.

e, eventualmente,

simplificando a expressão.

Aqui contém um teorema irreductível, numérico, que permite obter a soma canônica.

EXEMPLO 1

Projetar o circuito para as combinações da Fig. 1-15(a).

Vamos considerar o caso mais simples.

Simplificando a expressão.

O circuito minimizado é

a	b	s
0	0	1
0	1	0
1	0	0
1	1	0

TABELA DA VERDADE
DO NOR DE DUAS
SAÍDAS

circuito OR com um
único integrador: parte-se
de um exemplo mais sim-

plificando a saída.

$s = a(b + c)$

CIRCUITO SINTETIZADO

$s = a(b + c)$

CIRCUITO SINTETIZADO

Esse método, que exige muita arte do projetista, é um processo que não leva necessariamente ao mínimo, podendo-se obter expressões irreduzíveis de alto custo, dependendo do caminho seguido na simplificação. O problema, portanto, é saber se o circuito obtido é, realmente, o mais barato.

Para a síntese com blocos tipo *NAND* e *NOR*, segue-se um processo idêntico, com diferença no final. Depois que desenhamos o circuito com blocos *AND* e *OR*, começamos a transformá-lo em *NAND* e *NOR* utilizando os teoremas de De Morgan. Com esses teoremas, podemos complementar as saídas e as entradas de um bloco *AND* ou *OR* desde que invertamos a operação (troca-se *AND* por *OR*, e vice-versa). Na Fig. 1-15(c) vemos o circuito do exemplo anterior com blocos *NAND* e *NOR*.

Como o problema com *NAND* e *NOR* não tem solução viável, é prática comum usar-se uma tabela publicada por Hellerman⁽¹¹⁾ que fornece o circuito mínimo a partir de sua descrição pela transformada numérica. Com a evolução da tecnologia, obteve-se um bloco, economicamente viável com duas saídas, o bloco *OR/NOR* (*ECL*), e Baugh⁽¹²⁾ publicou uma tabela análoga para circuitos sintetizados com esse tipo de bloco.

Um caso particular de síntese que tem solução é a síntese em dois níveis *AND-OR* ou *NAND-NAND*. Esse problema foi estudado principalmente na segunda geração, pois o circuito combinatório em dois níveis *AND-OR* [Fig. 1-16(a)] podia ser realizado sem elementos ativos [Fig. 1-16(b)], o que era um grande fator de economia. Sua importância atual reside na velocidade; com circuito *NAND-NAND* [Fig. 1-16(c)] implementa-se qualquer circuito combinatório com dois níveis de atraso e, quando implementado com a técnica *DOT-OR* (veja à frente), consegue-se um atraso de apenas um nível.

Como o leitor já deve ter observado, o circuito em dois níveis, se analisado, terá como expressão uma soma de produtos. Então o problema é obter uma soma de produtos mais barata, que chamaremos de *soma mínima*. Vamos tratar das duas maneiras mais usadas para encontrar-se a soma mínima: pela simplificação booleana e pelo mapa de Karnaugh.

e. Determinação da soma mínima

Determinação da soma mínima por simplificação booleana

Dado o circuito combinatório por qualquer uma das descrições, escreveremos a sua soma canônica. A seguir, utilizando os teoremas

$$(T.16) \quad A \cdot B + A \cdot B' = A,$$

$$(T.21) \quad A \cdot B + A' \cdot C + B \cdot C = A \cdot B + A' \cdot C$$

e, eventualmente, o teorema

$$(T.18) \quad A + A' \cdot B = A + B$$

simplificamos a soma canônica até chegarmos à soma mínima.

Aqui continua o mesmo problema anterior: o projetista, após chegar a uma soma irreduzível, nunca sabe se é a mínima. Seguindo caminhos diferentes na simplificação, chega-se a somas irreduzíveis diferentes.

EXEMPLO 1

Projetar o circuito, em dois níveis *AND-OR*, mínimo, caracterizado pela tabela de combinações da Fig. 1-17(a).

Vamos escrever a soma canônica

$$s = a'b'c' + a'b'c + ab'c + abc.$$

Simplificando,

$$s = a'b' + ac \quad (\text{soma mínima}).$$

O circuito mínimo, em dois níveis *AND-OR*, pode ser visto na Fig. 1-17(b).

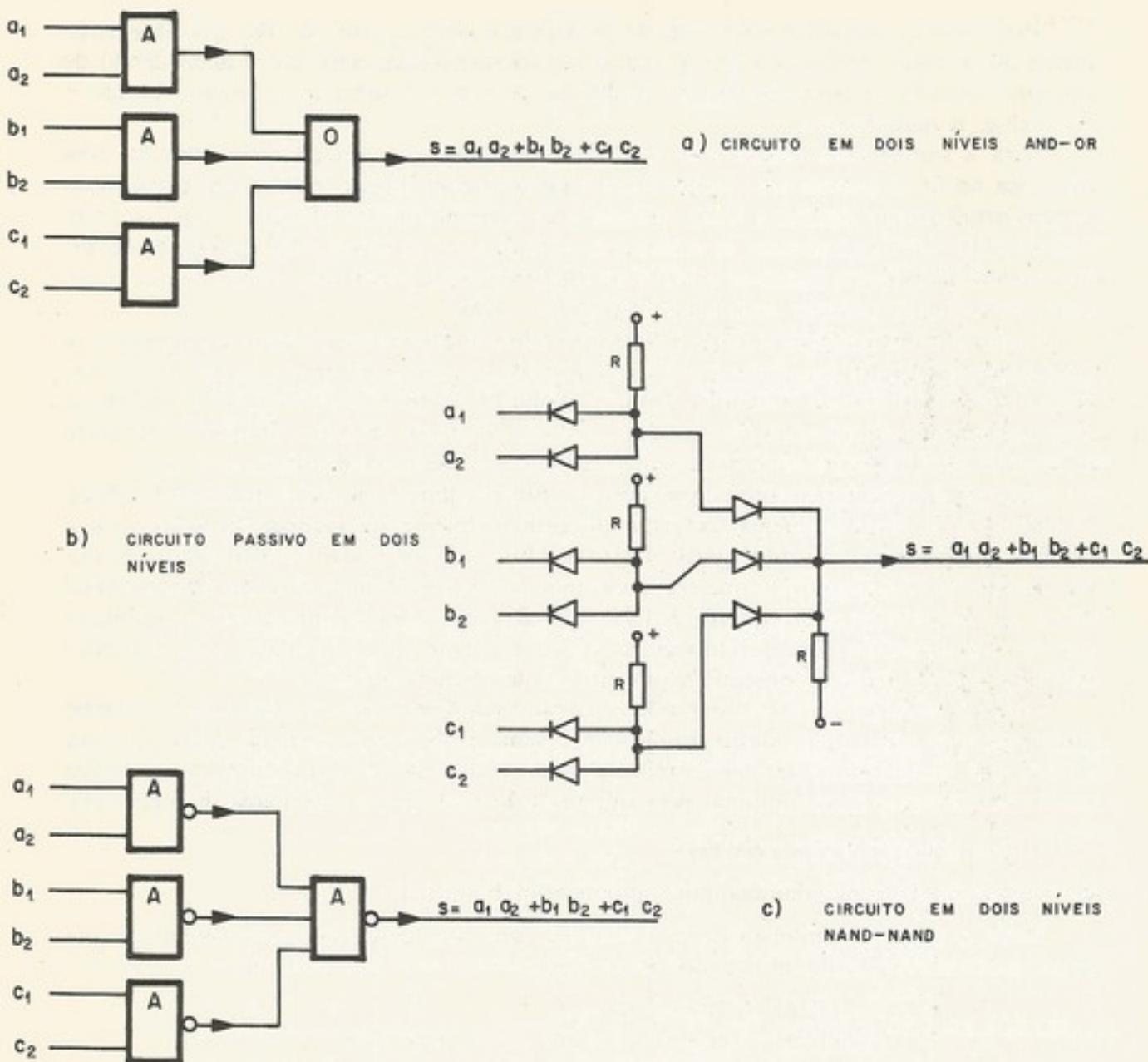


Figura 1-16. Várias implementações do mesmo circuito em dois níveis

a	b	c	s
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

a) TABELA DE COMBINAÇÕES

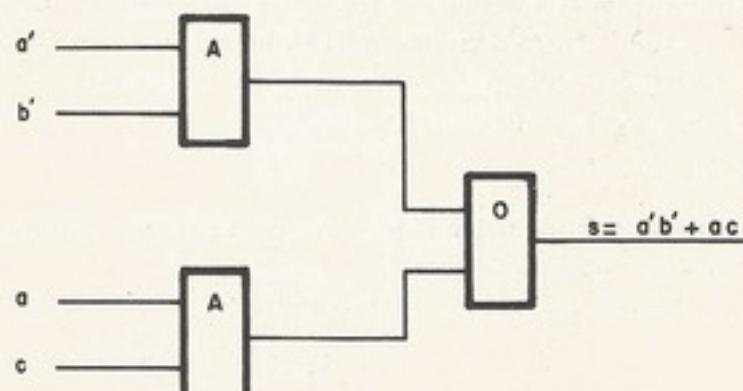


Figura 1-17. Circuito combinatório do Exemplo 1

EXEMPLO 2

Vamos ilustrar o problema de caminhos diferentes levarem a somas irreductíveis diferentes.

Projetar o circuito mínimo em dois níveis *NAND-NAND*, dado pela expressão booleana

$$s = a'b' + b'c' + ac + ab + bc'.$$

Primeira maneira. Simplificando,

$$s = a'b' + \underbrace{b'c + ac + ab}_{(b'c + ab)} + bc',$$

$$s = a'b' + b'c + ab + bc' \quad (\text{a soma irreductível não é mínima}).$$

Segunda maneira. Simplificando por outro caminho,

$$s = \underbrace{a'b' + b'c + ac}_{(a'b' + ac)} + ab + bc'$$

$$s = a'b' + \underbrace{ac + ab + bc'}_{(ac + bc')}$$

$$s = a'b' + ac + bc' \quad (\text{a soma irreductível é mínima}).$$

O circuito mínimo é mostrado na Fig. 1-18.

Determinação da soma mínima pelo mapa de Karnaugh

Vamos escrever a soma mínima da função dada pelo mapa da Fig. 1-19, para ilustrar as definições que seguem. A soma canônica será

$$s = \underbrace{a'b'c'd' + a'b'c'd}_{(\text{T.16}) - a'b'c'} + \underbrace{abc'd + abcd}_{(\text{T.16}) - abd} + \underbrace{ab'c'd + ab'cd}_{(\text{T.16}) - ab'd}$$

$$s = a'b'c' + \underbrace{abd + ab'd}_{(\text{T.16}) - ad},$$

$$s = a'b'c' + ad.$$

Olhando para a expressão da soma mínima, vemos que os produtos fundamentais correspondentes a "1" vizinhos se agrupam pelo (T.16). A técnica de determinação da soma mínima pelo mapa irá explorar essa característica.

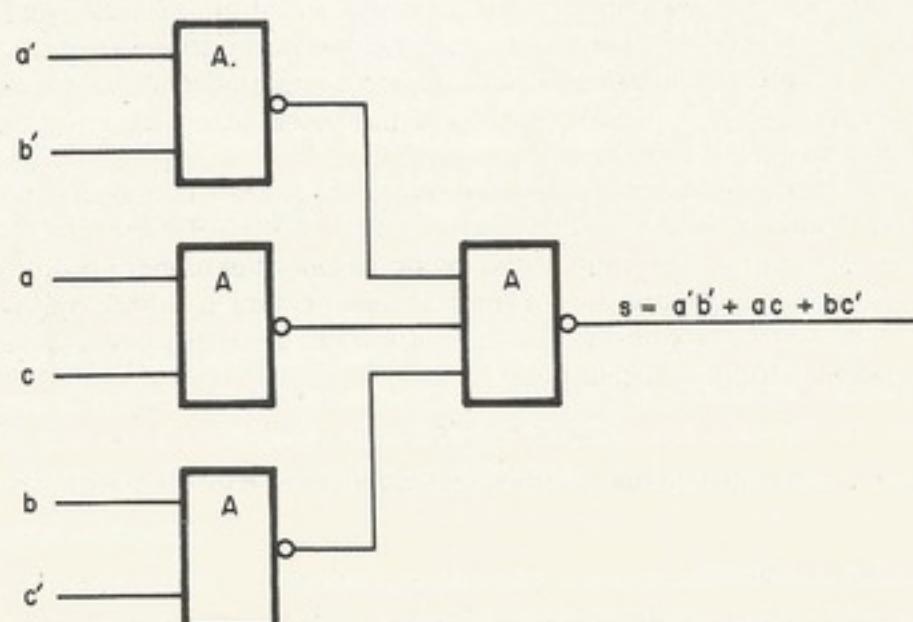


Figura 1-18. Circuito mínimo em dois níveis *NAND-NAND* do Exemplo 2

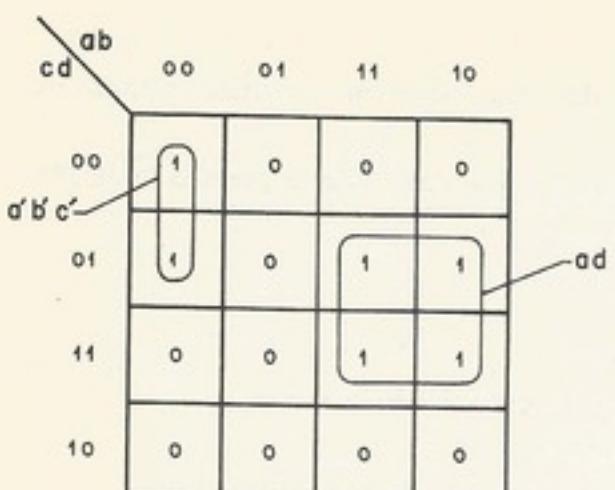


Figura 1-19. Prime implicant no mapa de Karnaugh

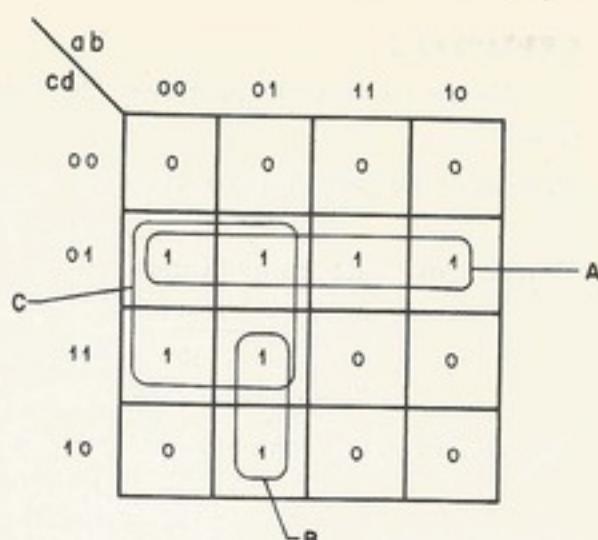


Figura 1-20. Mapa de Karnaugh do Exemplo 3

DEFINIÇÃO. Prime implicant

Dado um circuito combinatório descrito pelo mapa de Karnaugh, agrupamos as casas com "1" nos maiores grupos retangulares de tamanho 2^n . A cada grupo desses associamos um produto das variáveis (complementadas ou não) que pode ser obtido de soma canônica de todos os "1" do grupo, simplificados pelo (T.16). Esse produto é chamado de *prime implicant* (*p.i.*) da função. No exemplo anterior temos dois *prime implicants* (observe que está assinalado na Fig. 1-19):

$$a'b'c' \text{ e } ad$$

EXEMPLO 3

No mapa de Karnaugh da Fig. 1-20, indicamos três grupos retangulares com 2² células preenchidas com "1". Portanto essa função tem três *prime implicants*, que escreveremos a seguir.

$$p.i. A = a'b'c'd + a'bc'd + abc'd + ab'c'd = a'c'd + ac'd; \text{ portanto } p.i. A = c'd.$$

Na determinação dos *prime implicants* *B* e *C*, vamos mostrar a técnica usual, escrevendo diretamente do mapa; casas vizinhas são caracterizadas pelo fato de apenas uma variável mudar. Portanto a expressão do *prime implicant* é obtida multiplicando-se as variáveis constantes para todas as casas circundadas. (Variável complementada se o valor da entrada for "0", e não-complementada se for "1").

p.i. B. Para as casas desse grupo, as variáveis *a*, *b* e *c* são constantes, pois apenas a variável *d* muda. Como nesse caso *a* = 0, *b* = 1 e *c* = 1, o *p.i. B* é escrito *a'bc*.

p.i. C. Nesse caso, como temos um grupo maior, teremos um número menor de variáveis de entrada constantes e, portanto, uma expressão menor para o *prime implicant*.

Então, as variáveis constantes são *a* = 0 e *d* = 1. Logo, o *p.i. C* é escrito *a'd*.

Concluindo, os três *prime implicants* são

$$c'd, a'bc \text{ e } a'd.$$

Vamos agora conceituar soma mínima sem rigor, pois não é nosso propósito entrar em detalhes.

DEFINIÇÃO. Soma mínima

A soma mínima é a soma do menor número de *prime implicants* que cobre completamente a função. Admitimos como intuitivo o conceito de "cobrir". Então a técnica de deter-

minação da soma mínima consiste em marcar todos os *prime-implicants* e, a seguir, escolher o conjunto mínimo. É claro que se começa separando os essenciais (aqueles que cobrem *sozinhos* uma casa com "1") para, depois, escolherem-se os não-essenciais, levando-se em conta o conceito de custo.

EXEMPLO 4

"Implementar em dois níveis o circuito dado pelo mapa de Karnaugh da Fig. 1-21(a)."

Existem três *prime implicants* *A*, *B* e *C* que estão marcados no mapa, dos quais *A* e *C* são essenciais, pois cobrem uma casa (marcada com *) sozinhos. Como os dois são suficientes para cobrir toda a função, a soma mínima é dada pela soma deles:

$$s = p.i. A + p.i. C = ab' + a'c$$

O circuito mínimo pode ser visto na Fig. 1-21(b).

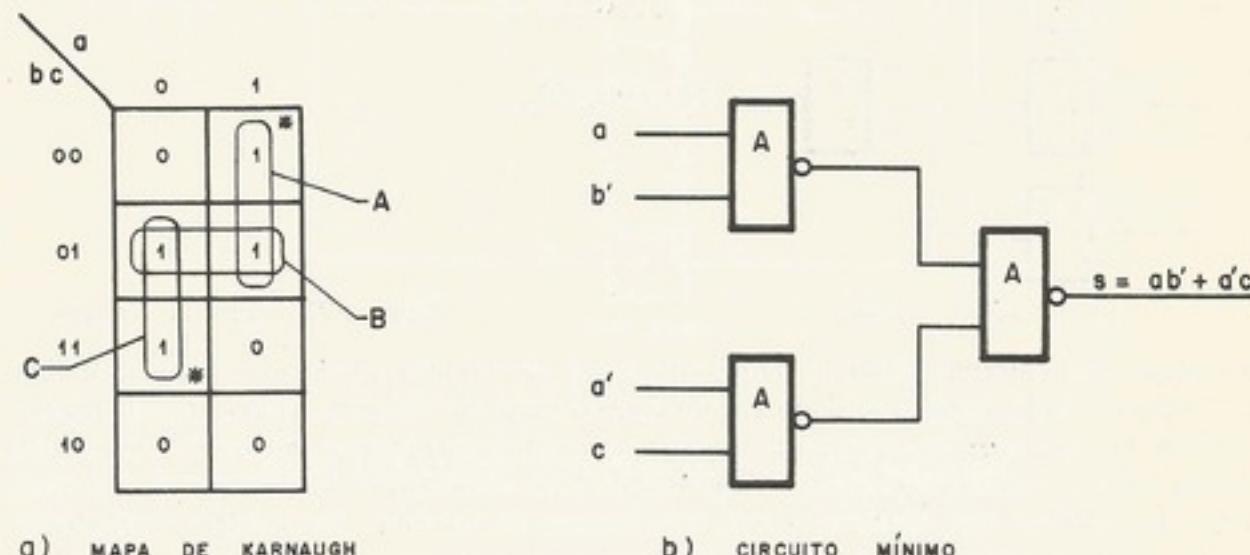


Figura 1-21. Figura do Exemplo 4

EXEMPLO 5

Implementar em dois níveis o circuito dado pelo mapa de Karnaugh da Fig. 1-22(a).

Marcados os *prime implicants*, vemos que apenas os *A* e *E* são essenciais, ou seja, necessariamente estarão presentes na soma mínima. Separando-os, para facilitar a escolha dos não-essenciais, obtemos o mapa visto na Fig. 1-22(b) (na escolha dos não-essenciais, deve-se levar em conta que um *prime implicant* envolvendo uma quantidade maior de "1" é mais barato, pois sua expressão é mais simples). Vemos na Fig. 1-22(b) que falta apenas cobrirmos dois "1" e a escolha mais barata é o *p.i. C*, pois ele cobre as duas casas que faltam. Então a soma mínima é dada pela soma dos *prime implicants* essenciais (*A* e *E*) e o *p.i. C*,

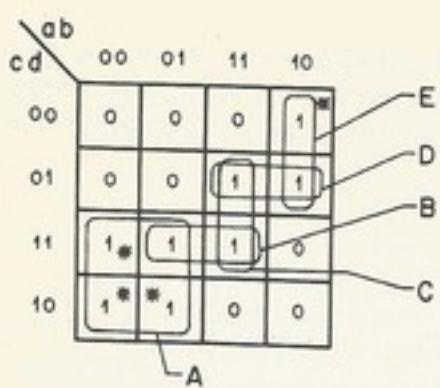
$$\begin{aligned} s &= p.i. A + p.i. E + p.i. C, \\ s &= a'c + ab'c' + abd \quad (\text{soma mínima}). \end{aligned}$$

O circuito mínimo pode ser visto na Fig. 1-22(c).

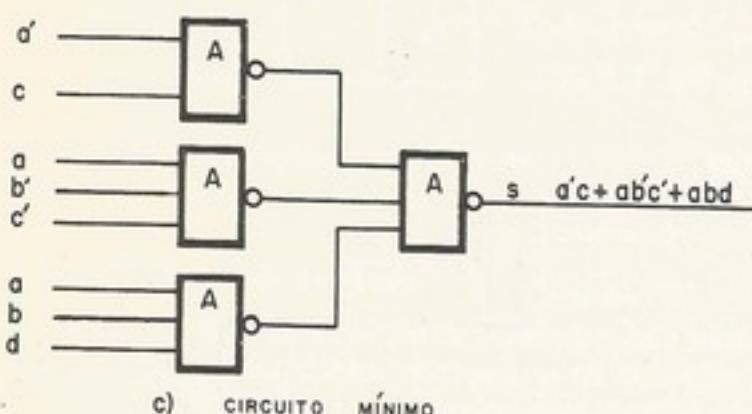
Acreditamos que essa segunda técnica de determinação da soma mínima seja a mais recomendada, por dar ao projetista uma boa visualização das alternativas, permitindo-lhe assegurar-se de que a escolha é realmente a melhor.

f. Circuitos seqüenciais

Se o circuito lógico tiver memória, então não bastará saber-se o estado das entradas atuais para se determinarem as saídas; será necessário, também, conhecer-se o *estado interno* do circuito. Circuitos desse tipo são chamados *circuitos seqüenciais*, pois a seqüência das

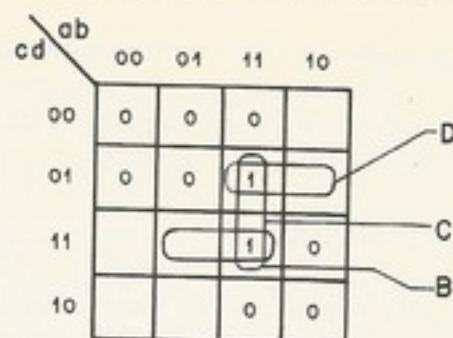


a) MAPA DE KARNAUGH



c) CIRCUITO MÍNIMO

projeto de computadores digitais



b) MAPA DE KARNAUGH SEM OS "PRIME IMPLICANTS" ESSENCIAIS

Figura 1-22. Figura do Exemplo 5

mudanças das entradas influem no comportamento do circuito. O circuito seqüencial tem um estado interno que é guardado em unidades funcionais de armazenamento, chamadas *flip-flop*. O *flip-flop* é uma memória que armazena dois estados, ou “0” ou “1”. O estado interno de um circuito seqüencial é determinado pela combinação dos estados de um conjunto de *flip-flops* que constitui a memória interna do circuito. Um modelo matemático de um circuito seqüencial aparece na Fig. 1-23.

É importante observar que o circuito seqüencial tem realimentação, isto é, parte das saídas são reinjetadas na entrada. O circuito combinatório da Fig. 1-23 tem dois conjuntos de saídas, o conjunto (Z_1, Z_2, \dots, Z_m) , que são as saídas do circuito seqüencial, e o conjunto (y_1, y_2, \dots, y_p) , que determina o estado seguinte do sistema (Y_1, Y_2, \dots, Y_p) . Então podemos dizer, em outras palavras, que o circuito seqüencial é caracterizado por uma memória que guarda o estado do sistema, e por um circuito combinatório cujas entradas são as entradas primárias do circuito seqüencial e o estado interno corrente. O circuito seqüencial fornece como saídas a saída do circuito combinatório e o estado seguinte do sistema. O conceito “seguinte” nesse caso, está relacionado com os atrasos intrínsecos dos blocos lógicos, mas o leitor, para melhor entender o funcionamento dos circuitos seqüenciais, pode imaginar como um tempo finito e determinado. Portanto vemos que, para conhecermos a saída de um circuito seqüencial, precisamos conhecer o valor das entradas e o estado interno, e não simplesmente as entradas, como era o caso dos circuitos combinatórios.

Existe uma classe de circuitos seqüenciais chamada “varredura finita” cuja saída poderá ser prevista, mesmo desconhecendo-se o valor do estado interno, se soubermos os n últimos valores das entradas. Podemos dizer, então, que esses n últimos valores das entradas determinam o estado atual.

g. Descrição de um circuito seqüencial

Como já vimos, um circuito seqüencial ficará definido se soubermos, para cada combinação dos valores do estado interno e da entrada, qual será a saída e o estado seguinte. As duas descrições que veremos a seguir têm essa característica. Elas são uma espécie de

tabela onde, dadas as entradas e do estado seguem, exprimem o resultado.

Descrição por tabelas.

O diagrama de bloco mostra que, para determinar o novo, fazemos combinações entre o valor das entradas e o valor do estado interno determinada.

A correspondência é feita de forma que, dado um certo valor das entradas a, b, c e o valor do estado interno x , saímos y e x' (pelo primeiro mapeamento) e, a partir disso, obtemos o novo valor do estado interno x' .

Por exemplo, se $x = 0000$ e $a, b, c = 000$, temos $y = 000$ e $x' = 0000$. A saída a é a nova saída a' e a saída b é a nova saída b' . A saída c é a nova saída c' . A saída x é a nova saída x' . A saída x' é uma parte da saída y . O novo valor do estado interno é obtido a partir da saída y e do novo valor do estado interno.

Observe, portanto, que o novo valor do estado interno é obtido a partir da saída y e do novo valor do estado interno.

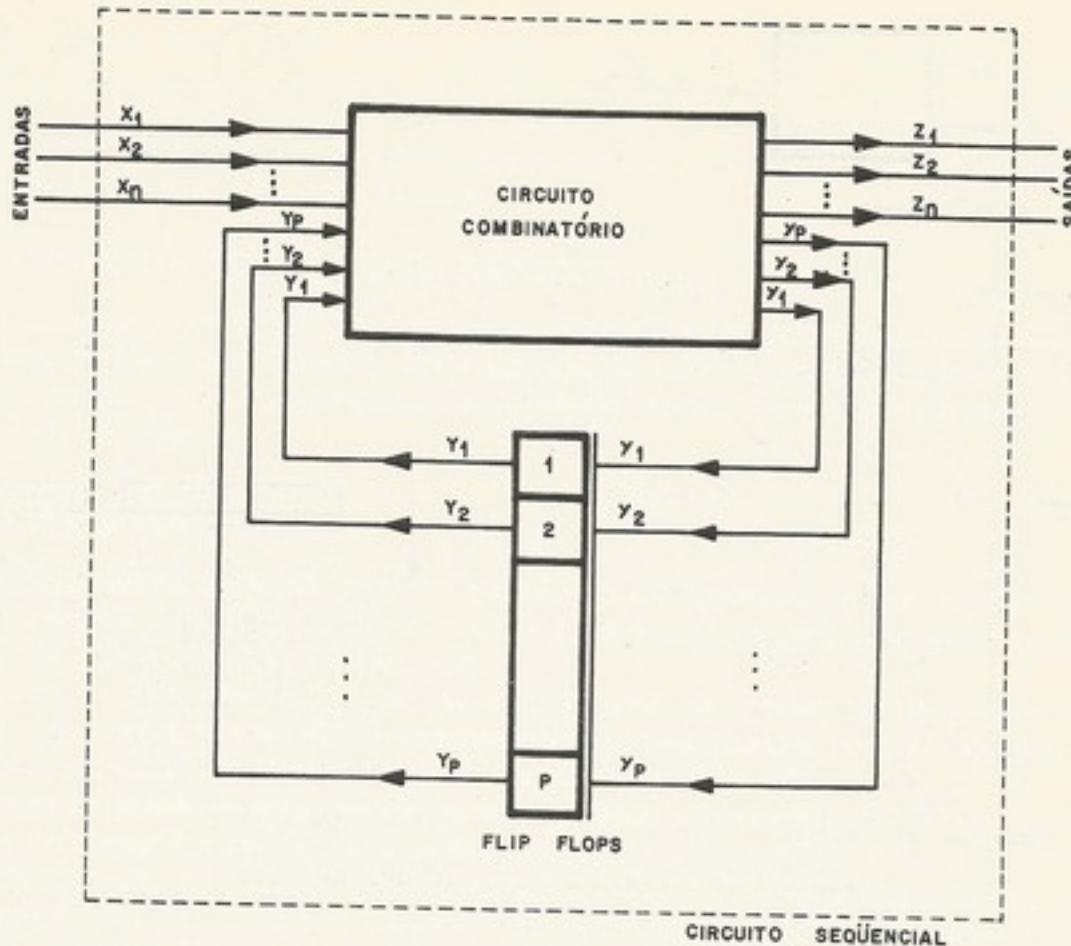


Figura 1-23. Modelo matemático de um circuito seqüencial

tabela onde, dado um certo estado e o valor da entrada, são fornecidos os valores da saída e do estado seguinte. Em síntese, podemos dizer que as descrições dos circuitos seqüenciais exprimem o funcionamento do circuito combinatório que faz parte dele.

Descrição pelo diagrama de estado

O diagrama de estado é um gráfico com tantos nós quantos forem os estados e, a cada nó, fazemos corresponder um estado. Ligando-se os nós, existem ramos que são determinados pelo valor das entradas. Esses ramos mostram o estado seguinte do circuito com aquela determinada entrada e, sobre o ramo, marcamos o correspondente valor da saída.

A correspondente descrição pelo diagrama de estado pode ser vista na Fig. 1-24(b). Dado um certo estado interno, situamos um dos três nós possíveis (existem três estados, a , b , e c) e, sabendo qual a entrada, localizamos um dos possíveis ramos que saem do nó (pelo primeiro número dos escritos no ramo). O final do ramo vai para o estado seguinte e a correspondente saída está indicada no segundo número escrito no ramo (depois de vírgula).

Por exemplo, vamos supor que estejamos no estado a e que a entrada seja "01" ($X_1 = 0$, $X_2 = 1$); como vemos na Fig. 1-24(b) (em negrito), o ramo que sai do nó a com a entrada "01" volta para a e a saída é "1". Portanto, com essa entrada, o estado seguinte é o mesmo e a saída permanece igual a "1". Admitamos agora que X_1 mude para "1", ou seja, a nova entrada passe a ser "11" (maior que a anterior). Então identificamos o ramo que sai de a com entrada "11". Ele indica que o estado seguinte será c (veja, na Fig. 1-24(c), que é uma parte da Fig. 1-24(b), ou seja, se estivermos no estado a e a entrada passa a ser "11" o estado seguinte será c e a saída será "1"). Depois disso, se mantivermos a entrada "11", o circuito se manterá no estado c com a saída "1".

Observe, portanto, que o diagrama de estado descreve completamente o circuito.

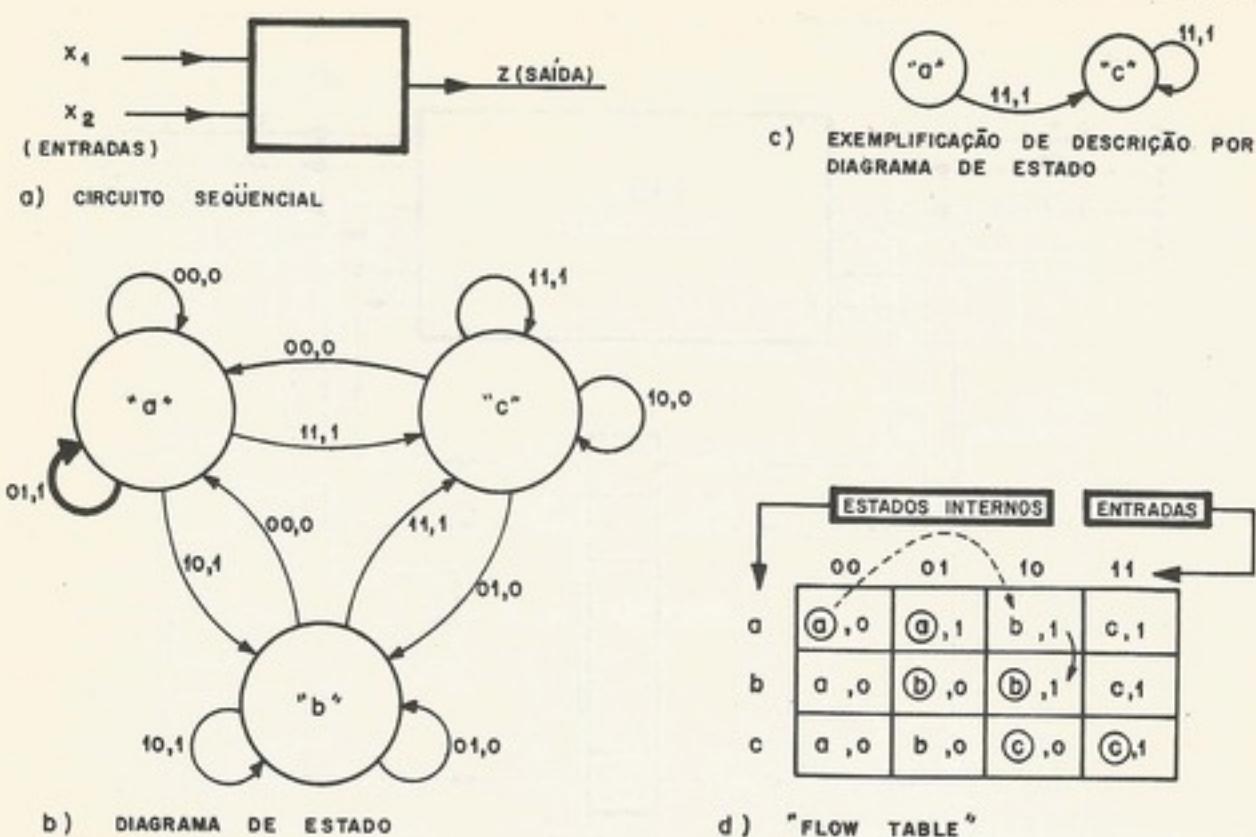


Figura 1-24. Descrições de um circuito seqüencial

Descrição pela "flow table"

O comportamento de um circuito seqüencial pode ser descrito por uma *flow table* (tabela de estados), uma tabela na qual as colunas são os valores das entradas e as linhas são os estados internos. Cada lugar da tabela corresponde à combinação dos dois componentes, a entrada e o estado interno. A essa combinação, daremos o nome de estado total do circuito seqüencial. A tabela é preenchida colocando-se o estado seguinte e a saída no lugar correspondente a cada estado total da tabela. Cada lugar na tabela, então, tem dois componentes, o estado interno seguinte e o valor da saída.

A *flow table* do circuito da Fig. 1-24(a) aparece na Fig. 1-24(d). O primeiro componente na tabela é o estado interno (a , b e c) e o segundo componente é a saída, ou "0" ou "1". O circuito do exemplo tem três estados internos. Então a *flow table* tem três linhas. Começando na primeira linha e primeira coluna, o estado total é $(00, a)$ significando que o valor da entrada é "00" (isto é, $X_1 = 0$, $X_2 = 0$) e o estado interno é " a ". A saída é "0". Supondo-se que a entrada mude de "00" para "10" (isto é, o sinal X_1 mude para "1"), o estado total será $(10, a)$ e notamos, na tabela da Fig. 1-24(d), que o estado seguinte será b e não a . Quando o estado interno seguinte não for igual ao atual, o estado total será chamado de *transitório* ou *instável*. Isso porque o circuito irá mudar de estado interno. Quando o estado interno mudar de a para b , estaremos no estado total $(10, b)$. O próximo estado interno para esse estado total é b , igual ao atual. Nesse caso, o estado total é chamado de *estável*. É costume, na *flow table*, indicar os estados estáveis por uma bolinha no próximo estado interno.

O comportamento do circuito, para esse exemplo está indicado na Fig. 1-24(d) pelas setas tracejadas. As setas indicam o movimento do chamado *ponto de operação (operating point)* do circuito; começando em $(00, a)$ com saída "0" ele muda para $(10, b)$, que é estável. A saída do circuito, que estava no estado "0", agora está no estado "1". Em termos gerais, uma mudança de entradas é um movimento horizontal do ponto de operação na *flow table*. Esse movimento horizontal provoca um movimento vertical quando o ponto de operação cai num estado instável.

Nesse último ...
table ou pelo dia...
do circuito, nem
circuitos seqüen...
(7), (9) e (12) da B

1.4 TECNOLOGIA

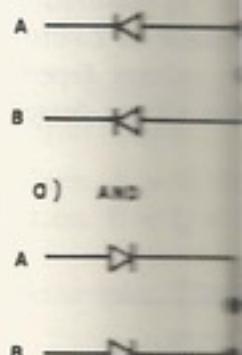
Estudaremos...
blocos lógicos. N...
dos assuntos. As
(15) e (16) da Bim

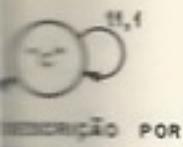
a. Circuitos

Um dispositivo...
usado, é o diodo
AND e *OR*, com

Na Fig. 1-22...
alta (mais possivel...
verificar o que)

Sempre há p...
circuitos *AND* se...
-se indefinidamente
circuitos com diodo...
com uma válvula...
e inversão é impo





Nesse último item não foi nossa intenção indicar *como* chegar à descrição pela *flow table* ou pelo diagrama de estados, a partir da descrição verbal do comportamento desejado do circuito, nem a partir do circuito. Não tratamos também do problema da síntese de circuitos seqüenciais. Maiores detalhes sobre o assunto poderão ser encontrados nos itens ⁽⁷⁾, ⁽⁹⁾ e ⁽²³⁾ da Bibliografia deste capítulo.

1.4 TECNOLOGIA: TRANSISTORES E DÍODOS

Estudaremos nesta seção e na próxima as técnicas usadas na implementação física dos blocos lógicos. Mantendo o espírito deste capítulo, ou seja dar apenas uma breve descrição dos assuntos. Ao leitor que desconheça o problema, recomendamos as referências^{(13), (14), (15) e (16)} da Bibliografia.

a. Circuitos lógicos com diodos

Um dispositivo usado na primeira geração de computadores, e que continua sendo usado, é o diodo. Junto com resistores, o diodo é usado para implementar os blocos lógicos *AND* e *OR*, como indicado na Fig. 1-25.

Na Fig. 1-25, o circuito (a) será um *AND* somente se o valor "1" representar tensão alta (mais positiva) e o valor "0" representar a tensão baixa (menos positiva). O leitor pode verificar o que aconteceria se os códigos fossem invertidos.

Sempre há perdas de níveis em circuitos como esses, sem amplificação. É comum termos circuitos *AND* seguidos por *OR* como indicado na Fig. 1-25(c), mas não é viável encadearmos indefinidamente os *gates AND* e *OR* sem amplificação. Na primeira geração, os circuitos com diodos, sintetizando os blocos *AND* e *OR*, eram seguidos de um inversor montado com uma válvula a vácuo para amplificar os sinais. Atualmente, a função de amplificação e inversão é feita com transistores.

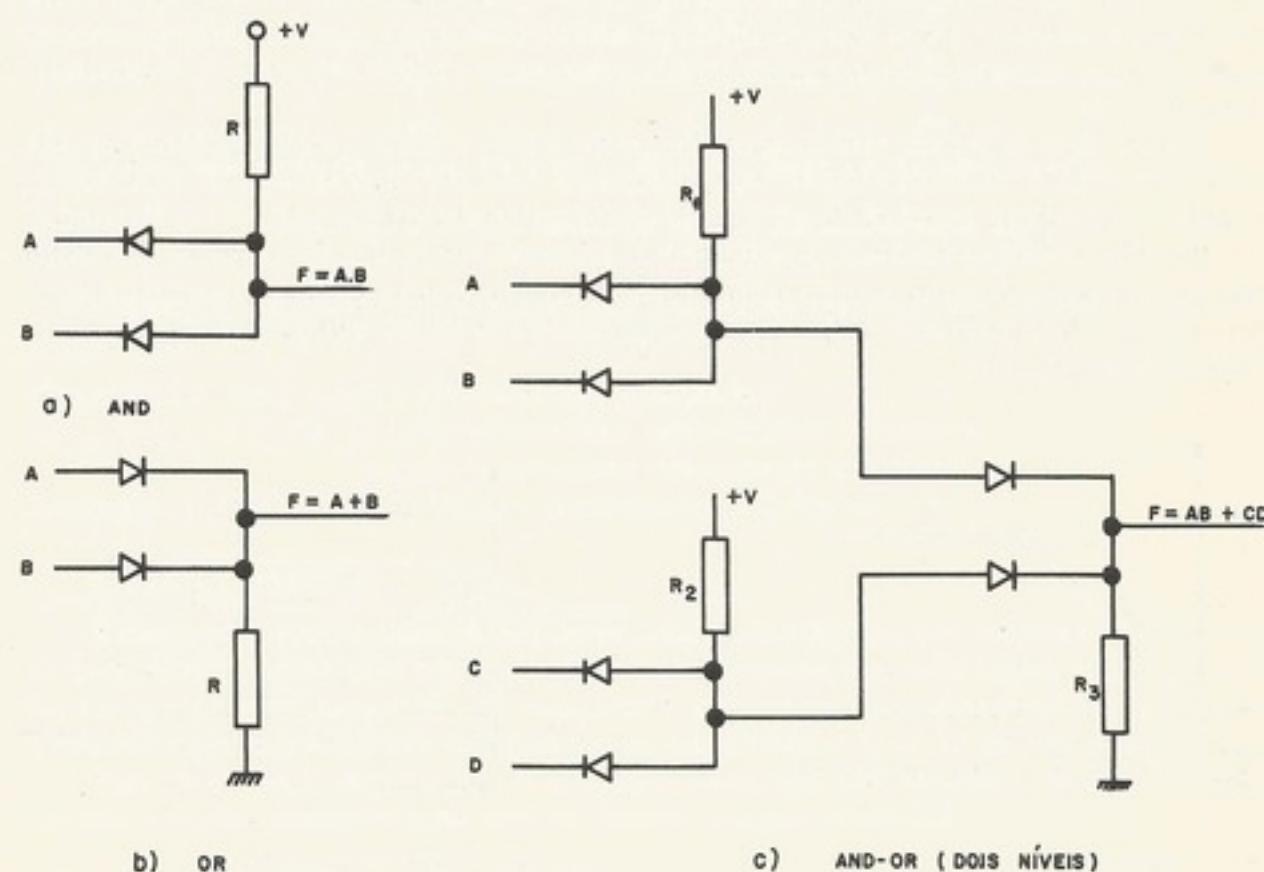


Figura 1-25. Circuitos lógicos com diodos

b. O transistor como chave

Estudo do regime

O transistor pode ser usado como chave, conforme visto na Fig. 1-26. Quando injetamos uma corrente, na base do transistor, suficientemente grande, a chave fecha, isto é, o ponto *C* é ligado para a terra. Quando a corrente de base é nula, a chave fica aberta, ou seja, o ponto *C* desligado da terra.

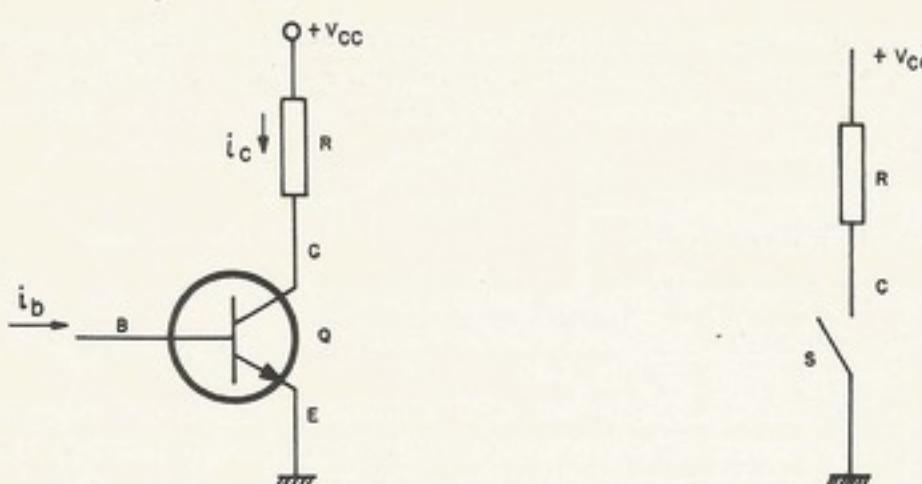


Figura 1-26. O transistor como chave

O transistor tem seu funcionamento caracterizado por três regiões (Fig. 1-27). Quando a corrente de base é nula, o transistor fica na *região de corte*, isto é, fica aberto. À medida que vamos aumentando a corrente da base (*i_b*), a corrente do coletor (*i_c = βi_b*) vai aumentando, caracterizando a *região ativa*, e o potencial do ponto *C* vai caindo, aproximando-se da terra. Até que se chega a um ponto onde *i_c · R = V_{cc}*, ou seja, a corrente do coletor é suficientemente grande a ponto de a queda de tensão no resistor ser igual à tensão de alimentação. Chegou-se à *região de saturação* do transistor, onde a tensão no ponto *C* é aproximadamente nula.

Então, para se operar com o transistor como chave, deve-se passar rapidamente pela região ativa. Se o transistor estiver cortado, deve-se saturá-lo imediatamente, passando-se o mais rapidamente possível da situação *i_b = 0* (cortado) para *i_b ≥ V_{cc}/(R · β)* (saturado).

Na Fig. 1-27(b), percebe-se que, quando o transistor está saturado, a tensão de coletor não é exatamente zero, nem quando cortado a corrente de coletor é nula. Os valores dependem do tipo de transistor e do ponto de operação. Na Fig. 1-28, fornecemos alguns desses valores característicos.

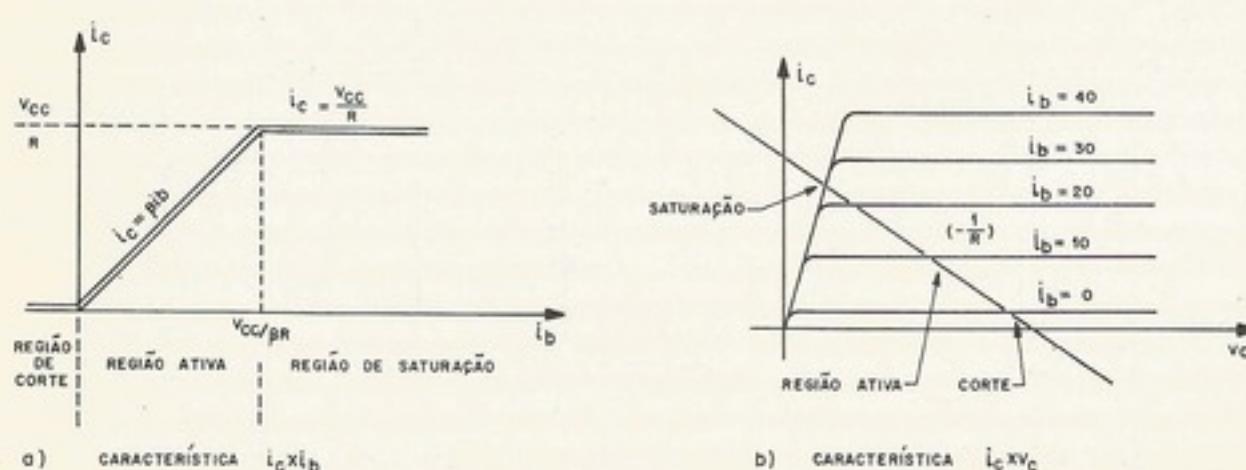
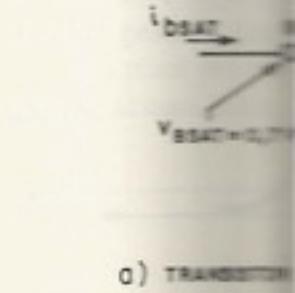
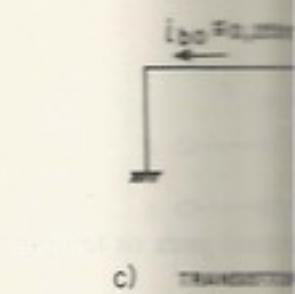


Figura 1-27. Curvas características do transistor como chave



a) TRANSMISOR



c) TRANSMISOR

Resumindo, para que o transistor funcione como chave, devemos ter:

Corte. $i_b = 0$, chama-se CORTADO.
Ativa. $0 < i_b < V_{cc}/R$, chama-se REGIÃO ATIVA.

Saturação. $i_b \geq V_{cc}/R$, chama-se SATURAÇÃO.

Estudo do regime

É importante notar que, quando o transistor está cortado, não consegue impedir que a corrente de base, nenhuma, flua. Isso faz com que a tensão de coletor fique negativa e por acumulação de portadores.

Para ilustrarmos, vamos considerar o circuito inversor da Figura 1-26.

Turn-on time. Forma de onda.

Quando injeta-se corrente na base, isto é, quando a base é aberta, ou

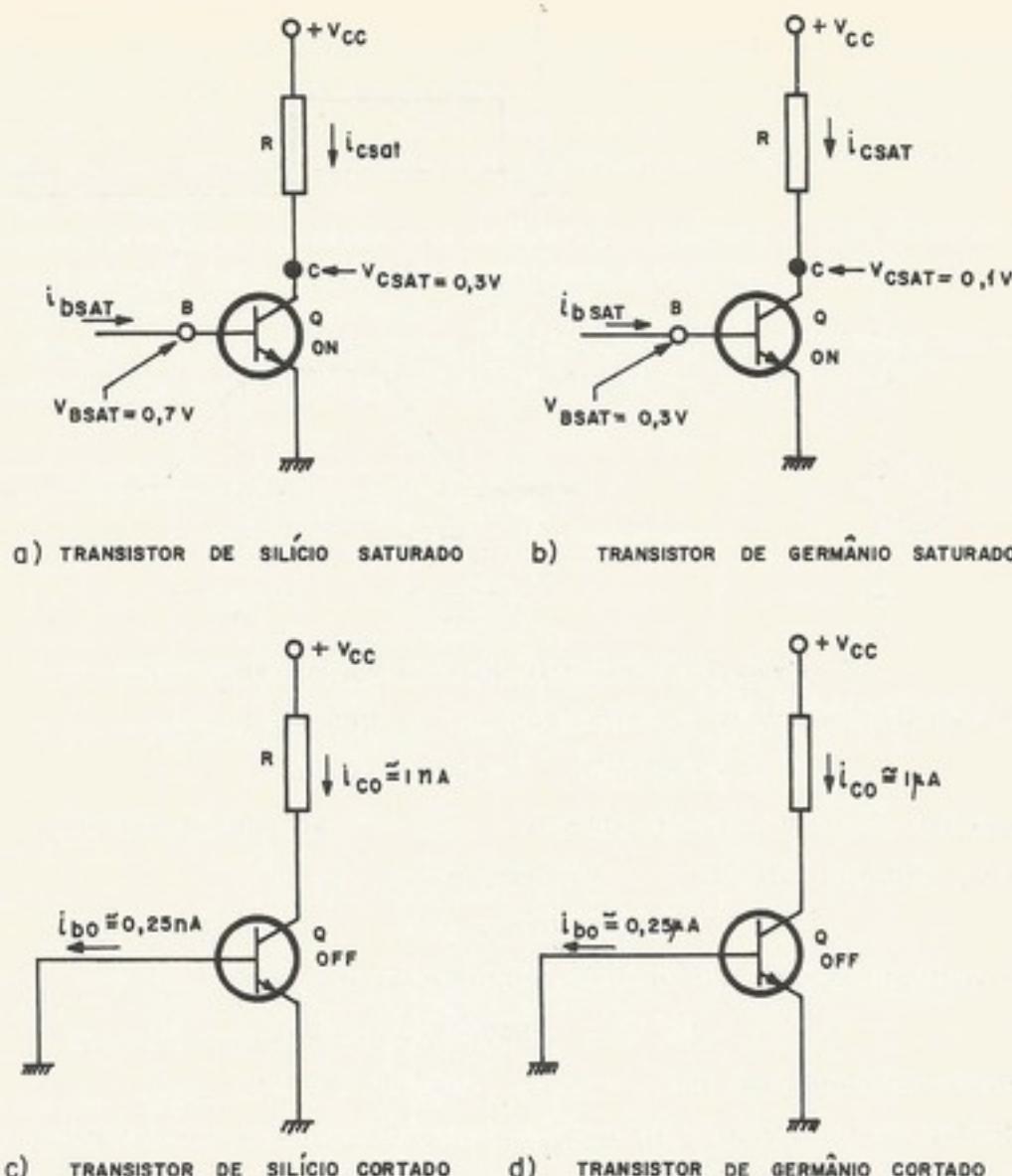


Figura 1-28. Valores característicos dos transistores como chave

Resumindo, podemos dizer que as três regiões de operação são caracterizadas conforme segue.

Corte. $i_b = 0$, duas junções reversamente polarizadas.

Ativa. $0 < i_b < V_{ce}/\beta R$ {junção coletor-base reversamente polarizada,
junção base-emissor diretamente polarizada.

Saturação. $i_b \geq V_{ce}/\beta R$, duas junções diretamente polarizadas.

Estudo do transitório

É importante para o leitor ter em mente que, no instante em que comutamos a corrente de base, nada acontece com a corrente de coletor. Somente depois de algum tempo é que se faz sentir no coletor a variação na base. Isso é provocado pelas capacitâncias das junções e por acúmulo de portadores minoritários na base. (Veja o Cap. 20 da referência⁽¹⁴⁾.)

Para ilustrarmos rapidamente o problema do atraso, apresentamos, na Fig. 1-29, um circuito inversor transistorizado com os tempos a seguir indicados.

Turn-on time. Formado por dois tempos,

$$t_d = \text{delay time},$$

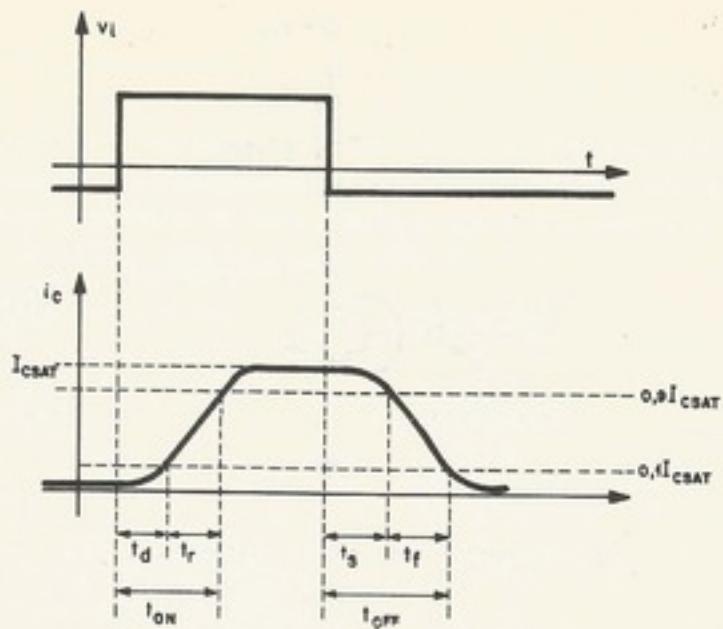
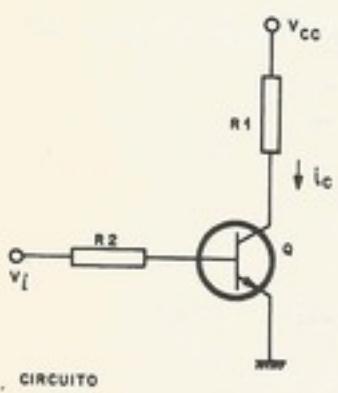


Figura 1-29. Transitório num circuito inversor

que é contado desde a subida da entrada até que i_c atinja $0,1 I_{c\text{sat}}$; e

$t_r = \text{rise time}$ (tempo de subida),

que é contado a partir do instante em que $i_c = 0,1 I_{c\text{sat}}$ até que $i_c = 0,9 I_{c\text{sat}}$.

Turn-off time. Também formado por dois tempos,

$t_s = \text{storage time}$,

contado a partir da descida do sinal de entrada até que $i_c = 0,9 I_{c\text{sat}}$; e

$t_f = \text{fall time}$ (tempo de descida),

contado desde o instante em que $i_c = 0,9 I_{c\text{sat}}$ até que $i_c = 0,1 I_{c\text{sat}}$.

É importante notar que, mais à frente, usaremos a palavra *delay* (atraso) para os tempos *turn-on* ou *turn-off* indistintamente.

c. Transistores em circuitos lógicos

Os circuitos lógicos usualmente implementados com transistores são os *NAND* e *NOR*. A técnica adotada é a da associação de um circuito inverter [Fig. 1-29(a)] na saída de um *AND* ou de um *OR* passivos (veja a Fig. 1-30). O funcionamento do inverter é entendido sem

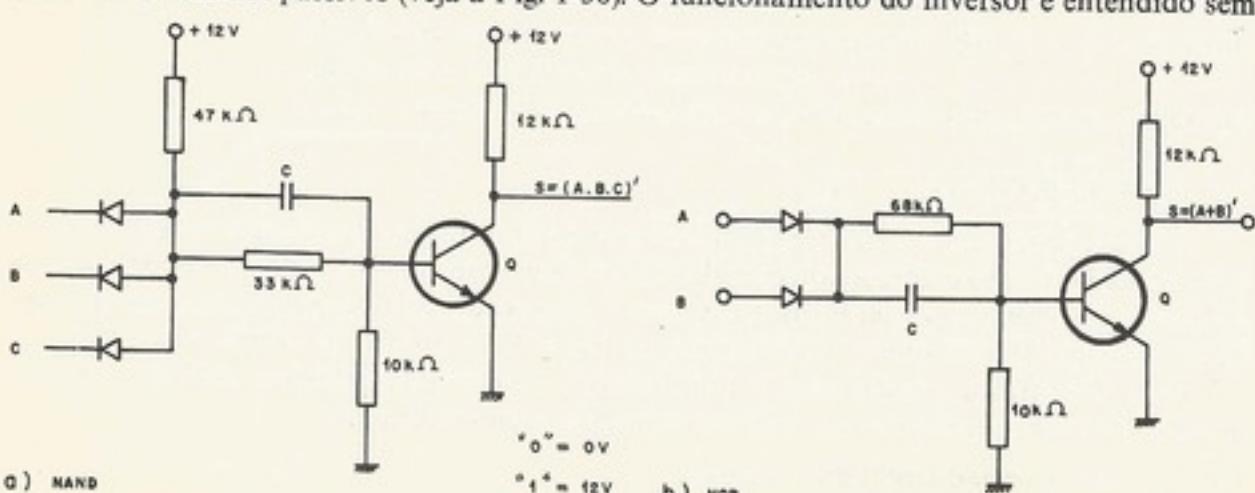


Figura 1-30. Circuitos lógicos transistorizados

problemas se pensa da Fig. 1-29) é alta, fará com que sua saída no coletor do transistor

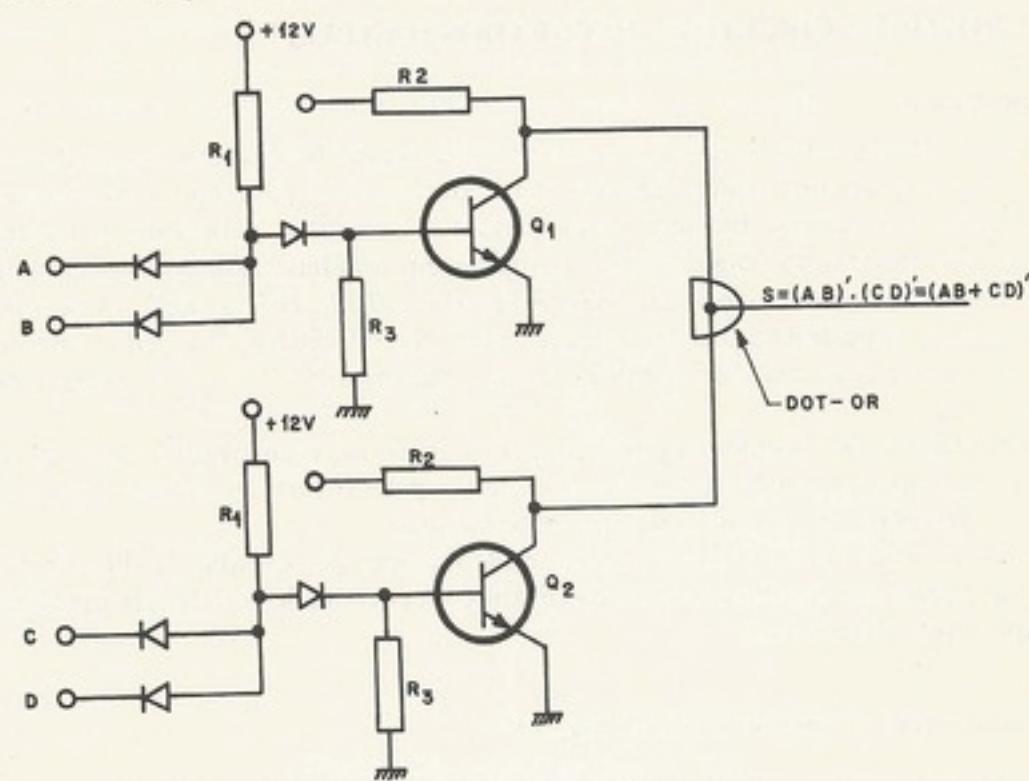
Os capacitâncias de *turn-on* e *turn-off* de *NAND* passa a ser

No nível de saída nulo e atraso zero

problemas se pensarmos da seguinte maneira: quando a tensão na entrada (V_i no circuito da Fig. 1-29) é alta, existe uma grande corrente na base do transistor que o satura, o que fará com que sua saída (tensão no coletor) caia para zero. Por outro lado, se a tensão de entrada for zero, a corrente na base será nula, o que cortará o transistor e, portanto, a tensão no coletor do transistor será alta.

Os capacitores foram colocados para melhorar a velocidade, ou seja, diminuir os tempos de *turn-on* e *turn-off*. Aqui também, se invertermos a convenção de "1" e "0", o circuito *NAND* passa a ser *NOR* e vice-versa.

No nível de transistores e resistores, pode-se conceber um novo bloco lógico de custo nulo e atraso zero, o *DOT-OR* ou *Wired-OR*, ou, ainda, *Implied-AND*⁽¹⁷⁾.



a) CIRCUITO COM DOT-OR NA SAÍDA

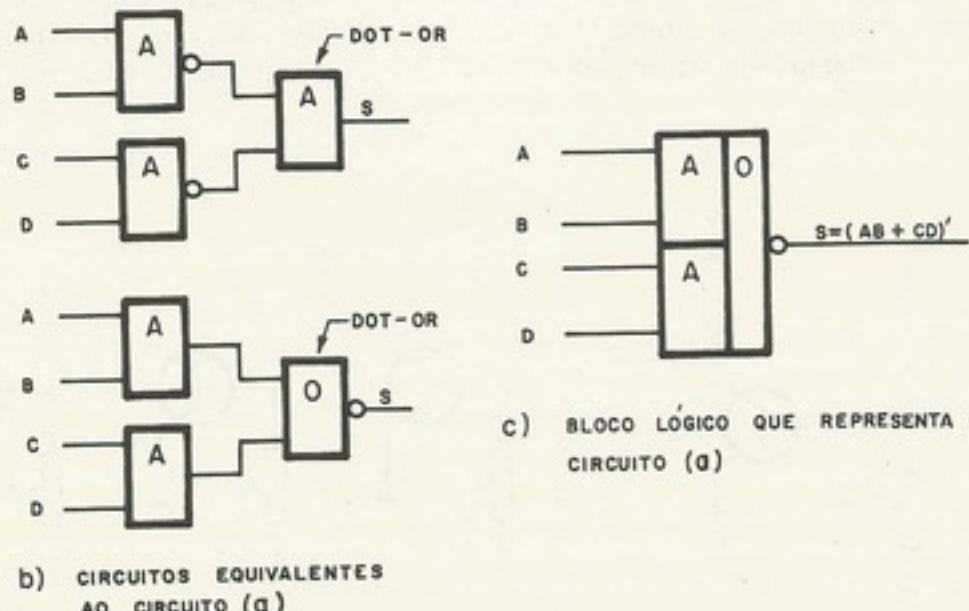


Figura 1-31. O DOT-OR

DOT-OR

Se tomarmos dois *NAND* transistorizados e ligarmos suas saídas em curto, tal como na Fig. 1-31, teremos, com esse ponto de ligação, realizado uma função *AND*, pois esse ponto só estará no nível “1” se os dois *NAND* tiverem suas saídas iguais a “1”. Um dos *NAND*, tendo sua saída igual a “0”, é suficiente para o ponto cair a zero, pois o seu transistor de saída saturado drena a corrente dos dois resistores, “derrubando” o ponto para a terra. Pode-se justificar o nome *DOT-OR* a esse ponto que realiza a operação *AND*, mostrando o circuito equivalente da Fig. 1-31(b), onde, pelo teorema de De Morgan, esse *AND* se transforma em um *OR*.

1.5 TECNOLOGIA: CIRCUITOS INTEGRADOS DIGITAIS**a. Introdução**

Todos os blocos lógicos estudados existem sob forma de circuito integrado e o projetista não mais se preocupa com o circuito propriamente dito, passando a considerar apenas as interligações dos blocos. Existem vários circuitos diferentes, realizando o mesmo bloco lógico. Por isso os circuitos digitais são agrupados em famílias segundo suas características elétricas. Vamos apresentar cinco famílias, *RTL*, *DTL*, *TTL*, *HTL* e *ECL*. A seguir, vamos falar um pouco sobre uma nova tecnologia, *FET*. Na referência⁽¹⁸⁾ o leitor encontra esse assunto com um pouco mais de detalhes e, nas referências⁽¹⁹⁾, ⁽²⁰⁾ e ⁽²¹⁾, poderá estudá-lo detalhadamente.

Vamos definir em seguida alguns conceitos úteis na comparação entre as famílias. *Delay (atraso)*. Usaremos, genericamente, tanto para o *turn-on* quanto para o *turn-off*.

Fan-in. Número de entradas de um bloco lógico.

Fan-out. *Fan-out* de um bloco é a quantidade máxima de entradas, de blocos da mesma família, que podem ser ligadas à sua saída. É um conceito ligado à potência de saída do circuito (capacidade de *driving*).

b. Família RTL (“resistor transistor logic”)

São circuitos usados principalmente com componentes discretos. Na Fig. 1-32, o leitor encontra dois circuitos diferentes realizando a mesma função lógica, *NOR*.

O resistor em série com a base torna esse circuito lento, já que, com a junção base-emissor, forma um circuito *RC*, aumentando os tempos de *turn-on* e *turn-off*. Para aumentar a velocidade, poderíamos pensar em adotar a mesma solução que já indicamos, ou seja,

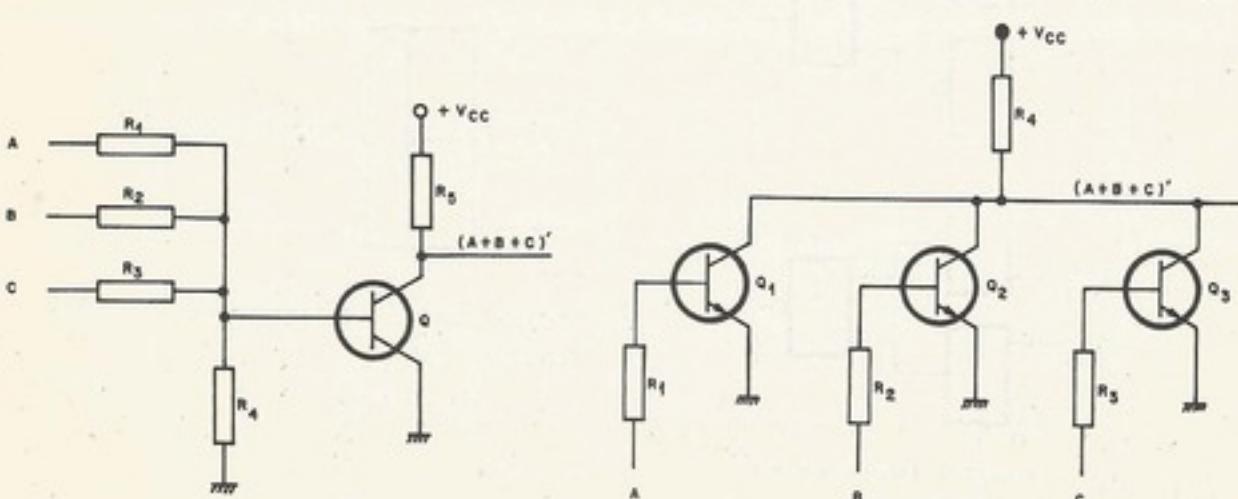


Figura 1-32. Circuitos *RTL*

introdução e ...

usar capacitors...
adotada em ...
volume exigido

Essa família
custo e pequena
monolíticas hige

c. Família DTL

Tem a estr...
pull-up, a capac...
aumentou-se a ...
O diodo em s...
liga a base à tem

O problema...
com a base, o p...
na base, e o m...
na entrada e ...
a sua saída. O ...
o transistor de ...

d. Família TTL

A Fig. 1-34...
do transistor m...
a vantagem do ...
o fan-out, geram...
push-pull, chama...
situação, durante...
grande consumo

Além de g...
realização da ...
resolver o pro...
de consumo

custo, tal como
pelo esse ponto
Um dos *NAND*,
transistor de saída
a terra. Pode-se
mundo o circuito
transforma em

projeto e o proje-
considerar apenas
o mesmo bloco
características
A seguir, vamos
encontra esse
podem estudá-lo
nas famílias.
para o *turn-off*.
bloco da mesma
saída do cir-

usar capacitores em paralelo com o resistor de base (família *RCTL*). Essa solução, que é adotada em componentes discretos, fica impraticável em circuitos integrados dado o grande volume exigido pelo capacitor.

Essa família ganha em custo de todas as outras, perdendo em *fan-out*. Dado o baixo custo e pequena quantidade de componentes, essa tecnologia tem sido usada em memórias monolíticas bipolares.

c. Família DTL ("diode transistor logic")

Tem a estrutura do tipo da que é vista na Fig. 1-33. Com a saída com resistores em *pull-up*, a capacidade de *drive* é muito maior no nível "0" e, por isso, para aumentar o *fan-out*, aumentou-se a impedância de entrada no nível "1" colocando-se os diodos vistos na figura. O diodo em série com a base serve para aumentar a imunidade ao ruído, e o resistor que liga a base à terra é importante para diminuir o tempo de *turn-off*.

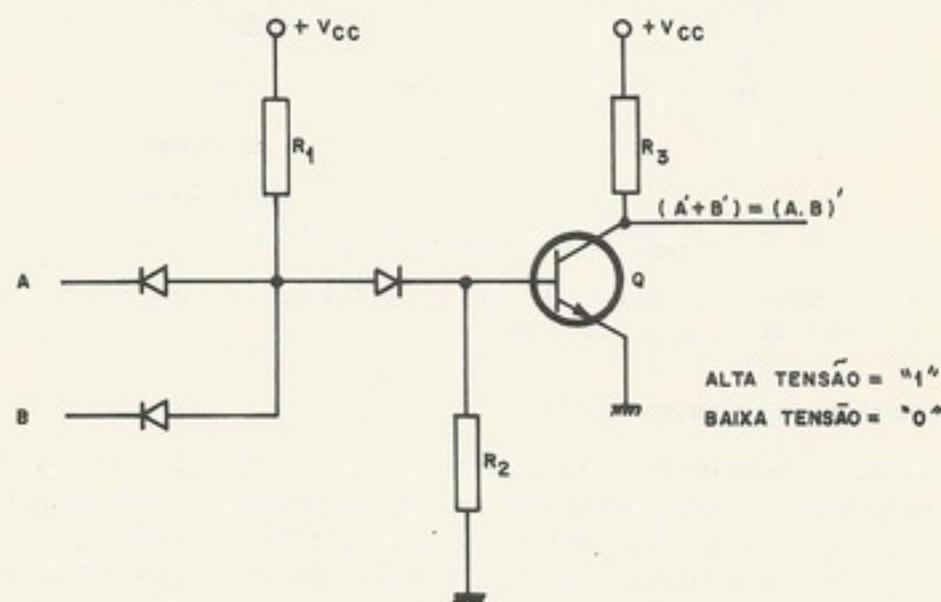


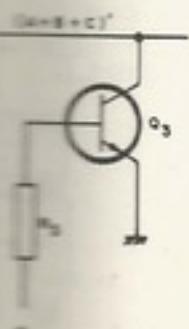
Figura 1-33. *NAND* da família *DTL*

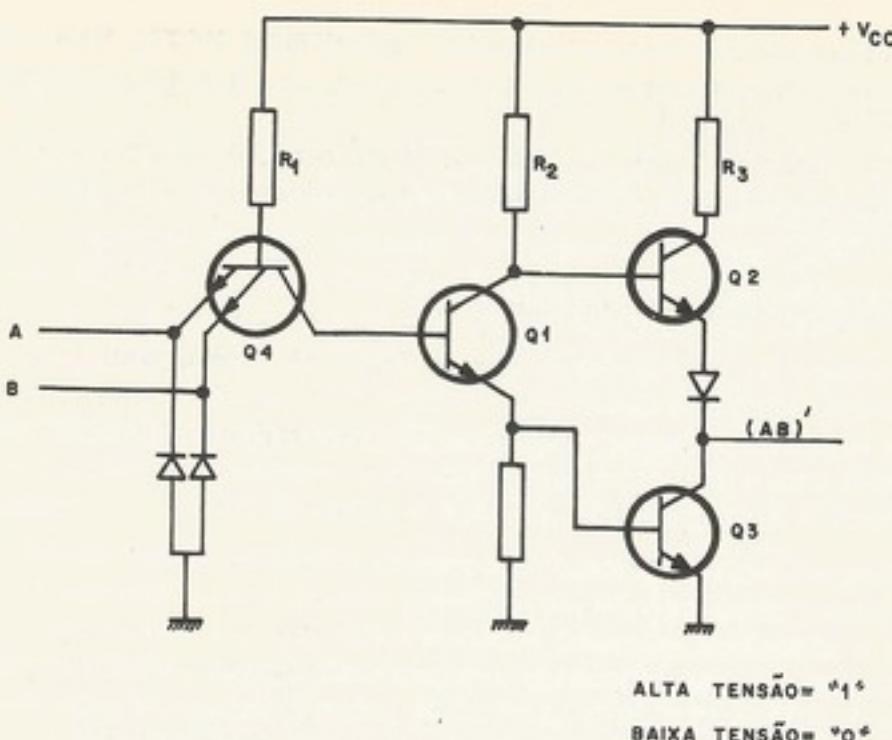
O problema da imunidade ao ruído é o que segue. Se não existisse o diodo em série com a base, o potencial na saída do *AND* quando igual a "0" (0,6 V, se silício), refletir-se-ia na base, e o transistor ficaria no limite entre o corte e a saturação, e qualquer ruído na entrada o levaria à região ativa; dependendo do ganho, poderia saturá-lo, mudando a sua saída. O diodo em série é como uma barreira de tensão (0,6 V, se silício) impedindo o transistor de conduzir se o sinal na saída do *AND* for de baixa amplitude (1,2 V, no caso).

d. Família TTL ("transistor transistor logic")⁽²²⁾

A Fig. 1-34 mostra um circuito apenas viável na forma integrada, dada a necessidade do transistor multiemissor na entrada. Seu funcionamento é parecido com o *DTL*, com a vantagem do transistor multiemissor aumentar o *turn-off* do transistor Q_1 . Para aumentar o *fan-out*, geralmente, esses circuitos são providos de um estágio de saída de potência, tipo *push-pull*, chamado aqui de *totem-pole*. Além do custo, têm a desvantagem de criar uma situação, durante a comutação, onde os transistores Q_2 e Q_3 ficam conduzindo, o que drena grande corrente da fonte de alimentação, gerando ruidos (*glitches*), muitas vezes indesejáveis.

Além de gerar ruidos na rede de alimentação, essa saída de potência não permite a realização da função *DOT-OR*, que, pelo seu custo e atraso nulos, é muito usada. Para resolver o problema, os projetistas de circuitos digitais criaram um bloco chamado "expan-



Figura 1-34. *NAND* da família *TTL*

sível" (dá acesso a dois pontos internos) e um outro "expansor", que, quando ligados, realizam a função *DOT-OR* (veja a Fig. 1-35).

e. Família HTL ("high threshold logic")

A característica fundamental desse circuito é a alta imunidade a ruídos, dado que o limiar entre as tensões que representam o "0" e as que representam o "1" é relativamente alto. Por exemplo, um circuito da família *HTL* pode funcionar com tensões entre 3 e 5 V, representando o nível "1" e com tensões entre 0 e 2 V, representando o nível "0". Se procurarmos operar o sistema com sinais do 0 V para o nível "0" e 5 V para nível "1", ruídos de até 2 V não atuarão no circuito.

Existem várias maneiras de se implementarem circuitos dessa família. A mais simples é vista na Fig. 1-36. O diodo zener é responsável pelo alto limiar.

f. Família ECL ("emitter coupled logic")

É uma família totalmente diferente das outras, já que, para diminuir o atraso, os transistores não operam saturados (técnica *current-switch*), o que descreveremos tomando um circuito típico como exemplo (Fig. 1-37).

O circuito funciona da maneira que segue. Pelo resistor de $1,24\text{ k}\Omega$ passa uma corrente mais ou menos constante; se nenhum dos transistores, Q_1 ou Q_2 , está conduzindo, então o transistor de referência, Q_3 , conduz, mas não satura. Com Q_3 conduzindo, o transistor Q_4 está cortado e o transistor Q_5 conduz. Com Q_1 ou Q_2 conduzindo, o transistor Q_4 conduz e o transistor Q_5 corta. A saída, sendo do tipo seguidor em emissor (*emitter follower*) dispõe-se de alta capacidade de *drive*.

Uma vantagem do circuito são as duas saídas de polaridade invertida, isto é, um mesmo bloco tem duas saídas, *OR* e *NOR*. O circuito é relativamente veloz porque os transistores não operam saturados, necessitando, porém, de bastante potência. Sua imunidade ao ruído é razoável, pois tem um limiar definido pela tensão na base de Q_3 . É, porém, incompatível com as famílias *TTL* e *DTL*, isto é, não pode ser ligado junto com blocos dessas famílias, por problemas de acoplamento de impedâncias e níveis.

g. Circuitos

O transistor de referência é usado na logia *MOS* (metálico-óxido-semicondutor), que assemelha à varistor. Os portadores de carga são portadores de elétrons, que são movidos pelo potencial negativo. Quando o potencial é positivo, os portadores de carga são movidos para o lado oposto, e também é chamado de varistor.

A Fig. 1-38 mostra o circuito de um varistor. Colocando o diodo zener em série obtém-se uma varistor com menor tensão de ruptura.

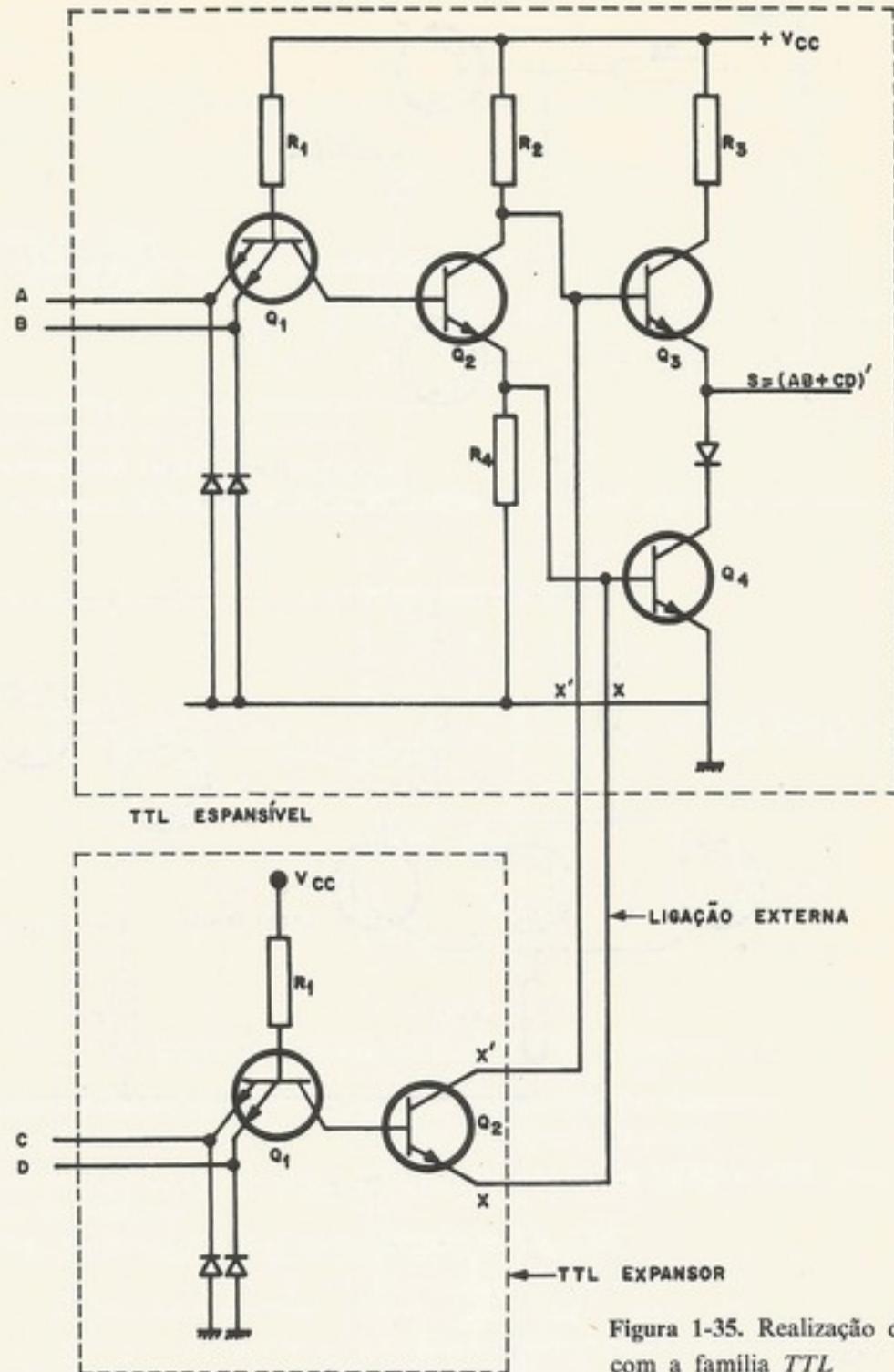
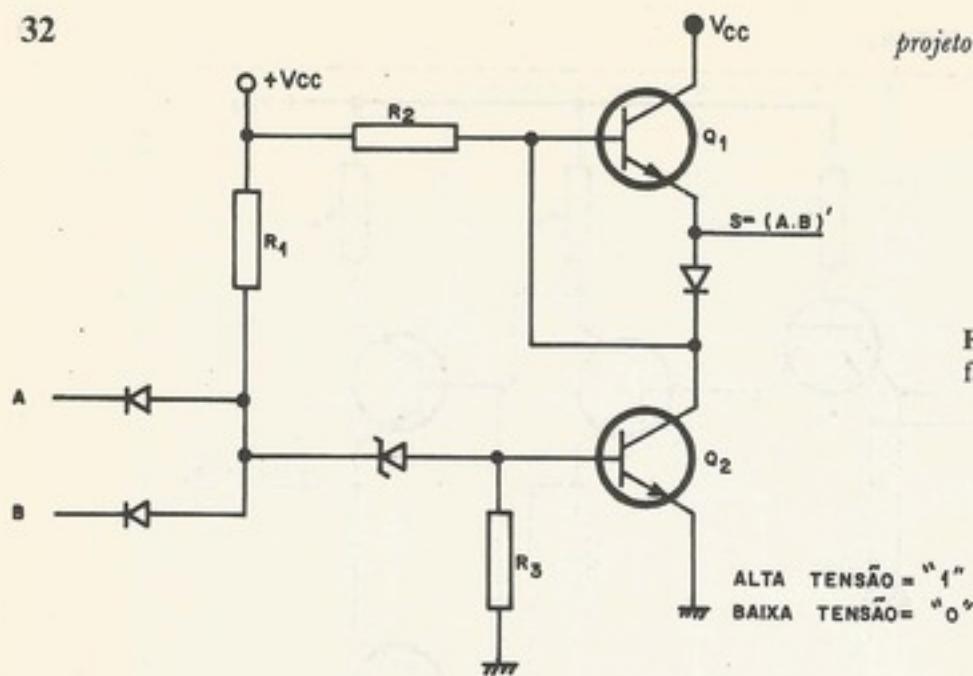
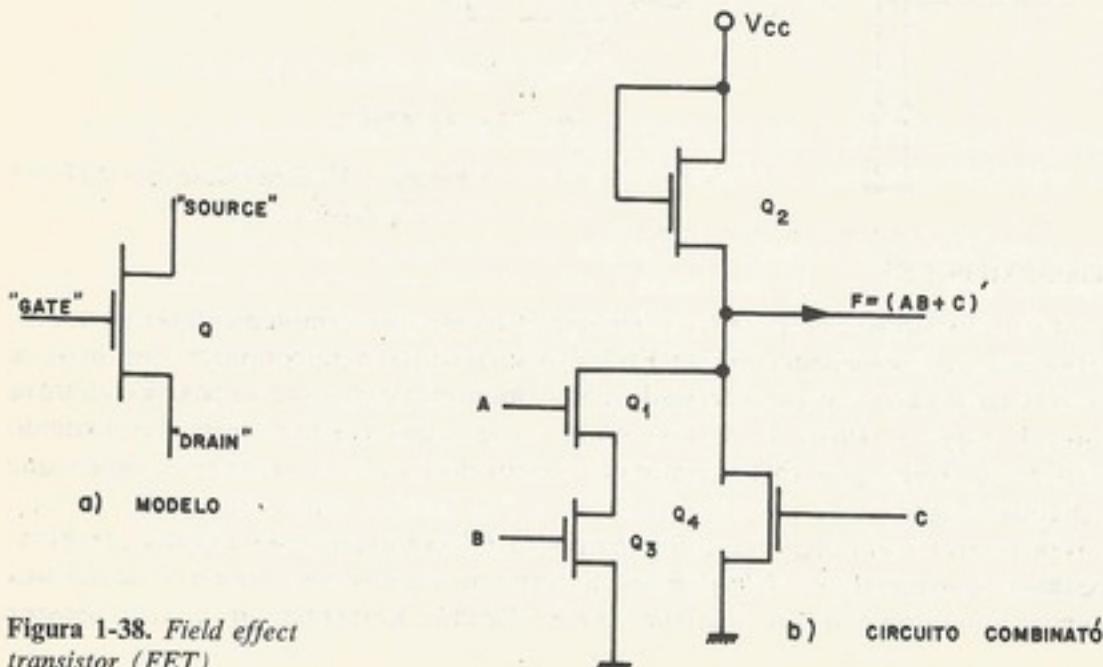
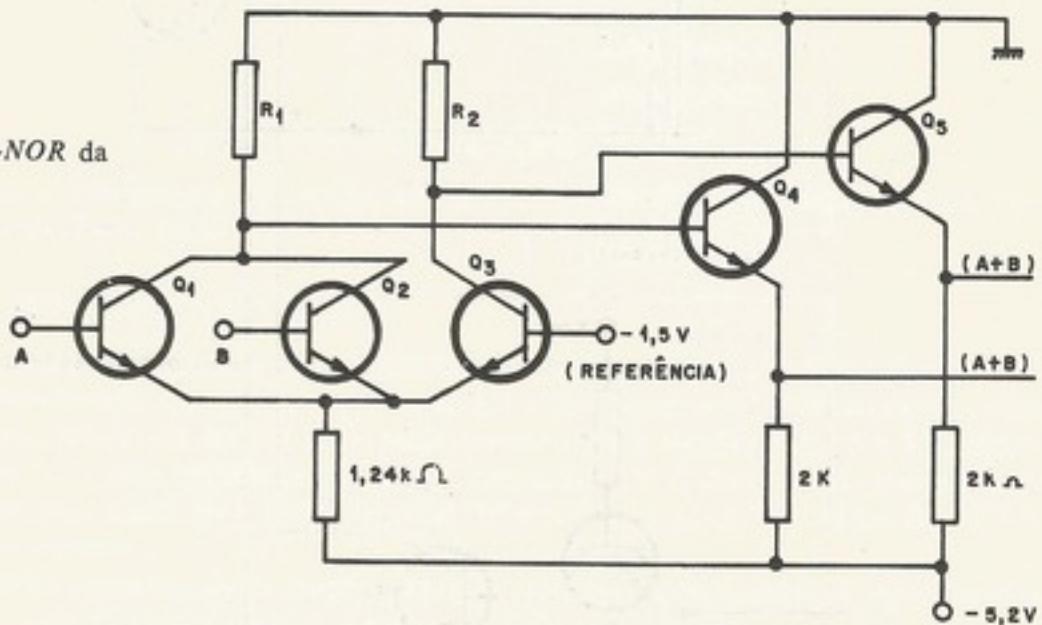


Figura 1-35. Realização de *DOT-OR* com a família *TTL*

g. Circuitos MOS-FET

O transistor de efeito de campo *FET* (*field-effect transistor*) é implementado na tecnologia *MOS* (*metal-oxide semiconductor*). O *FET* é o dispositivo semicondutor que mais se assemelha à válvula a vácuo. É caracterizado como monopolar porque apenas os elétrons são portadores de carga. O fluxo de elétrons entre os elétrodos *source* e *drain* é controlado pelo potencial no elétrodo *gate*. (Não confunda o elétrodo *gate* com o circuito lógico que também é chamado *gate*.)

A Fig. 1-38 mostra a aplicação de circuitos com *FET* como circuitos lógicos combinatórios. Colocando os dispositivos *FET* em paralelo formam-se *gates* tipo *NOR* e colocando-os em série obtém-se *gates* tipo *NAND*. Na figura, o *FET* ligado à fonte funciona como o resistor de carga.

Figura 1-36. *NAND* da família *HTL*Figura 1-37. *OR-NOR* da família *ECL*Figura 1-38. *Field effect transistor (FET)*

introdução e fundamentos

EXERCÍCIOS

- 1-1. Demonstre que
1-2. A igualdade

é uma identidade:
1-3. Dada a descrição

determine:

- (a) sua descrição
(b) sua descrição
(c) sua descrição
1-4. Dado o circuito

- 1-5. Sintetize os circuitos
(a) $s = a'(bc + d')$
1-6. Sintetize os circuitos
teoremas de De Morgan
1-7. Sintetize, em d

EXERCÍCIOS

- 1-1. Demonstre por manipulação da expressão booleana os teoremas (T.17) e (T.18).
 1-2. A igualdade

$$(A + B) \cdot (A' + C) \cdot (B + C) = (A + B) \cdot (A' + C)$$

é uma identidade? Prove.

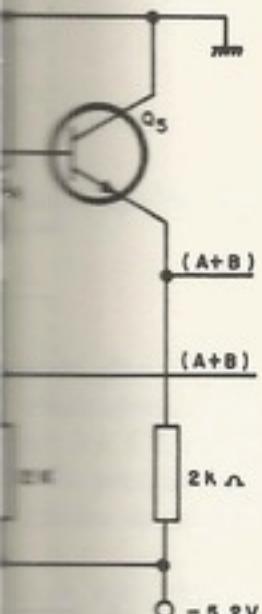
- 1-3. Dada a descrição de um circuito combinatório,

$$s = ab + a'b' + acb'$$

determine:

- (a) sua descrição pela tabela de combinações;
 (b) sua descrição pelo mapa de Karnaugh;
 (c) sua descrição pela transformada numérica.

- 1-4. Dado o circuito descrito pela tabela de verdade da Fig. 1-39, obtenha a sua soma canônica.



a	b	c	s
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

Figura 1-39. Tabela de combinações do Exercício 1-4

- 1-5. Sintetize os circuitos mínimos dados pelas seguintes expressões booleanas:

(a) $s = a'(bc + b'') + a'b$; (b) $s = x(y' + z) + z'(x + y)$; (c) $s = ab + a'b'$.

- 1-6. Sintetize os circuitos do exercício anterior apenas com apenas *NAND* e *NOR*. [Sugestão. Use os teoremas de De Morgan.]

- 1-7. Sintetize, em dois níveis, os circuitos descritos na Fig. 1-40.

x	y	z	s
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

a) CIRCUITO a

		ab	00	01	11	10
		cd	00	01	11	10
00	00		0	0	1	0
00	01		1	0	1	1
01	11		0	1	1	0
01	10		0	0	1	0

b) CIRCUITO b

Figura 1-40. Descrições dos circuitos do Exercício 1-7

1-8. Sabendo que o β do transistor do circuito da Fig. 1-41 é igual a 10, calcule o valor de R_1 .

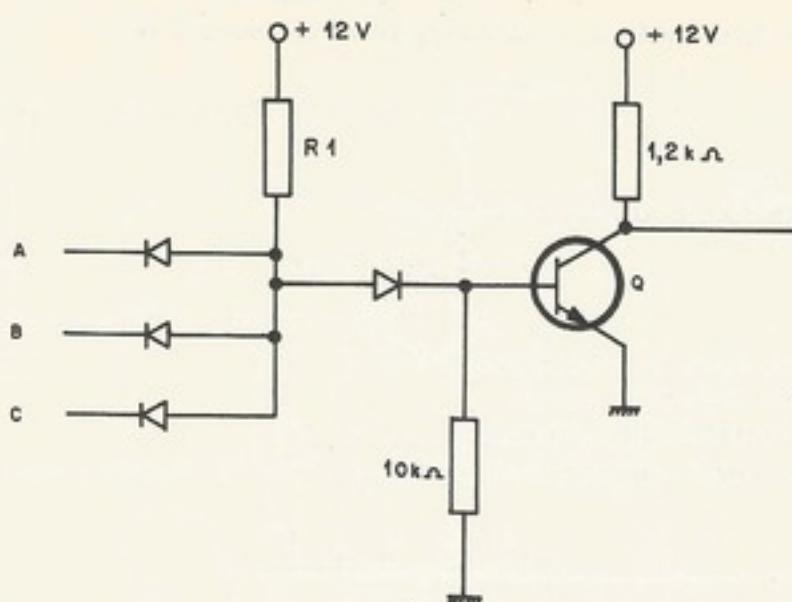


Figura 1-41. Circuito do Exercício 1-8

1-9. Projete dois circuitos que acoplem as famílias TTL com ECL, isto é, um que acopla a saída do TTL com a entrada do ECL, e outro que acople a saída do ECL com a entrada do TTL.

1-10. Faça um circuito apenas com FET que sintetize a função

$$s = A(C + D) + B.$$

BIBLIOGRAFIA

- (1) Nievergelt, J., "Computers and Computing – Past, Present, Future", *IEEE Spectrum*, janeiro de 1968, pp. 57-61
- (2) Laver, F. J. M., *Introducing Computers*, Her Majesty's Stationery Office, Londres, 1965
- (3) Aiken, H., "Proposed Automatic Calculating Machine", *IEEE Spectrum*, agosto de 1964, pp. 62-69
- (4) Burts, A. W. et al., *Preliminary Discussion of the Logical Design of an Electronic Computing Instrument*, junho de 1946. (Republicado na Datamation em setembro/outubro de 1962)
- (5) Boole, G., *An Investigation of the Laws of Thought...*, Londres, 1854. (Republicado pela Dover Publications, New York, 1951)
- (6) Shannon, C. E., "A Symbolic Analysis of Relay and Switching Circuits", *Electrical Engineering, Trans. Suppl.* 57, pp. 713-723, 1938
- (7) Wood, P. E., *Switching Theory*, Lincoln Laboratory Publications, McGraw-Hill, 1968
- (8) Miller, R. E., *Switching Theory*, John Wiley and Sons, N.Y. 1965
- (9) McCluskey, E. J., *Introduction to the Theory of Switching Circuits*, McGraw-Hill, 1965
- (10) Del Picchia, W., "A transformada numérica e sua aplicação à simplificação de funções e à resolução de equações booleanas", Tese do Departamento de Engenharia de Eletricidade da EPUSP, 1971
- (11) Hellerman, L., "A Catalog of Three Variable Or-Invert and And-Invert Logical Circuits", *IEEE Transactions on Electronic Computers*, junho de 1963, pp. 198-223
- (12) Baugh, G. R., et al., "Optimal Networks of NOR-OR Gates for functions of Three Variables", *IEEE Transactions on Computer*, fevereiro de 1972, pp. 153-160
- (13) Sifferlen, T. P., *Digital Electronics With Engineering Applications*, Prentice Hall, N.J., 1970
- (14) Millman, J., e Taub, H., *Pulse Digital and Switching Waveforms*, McGraw-Hill, 1965
- (15) Harris, J. N., et al, *Digital Transistor Circuits – Semiconductor Electronics Education Committee, SEEC – Vol. 6*, John Wiley, N.Y., 1966
- (16) Zuffo, J. A., *Subsistemas Digitais e Circuitos de Pulso*, EPUSP, 1972
- (17) Weger, Ph., "Wired-or With the FC Family of Integrated Circuits", Philips, Application Information, 847, Holanda, 1969
- (18) Comenzind, H. R., "Digital Integrated Circuit Design Techniques", *Computer Design*, novembro de 1968, pp. 52-62

- (19) Garret, L. S., "The Spectrum", outubro de 1968
- (20) Garret, L. S., "The Spectrum", novembro de 1968
- (21) Garret, L. S., "The Spectrum", dezembro de 1968
- (22) Beeson, R. H., *High-Speed Solid-State Circuits*, McGraw-Hill, 1968
- (23) Unger, S. H., *High-Speed Solid-State Circuits*, McGraw-Hill, 1968

- (19) Garret, L. S., "Integrated circuit Digital Logic Families — Part I, RTL and TTL devices", *IEEE Spectrum*, outubro de 1970
- (20) Garret, L. S., "Integrated Circuito Digital Logic Families — Part II, TTL devices". *IEEE Spectrum*, novembro de 1970, pp. 63-72
- (21) Garret, L. S., "Integrated Circuit Digital Logic Families — Part III, ECL and MOS devices", *IEEE Spectrum*, dezembro de 1970, pp. 30-42
- (22) Beeson, R. H., Ruegg, H. W., "New Forms of All-Transistor Logic", *Digest*, 1962, International Solid-State Circuits Conference, IEEE, pp. 10-11, 104
- (23) Unger, S. H., *Asynchronous Sequential Switching Circuits*, John Wiley Interscience, N.Y., 1969

capítulo 2

CÓDIGOS, NÚMEROS E ARITMÉTICA

2.1 CÓDIGOS

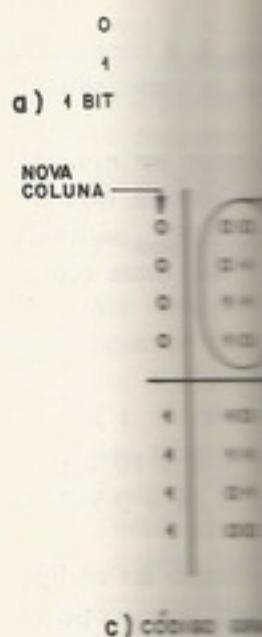
Para se processarem informações por um computador, é necessário representar as informações numa forma que ele reconheça. A unidade mais básica de informação é o dígito binário, ou *bit*. O *bit* pode ter valor "0" ou valor "1". Um conjunto de 7 a 9 *bits* (dependendo do veículo, meio ou código) é chamado um carácter. Esses códigos têm 1 *bit* de paridade; então 6 a 8 *bits* para representar conjuntos de 64 a 256 caracteres alfa-numéricos. Às vezes, carácter é usado como subdivisão de uma *palavra* de 16 a 64 *bits*. A troca de informações entre a máquina e o homem geralmente é feita em caracteres. Um código de 7 bits dispõe de 127 ou 128 caracteres distintos.

Um *código* é um conjunto de regras (ou tabela de combinações) pelo qual informações ou dados (por exemplo, números ou letras) podem ser convertidos a uma representação do código e vice-versa. Um dos códigos mais importantes é o código de cartões perfurados, chamado Hollerith, o nome do inventor. Esse código é de 12 *bits*, e tem representações para a maioria dos símbolos encontrados numa máquina de escrever.

O processo de codificação acontece quando uma tecla de uma perfuradora é batida, sendo codificada mecanicamente num código de 7 *bits*. Uma espécie de decodificação é a que o leitor está fazendo neste instante. Os conjuntos de letras que compõem estas palavras estão sendo reconhecidos, trazendo da memória o sentido da palavra.

Quando um carácter codificado é enviado de uma unidade a outra, por exemplo do computador à impressora, é costume enviar junto um *bit* a mais, chamado *bit* de "paridade". Num sistema com paridade "ímpar", para cada carácter junto com o *bit* de paridade, deve haver um número ímpar de *bits* no estado "1". Por exemplo, a combinação 0010110 tem três *bits* no estado "1", então o conjunto é "válido" num sistema de paridade ímpar. Um código padronizado para o intercâmbio de informação é chamado *ASCII* (código-padrão americano para intercâmbio de informação). Existem códigos *ASCII* para 7 *bits* de informação com 1 *bit* de paridade e, também, versões de 8 *bits* mais um de paridade para fitas magnéticas. As máquinas dos sistemas 360 e 370 da IBM usam um código de 8 *bits* de informação mais um *bit* de paridade chamado *EBCDIC* (*extended binary coded decimal interchange code*). Um outro código comum, usado em fita perfurada, é o código Baudot, de 5 *bits*.

Existe uma classe de códigos chamados *códigos de distância unitária* (*unit distance codes*). Esses códigos são úteis para se codificar uma posição mecânica ou angular, como um eixo. A distância Hamming entre dois caracteres de um código é o número de posições de *bit* em que os dois caracteres diferem. Por exemplo, caracteres 000111 e 001111 diferem em uma posição só, enquanto que caracteres 000111 e 111000 diferem em todas as seis posições. A idéia básica de códigos de distâncias unitárias é ser código ordenado (existe uma ordem 1, 2, 3, 4, etc.) de modo tal que caracteres vizinhos ($n, n + 1$) difiram em uma só posição.



O código mais popular é, também, comumente gerado de um

Para o código de paridade ímpar, o último carácter é gerado de um

O código de paridade ímpar é gerado de um

Uma aplicação comum de informática é usar a paridade para detectar erros. Para o código de paridade ímpar, o último carácter é gerado de um

O código de paridade ímpar é gerado de um

Uma aplicação comum de informática é usar a paridade para detectar erros. Para o código de paridade ímpar, o último carácter é gerado de um

O código de paridade ímpar é gerado de um

Uma aplicação comum de informática é usar a paridade para detectar erros. Para o código de paridade ímpar, o último carácter é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um

O código de paridade ímpar é gerado de um</p

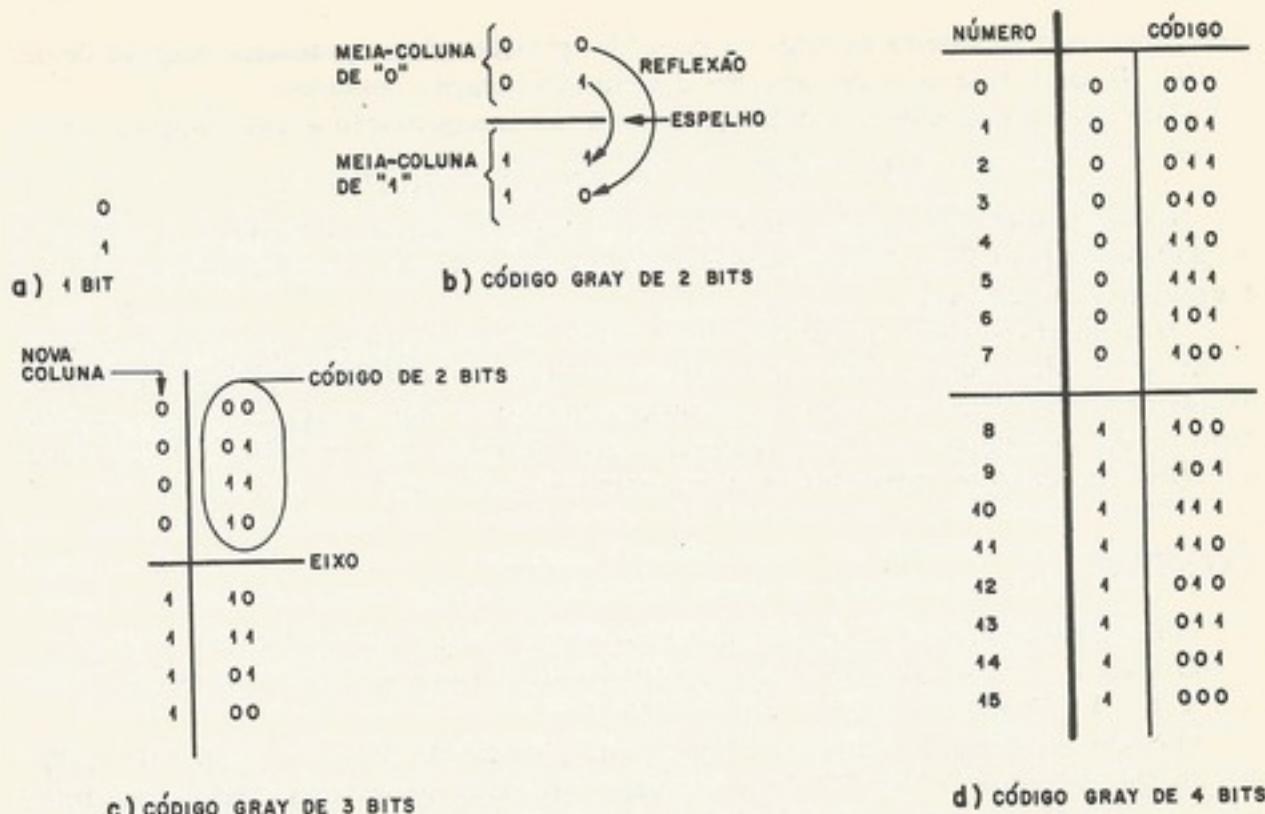


Figura 2-1. A geração de códigos de Gray

O código mais popular desse tipo é o código de Gray (*Gray code*, inventado por Gray), que é, também, conhecido como o código binário refletido (*reflected binary code*). Esse código é gerado de um modo bastante simples, como mostrado na Fig. 2-1.

Para o código de Gray de dois bits, a coluna "0,1" é refletida sobre um eixo embaixo do último carácter, fazendo a coluna 0,1,1,0. A seguir, aumenta-se uma nova coluna, na esquerda da qual, a metade superior consta de "0" e a metade inferior consta de "1".

O código de Gray também tem como propriedade o fato de a distância Hamming entre o último carácter na ordem e o primeiro ser uma unidade só. O código de Gray é útil, portanto, para contadores em que a primeira conta segue a última (veja Cohn e Even⁽¹⁾).

Uma aplicação de códigos em que existem mais combinações de bits do que caracteres de informação é para deteção e/ou correção de erros em transmissão⁽²⁾⁽³⁾. A idéia é acrescentar bits às palavras, aproveitando-se essa redundância para diminuir o efeito de erros no veículo, meio de armazenamento ou transmissão. Códigos de correção de erros simples (um bit só da palavra errada) têm a seguinte propriedade: cada palavra do código que dista uma unidade Hamming do carácter codificador corretamente está no conjunto que implica o carácter em questão. Por exemplo, se o código 101010 significa o dígito "1" num código de correção de erro simples, então 101011 é também interpretado como "1". Em termos do mapa de Karnaugh, os "vizinhos" das combinações corretas também são interpretados como tais combinações.

Para aplicações de códigos em sistemas digitais, recomendamos a referência⁽⁴⁾.

2.2 NÚMEROS

a. Sistemas

Um número binário é uma cadeia de bits em que cada posição tem um "peso". O peso é o valor associado com o bit quando no estado "1". O valor do número é o valor da soma dos pesos correspondentes às posições dos bits no estado "1". Para números binários, o valor de cada posição é 2 (a base do sistema) levada a uma potência inteira. Analogamente,

para números num sistema decimal, os pesos das posições são as potências integrais de dez (1, 10, 100, etc.). Esse tipo de esquema é chamado *sistema posicional*.

Para números positivos e inteiros, o valor da representação é calculado da seguinte forma:

$$N = A_0 B^n + A_1 B^{n-1} + \cdots + A_{n-1} B^2 + A_n B + A_n \quad (2.1)$$

Nessa equação, B é a base, sendo 2 para o sistema binário. O dígito A_i é coeficiente da posição de peso B^{n-1} . No caso de base dez, ou seja, números decimais, o coeficiente A_i é um dígito entre 0 e 9.

Os coeficientes A_i são agrupados numa cadeia para representar o número em notação posicional; conhecendo-se a base, não é necessário indicar os elementos B^i . Convém, quando há dúvida sobre a base, escrever a base como subíndice do número. Assim, $(110)_{10}$ significa "cento e dez" enquanto que $(110)_2$ significa "seis". Também $(6)_{10}$ e $(110)_2$ são duas representações ou *numerais* diferentes para o mesmo número (*seis*).

EXEMPLOS

O número decimal 946 [representado $(946)_{10}$] tem o valor de $9 \times 100 + 4 \times 10 + 6 = 900 + 40 + 6 = 946$. O número binário $(010110)_2$ tem o valor $0 \times 32 + 1 \times 16 + 0 \times 8 + 1 \times 4 + 1 \times 2 + 0 \times 1 = 16 + 4 + 2 = (22)_10$.

Quando o sistema de números é binário, não há dúvida de que a seleção do código dos dígitos está entre "0" ou "1". Mas, quando o sistema é decimal, existem várias opções para se representarem os dígitos 0 a 9. Isso porque os dispositivos de sistema digitais são binários e não decimais; por exemplo, um transistor conduzindo corrente (saturado) ou não-conduzindo (cortado). Então precisamos representar os dígitos decimais como combinações de sinais binários. O código mais óbvio para essa tarefa é a própria representação do dígito em binário. Essa representação é chamada *BCD* (*binary coded decimal*), que significa "decimal codificado em binário".

Uma outra representação que foi popular na primeira e segunda gerações, é o código BCD acrescentado de três. Assim, três em binário $(0011)_2$ representa o dígito decimal zero, e doze em binário $(1100)_2$ representa o dígito decimal nove. Esse código é chamado *XS-3* (*excess 3*), que significa “três em excesso”. Esses códigos são mostrados na Tabela 2.

Para simplificar a impressão de números binários, é costume aproveitar-se o dígito octal (3 bits) ou hexadecimal (4 bits). No sistema octal, os bits "000" a "111" são transcodificados para "0" a "7". No sistema hexadecimal, os bits "0000" a "1001" são transcodificados "0" a "9" e os bits "1010" a "1111" são transcodificados "A" a "F".

Tabela 2-1. Códigos decimais

Digito decimal	Código BCD ^(*)	Código XS-3
0	0000	0011
1	0001	0100
2	0010	0101
3	0011	0110
4	0100	0111
5	0101	1000
6	0110	1001
7	0111	1010
8	1000	1011
9	1001	1100

[*] Combinações “ilegais” em BCD: 1010, 1011, 1100, 1101, 1110 e 1111.

códigos, números e

b. Números impares

A representação de frações usando-se para tada como 1×10^{-1} da posição depende da sentação depende da 017890 é dezessete e o quarto dígi

c. Números negati

Para indicar que lado esquerdo da ampolha negativos é chamado o sinal Á vizinha de

Para números inteiros, com \exists^*

Existem mais dois autores. A grandeza de (1) complemento de sujeito de base diminuída nas teóricas para o êxito das tarefas foram adotadas, com a ajuda do computador.

Vamos superar essa limitação. Se o bit de sinal é “0”, os valores possíveis são de 0 a $2^n - 1$, ou seja, de 0 a mais 2^{n-1} unidades. Se o bit de sinal é “1”, os valores possíveis são de -2^{n-1} a -1 , ou seja, de mais 2^{n-1} a menos 2^{n-1} unidades.

d. Número de

Sendo $(+N)$ um em complemento de n , inclusive, o bá é:

EJEMPLO 5-4

A operação obviamente

EXEMPLO. $n = 4$

Uma desvantagem é que para o número 0 o

EXEMPLO

números inteiros de dez

simplificado na forma:

(2-1)

 A_0 é coeficiente A_1 é coeficiente

Número em notação

Convém, quando

 $(110)_2$ significa

não duas repre-

$$+ 4 \times 10 + 6 = \\ 1 \times 16 + 0 \times 8 +$$

representação do código

várias opções

sistema digitais são

saturado) ou

como com-

representação

que significa

é o código

decimal zero,

é chamado XS-3

Tab. 2-1.

ocorre-se o dígit

“111” são trans-

só transcodi-

b. Números inteiros e fracionários

A representação de números inteiros positivos pode ser estendida para representação de frações usando-se potências negativas da base. Por exemplo, a fração $(0,175)$ é representada como $1 \times 10^{-1} + 7 \times 10^{-2} + 5 \times 10^{-3}$. Assim para os números inteiros, o peso da posição depende da sua localização, ou seja, o valor de um número nesse tipo de representação depende da localização do ponto da base. Por exemplo, o valor da cadeia de dígitos 017890 é dezessete e oitenta e nove centésimos quando o ponto decimal está entre o terceiro e o quarto dígito da palavra.

c. Números negativos

Para indicar que um número é negativo, normalmente escrevemos o símbolo – ao lado esquerdo da amplitude do número, como -39 . Esse tipo de representação de números negativos é chamado de *sinal e amplitude*. Quando usado em sistemas decimais, às vezes o sinal é vizinho do dígito menos significativo, como no computador IBM 1401.

Para números binários, o sinal é normalmente o *bit* na posição mais significativa da palavra, com “0” indicando + e “1” indicando –.

Existem mais duas representações para sistemas de números encontrados em computadores. A grandeza de número será representada diferentemente para números negativos: (1) complemento de base (*complemento de dois* para o sistema binário) e (2) complemento de base diminuída (*complemento de um* para o sistema binário). Não tente procurar razões teóricas para o êxito dessas duas representações em sistemas digitais, pois as representações foram adotadas, como veremos, para simplificar e economizar circuitos da unidade aritmética do computador.

Vamos supor que a palavra binária disponha de n bits. Então temos 2^n combinações à nossa disposição. Em termos grosseiros, metade das combinações será negativa (a metade com o bit do sinal no estado “1”), então, com n bits, podemos representar números entre mais 2^{n-1} ou menos 2^{n-1} . No sistema binário para todas as três representações de números negativos, quando o bit de sinal é “0” as grandezas dos números de 0 a $2^{n-1} - 1$ são representadas em binário. A única diferença entre as representações acontece quando o bit de sinal é “1”.

d. Números binários em complemento de um

Sendo $(+N)$ um número binário de n bits entre 0 e $2^{n-1} - 1$, a representação de $(-N)$ em complemento de um é $[(+N)_2]'$ onde $[]'$ indica que cada bit de $(+N)$ é complementado, inclusive o bit de sinal.

EXEMPLO. $n = 4$

$$N = (+4)_10 = (0100)_2, -N = (1011)_2.$$

A operação obviamente é a mesma na passagem de um número negativo para positivo. Sendo N número negativo em complemento de 1, a representação de $-N$ é $[(N)_2]'$.

EXEMPLO. $n = 4$

$$N = (-5)_10 = (1010)_2, -N = (0101)_2 = (+5)_10.$$

Uma desvantagem da representação complemento de um é que existem duas representações para o número 0, o +0 e o -0. O problema é chamado de *ambigüidade de zero*.

EXEMPLO

Se $n = 4$ (4 bits), então zero = 0000. Fazendo o complemento de um de zero, obtemos 1111.

Alguns sistemas evitam o chamado "problema de menos zero" por hardware. A deteção da combinação -0 como resultado de qualquer operação aritmética provoca uma conversão imediata para +0.

Outros sistemas evitam -0 pela técnica de complemento da base (complemento de dois em nosso caso), que é bastante semelhante com complemento de um, gozando das mesmas vantagens (e até mais) em hardware, mas evitando a combinação -0.

e. Números em complemento de dois

Sendo N uma palavra de n bits, representando um número entre 0 e $2^{n-1} - 1$, $-N$ na representação de complemento de dois é dado por $[(N)_2]' + (1)_2$. O complemento de dois de um número binário é, então, o complemento de um do número acrescentado de um.

EXEMPLO. $n = 4$

$$\begin{aligned} N &= (+4)_{10} = (0100)_2, \\ -N &= (1011)_2 + (1)_2 = (1100)_2 = (-4)_{10}. \end{aligned}$$

Observa-se que aqui também, complemento de dois do complemento de dois de um número é o próprio número. Em resumo, usando-se n bits, a soma binária (não se tratando o bit de sinal como exceção) de N e o seu complemento de um são $11\dots1$ (n bits) = $2^n - 1$.

EXEMPLO. $n = 4$

$$\begin{aligned} N &= (+4)_{10} = 0100, \\ N + (-N) &= 0100 + 1011 = 1111. \end{aligned}$$

Usando-se n bits, a soma de N e o complemento de dois N dá $100\dots0$ ($n+1$ bits) = 2^n .

EXEMPLO. $n = 4$

$$\begin{aligned} N &= (+4)_{10} = 0100, \\ N + (-N) &= 0100 + 1100 = 10000. \end{aligned}$$

Uma comparação das três representações de números negativos para $n = 4$ aparece na Tab. 2-2.

Tabela 2-2. Números binários negativos

N	Sinal e amplitude	Complemento de um	Complemento de dois
+7	0111	0111	0111
+6	0110	0110	0110
+5	0101	0101	0101
+4	0100	0100	0100
+3	0011	0011	0011
+2	0010	0010	0010
+1	0001	0001	0010
+0	0000	0000	0001
-0	1000	1111	não existe
-1	1001	1110	1111
-2	1010	1101	1110
-3	1011	1100	1101
-4	1100	1011	1100
-5	1101	1010	1011
-6	1110	1001	1010
-7	1111	1000	1001
-8	não existe	não existe	1000

f. Ponto fixo e ponto flutuante
Em hardware, os números são representados por uma palavra de n bits. Muitas vezes, é mais fácil processar palavras de 16 ou 32 bits. A representação de ponto fixo é a mais simples, mas não é a mais significativa. Para adicionar ou subtrair em ponto fixo, precisamos multiplicar os operandos para que o resultado final seja o mesmo. Se a palavra é de 16 bits, ela fornece um resultado de 16 bits. A palavra de 16 bits é dividida em partes: a parte mais significativa (de 0 a 15 bits) é o resultado, e a parte mais significativa (de 16 a 31 bits) é o resultado final. A representação de ponto flutuante é a mais complexa, mas também é a mais utilizada. Se a palavra é de 32 bits, ela fornece um resultado de 32 bits. A palavra de 32 bits é dividida em partes: a parte mais significativa (de 0 a 23 bits) é o resultado, e a parte mais significativa (de 24 a 31 bits) é o resultado final. A representação de ponto flutuante é a mais utilizada em hardware.

Um número é representado por uma palavra de n bits. A palavra é dividida em partes: a parte mais significativa (de 0 a 15 bits) é o resultado, e a parte mais significativa (de 16 a 31 bits) é o resultado final. A representação de ponto flutuante é a mais utilizada em hardware.

EXEMPLO. Número

EXEMPLO. Número

O transbordamento ocorre quando a soma é maior que a faixa permitida, o que resulta em resultados errôneos.

No sistema IEEE-754, existem 32 bits em vez de 64.

A mantissa é normalizada, com 23 dígitos. A exponencial é representada por 8 bits, com 16, como deve ser.

f. Ponto fixo e ponto flutuante

Em hardware, as unidades aritméticas trabalham com palavras de um certo número de *bits*. Muitas vezes, o comprimento da palavra é igual ao da memória principal. É mais fácil processar palavras em que o ponto de base não aparece explicitamente. Para a representação de *ponto fixo*, o ponto de base fictício fica num lugar fixo, ou à esquerda da posição mais significativa (fração) ou à direita da posição menos significativa (inteiro). Para somar ou subtrair em ponto fixo, a localização do ponto de base não faz diferença, porém, quando utilizamos multiplicação e divisão, o resultado depende da localização do ponto de base. Se a palavra é de 16 bits, a operação de multiplicação de dois operandos (de 16 bits cada) fornece um resultado bruto de 32 bits. Se os operandos forem inteiros, o resultado final será a palavra de 16 bits menos significativa. Se a palavra de 16 bits mais significativa não for zero, terá havido transbordamento (*overflow*). No caso dos operandos serem frações, o resultado final será a palavra de 16 bits mais significativas; a palavra menos significativa representa uma perda de precisão através de truncamento. A dificuldade na utilização do ponto fixo é a faixa (*range*) limitada de números que podem ser representados. Por exemplo, com 16 bits usando um bit para o sinal, só é possível representar números entre $\pm 2^{15}$ (ou $\pm 32\,768$), ou frações entre $\pm 2^{-15}$. O problema é o da mudança de escala, e a representação de ponto flutuante é utilizada para superar esse problema.

Um número N em ponto flutuante tem o valor

$$N = m \times b^e \quad (2-2)$$

Na equação anterior, m representa a *mantissa*, que pode ser ou inteira ou fracionária, b é a *base* e e é o *expoente*. Observar que ambos, mantissa e expoente, possuem sinal. A precisão em algarismos significativos é a precisão da mantissa. A faixa depende da base e do expoente. Se a mantissa é fracionária e o dígito menos significativo é não-zero, a representação é *normalizada*. Números não-normalizados podem ser normalizados através de deslocamentos à esquerda da mantissa e uma diminuição do expoente.

EXEMPLO. Números normalizados

$$\begin{aligned} 2,1416 &= 0,21416 \times 10^{+1}, \\ 1\,024 &= 0,1024 \times 10^4, \\ 0,000985 &= 0,985 \times 10^{-3}. \end{aligned}$$

EXEMPLO. Números não-normalizados com normalização

$$\begin{aligned} 0,00123 \times 10^6 &= 0,123 \times 10^4, \\ 0,0370 \times 10^0 &= 0,370 \times 10^{-1}. \end{aligned}$$

O transbordamento do expoente acontece quando o resultado de uma operação é maior que a faixa permitida. Acontece também uma situação, chamada *underflow*, em que o resultado é não-zero, mas menor do que pode ser representado.

No sistema IBM S/360, o formato para números em ponto flutuante "curto" (de 32 bits em vez de 64), é visto na Fig. 2-2.

A mantissa é considerada um número de 6 dígitos em base 16. Então os deslocamentos da mantissa para fins de normalização são de 4 bits por vez. A base do expoente é também 16, como deve ser. O expoente dispõe de 7 bits, que pode representar expoentes entre 0 e

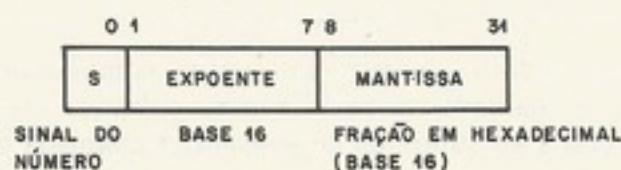


Figura 2-2. O formato de ponto flutuante curto no S/360

127. Contudo o expoente poderá também ser negativo. Para esse fim, o expoente $(0000000)_2$ representa -64 (10000000_2) e $(1111111)_2$ como $+63$. A faixa do expoente binário é, então, deslocada -64 . A faixa de variação da grandeza dos números normalizados em ponto flutuante curto do IBM S/360 vai de 16^{-65} a $(1 - 16^{-6}) \times 16^{63}$. Resultados não-nulos menores que 16^{-65} sofrem *underflow* do expoente.

Em muitas unidades centrais de sistemas não sofisticados, o conjunto de instruções só dispõe de adição e subtração em ponto fixo. Cabe ao programador, ou às sub-rotinas especiais, a implementação de multiplicação, divisão e aritmética em ponto flutuante.

2.3 ARITMÉTICA BINÁRIA

a. Números positivos

A soma de $(12)_{10}$ e $(26)_{10}$ é $(38)_{10}$, mostrada em binário na Fig. 2-3.

	32	16	8	4	2	1	PESO DA POSIÇÃO
	1	1					TRANSPORTE
12 =	0	0	1	1	0	0	
26 =	0	1	1	0	1	0	
38 =	1	0	0	1	1	0	

Figura 2-3. A soma de $(12)_{10}$ e $(26)_{10}$

Examinando cada posição, começando com a posição menos significativa (de peso 1), veremos que as parcelas "0" e "0" dão bit de soma igual a "0" e não têm transporte. As duas posições seguintes (de peso 2 e 4) indicam que as parcelas "0" e "1" resultam um bit de soma de "1", mas não têm transporte. A próxima posição (peso 8) mostra que as parcelas "1" e "1" resultam um bit de soma de "0", mas agora o transporte é "1". O transporte dessa posição é chamado de "vai um" (*carry-out*) dessa posição. Esse transporte influí na soma da posição vizinha onde é chamado de "vem um" (*carry-in*). Agora, com as parcelas de "0" e "1" mais o "vem um", o bit de soma dessa posição (peso 16) é "0", mas o transporte é propagado à próxima posição (peso 32). As regras de adição de números positivos em binário para uma posição determinada, seguem a tabela da verdade da Tab. 2-3.

Tabela 2-3. Tabela da verdade, adição binária

Primeira parcela	Segunda parcela	"Vem um"	Soma	"Vai um"
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

A implementação em circuitos lógicos do circuito combinatório de três entradas e duas saídas da Tab. 2-3 é chamada de circuito *soma completa* (*full adder*).

Na subtração, a parcela a ser subtraída do minuendo é chamada de *subtraendo*. O transporte é chamado de *emprestimo*, mas continuaremos a usar "vai um" e "vem um". A tabela de verdade para subtração é mostrada na Tab. 2-4.

Minuendo
0
0
0
0
1
1
1
1
1

Observamos que o resultado é completo. A diferença é mostrada a seguir.

Sabemos que a adição de dois números positivos resulta em um resultado que é sempre completo. A diferença é mostrada a seguir.

b. A técnica da subtração

No tratamento da subtração, usaremos a operação de subtração complementada. Por exemplo, se subtraímos B de A , obtemos o resultado da subtração ($A - B$).

A aritmética binária é feita da mesma forma que a decimal.

Nessa equação, "[...] e 2.2(e)] o número é negativo de A (ou B). É importante lembrar que o bit significativo da subtração é sempre 1.

EXEMPLO. $x = 10101010$

Podemos mostrar que

Tabela 2-4. Tabela da verdade, subtração binária

Minuendo	Subtraendo	"Vem um"	Diferença	"Vai um"
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Observamos que a função diferença é o *exclusive OR* das três entradas, como no somador completo. A diferença entre um minuendo positivo menor que um subtraendo é mostrada a seguir.

$$\begin{array}{r}
 & 4 & 1 & 4 & \\
 & \text{---} & & \leftarrow \text{EMPRESTIMO} \\
 12: & 0 & 0 & 1 & 1 & 0 & 0 \\
 -26: & 0 & 1 & 1 & 0 & 1 & 0 \\
 \hline
 & 1 & 1 & 0 & 0 & 1 & 0
 \end{array}$$

Figura 2-4. A diferença de $(12)_{10}$ e $(26)_{10}$

Sabemos que a diferença é $(-14)_{10}$ e notamos que $(-14)_{10}$ em complemento de dois com números de 6 bits é $(110010)_2$, como aparece na Fig. 2-4. Isso significa que as Tabs. 2-3 e 2-4 servem para números positivos com resultados positivos e, para tratar com números negativos precisamos derivar as respectivas regras de operação.

b. A técnica da complementação

No tratamento de adição e subtração com números positivos e negativos, é costume usar-se a operação de adição (através de um somador) junto com a operação de complementação. Por exemplo, para subtrair B de um número A , basta somar A a B com o sinal trocado ($-B$):

$$A - B = A + (-B). \quad (2-3)$$

A aritmética binária de n bits é baseada na equação

$$A + [A]' = 2^n - 1 = \underbrace{11\ldots1}_n \quad (2-4)$$

Nessa equação, "11...1" significa uma cadeia de n 1 e $[A]'$ indica [como nas Secs. 2.2(d) e 2.2(e)] o número binário A com todos os bits complementados. Observar que $[A]'$ é o negativo de A (ou seja, $-A$) em complemento de um.

É importante observar que o bit de sinal também entra nos cálculos como o bit mais significativo da palavra.

EXEMPLO. $n = 6$

$$\begin{aligned}
 A &= 000110 \\
 [A]' &= 111001 \\
 A + [A]' &= \overline{111111} = 2^6 - 1.
 \end{aligned}$$

Podemos mostrar um corolário,

$$[A]' = 2^n - 1 - A. \quad (2-5)$$

c. Aritmética usando complemento de um do subtraendo.

O corolário (equação anterior) é utilizado para se fazer economicamente subtração com equipamento para somar que possua a capacidade de complementar o subtraendo bit a bit (complemento de um).

Suponha que números A e B são positivos, de 6 bits cada [1 bit (o mais significativo) de sinal e 5 bits de grandeza] e desejamos obter a diferença $A - B$ (que chamaremos de R) usando a operação de adição. Aplicando as Eqs. (2-3) e (2-5)

$$R = A + [B]' \equiv A + 2^n - 1 - B \quad (2.6)$$

Notamos que $A + [B]'$ difere com $A - B$ pela quantia $(2^n - 1) = 11 \dots 1$. Se a diferença for positiva, o resultado de $A + [B]'$ virá acompanhado por um "vai um transbordo" que tem peso 2^n ou, nesse caso, $(64)_{10}$, e então os 6 bits de R representam $A - B - 1$. O "vai um transbordo" acontece porque o bit de sinal de A é "0" (pois A é positivo) e o bit de sinal de $[B]'$ é "1" (pois B é positivo e o bit de sinal foi complementado). Então, se o sinal do resultado é "0" através de uma soma, só poderia ter ocorrido um "vem um" na posição do sinal. Assim sendo, ocorreu um "vai um transbordo" (veja a Fig. 2-5). Para se corrigir o resultado, é necessário somar mais um na posição menos significativa (veja a Fig. 2-6).

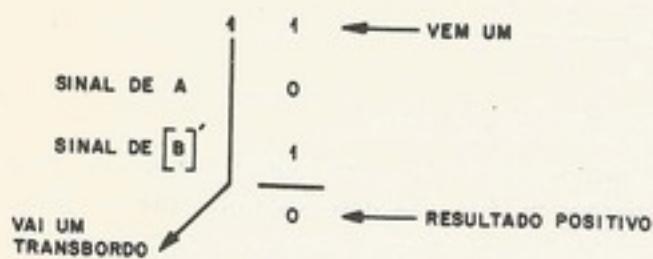


Figura 2-5. "Vai um transbordo" durante subtração usando somador e complementação

$$\begin{array}{r}
 A = 0 \ 0 \ 1 \ 1 \ 0 \ 1 \quad (13)_{10} \\
 B = 0 \ 0 \ 1 \ 0 \ 1 \ 1 \quad (14)_{10} \\
 \text{COMPLEMENTAÇÃO: } [B]' = 1 \ 1 \ 0 \ 1 \ 0 \ 0 \\
 A + [B]' = 0 \ 0 \ 1 \ 1 \ 0 \ 1 \\
 \hline
 1 \ 1 \ 0 \ 1 \ 0 \ 0 \\
 \hline
 0 \ 0 \ 0 \ 0 \ 0 \ 1 \\
 + \qquad \qquad \qquad 1 \quad \text{MAIS "1"} \\
 \hline
 \text{RESULTADO CERTO} \rightarrow \qquad 0 \ 0 \ 0 \ 0 \ 1 \ 0 \quad = (2)_{10}
 \end{array}$$

Figura 2-6. Exemplo de subtração em complemento de um

Vamos supor agora que o resultado seja negativo. Se B for maior que A , então $B - A$ será um número positivo e o resultado $R = A - B$ deve ser o complemento de um de " $B - A$ "; ou seja, deve ser $[B - A]'$. Segundo o Corolário (2-5), obtemos a Eq. (2-7), que é igual à Eq. (2-6).

$$R = [B - A]' = 2^n - 1 - B + A = A + 2^n - 1 - B \quad (2.7)$$

Então, quando o resultado é negativo, não há “vai um transbordo” e o resultado não necessita correção (veja a Fig. 2-7).

Resumindo, durante subtração, quando não há “vai um transbordo”, o resultado está certo; caso contrário, é necessário somar-se um ao resultado. Esse processo, na implementação, é chamado de realimentação ou *retorno de transporte* (*end-around carry*). Assim, o sinal de “vai um transbordo” sempre alimenta o “vem um” da posição menos significativa do somador. Quando o “vai um transbordo” é “0”, nada acontece e, quando o “vai um transbordo” é “1”, o resultado é corrigido.

SINAL DE A
SINAL DE B
NÃO TEM
VAI UM TRANSBORDO

EXEMPLOS

Foi mostrado que o circuito do processo de adição

EXEMPLO:

Neste exemplo, obser-

EXEMPLO Família

Neste exemplo, basta

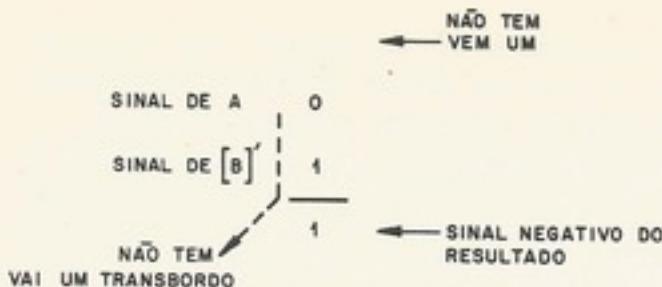


Figura 2-7. Subtração de dois positivos sem "vai um transbordo"

(2-6)

uma diferença for
"transbordo" que tem
"vai um trans-
de sinal de $[B]'$
sinal do resultado
do sinal. Assim
o resultado, é
.

"transbordo" durante
complemento-

EXEMPLOS

$$\begin{array}{r}
 1111 \\
 A = 001101 = (13)_{10}, \quad B = 010000 = (16)_{10} \\
 [B]' = 101111 \\
 \hline
 111100 \\
 \text{returno: 0} \\
 \hline
 111100 = (-3)_{10} \\
 1111 \ 1 \\
 A = 001101 = (13)_{10}, \quad B = 001010 = (10)_{10} \\
 [B]' = 110101 \\
 \hline
 000010 \\
 \text{returno: 1} \\
 \hline
 000011 = (+3)_{10}.
 \end{array}$$

Foi mostrado que o somador binário pode servir para subtração. Tendo-se em vista que o circuito do somador foi projetado para adicionar números positivos, será que esse processo de adição binária serve para somar um número positivo com um número negativo?

EXEMPLO. Resultado negativo

$$\begin{array}{r}
 1 \\
 A = 000111 \quad (7)_{10} \\
 B = 110100 \quad (-11)_{10} \\
 \hline
 111011 \quad (-4)_{10}.
 \end{array}$$

Neste exemplo, observar que o processo não deu "vai um transbordo" e o resultado é válido. A situação é descrita pela Eq. (2-7).

EXEMPLO. Resultado positivo

$$\begin{array}{r}
 1111 \\
 A = 001111 \quad (15)_{10} \\
 B = 110100 \quad (-11)_{10} \\
 \hline
 000011 \quad (3)_{10} \quad \text{resultado intermediário} \\
 \text{returno: 1} \\
 \hline
 000100 \quad (4)_{10} \quad \text{resultado certo.}
 \end{array}$$

Neste exemplo, houve "vai um transbordo", corrigido pelo retorno do transporte.

Observe que esse processo funciona tanto para adição quanto para a subtração de números positivos ou números negativos em representação complemento de um.

EXEMPLO

$$\begin{array}{r}
 (-9)_{10} - (3)_{10} \\
 1111 \\
 A = 110110 = (-9)_{10} \quad B = 000011 = (3)_{10} \\
 [B]' = \underline{111100} \\
 110010 \\
 \text{returno: 1} \\
 \hline
 110011 = (-12)_{10}.
 \end{array}$$

Acontece que, com esse esquema, subtraindo-se um número de si próprio, resulta -0 .

EXEMPLO

$$\begin{array}{r}
 (-9)_{10} - (-9)_{10} \\
 A = 110110 \\
 [A]' = 001001 \\
 \hline
 111111 \\
 \text{returno: 0} \\
 \hline
 111111 = (-0).
 \end{array}$$

Existe ainda um problema. Somando-se dois números de sinais iguais ou tomando-se a diferença de dois operandos com sinais diferentes, existe a possibilidade de a grandeza do resultado ser maior que $2^n - 1$ [ou menor que $-(2^n - 1)$]. Essa situação é chamada de estouro ou transbordamento (*overflow*). Existe uma regra simples para se descobrir um estouro: é quando o “vem um” que entra na posição do sinal não é igual ao “vai um transbordo”.

EXEMPLO. A e B positivos

$$\begin{array}{r}
 1 \\
 A = 010000 = + (16)_{10} \\
 B = + 010000 = (16)_{10} \\
 \hline
 100000 = (-31)_{10} \text{ (errado).}
 \end{array}$$

Neste exemplo, o “vem um” é igual a 1 e o “vai um transbordo” é igual a “0”. Sendo A e B negativos, o estouro acontece quando o “vem um” é igual a “0” e o “vai um transbordo” é igual a “1”, como no exemplo seguinte.

EXEMPLO. A e B negativos

$$\begin{array}{r}
 1 \\
 A = 100000 = (-31)_{10} \\
 B = + 110111 = (-8)_{10} \\
 \hline
 010111 \\
 1 \\
 \hline
 011000 = (+24)_{10} \text{ (errado).}
 \end{array}$$

Infelizmente, a regra não funciona em um caso: quando uma parcela é -0 e a outra é $-2^{n-1} + 1$. O exemplo com $n = 6$ segue.

EXEMPLO. Problema com -0

$$\begin{array}{r}
 1 \\
 100000 = (-31)_{10} \\
 111111 = (-0)_{10} \\
 \hline
 011111 \\
 1 \\
 \hline
 100000
 \end{array}$$

Neste exemplo, normalmente esse fluxograma é mostrado na Figura 4.10.

Figura 4.10

d. Aritméticas

Nesta seção, simplesmente o parâmetro é um bit.

Os processos de subtração e adição são quase iguais, com exceção de um complemento que é usado.

Neste exemplo, deu sinal de transbordamento (*overflow*) apesar de certo. Esse problema é normalmente resolvido através de circuitos especiais para evitar o -0 .

O fluxograma para aritmética (adição e subtração) binária em complemento de um é mostrado na Fig. 2-8.

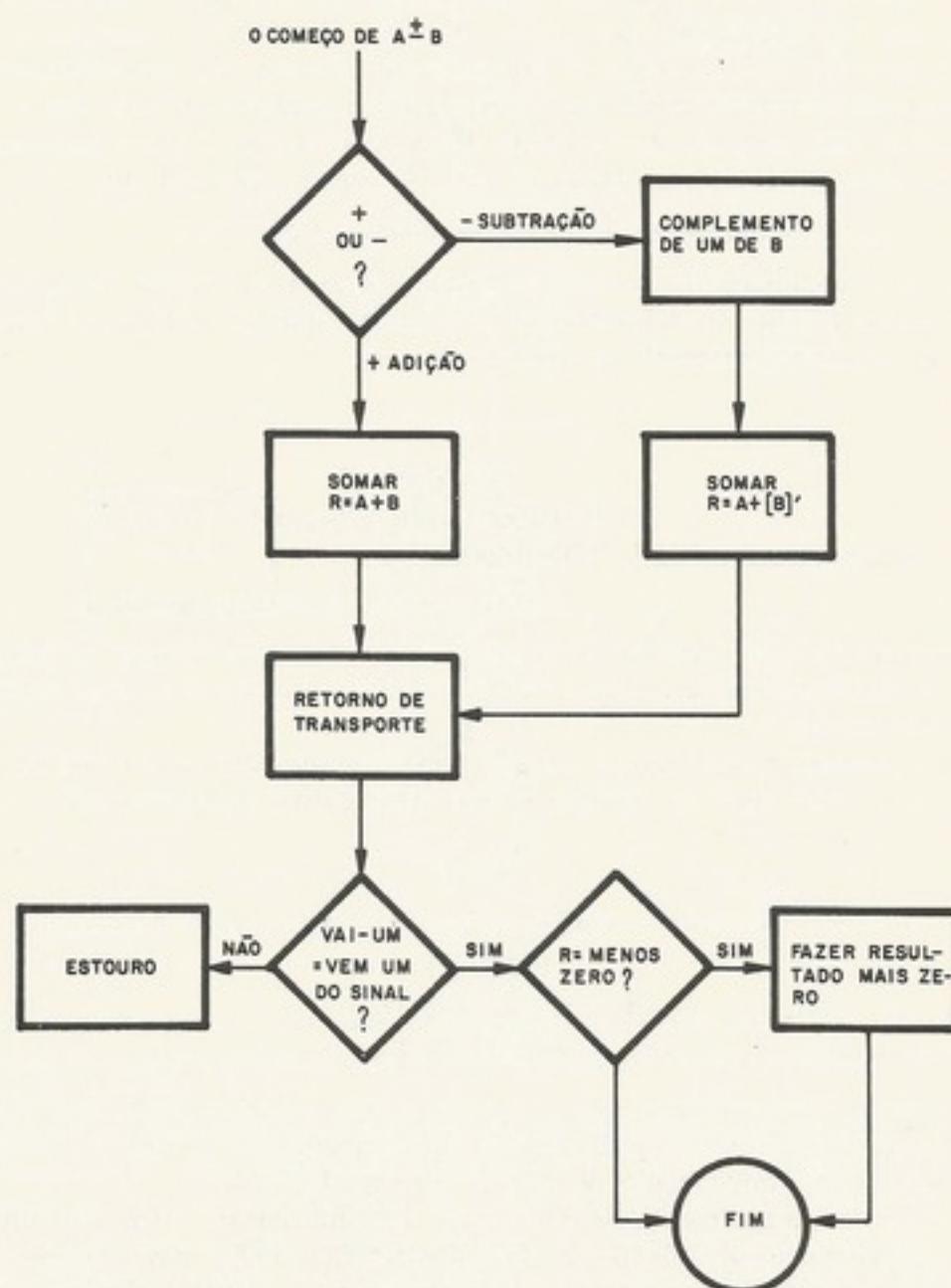


Figura 2-8. Fluxograma de aritmética binária em complemento de um

d. Aritmética usando complemento de dois

Nesta seção, como na anterior, queremos tratar a aritmética sob o aspecto de utilizar-se simplesmente o processo de adição binária e o processo de complementar um número *bit a bit*.

Os processos aritméticos com números negativos representados em complemento de dois são quase iguais aos do complemento de um. Uma diferença existe na equação do complemento que é alterada para

$$[A]' = 2^n - A. \quad (2-8)$$

Agora, para subtrair fazendo a soma com o minuendo e o complemento do subtraendo, obtemos

$$R = A + [B]' = A + (2^n - B) = A - B + 2^n. \quad (2-9)$$

Se $A - B$ for positivo, então 2^n mais uma quantidade 2^e irá estourar a capacidade de n bits, que significa um "vai um transbordo". Caso contrário, se $A - B$ for negativo, o resultado é $2^n - (B - A)$, que é a representação da grandeza da diferença em complemento de dois, ou seja, o resultado correto. A aritmética binária em complemento de dois, tem a vantagem de não necessitar uma correção do resultado. Uma desvantagem é que o complemento de dois necessita, além da complementação de bit a bit, uma soma por um,

$$(\text{complemento de } 2) = (\text{complemento de } 1) + 1. \quad (2-10)$$

Na prática, o complemento de dois é efetuado através da técnica "vem um quente" (*hot carry*), durante o processo de subtração. O "vem um quente" (*VUQ*) é o "vem um" da posição do bit menos significativo da palavra.

EXEMPLO

$$\begin{array}{r} & 1 \leftarrow VUQ \\ A = 001000 & = (8)_{10} \quad B = 001011 = (11)_{10} \\ B \text{ (complemento de 1)} = 110100 \\ \hline 111101 & = (-3)_{10} \text{ (complemento de 2).} \end{array}$$

A combinação de complementar bit a bit o subtraendo e de "forçar" o "vem um" da posição menos significativa (que possui o peso de um) tem o efeito de se fazer complemento de dois do subtraendo.

Quando o processo é de adição, a complementação é desnecessária e não há "vem um quente", isto é, o "vem um" da posição menos significativa permanece "0".

EXEMPLO

$$\begin{array}{r} 11 \\ A = 110000 = (-16)_{10} \\ B = \underline{111111} = (-1)_{10} \\ \hline 101111 = (-17) \end{array}$$

No exemplo, o "vai um transbordo" é ignorado, pois é igual ao "vem um" da posição do sinal. A deteção de estouro é igual ao caso do complemento de um. O fluxograma de aritmética binária em complemento de dois é visto na Fig. 2-9.

A aritmética em complemento de dois tem a desvantagem de o valor de um número negativo em complemento de dois não ser facilmente reconhecido; mas a aritmética dispõe de certas vantagens sobre o complemento de um. Por exemplo, o problema de -0 não existe. O -0 pode ser inconveniente para instruções que testam o sinal porque o resultado será ambíguo se o número for zero. Além do mais, o -0 cria problemas quando somado com $(-2^{n-1} + 1)$. O processo do retorno do transporte é inconveniente quando a adição é feita seriamente em etapas; por exemplo, quando a palavra tem 16 bits e o somador é de 4 bits. Tendo retorno de transporte, o processo volta ao começo.

c. Aritmética com números em sinal e amplitude

A aritmética com números em sinal e amplitude pode aproveitar-se as técnicas de complemento de um ou de complemento de dois para subtração. Primeiro é importante observar que o processo de obtenção da diferença das grandezas ocorre quando se estão adicionando números de sinais diferentes ou subtraindo-se números de sinais iguais. Por

isso, antes da adição, as grandezas devem ter o mesmo complemento de um.

O estouro só ocorre quando "vai um transbordo". Mas, quando se está subtraindo, o que se deve fazer é complementar o subtraendo. De modo similar, se se está somando, o que se deve fazer é complementar o somando.

EXEMPLO. Subtração

O processo é:

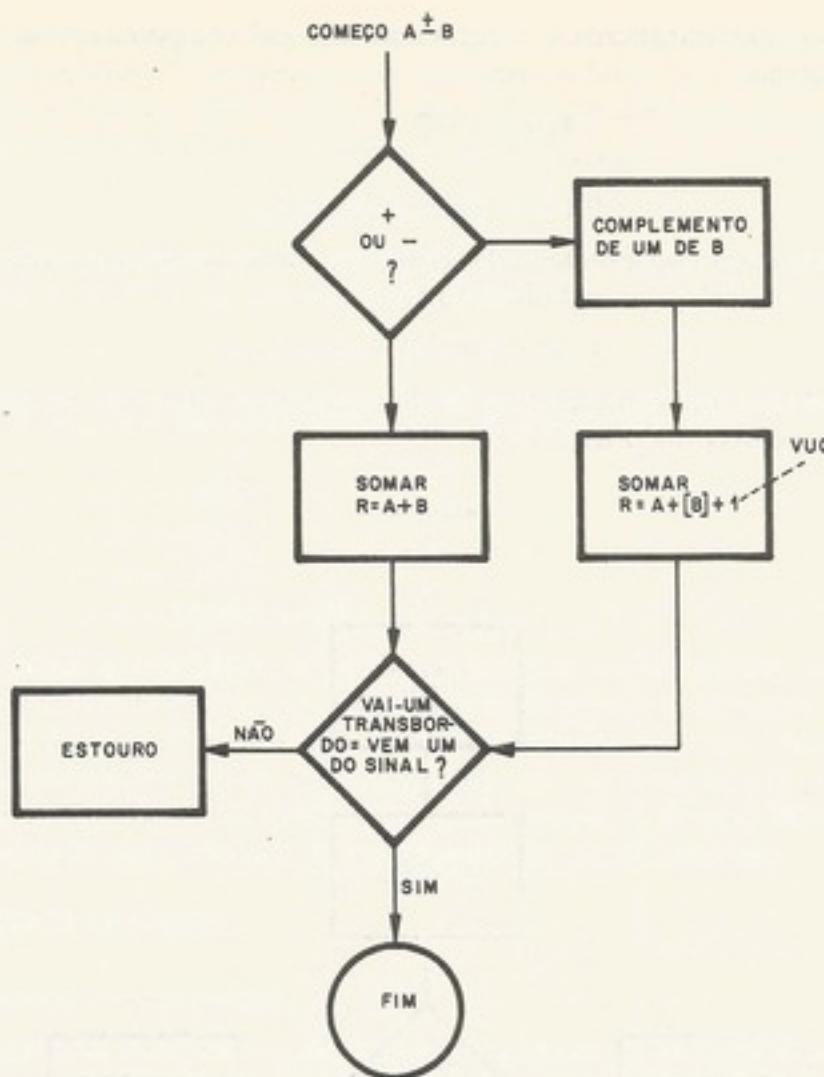


Figura 2-9. Fluxograma de aritmética binária em complemento de dois

isso, antes da aritmética, é necessário examinar os sinais das grandezas. Portanto somente as grandezas entram na aritmética (em contraste com sistemas de complemento de um ou complemento de dois).

O estouro só pode acontecer na obtenção da soma das grandezas, e é indicado pelo "vai um transbordo". Na obtenção da diferença entre grandezas, é impossível haver estouro. Mas, quando se subtrai uma grandeza da outra, pode ocorrer a necessidade do *recomplementar*, o que se indica pela ausência do "vai um transbordo". Usaremos complemento de dois no exemplo a seguir, com $n = 6$, onde o bit mais significativo é o sinal e não entra na aritmética.

EXEMPLO. Soma A e B , números em sinal e amplitude

$$A = 000011 = (3)_{10}, \quad B = 101101 = (-13)_{10}$$

O processo é realmente de subtrair, sendo B negativo,

$$\begin{array}{r} 110 \leftarrow VUQ \\ A = 00011 \\ [B]' = \underline{10010} \\ \hline 10110 \end{array}$$

Faltando "vai um transbordo", o resultado está em complemento de dois e tem de ser recomplementado,

$$\begin{array}{r} 1\text{ } \leftarrow VUQ \\ 01001 \\ \hline 01010 = \text{amplitude } (10)_{10}. \end{array}$$

A recomplementação só é necessária quando o sinal da diferença difere do sinal do minuendo; então o sinal do resultado é "1",

$$A - B = 101010 \leftarrow (-10)_{10}.$$

Também podemos usar complemento de um para subtrair números em sinal e amplitude. Esse fluxograma aparece na Fig. 2-10.

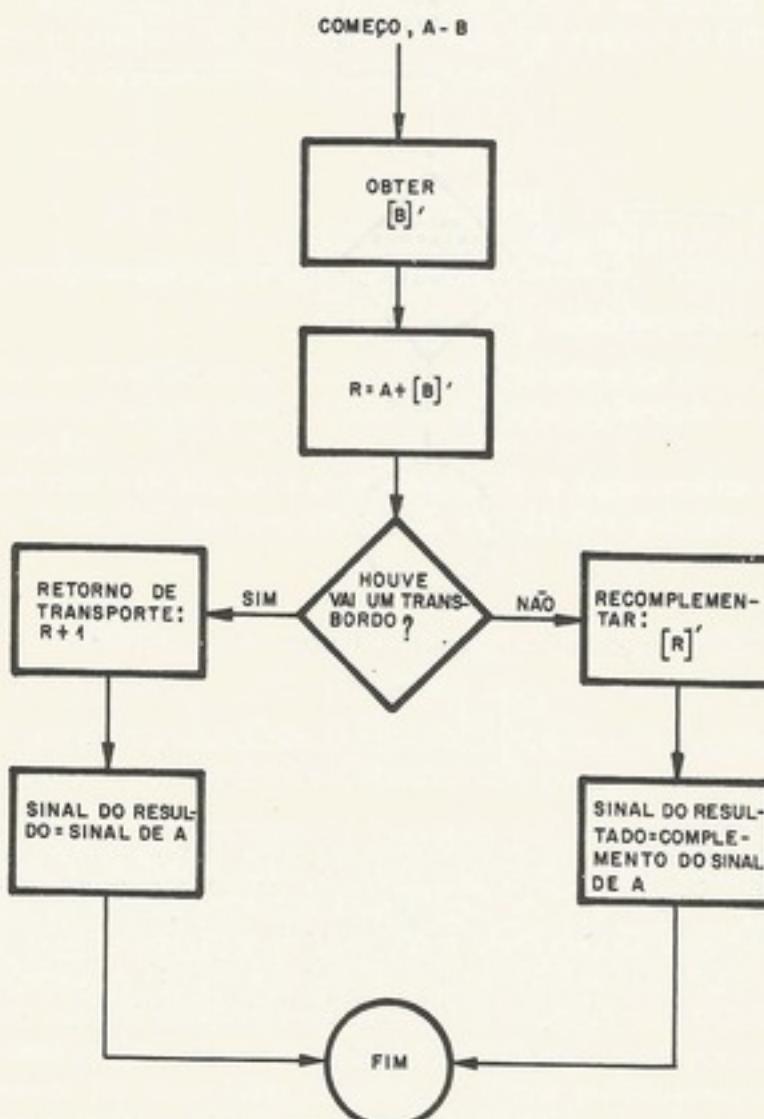


Figura 2-10. Subtração com números binários em sinal e amplitude usando-se complemento de um

f. Variações

Até agora, o nosso interesse foi aproveitar a adição binária com a técnica de complementar o subtraendo, para realizar subtrações.

Descobrimos que, com o processo de adição associado à complementação de um operando bit a bit, obtemos a generalidade de somar ou subtrair números positivos ou negativos. Obteremos essa mesma generalidade se usarmos a técnica de complemento do

minuendo⁽⁵⁾. Essa técnica pode ser complementada bit a bit A antes de se desejar; então compre-

Não examinaremos os vários exemplos, e ve-

EXEMPLO

Também existe a subtração bit a bit. Para somar ou subtrair, depende ou "retorno de empréstimo". basta dizer que opera com UCP projetada

2.4 ARITMÉTICA II

a. Representações

A aritmética com existem as representações. O complemento de um de um dígito, isto é, subtração de dígito "6" é "3", de dígitos do código XS-3 (negativo) a bit do código para o é "1001". Essa propriedade goza dessa vantagem de complementar um operando.

O complemento de um é isso porque para um número de nove vale $(10^3 - 1) = 999$ em sistemas decimais, ne-

EXEMPLO

b. Exemplos de subtração

Como em sistemas de base problema de -0 e da subtração

minuendo⁽⁵⁾. Essa técnica é usada para decrementar por um a conta de um contador que pode ser complementado. Suponhamos que seja para subtrair $A - B$. Se complementarmos bit a bit A antes de somar, obtemos $R = (-A + B)$. O resultado parece ser o negativo do desejado; então complementarmos bit a bit o resultado,

$$-R = -(-A + B) = A - B. \quad (2-11)$$

Não examinaremos esse assunto em profundidade, mas o leitor pode experimentar com vários exemplos, e verificar as características da técnica.

EXEMPLO

$$\begin{array}{r} A = (001110)_2 = (14)_{10}, \quad B = (000101)_2 = (5)_{10} \\ \hline -A = 110001 \\ B = 000101 \\ \hline R = 110110 \\ -R = 001001 = (9)_2. \end{array}$$

Também existe generalidade se podemos subtrair e, juntamente, complementar bit a bit. Para somar com um subtrator, basta só complementar o operando que serve como subtraendo. Dependendo da representação de negativos, ou ocorrerá "empréstimo quente" ou "retorno de empréstimo". Essa técnica também não será examinada em profundidade, basta dizer que computadores de médio porte (por exemplo, o Prodac 580 da Westinghouse com UCP projetada pela Univac) foram construídos neste princípio.

2.4 ARITMÉTICA DECIMAL

a. Representação dos negativos

A aritmética com números em decimal é completamente análoga à aritmética binária. Existem as representações sinal e amplitude, complemento de nove, e complemento de dez. O complemento de nove, análogo ao complemento de um, é feito complementando-se cada dígito, isto é, subtraindo-se cada dígito de nove. Por exemplo: o complemento de nove do dígito "6" é "3", do dígito "0" é "9", e do dígito "2" é "7". Agora podemos ver uma vantagem do código XS-3 (veja a Tab. 2-1): o complemento de nove do dígito é o complemento bit a bit do código para o dígito. Por exemplo: o código de "3" é "0110" e o complemento "6" é "1001". Essa propriedade do código é chamada *autocomplementação*. O código BCD não goza dessa vantagem, mas, como veremos, podemos também aproveitar a capacidade de complementar um operando em BCD bit a bit.

O complemento de dez de um número decimal é o complemento de nove mais um; isso porque para um número A de n dígitos (excluindo-se o sinal) o seu complemento de nove vale $(10^n - 1 - A)$ e seu complemento de dez vale $(10^n - A)$. O sinal, muitas vezes, em sistemas decimais, necessita um dígito para si.

EXEMPLO

$$\begin{array}{r} A = +103 \\ -A \text{ (complemento de 9)} = -896 \\ -A \text{ (complemento de 10)} = -897. \end{array}$$

b. Exemplos de aritmética decimal

Como em sistemas binários, a aritmética decimal em complemento de nove tem o problema de -0 e do "retorno do transporte" quando se utiliza o somador para subtrair.

Sistemas em complemento de dez, para subtração, podem aproveitar a técnica de fazer o complemento de nove para o subtraendo e forçar o "vem um quente" na posição menos significativa. Sistemas decimais usando a representação sinal e amplitude podem subtrair usando o complemento de dez ou complemento de nove do subtraendo, como se quiser, mas, de qualquer maneira, se o resultado for negativo, será necessário *recomplementá-lo*.

EXEMPLO. Complemento de nove

$$A = +0367, \quad B = -9831 = (-0168)_{10}$$

$$\begin{array}{r} 1 \\ +0367 \\ -9831 \\ \hline +0198 \\ \hline 1 \\ +0199 \end{array} \quad (367 - 168 = 199).$$

O sinal pode ser tratado como $+ = 0$ e $- = 1$.

$$B - A$$

$$\begin{array}{r} 111 \\ -9831 \\ -9632 \quad (\text{complemento de } 9 \text{ de } A) \\ \hline -9463 \\ \hline 1 \\ -9464 \quad (-168 - 367 = -535). \end{array}$$

EXEMPLO. Complemento de dez

$$A = +0367, \quad B = -9832 = (-0168)_{10}$$

$$A + B$$

$$\begin{array}{r} 111 \\ +0367 \\ -9832 \\ \hline +0199 \end{array}$$

$$B - A$$

$$\begin{array}{r} 111 \quad \text{①} \leftarrow \text{"vem um quente" (VUQ)} \\ -9832 \\ -9632 \quad (\text{complemento de } 9 \text{ de } A) \\ \hline -9465 \quad (-168 - 367 = -535). \end{array}$$

EXEMPLO. Sinal e amplitude

$$A = +0367, \quad B = -0168$$

$A + B$: a operação é de subtração; então

$$A = 0367$$

$$\text{complemento de } 9 \text{ de } B = \begin{array}{r} 9831 \\ +0199 \end{array}$$

Aqui, o "vem um" na posição do sinal indica que o sinal do resultado é o sinal do minuendo.

$$B + A$$

$$\begin{array}{r} 110 \leftarrow \text{"vem um quente"} \\ 0168 \\ 9632 \\ \hline 9801 \end{array}$$

códigos, números

A ausência de que o resultado

c. Aritmética

Como em dos operandos das amplitudes possibilidade de O sistema "ilegais". Quais casos:

O caso I quando o resultado é 19. O caso II é tado. No caso

EXEMPLO

$A + B$
Primeira etapa

Para o caso para o caso II Segunda etapa

resultado certo. Uma desvia "vai um" com a EXEMPLO

Primeira etapa

A ausência do sinal indica que o sinal do resultado não é o sinal do minuendo, significando que o resultado tem de ser recomplementado:

$$\begin{array}{r} \text{complemento de 9: } \quad 0198 \\ + 0199 \\ \hline \text{resultado certo.} \end{array}$$

① ← “vem um quente”

c. Aritmética em BCD usando representação sinal e amplitude

Como em números binários para sinal e amplitude, é necessário examinar os sinais dos operandos, antes de fazer aritmética, para descobrir se a operação é basicamente a soma das amplitudes ou a diferença delas. No caso de somar amplitudes como sempre, tem a possibilidade de estouro de capacidade do resultado.

O sistema *BCD* usa 10 das combinações disponíveis com 4 bits, sendo as outras seis “ilegais”. Quando somamos dois dígitos *BCD* em binário, o resultado, pode estar em três casos:

- caso 1, dígito legal sem “vai um”;
- caso 2, dígito ilegal sem “vai um”;
- caso 3, dígito legal com “vai um”.

O caso 1 acontece quando o dígito do resultado é nove ou menos. O caso 2 acontece quando o resultado fica entre 10 e 15, e o caso 3 acontece quando o resultado cai entre 16 e 19. O caso 1 não precisa de correção, mas para os casos 2 e 3 precisamos corrigir o resultado. No caso 2, também precisamos propagar o “vai um”.

EXEMPLO

$$A = 0832, \quad B = 0983$$

$$A + B$$

Primeira etapa

$$\begin{array}{r} & & 1 \\ A = 0000 & 1000 & 0011 & 0010 \\ B = 0000 & 1001 & 1000 & 0011 \\ \hline 0001 & 0001 & 1011 & 0101 \\ \text{caso 3} & \text{caso 2} & \text{caso 1} \end{array}$$

Para o caso 2, precisamos somar 6, para corrigir o dígito e propagar o “vai um”; e, para o caso 3, só precisamos somar 6, pois o “vai um” já foi propagado.

Segunda etapa

$$\begin{array}{r} 1111 \quad 11 \\ 0001 \quad 0001 \quad 1011 \quad 0101 \\ 0110 \quad 0110 \\ \hline 0001 \quad 1000 \quad 0001 \quad 0101 \end{array}$$

resultado certo, 1815.

Uma desvantagem da correção do resultado dessa maneira é que o tratamento do “vai um” com o caso 2 pode criar correções não previstas na primeira etapa.

EXEMPLO

$$A = 0372, \quad B = 0633$$

Primeira etapa

$$\begin{array}{r} 11 \quad 111 \quad 1 \\ 0000 \quad 0011 \quad 0111 \quad 0010 \\ 0000 \quad 0110 \quad 0011 \quad 0011 \\ \hline 0000 \quad 1001 \quad 1010 \quad 0101 \\ \text{caso 2} \end{array}$$

Segunda etapa

$$\begin{array}{r}
 & 11 & 11 \\
 \begin{array}{r} 0000 \\ + 0101 \\ - 0110 \end{array} & \hline
 & 0000 & 1010 & 0000 & 0101 \\
 & \text{caso 2} & & &
 \end{array}$$

Para evitar esse tratamento especial, existe um outro algoritmo (veja Hellerman⁽¹¹⁾), o da soma de 6 em todos os dígitos de um dos operandos antes da soma das duas parcelas. Assim, só existem dois casos, distinguidos pelo "vai um" do dígito:

- caso 1: o resultado não deu "vai um" e então caiu entre 6 e 15 — tem de subtrair 6;
 caso 2: o resultado deu "vai um"; então o "vai um" já foi propagado e o dígito está correto entre 0 e 9.

EXEMPLO

$$A = 0372, \quad B = 0633$$

Primeira etapa

$$\begin{array}{r}
 & 11 & 11 & 11 \\
 \begin{array}{r} A = 0000 \\ + 0110 \\ - 0110 \end{array} & \hline
 & 0110 & 1001 & 1101 & 1000 \\
 & \text{caso 1} & \text{caso 2} & \text{caso 2} & \text{caso 1}
 \end{array}$$

Se a primeira etapa der "vai um" entre dígitos, o dígito original do operando terá sido ilegal.

Segunda etapa

$$\begin{array}{r}
 & 1 & 1111 & 111 \\
 \begin{array}{r} A + 6 = 0110 \\ + 0000 \\ - 0111 \end{array} & \hline
 & 0111 & 0000 & 0000 & 1011 \\
 & \text{caso 1} & \text{caso 2} & \text{caso 2} & \text{caso 1}
 \end{array}$$

Terceira etapa (em vez de subtrair "0110", somamos "1010" e ignoramos o "vai um").

$$\begin{array}{r}
 & 11 & & 1 \\
 & 0111 & 0000 & 0000 & 1011 \\
 & 1010 & & & 1010 \\
 \hline
 & 0001 & 0000 & 0000 & 0101
 \end{array}$$

resultado certo, 1005.

Para obter a diferença entre duas amplitudes decimais em BCD, podemos usar a técnica de complementar o dígito BCD bit a bit. Se o dígito do minuendo for maior que o dígito do subtraendo, a diferença em binário cairá entre 0 e 9 e não haverá problema de correção. Como no caso binário, o "vai um" transbordo indica que não precisa recomplementar.

EXEMPLO. Complemento de dez

$$A = 9865, \quad B = 3112$$

$$A - B$$

$$\begin{array}{r}
 & 1 & 11 & 1 & 11 & 1 & 1 & 1 \oplus \leftarrow \text{"vem um quente"} \\
 \begin{array}{r} A = 1001 \\ + 1100 \\ - 0110 \end{array} & \hline
 & 0110 & 0111 & 0101 & 0011 \\
 & 6 & 7 & 5 & 3
 \end{array}$$

O resultado 6753 está certo e não precisa de correção. Em cada posição, se o dígito do subtraendo for menor, a soma do dígito do minuendo e o complemento do dígito do subtraendo

códigos, números

sempre dá "vai um" que implementa

EXEMPLO

Primeira etapa

Segunda etapa

resultado certo.
Nos casos certos
e precisamos subtrair

EXEMPLO

O resultado 4007
precisamos subtrair

A Fig. 2-11 mostra
números em símbolos
mendamos os resultados

2.5 CONVERSÃO

Em alguns sistemas
para fazer conversões
supomos que os números
em código BCD

sempre dá “vai um” do dígito. Caso contrário, o dígito necessita a correção de subtrair “6”, que implementamos no exemplo pela técnica de somar dez ($(1010)_2$) e ignorar o “vai um”.

EXEMPLO

$$A = 9865, \quad B = 3972$$

Primeira etapa

$$\begin{array}{r} & 1 & & 1 & 1 \\ A = & 1001 & 1000 & 0110 & 0101 \\ [B]' = & 1100 & 0110 & 1000 & 1101 \\ & \hline & 0101 & 1110 & 1111 & 0011 \end{array} \quad \text{①} \leftarrow \text{“vem um quente” (VUQ)}$$

Segunda etapa

$$\begin{array}{r} 1010 & 1010 \\ \hline 0101 & 1000 & 1001 & 0011 \end{array}$$

resultado certo, 5893

Nos casos em que não há “vai um transbordo”, o resultado está na forma complementar, e precisamos recomplementar.

EXEMPLO

$$\begin{array}{r} A = 3972, \quad B = 9865 \\ & 1111 & 1111 & 111 & 1 \\ A = & 0011 & 1001 & 0111 & 0010 \\ [B]' = & 0110 & 0111 & 1001 & 1010 \\ & \hline & 1010 & 0001 & 0000 & 1101 \\ & 1010 & & & 1010 \\ R = & 0100 & 0001 & 0000 & 0111 \end{array} \quad \text{①} \leftarrow \text{VUQ}$$

O resultado 4107 é o complemento de dez do resultado certo, 5893. Para recomplementar, precisamos subtrair 4107 de zero:

$$\begin{array}{r} & & & \text{①} \leftarrow \text{VUQ} \\ & 0000 & 0000 & 0000 & 0000 \\ [R]' = & 1011 & 1110 & 1111 & 1000 \\ & \hline & 1011 & 1110 & 1111 & 1001 \\ & 1010 & 1010 & 1010 & 1010 \\ & \hline & 0101 & 1000 & 1001 & 0011 \\ & 5 & 8 & 9 & 3 \end{array}$$

A Fig. 2-11 representa um resumo das técnicas de fazer-se aritmética BCD com os números em sinal e amplitude. Para demais informações sobre números e aritmética, recomendamos os textos das referências⁽⁶⁾ e⁽⁷⁾.

2.5 CONVERSÃO ENTRE NÚMEROS BINÁRIOS E DECIMAS

Em alguns sistemas, como o IBM S/360, existem instruções específicas em hardware para fazer conversões; mas a maioria dos sistemas usa sub-rotinas para esse fim. Aqui, supomos que os números sejam inteiros e que a representação dos dígitos decimais esteja em código BCD.

EXEMPLO. Conversão

1) 11
2) 10
3) 00

a conversão: 0011

2)

Se não houver (veja Couleur¹⁰). O decimal pode ser

1. Começar com
2. Se o segundo
3. Se o segundo
4. Continuar

EXEMPLO

Começando com bit é "1". O terceiro é "1"; então temos 1. O sexto bit é "0"; 0 (dobrando, + 1). O processo é justificável.

$$A_0 2^{n-1} + \dots$$

Um somador de sinal e amplitude pode ser usado para apenas somar o resultado das parcelas. Durante o processo, o número binário deve ser implementado através de 4 bits cada. Caso a combinação seja maior que seis (0110_2), o resultado deve ser convertido em uma nova alocação, se um ou mais bits na posição do vizinho mais significativo devem ser tratados até que o resultado seja menor que seis.

No exemplo anterior, devemos examinar e corrigir as sequências (veja Seções F8, F4, F2 e F1) entre 0000 e 1000, e o "vai um". Essa é a mesma lógica da Fig. 1-24 no sentido de melhor, pulso de sincronização interna do sistema.

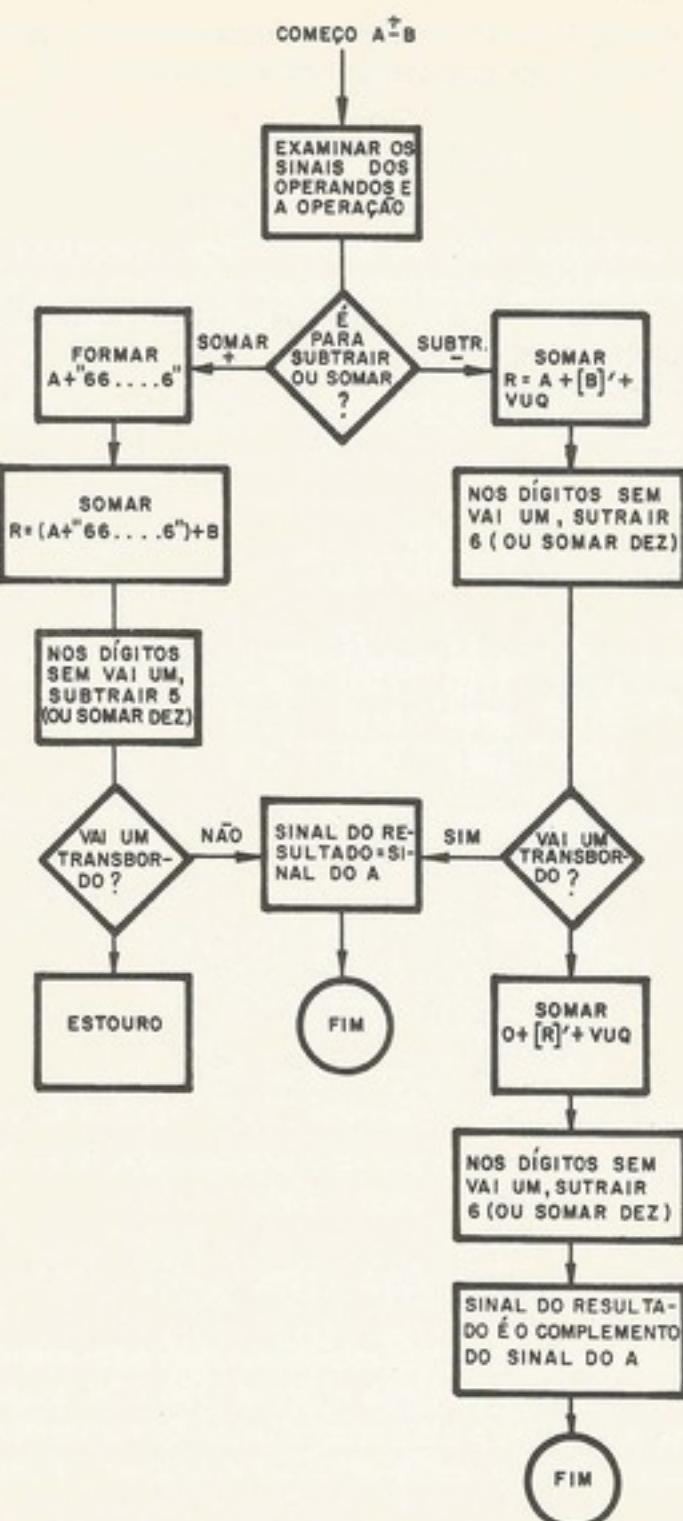


Figura 2-11. Aritmética BCD em sinal e amplitude

a. Conversão binária a decimal

Se o sistema tem meios de fazer divisão binária, a conversão pode ser efetuada através de divisões sucessivas por dez (1010_2). O resto da primeira divisão é o dígito decimal menos significativo. O quociente da primeira iteração é o dividendo da segunda, e o resto da segunda iteração é o dígito decimal segundo menos significativo. O processo continua reiterativamente até que o quociente fique zero. Esse último resto é o dígito decimal mais significativo do número convertido.

EXEMPLO. Converter $(11011011)_2$

$$\begin{array}{lll} 1) & 11011011 & 1010 = 10101 \\ 2) & 10101 & 1010 = 0010 \\ 3) & 0010 & 1010 = 0 \end{array} \quad \begin{array}{l} \text{quociente com 1001 resto,} \\ \text{quociente com 0001 resto,} \\ \text{quociente com 0010 resto;} \end{array}$$

$$\begin{array}{lll} \text{a conversão:} & 0010 & 00001 & 1001 \\ & 2 & 1 & 9 \end{array}$$

Se não houver meios de divisão, existe o algoritmo que segue, chamado *double-dabble* (veja Couleur⁽⁹⁾). Dado um número binário, positivo e inteiro de n bits, a conversão a decimal pode ser efetuada da maneira que segue.

1. Começar com o bit mais significativo.
2. Se o seguinte bit é 0, dobrar o resultado intermediário.
3. Se o seguinte bit é "1", dobrar o resultado intermediário, a acrescentar um.
4. Continuar o processo para todos os bits até o fim; o resultado é em decimal.

EXEMPLO

$$A = (11011011)_2 = (219)_{10} .$$

Começando com 1, dobramos e acrescentamos um para obter 3, porque o segundo bit é "1". O terceiro bit é "0" e, então, simplesmente dobramos 3 para obter 6. O quarto bit é "1"; então temos 13 (dobrando, + 1). O quinto bit é "1" e agora temos 27 (dobrando, + 1). O sexto bit é "0"; então só dobramos para obter 54. O sétimo bit é "1"; então temos 109 (dobrando, + 1). O último bit é "1"; então, dobrando, mais 1, dá o resultado certo 219. O processo é justificado pela seguinte identidade:

$$A_0 2^{n-1} + A_1 2^{n-2} + \cdots + A_{n-1} 2^0 = (\cdots ((A_0)2 + A_1)2 + \cdots A_{n-2})2 + A_{n-1} .$$

Um somador decimal (ou um somador binário com meios de fazer correção para *BCD*) pode ser usado para implementar o algoritmo. Para dobrar o resultado intermediário, basta apenas somar o resultado a si mesmo, isto é, o resultado intermediário serve como ambas as parcelas. Durante esse processo, o "vem um quente" ficará "1" só se o seguinte bit do número binário for "1"; caso contrário, ficará "0". O mesmo algoritmo também pode ser implementado através de um registrador especial de deslocamento que é dividido em dígitos de 4 bits cada. Cada seção de 4 bits tem meios de descobrir se o dígito guardado é uma combinação ilegal (veja a Tab. 2-1). Cada seção também tem meios de somar a parcela de seis $(0110)_2$. O registrador começa limpo. A cada iteração, um novo bit do número binário a ser convertido entra na posição menos significativa do registrador. Depois de cada deslocamento, se um dígito decimal de 4 bits apresentar uma combinação *BCD* ilegal ou se um bit na posição de peso "8" desse dígito passar para a posição de peso "1" do dígito decimal vizinho mais significativo, então a parcela de $(0110)_2$ será somada a esse dígito. Continuar até tratar o último bit (o bit menos significativo) do número binário.

No exemplo a seguir, cada iteração consta de dois passos: primeiro, deslocar; segundo, examinar e corrigir se for necessário. Rhyne⁽¹⁰⁾ estudou o problema em termos de circuitos seqüenciais (veja Sec. 1.2 e Fig. 1-11). Supomos que cada dígito conste de quatro *flip-flops*, $F8$, $F4$, $F2$ e $F1$ (conforme o peso da posição). Em termos do estado interno (o dígito atual, entre 0000 e 1001) e o "vem um" do vizinho, podemos predizer o próximo estado interno e o "vai um". Essa informação é mostrada na *flow table* da Tab. 2-5. A tabela difere daquela da Fig. 1-24 no sentido que esse circuito seqüencial é sincronizado através de um sinal, ou melhor, pulso de controle não mostrado aqui explicitamente. O circuito faz uma só transição interna do estado para cada pulso de controle.

Tabela 2-5. Circuito seqüencial para realizar o algoritmo *double dabble* (Rhyme)

Estado interno				"Vem um"	
F8	F4	F2	F1	0	1
0	0	0	0	0000,0	0001,0
0	0	0	1	0010,0	0010,0
0	0	1	0	0100,0	0101,0
0	0	1	1	0110,0	0111,0
0	1	0	0	1000,0	1001,0
0	1	0	1	0000,1	0001,1
0	1	1	0	0010,1	0011,1
0	1	1	1	0100,1	0101,1
1	0	0	0	0110,1	0111,1
1	0	0	1	1000,1	1001,1

EXEMPLO. *Double dabble*

$A = (11011011)_2 = (219)_{10}$	8421	8421	8421	A
Começo	0000	0000	0000	11011011
Deslocar			1	1011011.
Deslocar			11	011011..
Deslocar			110	11011...
Deslocar			1101	1011....
Corrigir (+ 6)	1	0011		
Deslocar	10	0111	011....	
Deslocar	100	1110	11.....	
Corrigir (+ 6)	101	0100		
Deslocar	1010	1001	1.....	
Corrigir (+ 6)	1	0000	1001	
Deslocar	10	0001	0011
Corrigir (+ 6)	10	0001	1001	← conversão
Resultado	2	1	9	

Observar a linha do estado interno "0100", que significa $(4)_{10}$. Se o "vem um" é "0", nessa iteração, simplesmente dobramos 4 para obter $(8)_{10}$, que é $(1000)_2$; e, se o "vem um" é "1", "dobro + 1" é $(9)_{10}$, que é $(1001)_2$. Em cada caso, o "vai um" é "0". No caso do estado interno "0110", que significa $(6)_{10}$, o dobro é $(12)_{10}$, que já cria o "vai um" de "1" ao digito vizinho, e o próximo estado interno é $(2)_{10}$ se o "vem um" é "0", ou $(3)_{10}$ se o "vem um" é "1".

b. Conversão decimal a binária

Rhyne⁽¹⁰⁾ também mostra que um processo iterativo baseado em deslocamento e correção pode ser realizado aplicando a técnica de *flow table*. Aqui, o processo começa com o bit menos significativo do número decimal e o sentido do deslocamento é para a direita.

Também existe um algoritmo para a conversão de *BCD* a binário, que só usa deslocamento e adição binária. O algoritmo aproveita o fato que para multiplicar um número binário por dez [multiplicador de $(1010)_2$], basta somar o multiplicando deslocado à esquerda três posições ao multiplicando deslocado à esquerda uma posição.

A conversão *BCD* para binário pode começar com o bit mais significativo. Para multiplicar o multiplicando por dez, basta somar com o multiplicador deslocado à esquerda três posições. Caso o resultado seja maior que 9, é necessário subtraí-lo de 10.

EXEMPLO. Transformação de *BCD* para binário

2.6 SUMÁRIO

Estudamos a representação de números inteiros em sinal e amplitude, a aritmética que usamos e exemplos e funções. Não indica estudo da binária, formando parte de um assunto efetuado através da adição, subtração, multiplicação e divisão.

EXERCÍCIOS

- 2-1. É possível representar $(-10)_{10}$ em binário? (veja a referência 7)
- 2-2. (a) Qual é o resultado da adição de $(1010)_2$ e $(1101)_2$? (b) Representar o resultado da adição de $(1010)_2$ e $(1101)_2$ em binário.
- 2-3. O que é $(1010)_2$ em decimal?
- 2-4. (a) Qual é o resultado da subtração de $(1010)_2$ e $(1101)_2$? (b) Representar o resultado da subtração de $(1010)_2$ e $(1101)_2$ em binário.
- 2-5. Como passar de $(1010)_2$ para $(101010)_2$?
- 2-6. Mostre como converter $(1010)_2$ para $(101010)_2$.
- 2-7. Repetir o exemplo da conversão de $(1010)_2$ para $(101010)_2$ usando a técnica de *flow table*.

EXEMPLO. Multiplicar $M = (000110)_2 = (6)_{10}$ por dez

$$\begin{array}{rcl} M \times 2^3 & = & 110000 \quad (M \text{ deslocado por } 3) \\ M \times 2^1 & = & 001100 \quad (M \text{ deslocado por } 1) \\ \text{produto} & = & \underline{111100} = (60)_{10}. \end{array}$$

A conversão *BCD* a binário é feita da seguinte maneira:

- começar com o dígito *BCD* mais significativo;
- multiplicar o resultado intermediário por dez ($1010)_2$;
- somar com o próximo dígito *BCD*;
- se, no passo anterior, o dígito *BCD* não foi o dígito menos significativo, voltar para segundo passo. Caso contrário, o resultado é a conversão.

EXEMPLO. Transcodificar $(219)_{10}$ em *BCD* ($0010\ 0001\ 1001$) para binário

primeiro resultado intermediário	0	0000	0010	$(2)_{10}$
multiplicar	0	0001	0000	
	0	0000	0100	
	0	0001	0100	
somar o próximo			0001	$(1)_{10}$
segundo resultado intermediário	0	0001	0101	
multiplicar	0	1010	1000	
	0	0010	1010	
	0	1101	0010	
somar o próximo			1001	$(9)_{10}$
resultado $(219)_{10}$	0	1101	1011	

2.6 SUMÁRIO

Estudamos alguns dos códigos mais comuns em sistemas digitais, bem como a representação de números. Em geral, existem três representações para os números negativos: sinal e amplitude, complemento da base e o complemento da base diminuída. Sistemas de aritmética que usam o processo de adição e complementação foram demonstrados com exemplos e fluxogramas. Dependendo da operação, o “vai um transbordo” normalmente não indica estouro de capacidade. Da mesma forma, conversões binária a decimal, e decimal a binária, foram estudadas com vários métodos ilustrados.

Um assunto não tratado foi a multiplicação e a divisão. A multiplicação é normalmente efetuada através de adições e deslocamentos. A divisão, que é muito mais complicada, consta de adição, subtração e deslocamentos. Esses assuntos são amplamente tratados em Flores⁽⁶⁾.

EXERCÍCIOS

- 2-1. É possível representar os inteiros com base negativa? Explique usando a base (-2) como exemplo (veja a referência⁽⁵⁾).
- 2-2. (a) Qual o valor decimal de $(01101101)_2$? (b) Qual a representação binária de 654?
- 2-3. Representar o número 12,1 em binário de dez bits, 5 bits inteiros e 5 bits fracionais.
- 2-4. O que é $(-47)_{10}$, usando representações binárias de 8 bits de:
 - (a) sinal e amplitude; (b) complemento de um; (c) complemento de dois.
- 2-5. Como parece o número 32000 em ponto flutuante “curto” normalizado do S/360? (veja a Fig. 2-2.)
- 2-6. Mostre como somar em complemento de um, $n = 6$ bits, as seguintes parcelas:
 - (a) $A = (-27)_{10}$, $B = (-7)_{10}$; (b) $A = (+27)_{10}$, $B = (+8)_{10}$; (c) $A = (+1)_{10}$, $B = (+5)_{10}$.
- 2-7. Repetir o exercício 2-6, com as parcelas codificadas em complemento de dois (6 bits).

2-8. Usando a técnica de subtrair através de complementação do subtraendo, mostrar como obter as seguintes diferenças $A - B$, em binário complemento de um, 6 bits:

- (a) $A = (+8)_{10}$, $B = (+7)_{10}$; (b) $A = (-16)_{10}$, $B = (+16)_{10}$; (c) $A = (+15)_{10}$, $B = (+24)_{10}$.

2-9. Repetir o Exercício 2-8 em complemento de dois de 6 bits.

2-10. Repetir o Exercício 2-8 em sinal e amplitude, (1 bit de sinal, 5 de amplitude).

2-11. Vamos supor que tenhamos um sistema decimal de 4 dígitos, mais um bit de sinal, que usa a representação de negativos em complemento de nove. Mostrar como fazer:

- (a) $A + B$, $A = +0136$, $B = -7654$;
 (b) $A + B$, $A = -9998$, $B = +7777$; [Observação. -9998 é $(-1)_{10}$.]
 (c) $A - B$, $A = -0010$, $B = +0108$. [Observação. -0010 é $(-9989)_{10}$.]

2-12. Transcodificar os números negativos (em complemento de nove) do Exercício 2-11 para a representação sinal e amplitude e repetir, mostrando os resultados em sinal e amplitude.

2-13. Repetir o Exercício 2-12, mas usando o código BCD com adição binária, complementação bit a bit e a técnica de correção de ± 6 .

2-14. Transformar a conversão binária a BCD dos seguintes números binários:

- (a) 001101; (b) 00100110111; (c) 01010101.

2-15. Efetuar a conversão BCD a binário dos seguintes números BCD:

- (a) 0010 0110 1001; (b) 0110 1000 0011; (c) 0100 0001 0111.

BIBLIOGRAFIA

- (1) M. Cohn e S. Even, "A Gray Code Counter", *IEEE Trans. Computers*, Vol. C-18, pp. 662-664 (julho de 1969)
- (2) R. W. Hamming, "Error Detecting and Correcting Codes", *Bell Systems Technical Journal*, Vol. 29, n.º 2, pp. 147-160 (abril de 1950)
- (3) W. W. Peterson, *Error-Correcting Codes*, The M.I.T. Press e John Wiley and Sons, Inc., New York, 1961
- (4) *IBM Journal of Research and Development*, Vol. 14, n.º 4 (julho de 1970). Exemplar especial em teoria e aplicações de codificação
- (5) Glen G. Langdon, Jr., "Subtraction by Minuend Complementation", *IEEE Trans. Computers*, Vol. C-18, n.º 1, pp. 74-75 (janeiro de 1969)
- (6) Ivan Flores, *The Logic of Computer Arithmetic*, Prentice-Hall, Englewood Cliffs, N.J., 1963
- (7) Y. Chu, *Digital Computer Design Fundamentals*, McGraw-Hill Book Co., New York, 1962
- (8) Peter Calingaert, *Princípios de computação*, Parte II. Ao Livro Técnico, Rio de Janeiro, 1969
- (9) J. F. Couleur, "BIDEC - a binary-to-decimal or decimal-to-binary converter", *IRE Trans. Elec. Computers*, Vol. EC-7, pp. 313-316 (dezembro de 1958)
- (10) V. T. Rhyne, "Serial Binary-to-Decimal and Decimal-to-Binary Conversion", *IEEE Trans. Computers*, Vol. C-19, n.º 9, pp. 808-812 (setembro de 1970)
- (11) H. Hellerman, *Digital System Design Principles*, McGraw-Hill, 1967

capítulo 3

ELEMENTOS

3.1 INTRODUÇÃO

O engenheiro de computadores e também a tecnologia depende de seu conhecimento em componentes eletrônicos chamados unidade lógica. Com o desenvolvimento preciso pensar em circuitos que são do tipo somador, multiplicador, divisor, ligações representadas por resistores, tais como resistores, que o transistores.

Os circuitos lógicos são os circuitos de fluxo de dados geralmente usados na comunicação entre os tipos de circuitos.

Os circuitos lógicos que regulam o ciclo da unidade de processamento do fluxo de dados é estudo no Capítulo 3.

O projeto de circuitos lógicos deve considerar os caminhos de dados, os tempos de atraso, os circuitos serials indicando o fluxo de dados.

3.2 CIRCUITOS

Uma rápida visão para mostrar que os circuitos lógicos de seleção, decodificação, de controle, etc.

capítulo 3

ELEMENTOS DO PROJETO LÓGICO

3.1 INTRODUÇÃO⁽⁹⁾

O engenheiro que faz um projeto lógico deve conhecer a função a ser implementada e também a tecnologia. A tecnologia sempre influí muito no projeto, pois o custo do sistema depende de seu bom aproveitamento. Nas primeiras duas gerações, a meta era a economia em componentes básicos, como circuitos *NAND*, *NOR* e *flip-flops*. Esses blocos lógicos, chamados *units logics*, são os blocos unitários com que foram construídos os sistemas digitais. Com o desenvolvimento de circuitos integrados em grande escala, o engenheiro agora precisa pensar mais em termos de "unidades funcionais", onde os "módulos" ou unidades são do tipo somador, registrador, contador, gerador de paridade e decodificador⁽¹⁾. O problema de ligação ou fiação (*wiring*) está ficando mais crítico em todos os níveis. As interligações representam uma grande parte do custo de *hardware*. Então o engenheiro projetista de computadores digitais também procura diminuir as ligações, e não é mais verdade que o transistor custa caro e o fio custa nada (referência⁽²⁾, p. 178).

Os circuitos lógicos de um computador dividem-se em mais ou menos duas categorias, os circuitos de fluxo de dados (*data flow*) e os circuitos da unidade de controle. O fluxo de dados geralmente consta dos registradores centrais, da unidade aritmética e dos meios de comunicação ou transferidor de dados entre eles (*busses*) que são chamados de vias. São esses tipos de circuitos que estudaremos neste capítulo.

Os circuitos da unidade de controle constam do relógio-mestre ou relógio central, que regula o *ciclo básico da máquina* (*machine cycle*). Os sinais sincronizados provenientes da unidade de controle controlam a operação da unidade aritmética e as transferências do fluxo de dados. A unidade de controle será abrangida através de um exemplo no Cap. 5, e estudada no Cap. 8.

O projetista da lógica também reconhece os problemas de *timing*, isto é, que certos eventos devem ocorrer dentro de certos prazos. Entre os registradores do fluxo de dados, os caminhos têm certos atrasos. O projetista deverá evitar que os atrasos dos circuitos superem aos tempos designados. Neste capítulo, onde são aplicados, esses atrasos relativos dos circuitos serão indicados. É importante observar que o ciclo básico da máquina e os atrasos do fluxo de dados são interdependentes.

3.2 CIRCUITOS COMBINATÓRIOS

Uma rápida análise nos esquemas lógicos de um sistema de computação é suficiente para mostrar que muita coisa é feita com circuitos combinatórios: somadores, portas de seleção, decodificadores, árvores de paridade, árvores de prioridade, geração dos sinais de controle, etc.

No projeto desses circuitos, muita coisa é levada em conta além da economia: relação pinos/circuitos, para facilitar a fiação e aumentar a confiabilidade (também ligada a custo), atrasos (em muitas situações, paga-se mais caro para diminuir o número de níveis lógicos), *fan-in* máximo das portas, *fan-out* máximo, etc.⁽²⁾ Isso tudo leva o projetista a definir determinados esquemas que, para um engenheiro não-especialista, pareceria caro e complexo. Para ilustrar essas idéias vamos analisar alguns circuitos combinatórios típicos.

a. Decodificadores^(2,3)

Em sua forma mais geral, um decodificador é um circuito com n entradas e 2^n saídas. Em cada instante, apenas uma das saídas está no nível "1" e todas as outras no nível "0", dependendo das entradas. Para cada combinação diferente das entradas, existe uma saída correspondente.

A implementação física desses circuitos oferece muitas opções. Podemos, por exemplo, escolher, para saída com *NOR* ou *NAND*, para termos saída invertida (*NAND*) ou não-invertida (*NOR*), podemos sacrificar o custo e a velocidade para ganharmos outros sinais intermediários.

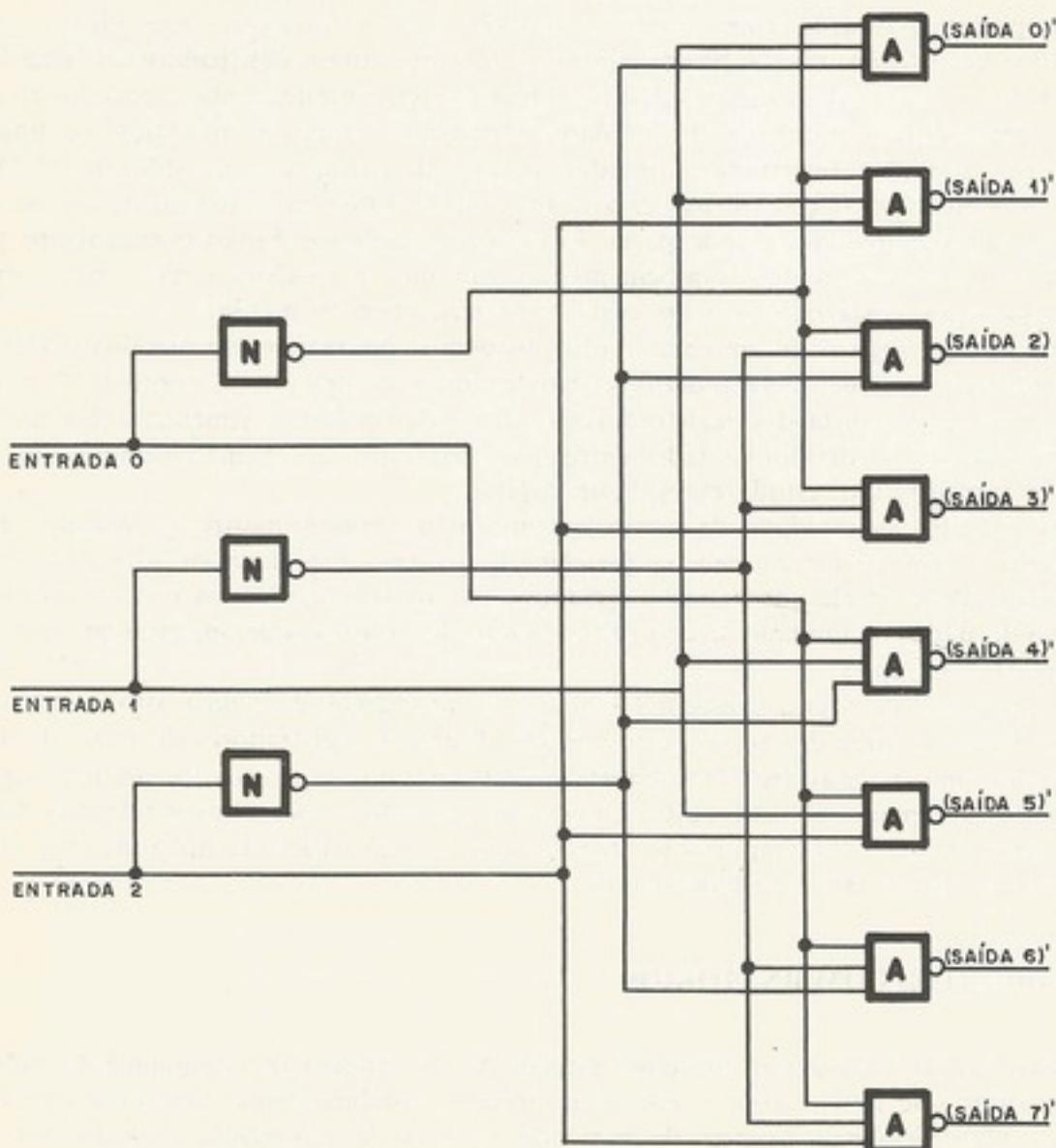


Figura 3-1. Decodificador de três entradas e oito saídas (saídas invertidas)

Compare os esquemas das Figs. 3-1 e 3-2 e observe que a vantagem da segunda sobre a primeira está no fato de esta apresentar sinais elétricos da decodificação de alguns bits independentemente dos outros.

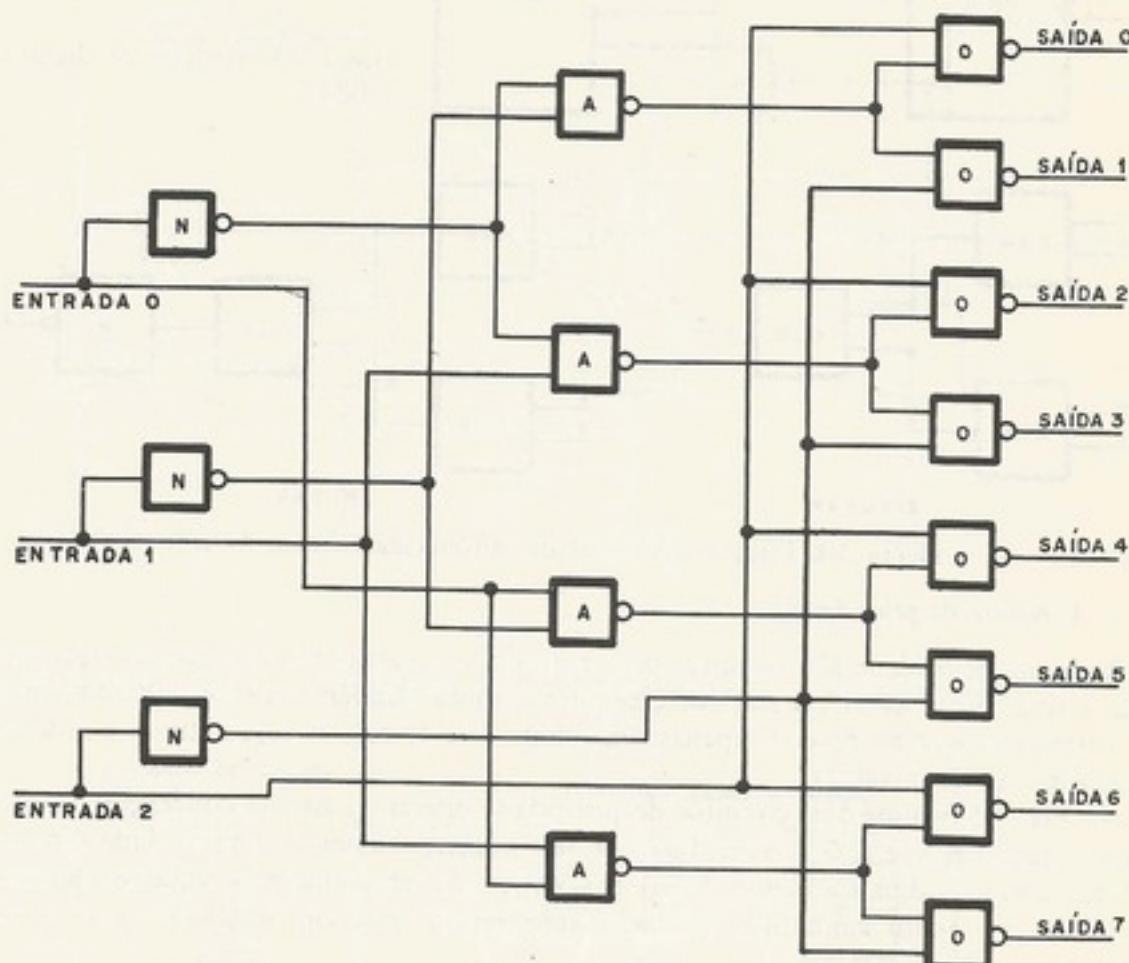


Figura 3-2. Decodificador em árvore, de três entradas e oito saídas

Muitas vezes, dependendo do número de entradas do circuito, optamos por um misto entre os dois para contornar o problema do *fan-in* elevado das portas.

b. Circuitos de paridade⁽¹⁰⁾

Circuito de paridade, genericamente, é um circuito de n entradas e uma saída que fica no nível “1” ou “0”, conforme a paridade do número de bits de entrada que esteja no nível “1”. Os circuitos que têm a saída “1” para a paridade de entrada par são chamados de *par* (*even*), e o de saída complementar, de *ímpar* (*odd*).

Esses circuitos são implementados com *exclusive OR*, pois *circuito ímpar* (Fig. 3-3),

$$e = a \oplus b \oplus c \oplus d;$$

circuito par (Fig. 3-3),

$$e = (a \oplus b \oplus c \oplus d)'.$$

Essas expressões sugerem os circuitos da Fig. 3-4.

Observe que, a partir do circuito ímpar, podemos obter o circuito *par* complementando a sua saída ou qualquer uma de suas entradas.

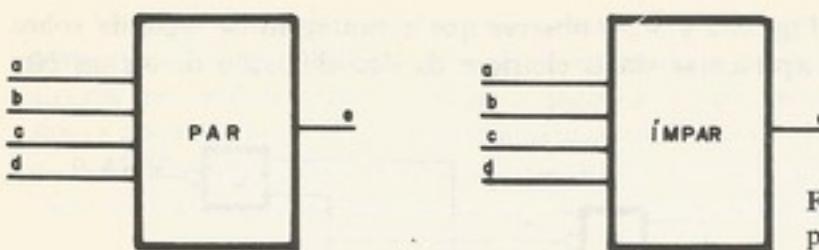
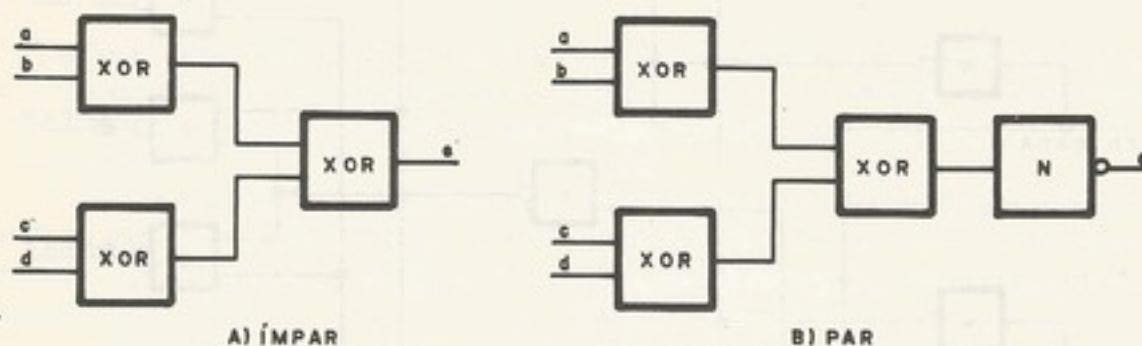


Figura 3-3. Modelos de circuitos de paridade

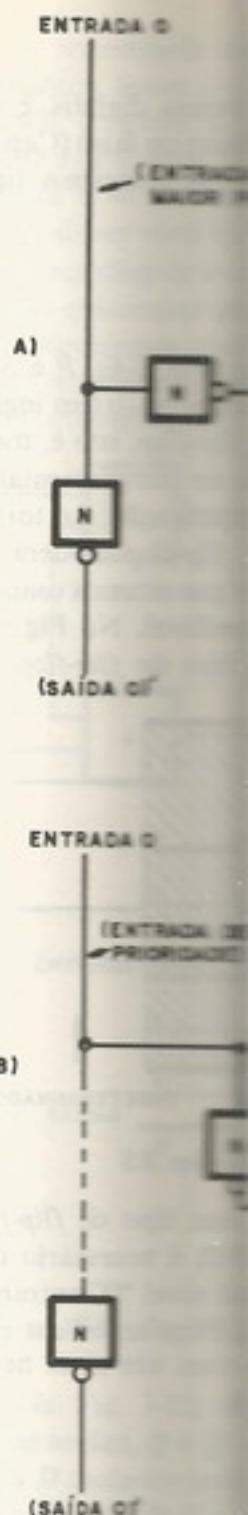
Figura 3-4. Circuitos de paridade implementados com *XOR*c. Circuitos de prioridade⁽²⁾

Redes de prioridade são circuitos de n entradas e n saídas tal que, apenas uma das entradas estiver no nível "1", a sua correspondente saída também será "1". Quando mais de uma entrada estiver no nível 1, apenas uma saída será 1, aquela cuja entrada é a de maior prioridade das de nível "1".

Na Fig. 3-5, vemos dois circuitos de prioridade diferentes na sua concepção. A entrada de maior prioridade é a O e, à medida que seu número aumenta, a prioridade diminui. A diferença entre os dois esquemas 3-5(a) e 3-5(b) é que, enquanto em 3-5(a) ganhamos em velocidade, em 3-5(b) ganhamos no *fan-in* das portas, que, em 3-5(a), às vezes se torna proibitivo.

EXERCÍCIOS

- 3-1. Projete um decodificador de duas entradas (e quatro saídas) usando apenas dois inversores e quatro *NOR* de duas entradas.
- 3-2. Projete um decodificador com quatro linhas de entrada e dez linhas de saída (não é um decodificador completo). Suponha que as quatro linhas de entrada sejam números em *BCD* e que a saída seja a decimal equivalente, de 0 a 9. Esse decodificador poderia ser chamado de decodificador decimal. [Sugestão. Leve em conta as combinações das entradas que não são usadas e simplifique o circuito (se possível).]
- 3-3. (a) Projete um decodificador de três entradas e oito saídas (use circuitos *NOR* na saída). Inclua uma quarta entrada (chame-a de "inibe") tal que, quando "inibe" = 1 as linhas de saída são zero (não importando o valor das três entradas) e, quando "inibe" = 0, o decodificador funciona como padrão. (b) Interligue dois desses decodificadores do item (a), de tal modo a obter um decodificador de quatro entradas e dezenove saídas. Use um número mínimo de blocos lógicos extras. (c) Interligue nove decodificadores do tipo definido no item (a), de tal modo a obter um decodificador de seis entradas e 64 saídas. Não é necessário usar blocos lógicos extras. (d) Comente sobre os atrasos dos circuitos em questão [itens (a), (b) e (c)].
- 3-4. (a) Projete um circuito ímpar de três entradas em dois níveis *NAND* × *NAND* (admita que se dispõe dos sinais de entrada complementados e não-complementados). (b) Modifique algumas ligações das entradas de modo tal que o circuito do item (a) se torne um circuito par (não mude o circuito).
- 3-5. Suponha que a rede de prioridade da Fig. 3-5(b) tenha oito entradas (e oito saídas). (a) Qual é o atraso máximo do circuito? (b) A que frequência (sinais por segundo) máxima podemos usar essa rede de prioridade, admitindo que o atraso de um bloco lógico seja de 10 ns?
- 3-6. Projete uma rede de prioridade com dezenove entradas que tenha um atraso máximo menor que $8d$ (d é o atraso de um nível lógico). Use blocos lógicos tipo *NAND*, *NOR* e inversor, com *fan-in* máximo de 4. [Sugestão. Use um mixto das técnicas apresentadas nas Figs. 3-5(a) e 3-5(b).]



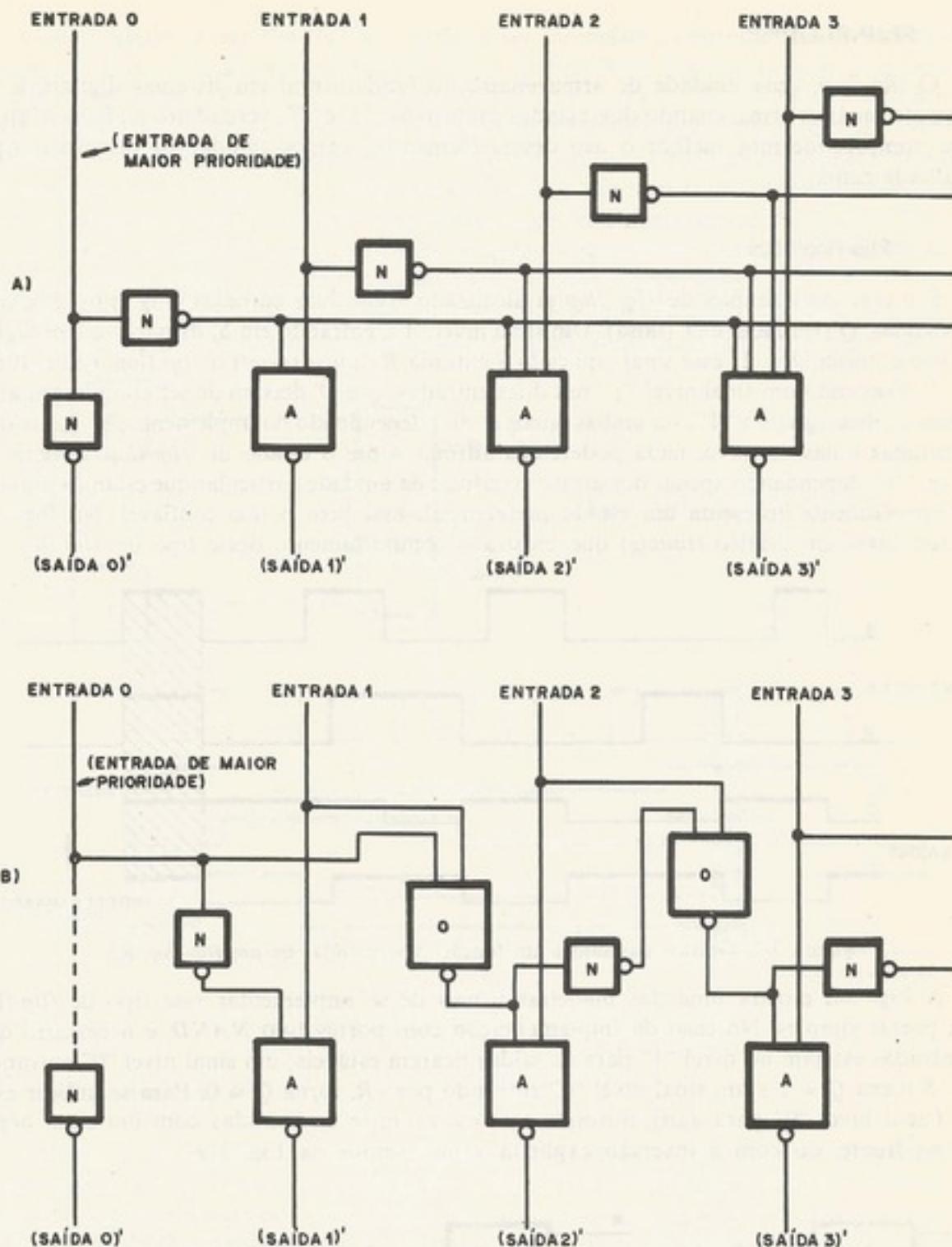


Figura 3-5. Redes de prioridade (saídas invertidas)

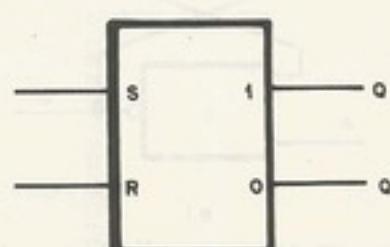


Figura 3-6. Flip-flop tipo RS

3.3 "FLIP-FLOP"⁽⁷⁾

O flip-flop, uma unidade de armazenamento fundamental em sistemas digitais, é um elemento binário, armazenando dois estados exclusivos: "1" e "0", verdadeiro ou falso (Cap. 1). Para compreendermos melhor o uso desses elementos, vamos estudar os principais tipos detalhadamente.

a. "Flip-flop" RS

É o tipo mais simples de *flip flop* já idealizado. Tem duas entradas (Fig. 3-6), *R* e *S*; e duas saídas, *Q* (verdade) e *Q'* (falsa). Um sinal nível "1", entrando em *S*, dispara -o (em inglês, *set*) isto é, torna *Q* = 1; esse sinal, aplicado à entrada *R*, limpa (*reset*) o flip-flop, isto é, torna *Q* = 0. Forçando um sinal nível "1" nas duas entradas, *Q* e *Q'* deixam de ser complementares, e ficam as duas iguais a "1", ou ambas iguais a "0", dependendo da implementação. Ao tornar as entradas nulas de novo, nada poderemos afirmar sobre o estado do flip-flop. Poderá ser "1" ou "0", dependendo apenas dos atrasos relativos da unidade particular que estamos usando, que normalmente apresenta um estado preferencial, mas bem pouco confiável. Na Fig. 3-7 apresentamos um gráfico (*timing*) que mostra o comportamento desse tipo de *flip-flop*.

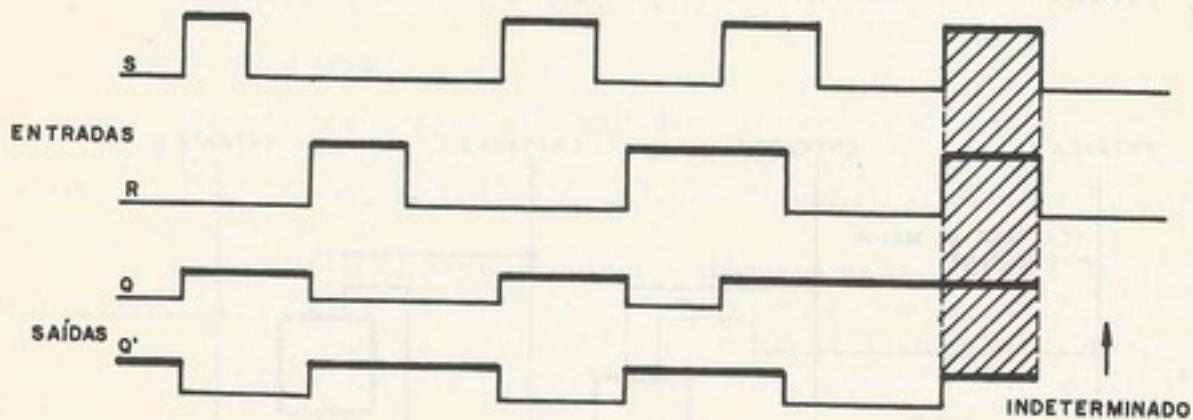


Figura 3-7. Gráfico das saídas em função das entradas de um flip-flop RS

A Fig. 3-8 mostra uma das maneiras usuais de se implementar esse tipo de *flip-flop* com portas simples. No caso da implementação com portas tipo *NAND*, é necessário que as entradas estejam no nível "1" para as saídas ficarem estáveis; um sinal nível "0" entrando por $-S$ torna *Q* = 1 e um sinal nível "0" entrando por $-R$, torna *Q* = 0. Para se indicar esse fato (sinal nível "0" para agir), normalmente escrevem-se as entradas com um sinal negativo na frente, ou com a inversão explícita, como vemos na Fig. 3-9.

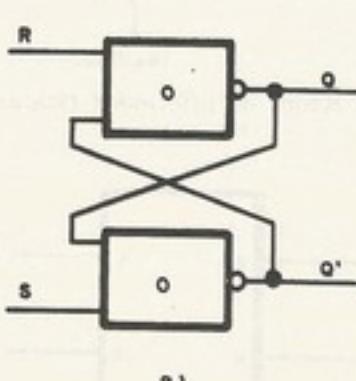
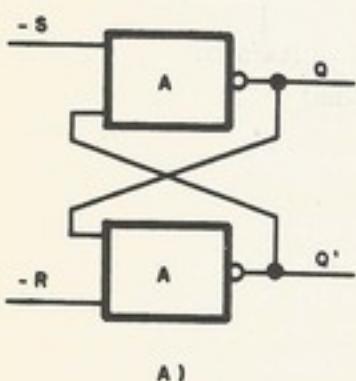


Figura 3-8.(a) Flip-flop tipo RS com portas tipo *NAND*; (b) Flip-flop RS com portas tipo *NOR*

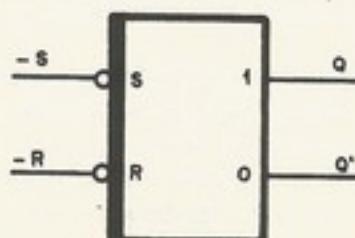
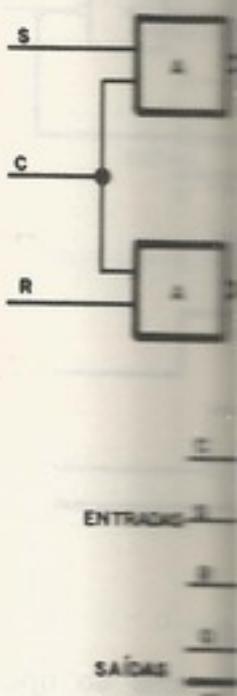


Figura 3-9. Flip-flop tipo $\bar{R}\bar{S}$ obtido com a implementação com portas tipo *NAND*

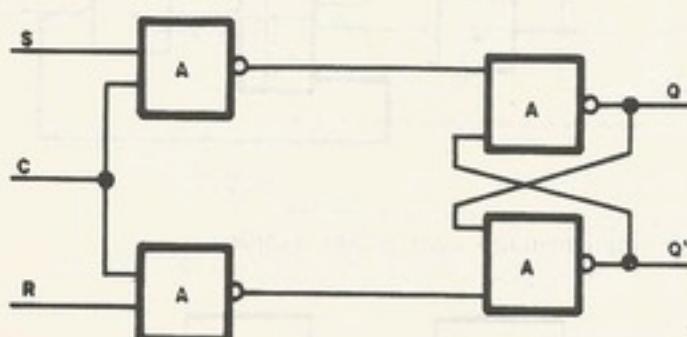
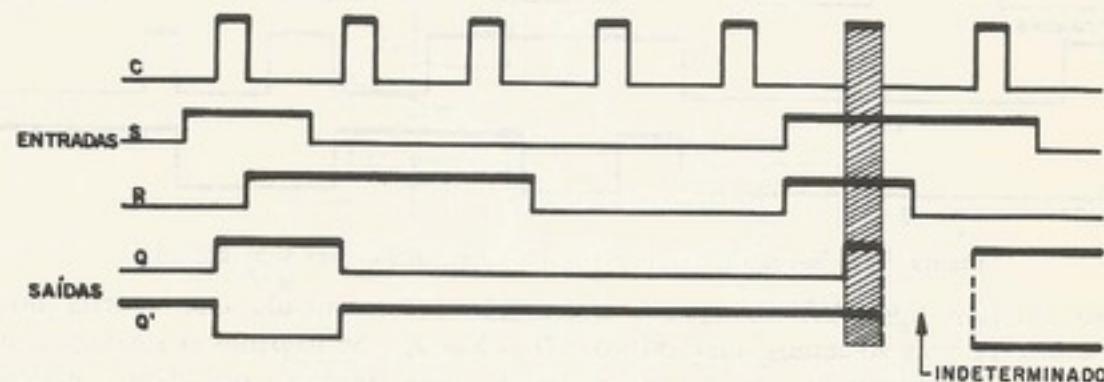
A necessidade de momentos levanta a questão da entrada, com nível de sinal diferente das entradas normais. Neste caso, fazemos $C = 1$. Isso pode ser visto isto é, restringindo o tempo em que queremos mudar o caso, devemos ter uma porta de controle que permaneça ligada durante o tempo desejado.

b. "Flip-flop" $\bar{R}\bar{S}$

Na Fig. 3-10, mostramos a implementação de um flip-flop tipo $\bar{R}\bar{S}$ com duas saídas, Q e Q' , e uma terceira saída, Q'' , obtida a partir da saída Q . A saída Q é a saída principal, com valor lógico "1" quando o flip-flop está armazenando "1". A saída Q' é a saída complementar, com valor lógico "1" quando o flip-flop está armazenando "0". A saída Q'' é a saída de inversão da saída Q , com valor lógico "1" quando o flip-flop está armazenando "0".

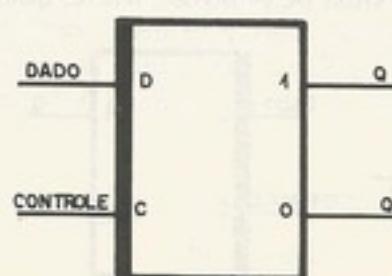
A necessidade de um *flip-flop* que pudesse ser insensível às entradas em determinados momentos, levou à concepção de *flip-flop* visto na Fig. 3-10, onde se incluiu uma terceira entrada, controle (*C*). Enquanto a entrada *C* estiver no estado "0", o *flip-flop* ficará indiferente às entradas *S* e *R*, já que os dois *NAND* incluídos têm uma entrada *C* nula. Quando fazemos *C* = 1, abrimos as portas *NAND*; os sinais em *S* e *R* passam e atingem o *flip-flop*. Isso pode ser visto na Fig. 3-11. A idéia desse *flip-flop* é "sincronizar" a mudança do *flip-flop*, isto é, restringi-la a certos instantes; colocamos sinais convenientes em *S* e *R* e, no instante em que quisermos que o *flip-flop* registre as entradas, forçamos um pulso em *C* (mesmo nesse caso, deveremos cuidar para que não aconteça o caso *S* = *R* = 1).

Continuando com a idéia de fazermos um *flip-flop* que, em certo instante, copie a entrada, construiu-se o *flip-flop* tipo *PH* (*polarity hold*), também conhecido como *latch*.

Figura 3-10. *Flip-flop RS* com controleFigura 3-11. *Flip-flop tipo RS* com controle-timing

b. "Flip-flop" tipo *PH*⁽⁷⁾

Na Fig. 3-12, encontra-se um *flip-flop* com duas entradas, *D* (dados) e *C* (controles), e duas saídas, *Q* e *Q'*. Um pulso positivo em *C* faz com que o *flip-flop* registre o valor da entrada *D* (*polarity hold*, guarda-polaridade). Se, enquanto *C* for igual a 1, a entrada *D* mudar de valor, a saída *Q* a acompanhará na mudança. Isso é o que caracteriza a classe de *flip-flops* chamada *sensíveis ao nível*, isto é, esse é um *flip-flop* que, enquanto *C* se mantiver no nível "1" ele ficará com a entrada sensível. O sinal "controle" é usualmente referido como *clock*.

Figura 3-12. *Flip-flop tipo PH*

Na Fig. 3-13 vemos esse tipo de flip-flop construído com portas simples, em duas das inúmeras versões possíveis. Na versão (a), apenas tomamos o flip-flop tipo RS com controle e, à entrada S, ligamos o "dado" e, à R, o "dado negado". Na versão (b), é construído um circuito com elo de realimentação positiva (em negrito) que, para $C = 0$, "guarda" um certo estado ("1" ou "0") e, quando C se torna igual a "1", esse elo de realimentação é interrompido e ficamos com $Q = D$, que realimentará quando C se tornar novamente igual a "0". Na Fig. 3-14 apresentamos gráficos de variação da saída Q em função de sinais nas entradas C e D .

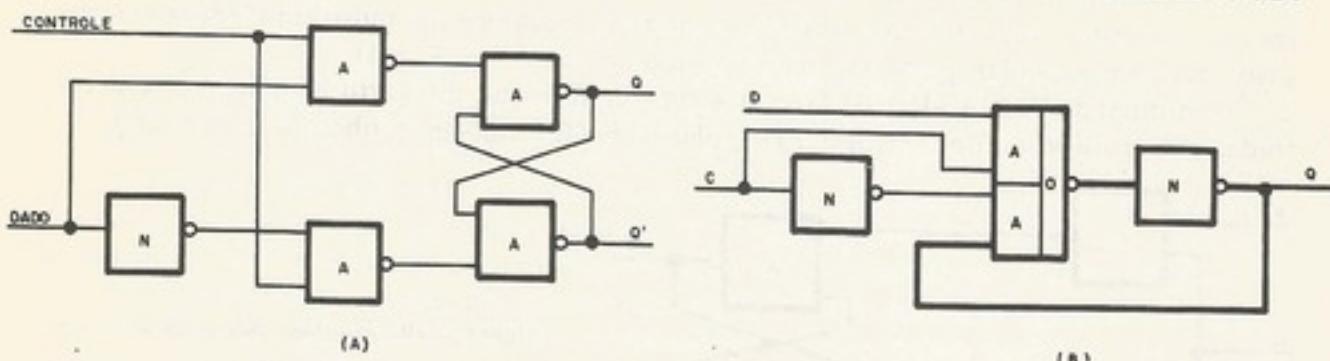


Figura 3-13. Flip-flop tipo PH implementado com portas simples

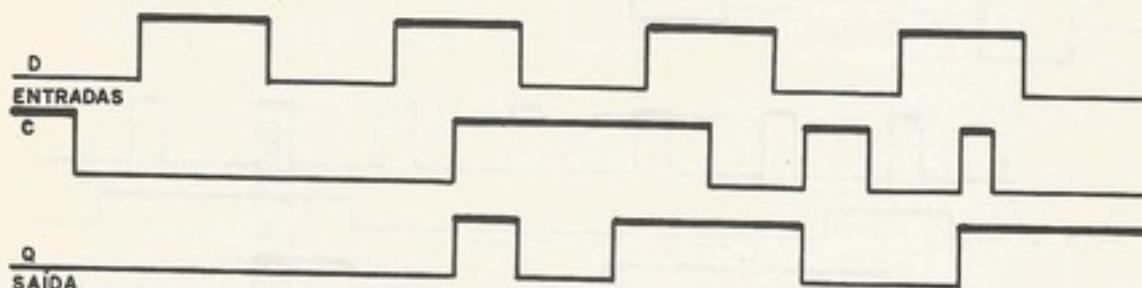


Figura 3-14. Saída do flip-flop tipo PH em função das suas entradas

No flip-flop tipo PH, não temos a situação indeterminada que ocorria no tipo RS ($R = S = 1$), pois só temos uma entrada ($D = S = R'$). Se fizermos o sinal de controle bem estreito e garantirmos que ele não ocorrerá numa transição do "dado", poderemos imaginar que temos um flip-flop que mostra a entrada num instante preciso, o que é de grande utilidade em sistemas digitais. Contudo um outro tipo de flip-flop tornou desnecessárias as tentativas de fazer-se o sinal em C bastante estreito para definir o instante de amostragem dos "dados" e transição do flip-flop, o flip-flop tipo D sensível à borda (edge sensitive D).

c. "Flip-flop" tipo D, sensível à borda

O flip-flop tipo D sensível à borda é muito usado no projeto de sistemas digitais, dadas as suas ótimas características funcionais. Como vemos na Fig. 3-15, ele tem duas entradas, D e C , e duas saídas, Q e Q' . O importante nesse elemento de armazenamento é que ele copia a entrada D durante a borda do sinal de controle, isto é, quando o sinal em C muda de "0"

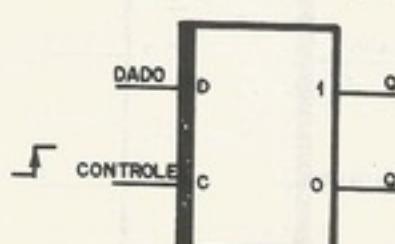


Figura 3-15. Flip-flop tipo D sensível à borda

elementos de projeto

para "1", por
borda (edge sen-
da entrada no)

O fato de
extrema utili-
muitas vezes cia

Assim como
cado sob forma

Figura 3-16.

Figura 3-17. Flip-flop

A análise des-
o mesmo circuito
flip-flops 1 e 2 em
quanto isso, a sua
figuração:

flip-flop 1 $-S = 0$
 $-R = 0$
 $Q = 1$
 $Q' = 0$

para "1", por exemplo. Isso caracteriza uma outra classe de flip-flop, chamada sensíveis à borda (*edge sensitive*). Nessa classe, encaixam-se todos os flip-flops que registram o valor da entrada no instante de variação do sinal de controle, seja na subida seja na descida.

O fato de o flip-flop armazenar a entrada na borda do sinal de controle, torna-o de extrema utilidade em sistemas sincronos e o elemento sincronizador é o controle *c*, por isso, muitas vezes chamado de *clock*. Na Fig. 3-16, mostramos as curvas de *Q* em função de *C* e *D*.

Assim como outros tipos estudados até agora, esse flip-flop existe disponível no mercado sob forma de integrado. Na Fig. 3-17, vemos um flip-flop tipo *D* sensível à borda.

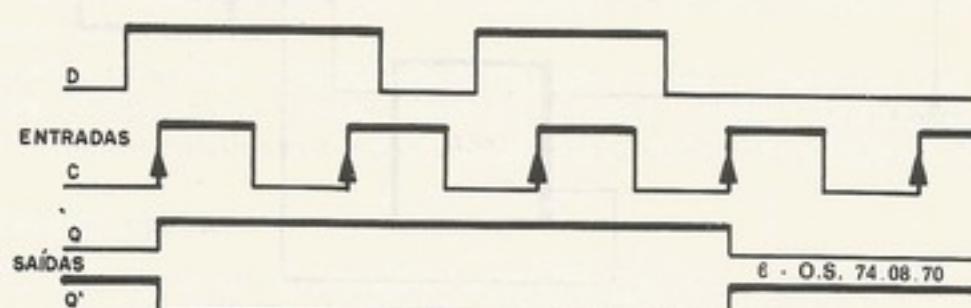


Figura 3-16. Curvas de *Q* e *Q'* em função de *D* e *C* num flip-flop tipo *D* sensível à borda

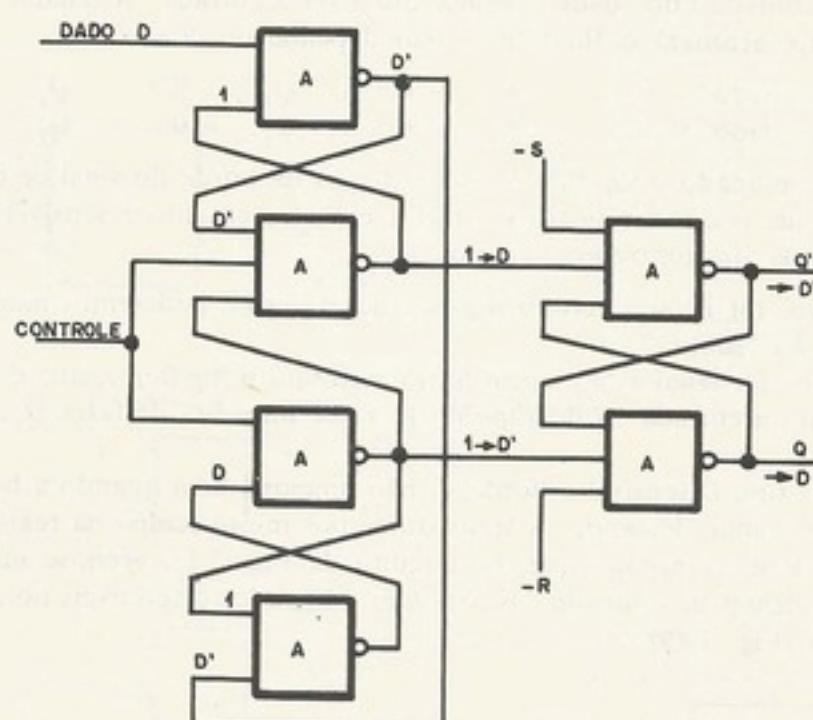


Figura 3-17. Flip-flop tipo *D* sensível à borda obtido com portas simples (projeto por John Earle⁽⁷⁾)

A análise desse circuito é um pouco complicada e, por isso, na Fig. 3-18, apresentamos o mesmo circuito em blocos maiores. Vemos aí que, quando *C* = 0, as entradas *-S* dos flip-flops 1 e 2 são nulas; portanto suas saídas são iguais a "1" e o flip-flop 3 é estável. Enquanto isso, a saída *Q* do flip-flop 1 é igual a "dado" e, portanto, temos a seguinte configuração:

$$\begin{array}{ll} \text{flip-flop 1} & -S = 0 \\ & -R = \text{"dado"} \\ & Q = 1 \\ & Q' = \text{"-dado"}; \end{array} \quad \begin{array}{ll} \text{flip-flop 2} & -S = 0 \\ & -R = \text{"-dado"} \\ & Q = 1; \end{array} \quad \begin{array}{ll} \text{flip-flop 3} & -S = 1 \\ & -R = 1 \end{array} \left. \right\} \text{estável.}$$

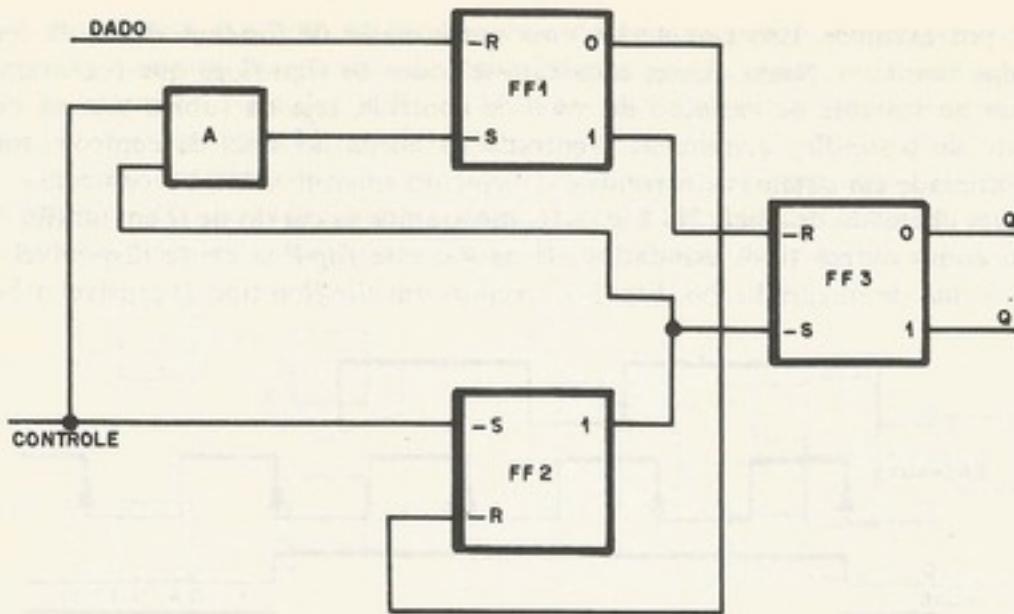


Figura 3-18. Flip-flop tipo D sensível à borda

Façamos agora o controle subir de "0" para "1". Liberamos então os dois flip-flops, 1 e 2, para o comando do "dado"; aquele que tiver a entrada $-R$ igual a "0" terá Q igual a "0" e, portanto, acionará o flip-flop 3; isso depende do "dado":

$$\begin{array}{lll} \text{se "dado"} = 0, & -R_{ff1} = 0, & Q_{ff1} = 0, \quad Q_{ff3} = 0; \\ \text{se "dado"} = 1, & -R_{ff2} = 0, & Q_{ff2} = 0, \quad Q_{ff3} = 1. \end{array}$$

Depois de mudado o flip-flop 3, isto é, depois da borda do sinal de controle, o "dado" pode mudar, que o circuito fica insensível a ele (esse circuito é sensível apenas durante a borda de subida do controle). Vejamos,

se o "dado" for igual a zero, o flip-flop ficará zero e poderemos mudar o "dado" para "1" que ele não "sentirá";

se o "dado" for igual a "1", quem ficará zero será o flip-flop 2, que já terá realimentado o sinal "0" para a entrada $-S$ do flip-flop 1, o que impedirá de fazer $Q_{ff1} = 0$ mudando o "dado".

O flip-flop tipo D sensível à borda só não funciona bem quando a borda de ataque do sinal C é muito lenta, deixando os transistores por muito tempo na região limiar.

O leitor deve ter notado que, no circuito da Fig. 3-17, vêem-se mais duas entradas, S e R , independentes do controle. Os flip-flops integrados disponíveis no mercado têm essas duas entradas (Fig. 3-19).

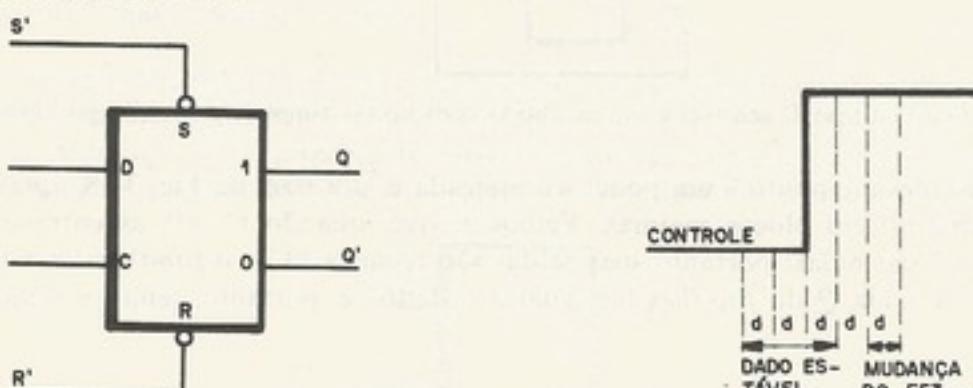
Figura 3-19. Flip-flop tipo D sensível à borda com entradas suplementares S' e R' para limpar ou disparar independentemente do controle

Figura 3-20. Comutação no flip-flop da Fig. 3-17

Seja d o ...
família TTL. Se
precisam estarem
de controle. A
controle.

O fato de
funcional muito
deslocadores, d
à frente é o m
que a entra

Para mui
cidade de seu
cismos de um

d. "Flip-flop"

Na Fig. 3-18
controladas pe
 Q e Q' . O fun
tabela supõe m

Seja d o atraso de uma porta (gate) simples com apenas uma inversão (20 ns para a família TTL). Se analisarmos o circuito da Fig. 3-17 com mais detalhes, veremos que os dados precisam estar estáveis durante o tempo de $2d$ antes e d depois da borda de subida do sinal de controle. A saída do flip-flop 3 muda somente $2d$ depois da borda de subida do sinal de controle.

O fato de a entrada D ficar insensível antes que a saída Q mude dá uma característica funcional muito boa para os flip-flops sensíveis à borda, pois eles podem ser usados para deslocadores, divisores de freqüência, etc. Um outro tipo de flip-flop que estudaremos mais à frente é o *master-slave*, que apresenta a característica de a saída mudar somente depois que a entrada fica insensível.

Para muitas aplicações, o flip-flop tipo D sensível à borda é ineficiente, dada a simplicidade de seu funcionamento: "subiu o controle e copiou o 'dado'". Aplicações onde precisamos de um pouco mais de sofisticação, um tipo muitas vezes mais eficiente é o flip-flop JK .

d. "Flip-flop" tipo JK

Na Fig. 3-21 vemos um tipo padrão de flip-flop tipo JK , com duas entradas, J e K , controladas pelo sinal C , com as duas saídas usuais dos elementos armazenadores binários, Q e \bar{Q} . O funcionamento desse tipo está na tabela da Fig. 3-21. Entenda-se aqui que essa tabela supõe os valores das entradas J e K quando o controle as torna sensíveis. Observe

TABELA DE FUNCIONAMENTO			
J	K	Q	\bar{Q}
0	0	ESTÁ - VEL	
1	0	1	0
0	1	0	1
1	1	INVERTE O ESTADO	

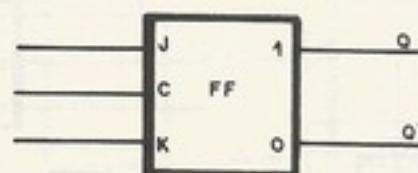


Figura 3-21. Flip-flop tipo JK

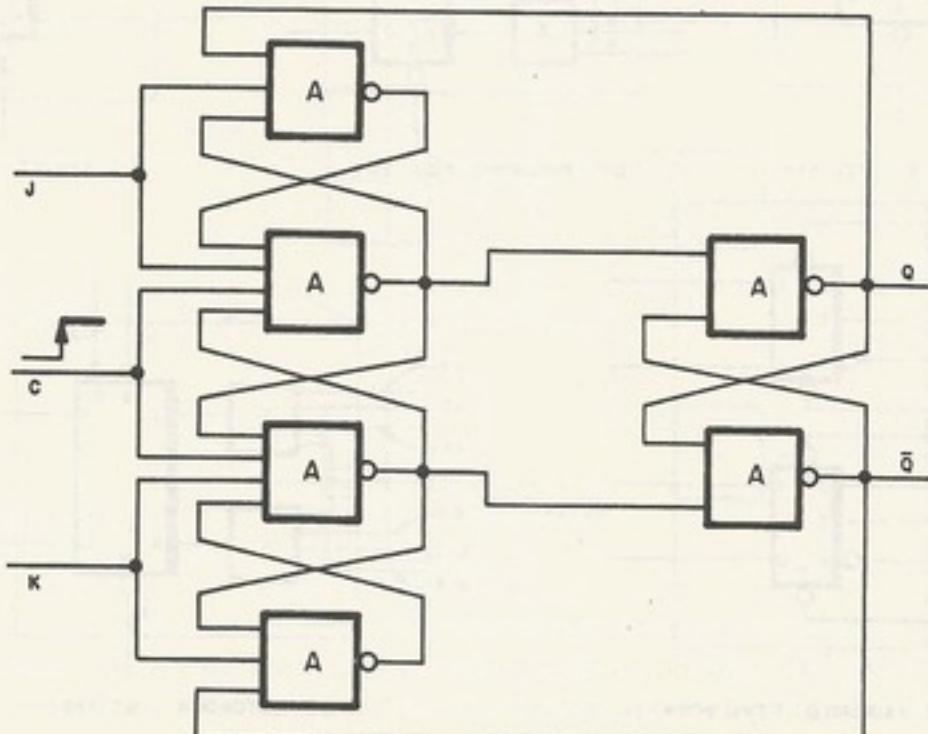


Figura 3-22. Flip-flop tipo JK sensível à borda (Fairchild)

que o funcionamento é parecido com o do flip-flop RS, com a grande diferença de que a situação indeterminada deste, no JK, faz com que o flip-flop mude de estado ($J = K = 1$). O flip-flop tipo JK pode ser da classe dos sensíveis à borda, como vemos na Fig. 3-22, um esquema com portas simples. O leitor deverá analisar com detalhes esse esquema, para verificar o seu comportamento e confirmar a correção das curvas apresentadas na Fig. 3-23. Como em outros flip-flops JK, este tem o inconveniente de, quando $C = 1$, mudando-se J e K , a saída muda.

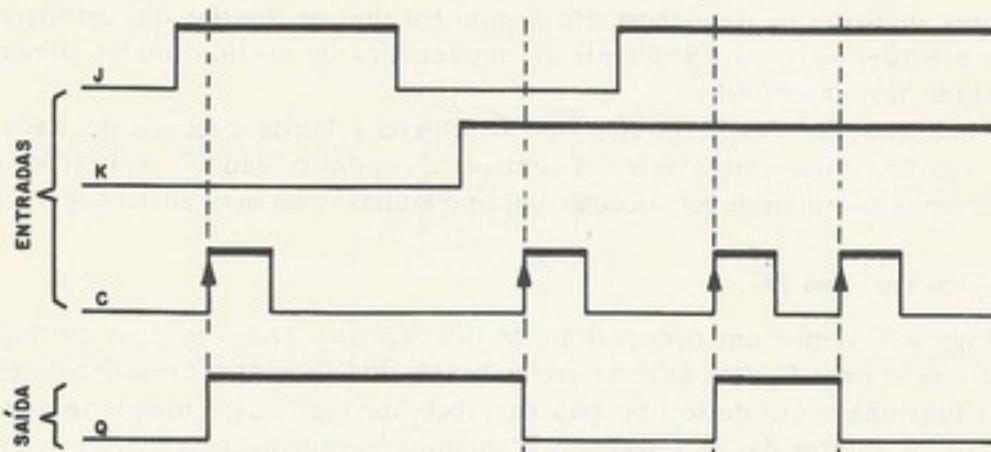


Figura 3-23. Saída Q em função das entradas J , K e C do flip-flop da Fig. 3-22

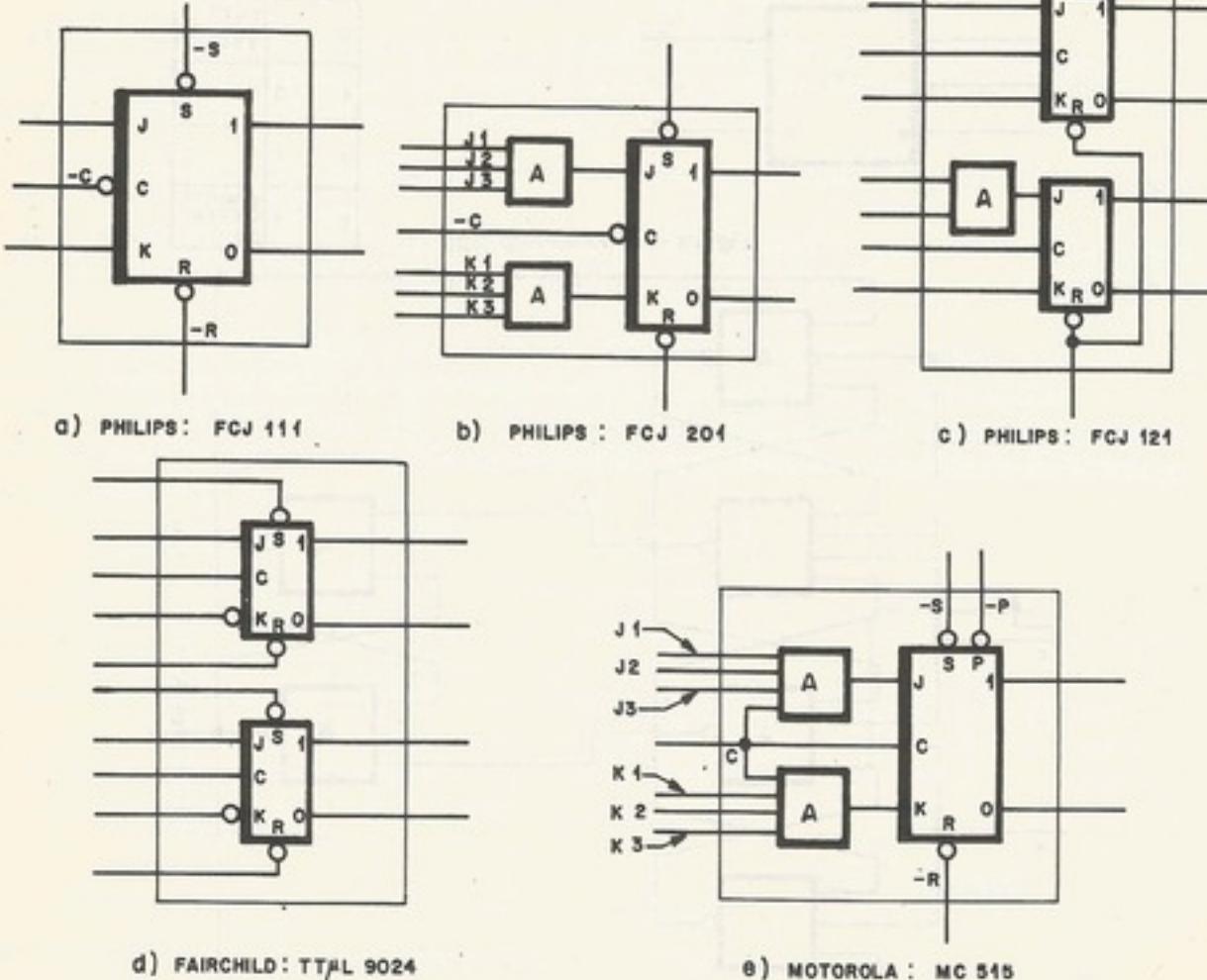


Figura 3-24. Alguns tipos de flip-flop JK disponíveis no mercado sob forma de circuito integrado

elementos do projeto lógico

Disponíveis no mercado da classe dos flip-flops alguns tipos diferentes

O flip-flop tipo JK, classe dos sensíveis à borda, controlando independentemente que existe no mercado à borda, tem a característica de que J e K já estão inseridos

c. "Flip-flop master-slave"

Um esquema simples permite-nos verificar a estrutura interna, ele dispõe de um controle que está no nível "1" quando desce para o nível "0", o que é o fato de a saída mudar

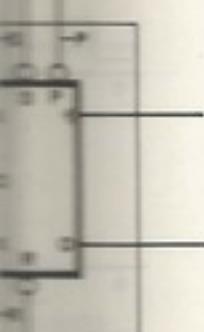
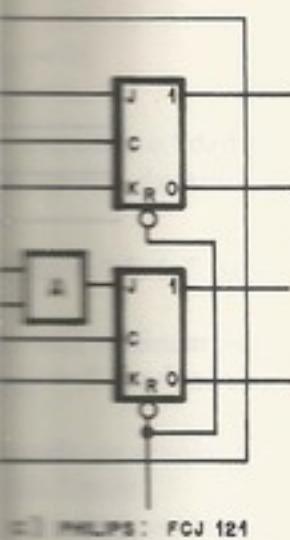


Figura 3-26. Tipico JK master-slave (Fairchild)

Da classe dos flip-flops tipo RS, etc. Nas Figs. 3-22 e 3-23, o flip-flop JK master-slave. É fácil verificar que a estrutura é muito mais complexa

diferença de que a saída ($J = K = 1$). Vemos na Fig. 3-22, esse esquema, para entradas na Fig. 3-23. mudando-se J e K ,

Fig. 3-22



de circuito integrado

Disponíveis no mercado, existem os mais variados tipos de *flip-flops JK*, na sua maioria da classe dos *flip-flops master-slave* (analisados no item seguinte); na Fig. 3-24, mostramos alguns tipos diferentes de circuitos integrados.

O *flip-flop* tipo *JK* leva uma grande vantagem sobre o tipo *D* (se tomarmos ambos da classe dos sensíveis à borda), pois permite realizações de funções lógicas mais sofisticadas, controlando independentemente as duas entradas, *J* e *K*. O tipo mais comum de *flip-flop JK* que existe no mercado é o *JK master-slave*, que, apesar de não ser da classe dos sensíveis à borda, tem a característica fundamental de a saída *Q* mudar somente quando as entradas *J* e *K* já estão insensíveis.

e. "Flip-flop master-slave"

Um esquema simplificado da estrutura do *flip-flop* da classe dos *master-slave* (Fig. 3-25) permite-nos verificar o funcionamento bastante simples desse tipo de *flip-flop*. Em sua estrutura interna, ele dispõe de dois *flip-flops*, o *master* (mestre) e o *slave* (escravo). Quando o controle está no nível "1", o *master* muda, sob influência das entradas, e, quando o controle desce para o nível "0", o *slave* copia o *master*. A principal característica desse tipo de *flip-flop* é o fato de a saída mudar somente depois que a entrada fica insensível.

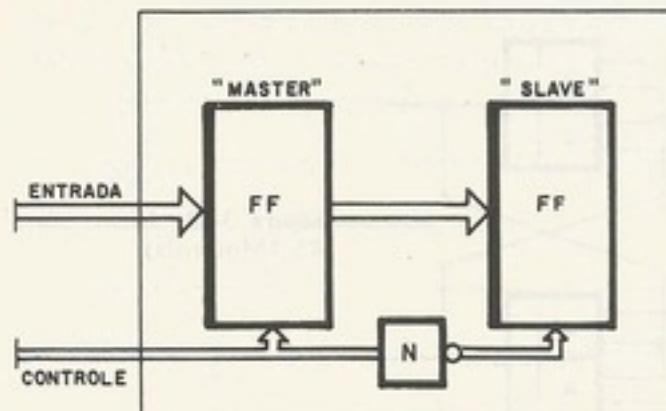


Figura 3-25. Esquema simplificado do *flip-flop* tipo *master-slave*

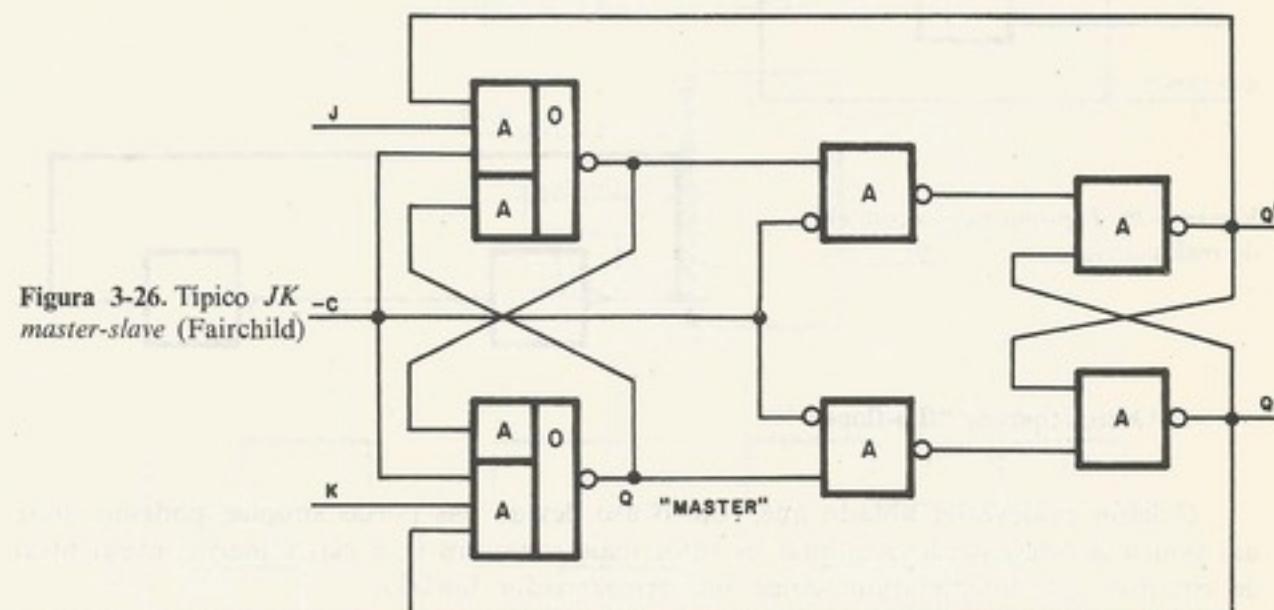


Figura 3-26. Típico *JK* -c *master-slave* (Fairchild)

Da classe dos *master-slave*, encontramos *flip-flops* do tipo *JK* (principalmente), do tipo *RS*, etc. Nas Figs. 3-26, 3-27 e 3-28 são vistos alguns tipos de *flip-flops* da classe dos *master-slave*. É fácil verificar que esses *flip-flops* têm o inconveniente de serem lentos, um fato importante.

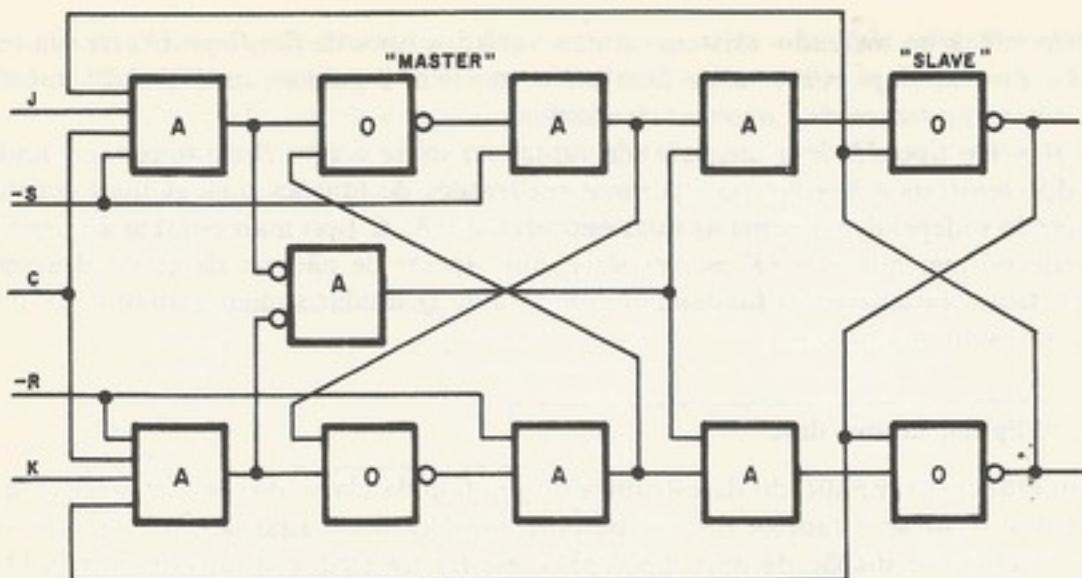


Figura 3-27. Flip-flop JK master-slave (Philips)

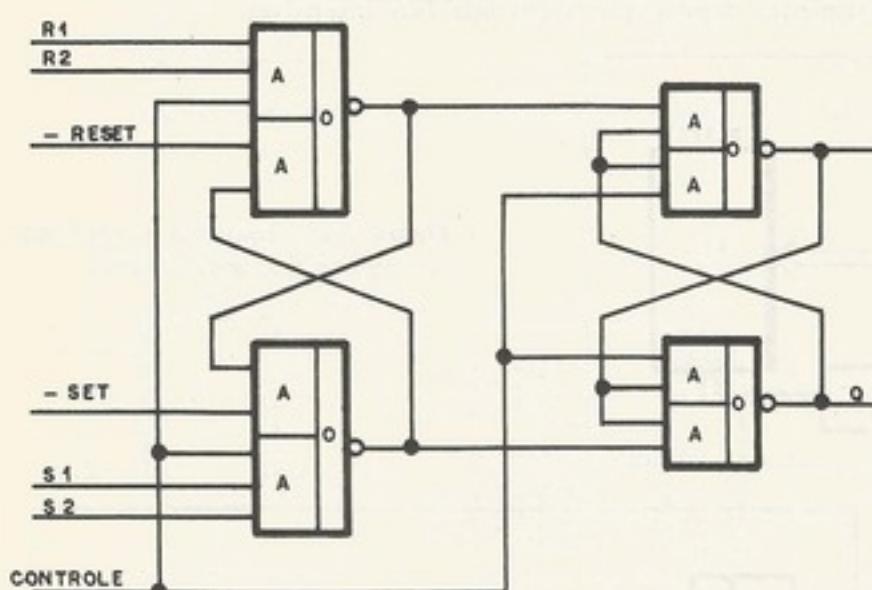
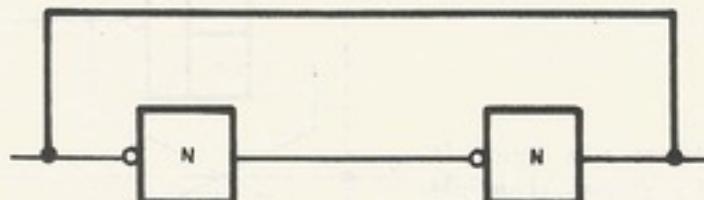


Figura 3-28. Master-slave RS (Motorola)

Figura 3-29. Flip-flop básico com elo de realimentação

f. Outros tipos de "flip-flops"⁽⁷⁾

O leitor já deve ter notado que, com o uso devidamente das portas simples, podemos usar um pouco de imaginação e montar os tipos mais estranhos (e, é claro, mais convenientes) de circuitos que funcionariam como um armazenador binário.

Uma técnica usual de montar essas estruturas é encarar um flip-flop como um circuito que tem um elo de realimentação positiva (Fig. 3-29) que garante a sua estabilidade num estado qualquer ("1" ou "0"). A estrutura vista na Fig. 3-29 não nos permite alterar o estado armazenado. O que temos a fazer é alterá-la de modo a podermos forçar o estado desejado de uma maneira conveniente. Na Fig. 3-8 vemos um exemplo típico. É nesse ponto que po-

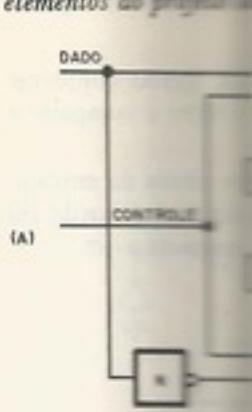
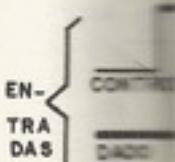
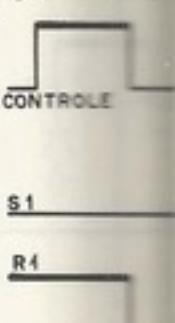


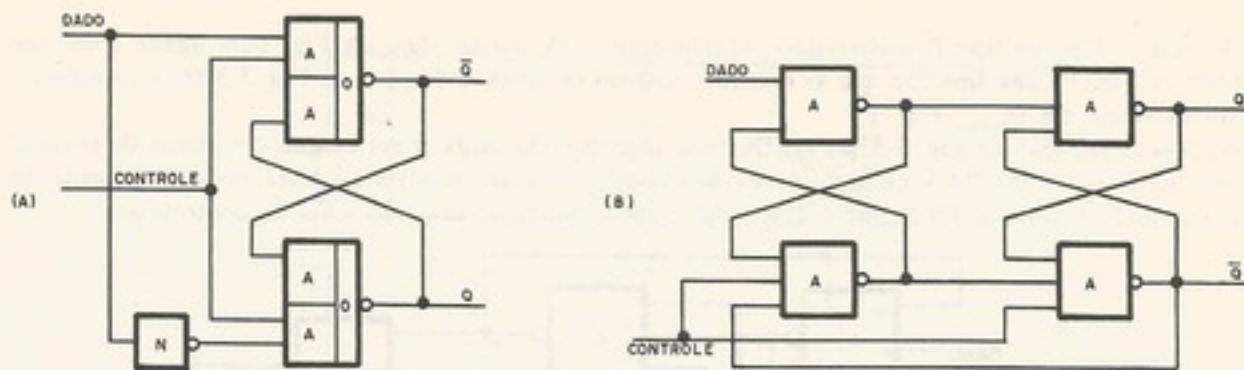
Figura 3-

demos "inventar" e aconselhamos a visto na Fig. 3-38 um para o estudo

EXERCÍCIOS

- 3-7. Seja o flip-flop que as outras entradas saída Q (admita que



Figura 3-30. Outros exemplos de *flip-flop*: (a) Fairchild TTL (PH); (b) tipo PH

demos "inventar" como agir no elo de realimentação. Na Fig. 3-30 temos dois exemplos, e aconselhamos ao leitor analisá-los com detalhes. Observar, por exemplo, que o *flip-flop* visto na Fig. 3-30(b) é do tipo sensível ao nível, com os dois elos de realimentação distintos, um para o estado "1" e outro para o estado "0".

EXERCÍCIOS

3-7. Seja o *flip-flop* tipo *RS master-slave* da Fig. 3-28. Admita que $S_2 = R_2 = 1$, $-reset = -set = 1$ e que as outras entradas tenham sinais variando conforme os gráficos da Fig. 3-31. Faça um gráfico da saída Q (admita que, no instante inicial, $Q = 0$).

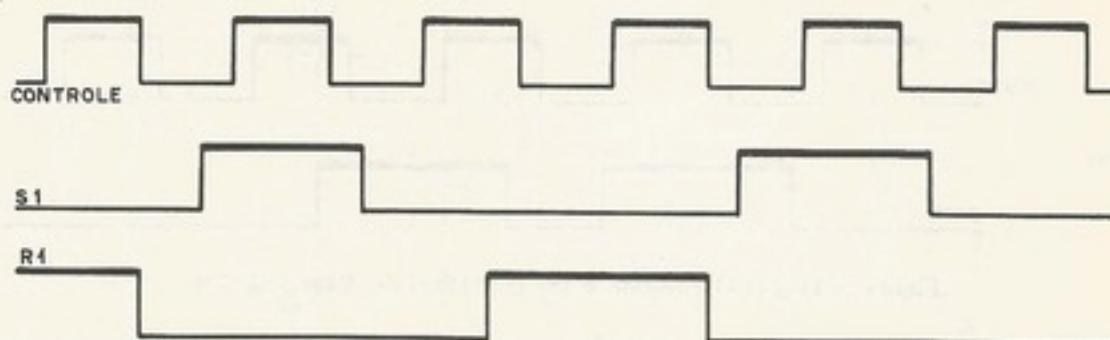
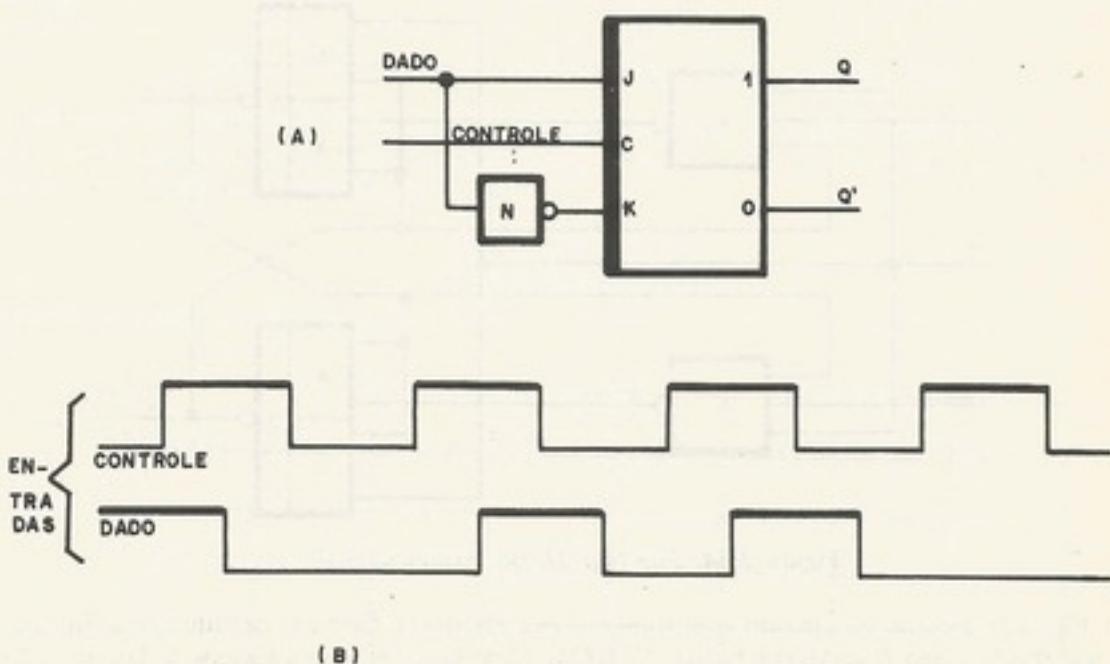


Figura 3-31. Figura do Exercício 3-7

Figura 3-32. (a) O *flip-flop* e (b) a forma de onda do Exercício 3-8

3-8. Seja o flip-flop tipo D master-slave, obtido com o JK master-slave da Fig. 3-26, ligado conforme mostra a Fig. 3-32(a). Imagine que as entradas tenham os sinais indicados na Fig. 3-32(b) e complete-a com os sinais em Q_{master} e Q .

3-9. Seja o flip-flop da Fig. 3-33(a). (a) Desenhe o gráfico da saída Q em função dos sinais de entrada vistos na Fig. 3-33(b). (b) Você seria capaz de classificá-lo entre sensível ao nível, sensível à borda ou master-slave? Justifique. (c) Se for o caso, diga qual a borda de ação do sinal de controle.

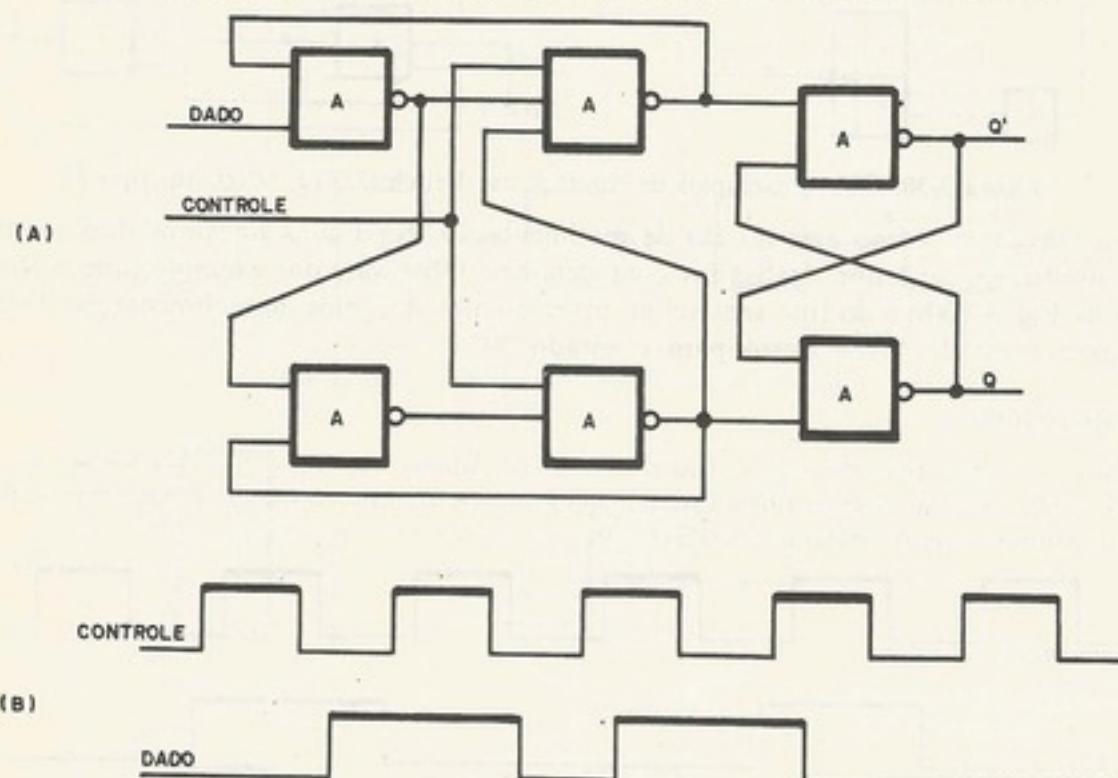


Figura 3-33. (a) O *flip-flop* e (b) o gráfico do Exercício 3-9.

3-10. Seja o flip-flop JK da Fig. 3-34. (a) Classifique-o entre as classes sensível à borda e master-slave. (b) Se for o caso, diga qual a borda de ação do sinal de controle.

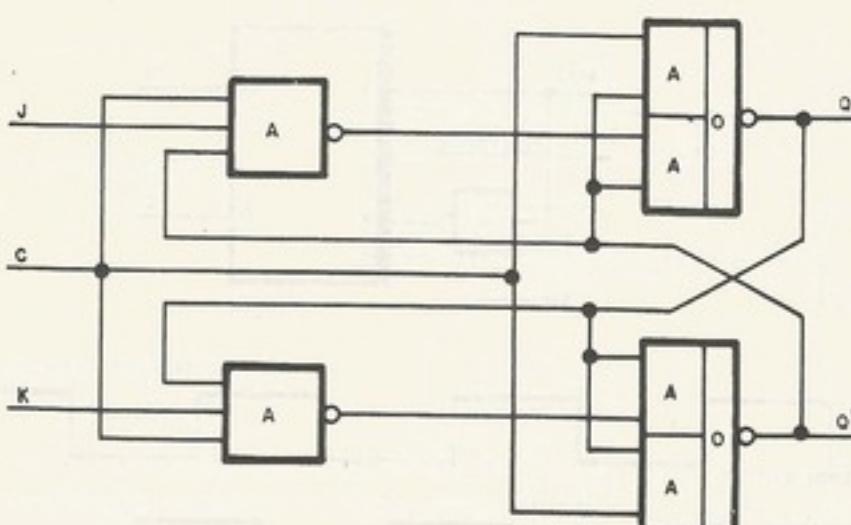
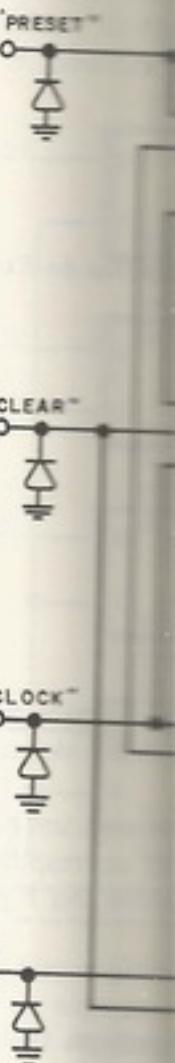


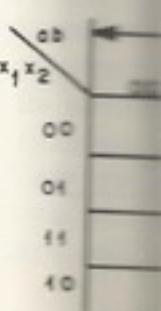
Figura 3-34. Flip-flop JK do Exercício 3-10.

3-11. A Fig. 3-35 mostra um circuito com transistores e resistores. Esse é o circuito que a Signetics usa para o seu *flip-flop* tipo *D* sensível à borda (*N74H74*). Identifique os blocos lógicos da família *TTL* que estão interligados nesse esquema (cuidado que alguns deles não têm estágio de saída). Redesenhe o circuito em termos dos blocos lógicos. Já estudamos tal circuito; identifique-o.



3-12. Esse exercício

(b) Preencha a tabela



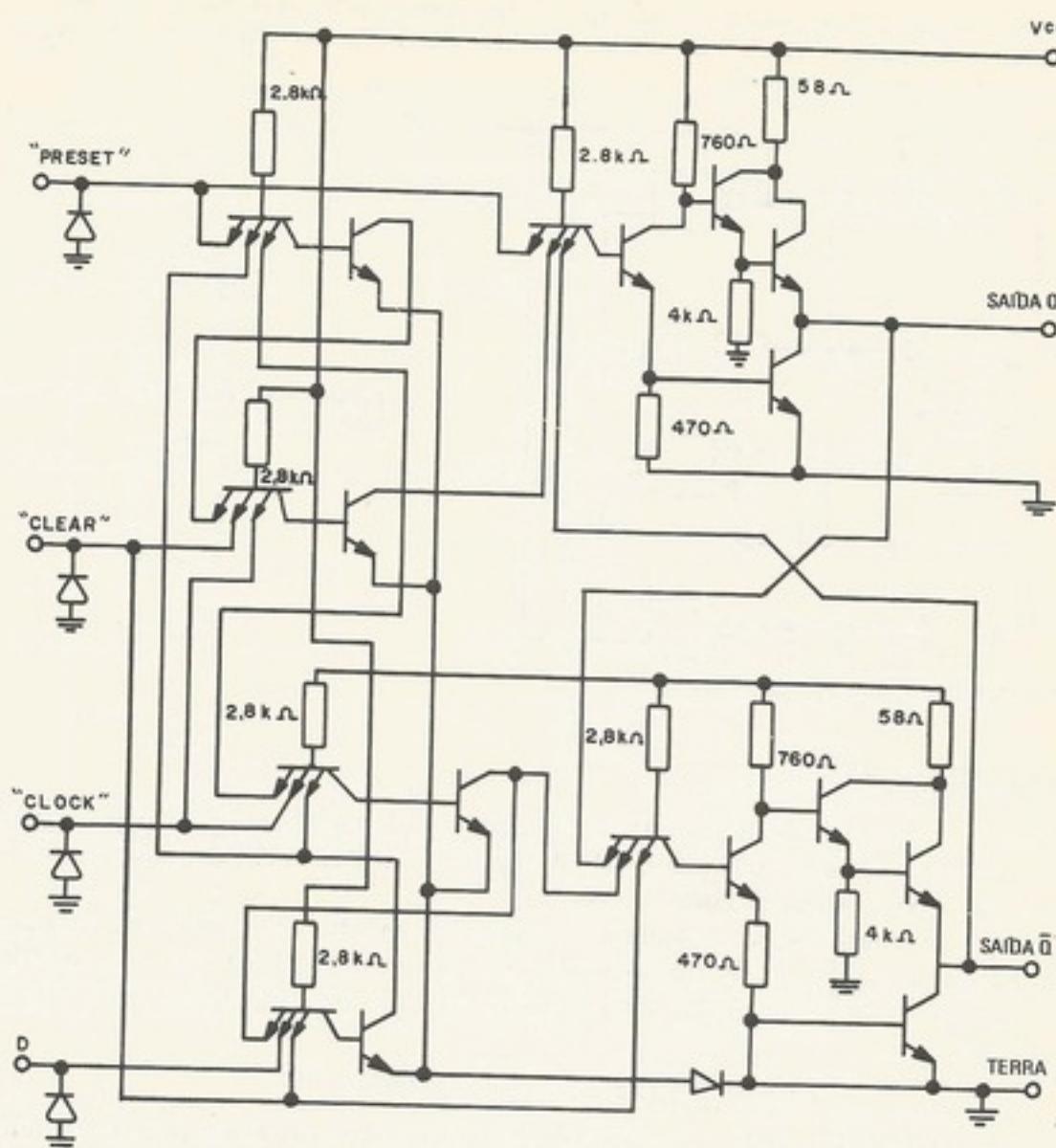


Figura 3-35. Flip-flop tipo D sensível à borda, família TTL

3-12. Esse exercício requer conhecimentos de teoria da comutação (*switching theory*). Seja o circuito da Fig. 3-36, onde estão assinaladas duas variáveis, x_1 e x_2 , que devem ser tomadas como variáveis de estado.

(a) Escreva as expressões algébricas booleanas

$$X_1 = F(a, b, c, x_1, x_2),$$

$$X_2 = F_2(a, b, c, x_1, x_2).$$

(b) Preencha a tabela de transição

		C = 0				C = 1			
		00	01	11	10	00	01	11	10
x ₁	x ₂	00							
		01							
01	00								
	01								
11	00								
	01								
10	00								
	01								

(DENTRO DAS CÉLULAS X₁ X₂)

(c) Admitindo que esse circuito seja um *flip-flop* tipo *JK*, qual a correspondência entre as entradas *a*, *b* e *c* com as entradas do padrão *J*, *K* e *C*? De que classe é esse *flip-flop* (*master-slave*, sensível à borda de subida; sensível à borda de descida)?

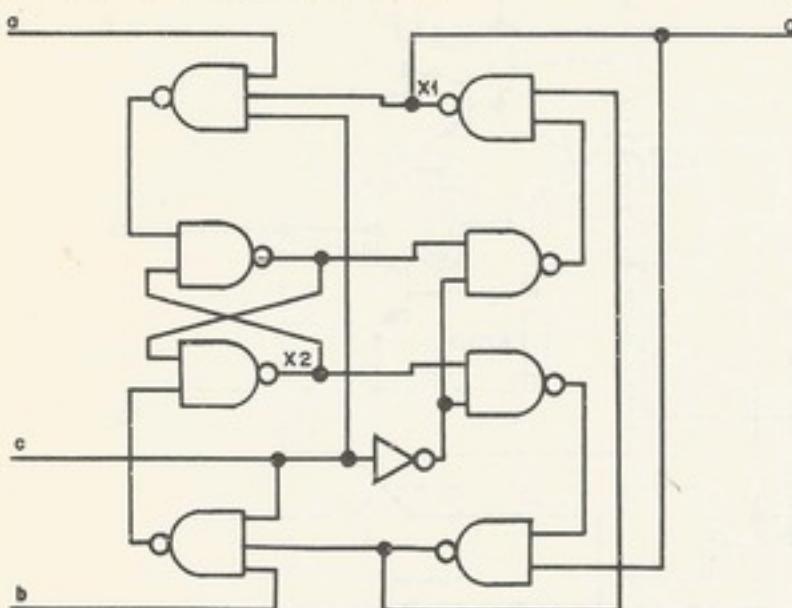


Figura 3-36. O *flip-flop* do Exercício 3-12

3.4 REGISTRADORES

Podemos entender como registrador a um conjunto de *flip-flops* utilizado para guardar uma informação codificada em binário^(3,4). A Fig. 3-37 mostra um registrador de três bits, onde está gravado o número cinco (=101).

Convencionaremos a seguir a seguinte notação: o nome do registrador com um número entre parênteses especifica o bit correspondente a esse número [*DAD(2)* corresponde ao bit 2 do registrador (*DAD*)] e, ainda, que o número zero é o mais significativo. Na Fig. 3-37, o número três seria

$$Q(0) = "0", Q(1) = "1" \quad \text{e} \quad Q(2) = "1".$$

O registrador da Fig. 3-37 é do tipo convencional, onde qualquer palavra codificada na entrada (3 bits) fica registrada nele sob comando do pulso de controle. Podemos repetir

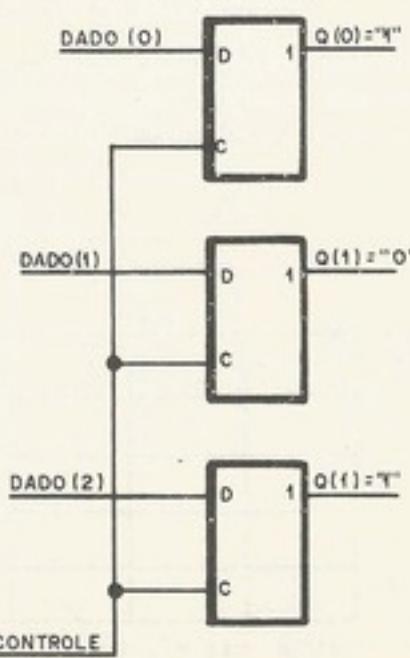
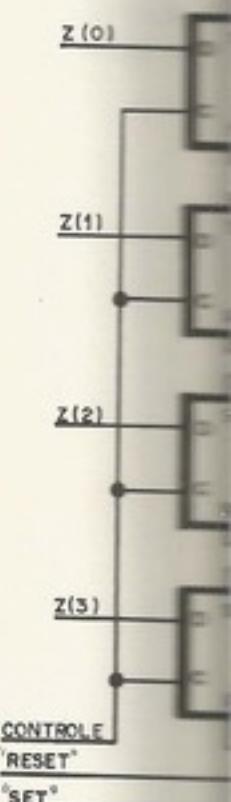


Figura 3-37. Um registrador de 3 bits

elementos de j...
o conceito de...
uma palavra.
Um regis...
ter duas limi...
o estado de...
3-39, podem...



Um regis...
efetuar alguma...
como o regis...
lizar e sem...
tante, conform...
classes de ...

a. Regis...

Um regis...
apenas de des...
circuitos com...
de um pulso...
exemplo, um...
à direita, p...
ser 1) foi int...
3-40 vemos ...

Como um...
a) um des...
o número cod...
dois;

b) um des...
informações ...

o conceito de registrador como sendo um grupo de *flip-flops* utilizados para armazenar uma palavra binária.

Um registrador, além dos *bits* de entrada e saída (complementados ou não) pode ainda ter duas linhas. A linha *set* (torna "1" todos os *bits* do registrador) e a linha *reset* (torna "0" o estado de todos os *bits* do registrador). Um registrador será indicado como visto na Fig. 3-39, podendo ou não estar explícito a quantidade de *bits* que ele armazena.

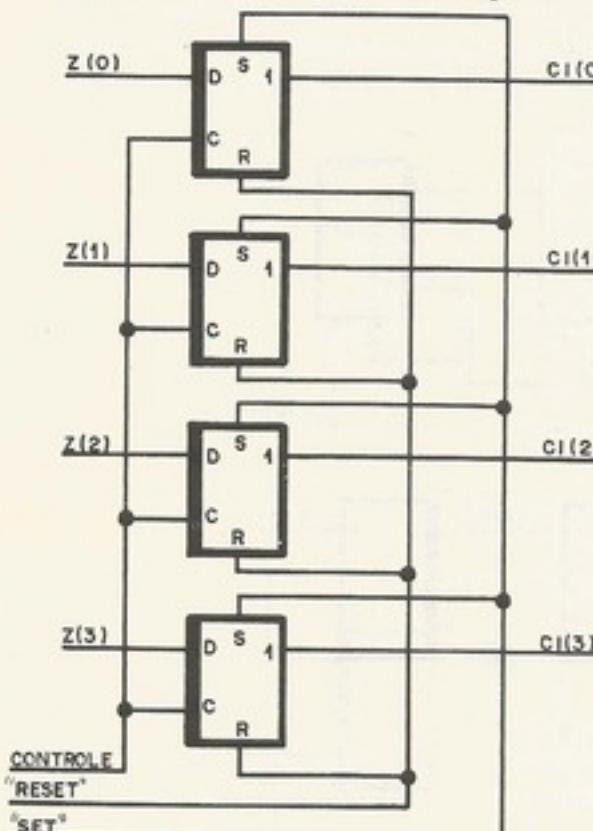


Figura 3-38. Um registrador de 4 bits com as linhas de *set* e *reset*

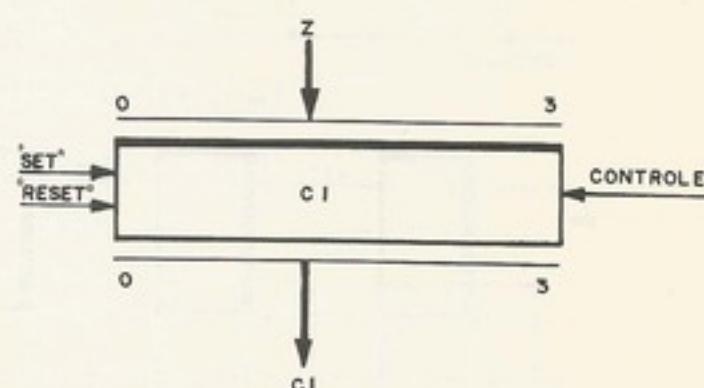


Figura 3-39. Representação de um registrador de 4 bits

Um registrador, além de armazenar um conjunto de *bits*, pode ser usado também para efetuar algumas transformações com esses dados. Para isso existem registradores especiais, como o registrador-deslocador (*shift-register*) usado para muitas coisas, entre as quais paralelizar e seriar informações; os contadores (*counter*) são usados numa função muito importante, conforme o seu próprio nome diz, isto é, contar. A seguir estudaremos essas duas classes de registradores com mais detalhes.

a. Registrador deslocador⁽⁷⁾

Um registrador-deslocador ou registrador com deslocamento, que passaremos a chamar apenas de deslocador (não confundir com *shifter*, que, na maioria das vezes, é usado para circuitos combinatórios com capacidade de deslocar), é um registrador que, com o comando de um pulso (controle) desloca todos os *bits* num dado sentido (direita ou esquerda). Por exemplo, um registrador de 4 bits que tenha gravado a palavra 1001, após um deslocamento à direita, passa a ter a palavra 0100, onde o 1 menos significativo perdeu-se, e um 0 (poderia ser 1) foi introduzido na posição mais significativa⁽⁴⁾. Para ilustrar melhor a idéia, na Fig. 3-40 vemos um exemplo.

Como aplicações bastante importantes desses registradores, podemos ressaltar⁽²⁾:

- um deslocamento à esquerda de 1 bit introduzindo zeros é equivalente a multiplicar o número codificado por 2. O deslocamento análogo à direita corresponde à divisão por dois;
- um deslocador tem a capacidade de seriar informações paralela ou paralelizar informações em série.

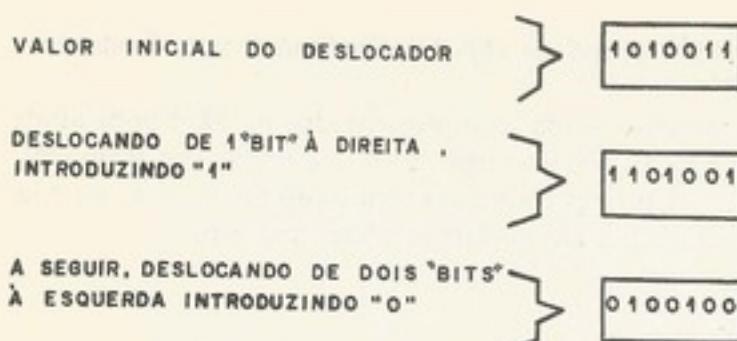


Figura 3-40. Exemplos de deslocamentos

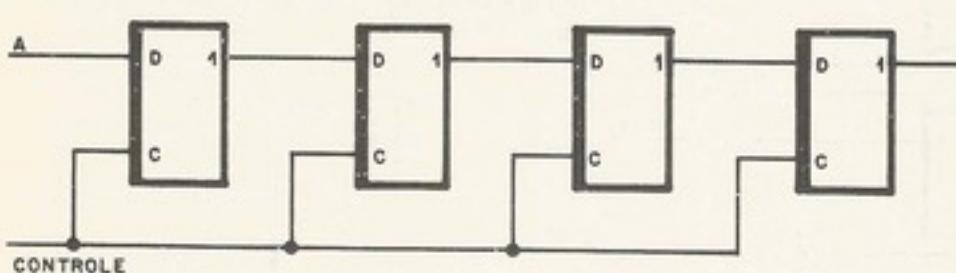
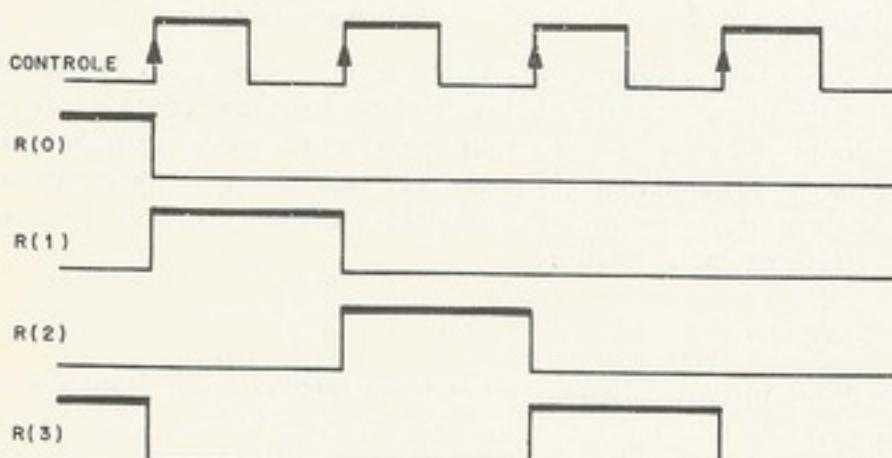
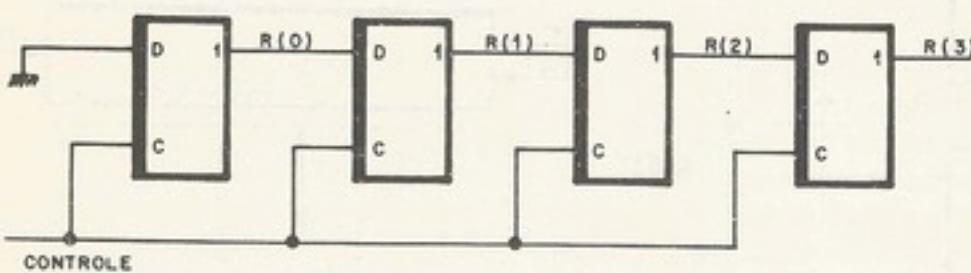


Figura 3-41. Deslocador à direita



SITUAÇÃO INICIAL	1 0 0 1
APÓS 1º PULSO DE CONTROLE:	0 1 0 0
APÓS 2º PULSO DE CONTROLE:	0 0 1 0
APÓS 3º PULSO DE CONTROLE:	0 0 0 1
APÓS 4º PULSO DE CONTROLE:	0 0 0 0

Figura 3-42. Exemplo do funcionamento do deslocador da Fig. 3-41

elementos de g...
A imprensa
Dependendo
é obtida com
onde se pode
o flip-flop tem
trole, a saída
problemas em
registram os
querida, e, em

Figura 3-44. Di...
com giro

A implementação física desses deslocadores é obtida com os tipos usuais de *flip-flops*. Dependendo deles, alguns cuidados são necessários. Uma implementação bastante simples é obtida com o *flip-flop* tipo *D* sensível à borda, como na Fig. 3-41, um deslocador à direita, onde se pode introduzir "0" ou "1", dependendo de *A*. Como vimos no item anterior (*flip-flop*) o *flip-flop* tipo *D* sensível à borda tem a importante característica de, após a subida do controle, a saída mudar, depois que a entrada fica insensível ao dado. Portanto não existem problemas com o deslocador da Fig. 3-41, pois, quando o controle sobe, todos os *flip-flops* registram os valores dos dados que, no caso, são as saídas dos *flip-flops* da vizinhança esquerda, e, antes que eles mudem, as entradas ficam insensíveis. Assim, quando as saídas

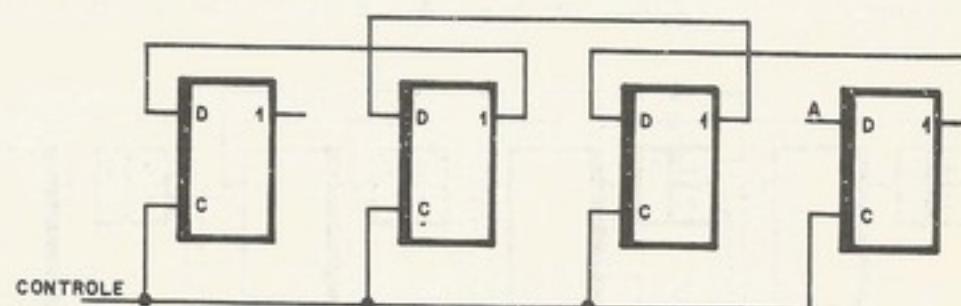


Figura 3-43. Deslocador à esquerda

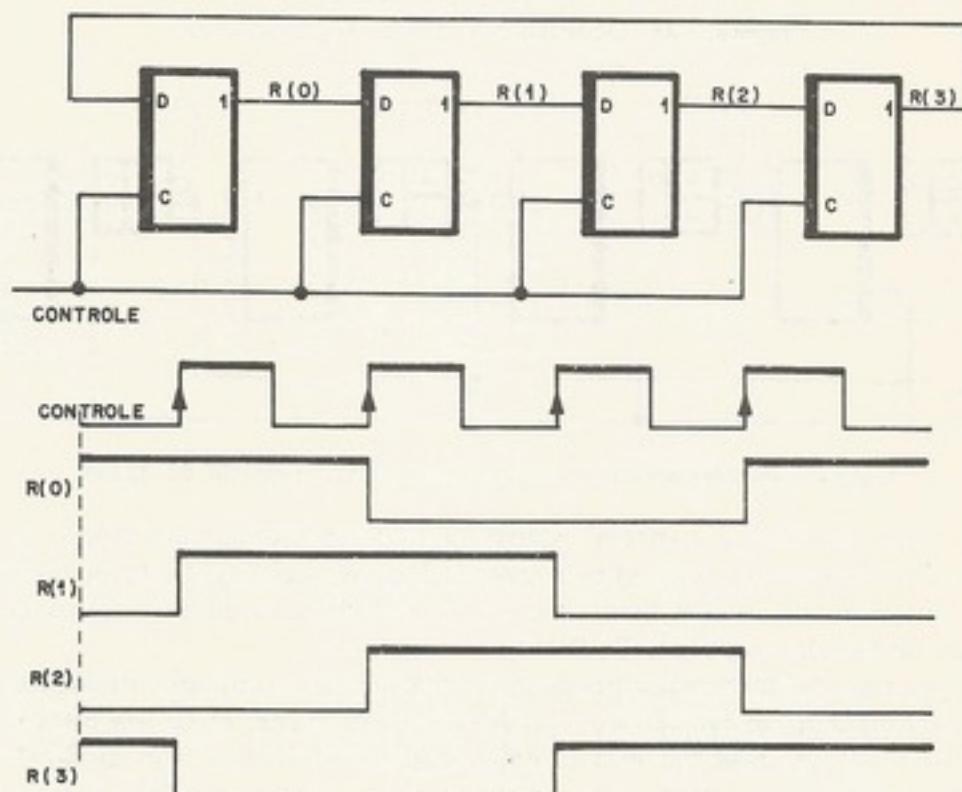


Figura 3-44. Deslocador à direita com giro

SITUAÇÃO INICIAL	1	0	0	1
APÓS 1º PULSO DE CONTROLE:	1	1	0	0
APÓS 2º PULSO DE CONTROLE:	0	1	1	0
APÓS 3º PULSO DE CONTROLE:	0	0	1	1
APÓS 4º PULSO DE CONTROLE:	1	0	0	1

passam a ter os novos valores, não existe a possibilidade de algum *flip-flop* registrar o novo valor do seu vizinho à esquerda.

Com o *flip-flop* tipo *D* sensível à borda, podemos obter outras estruturas, conforme se vê nas Figs. 3-43, 3-44, 3-45 e 3-46.

Na Fig. 3-44, vimos que podemos realimentar o bit que se perde (menos significativo no deslocamento à direita e mais significativo no deslocamento à esquerda) para o outro extremo do registrador, criando-se um novo tipo de deslocador, o deslocador com giro, muito usado em sistemas digitais. Podemos introduzir algumas portas (*gate*) na estrutura do deslocador, para torná-lo mais versátil, como na Fig. 3-45, que mostra um registrador que desloca à direita ou à esquerda, e na Fig. 3-46, um deslocador à direita com entrada paralela de dados.

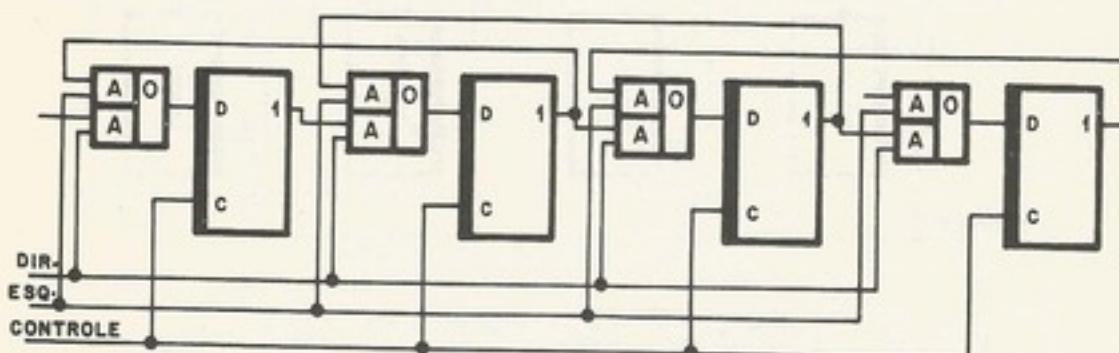


Figura 3-45. Deslocador à direita ou à esquerda

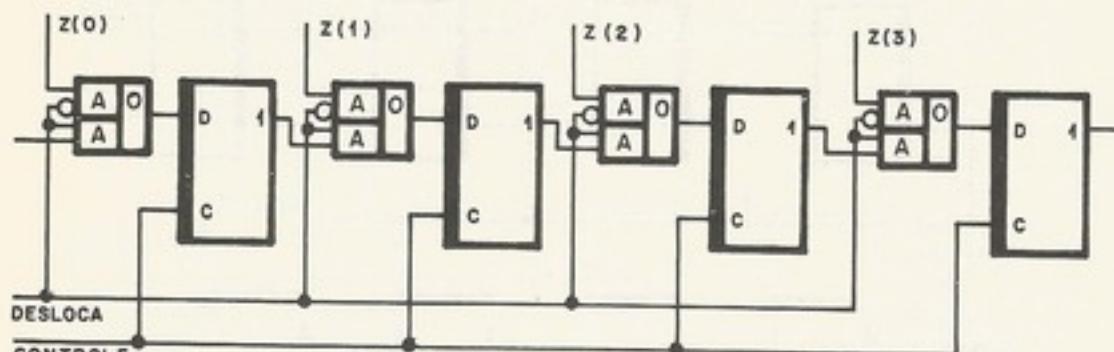


Figura 3-46. Deslocador à direita com entrada paralela de dados

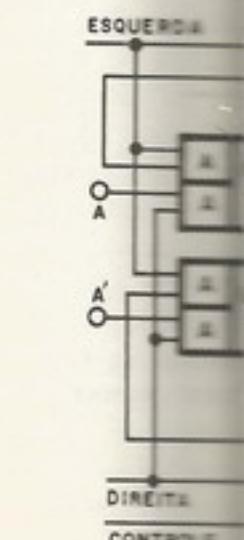
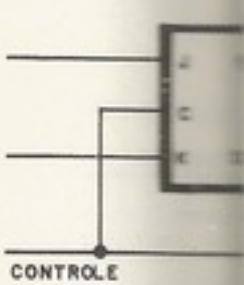
Como vimos no Cap. 2, existem sistemas digitais que funcionam com o código *BCD*. Um deslocador importante seria aquele que deslocasse dígito a dígito (4 bits). Isso pode ser obtido com um deslocador comum bit a bit, dando-se 4 deslocamentos consecutivos ou com um deslocador especial, *BCD* (Fig. 3-47).

O *flip-flop* tipo *JK* também se presta com eficiência para circuitos deslocadores. Porém tem uma desvantagem em relação ao tipo *D* sensível à borda, exige um número maior de ligações. Veja nas Figs. 3-48 e 3-49 e compare com os anteriores. Um modo de atenuar a desvantagem desses deslocadores é usarmos circuitos integrados, que, numa só pastilha, têm o *flip-flop JK* e as portas necessárias para a implantação desse modelo.

Alguns problemas surgem quando tentamos a implementação de deslocadores com *flip-flops* da classe dos sensíveis ao nível, pois, nesse caso, a saída muda enquanto a entrada ainda está sensível. Exceptuando-se o caso dos *flip-flops* da classe dos *master-slaves*, esse deslocador exige cuidados especiais^(4, 5, 7). (Veja nas Figs. 3-50 e 3-51.)

É muito importante notar que, nesse tipo de deslocador (Figs. 3-50 e 3-51), é necessário o dobro de *flip-flops* do que se usa em deslocadores equivalentes com *flip-flops* sensíveis à borda. Esse número é o dobro, pois (Fig. 3-50), quando da chegada do primeiro pulso no controle *par*, os *flip-flops* nas posições pares (0 e 2) perdem a informação que tinham para

Figura 3-47. Deslocador *BCD* de 3 dígitos



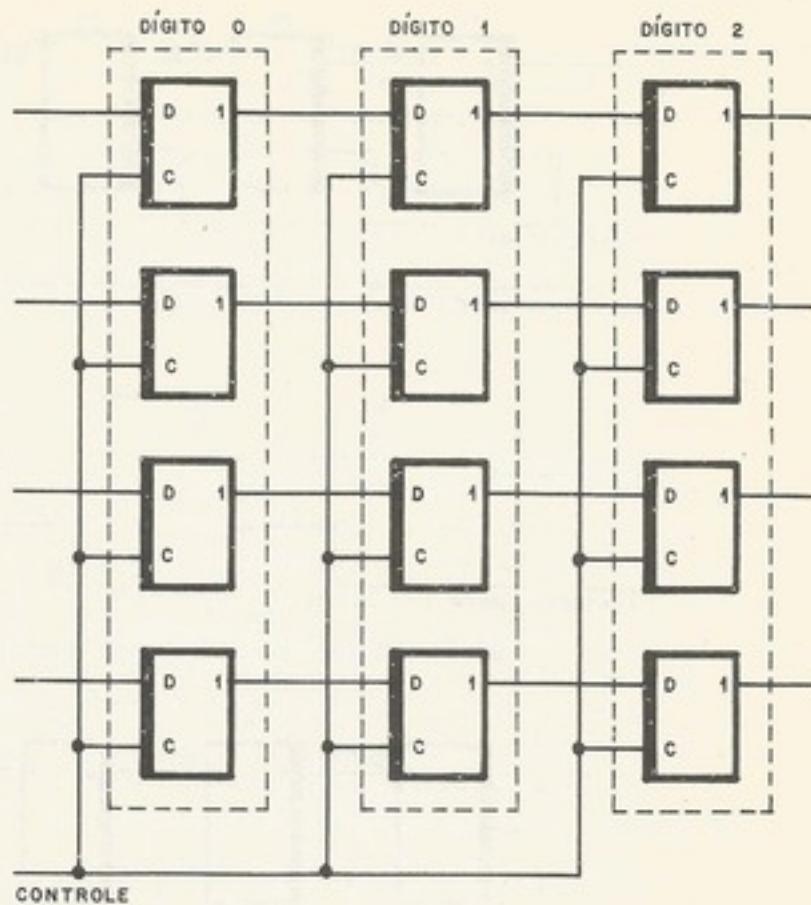


Figura 3-47. Deslocador à direita BCD de 3 dígitos

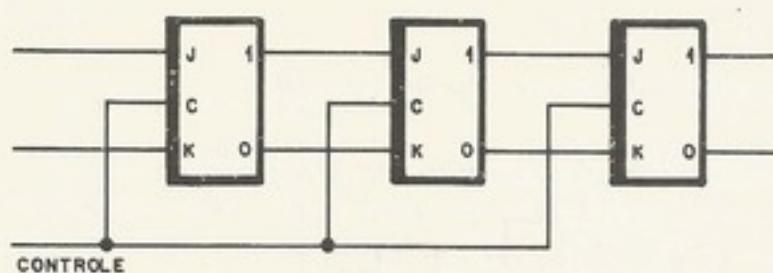


Figura 3-48. Deslocador à direita

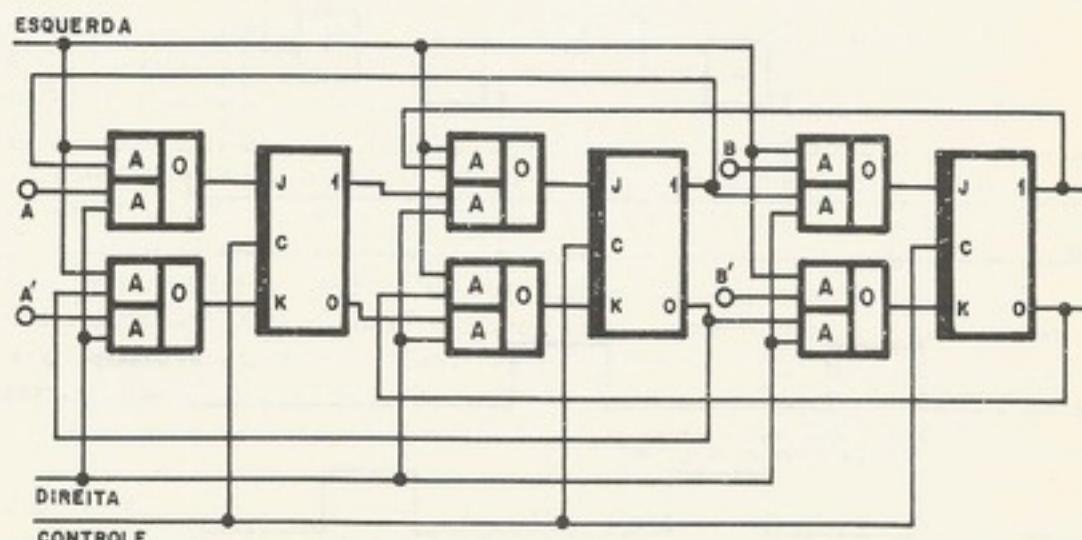


Figura 3-49. Deslocador à direita ou à esquerda

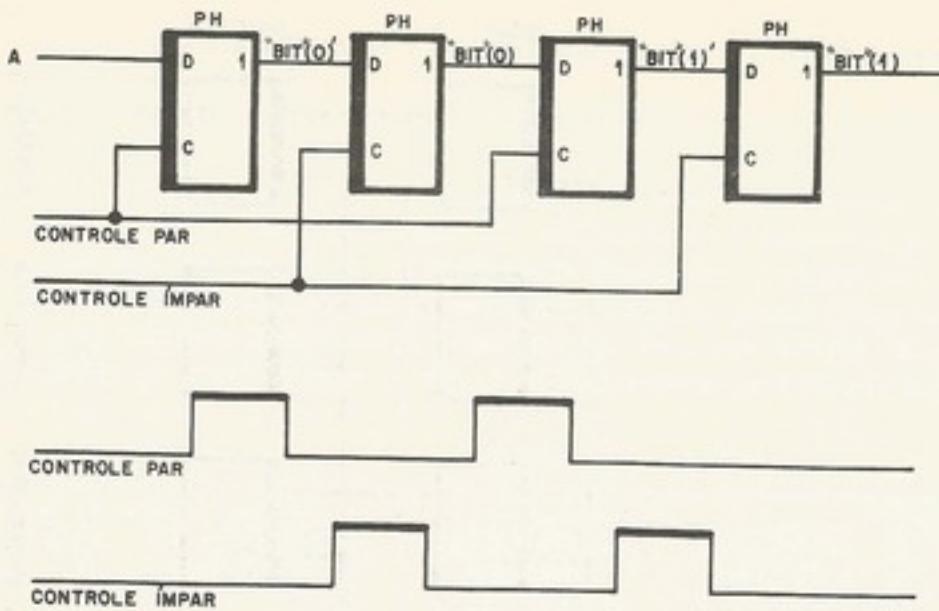


Figura 3-50. Deslocador à direita com flip-flop PH

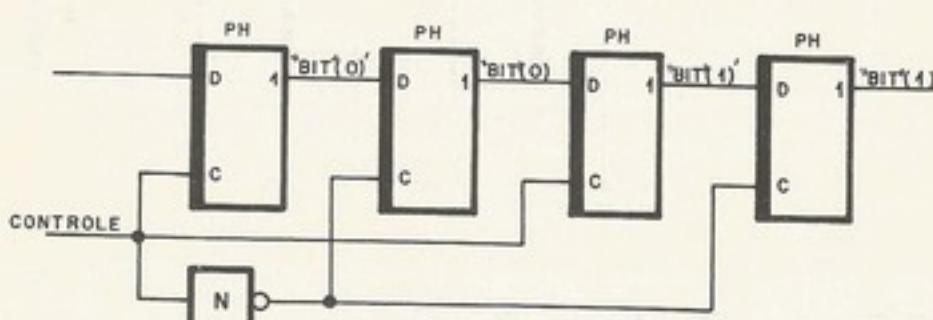


Figura 3-51. Deslocador à direita com flip-flop tipo PH

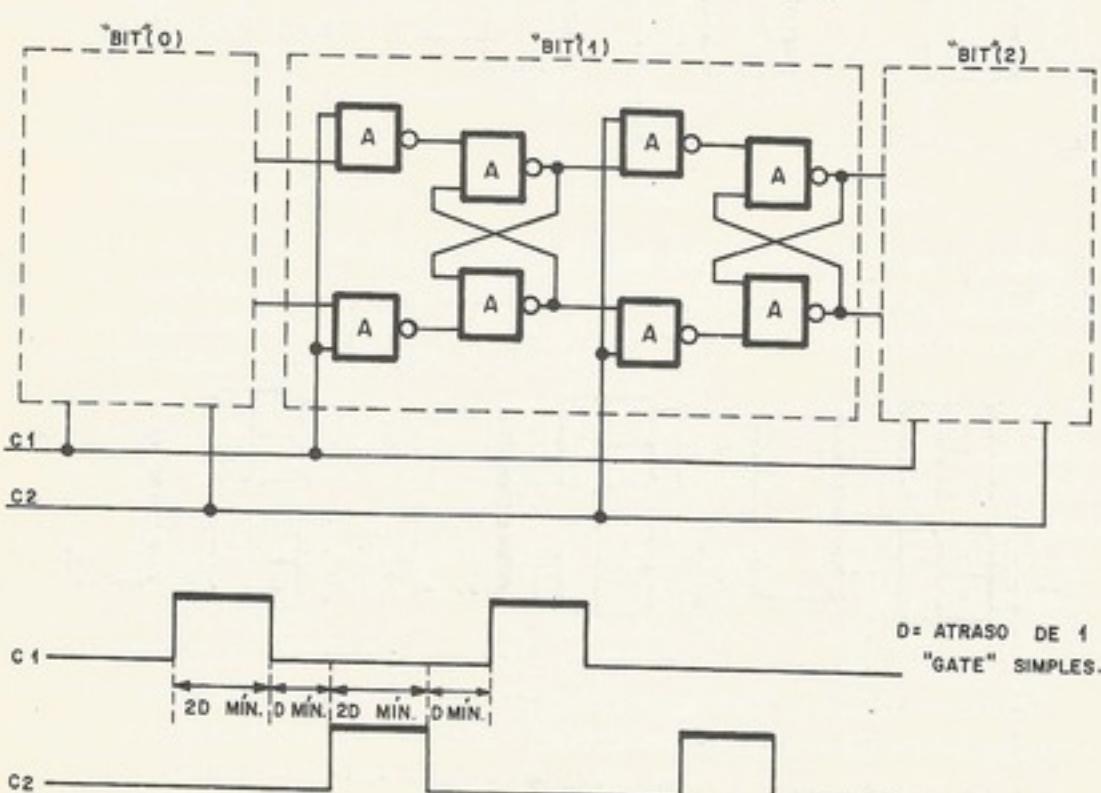


Figura 3-52. Deslocador implementado com portas tipo NAND

register e receber a informação útil. O leitor já deve ser idêntica à dos flip-flops.

Existem inúmeras tipos de flip-flops.

Convém que o designer poderá utilizar os tipos mais comuns e modelos.

b. Registradores

Habitualmente, o registrador é um registrador que armazena um número em seu conteúdo binário, que poderá ser conservado a zero com o pulso de reinício.

Mais genericamente, é que, com a chegada de uma sequência de pulsos, o conteúdo muda.

Dentre as inúmeras possibilidades:

- em sistema de contagem
- como divisor de frequência

Inicialmente analisaremos o registrador binário.

c. Contadores

O tipo mais comum de flip-flop que, combinado com outros flip-flops, pode fornecer números módulo 2 ou 5.

Para contadores de grande capacidade, isto é, que contam até 1000, chamados de *counters*, é o uso do flip-flop tipo SR, que tem a função de inverter o sinal de controle.

A partir de dois flip-flops simplesmente ligados em paralelo, obteremos um contador de 2 bits, chamados de *counter modulo 2*. Se o número 1011 (que é 11 em binário) for o número final que o contador passará, teremos:

Existem contadores de "contagem baixa" (*down-counters*) e contadores de "contagem ascendente" (*up-counters*). No caso de um contador "par", todos os bits são invertidos.

Todos os contadores de "contagem ascendente" (*ripple carry counters*) só se tivermos registros de

registrar e recebem a dos *flip-flops* à esquerda deles (posições ímpares), que realmente têm a informação útil, que continuará sendo deslocada com a chegada do pulso *controle impar*. O leitor já deve ter notado que, nesse esquema de deslocadores, forçou-se uma estrutura idêntica à dos *flip-flops* da classe *master-slave* (veja a Fig. 3-51).

Existem inúmeras maneiras de se implementarem deslocadores com portas simples tipo *NAND* ou *NOR*. A maioria delas é com estruturas tipo *master-slave* (Fig. 3-52)⁽⁵⁾.

Convém que o leitor tenha em mente que, quando necessitar de deslocadores tipo padrões, poderá utilizar circuitos integrados que tenham esses deslocadores em inúmeros tamanhos e modelos.

b. Registrador-contador^(2, 6, 8)

Habitualmente, um registrador-contador, que passará a ser chamado de *contador* apenas, é um registrador que, com a chegada de um pulso de controle, incrementa (ou decrementa) um em seu conteúdo. Por isso, é lógico que, dependendo do tipo de codificação (binária, *BCD*, biquinária, etc.), o contador terá uma estrutura particular. Ainda mais, um contador poderá ser construído de modo tal que, quando atingir um certo número $N - 1$, ele volta a zero com o pulso seguinte; nesse caso, dizemos que o contador é *módulo N*, ou *divide por N*.

Mais genericamente, contador módulo N é um registrador que tem N estados internos e que, com a chegada de um pulso, muda de estado de maneira que, com a chegada consecutiva de pulsos, o contador fica passando pelos seus N estados internos.

Dentre as inúmeras aplicações dessas estruturas contadoras podemos ressaltar duas⁽²⁾:

- a) em sistemas digitais onde se quer contar um certo número de pulsos;
- b) como divisor de frequência.

Inicialmente analisaremos os tipos de *contadores binários* que contam segundo o código binário.

c. Contadores binários

O tipo mais simples, talvez a unidade fundamental dos contadores, é composto de um *flip-flop* que, com a chegada de um pulso, muda de estado (Fig. 3-53). São contadores binários módulo 2 ou divide por $2^{(2)}$.

Para contadores, é bastante útil a utilização de *flip-flop* tipo *D* sensível à borda de descida, isto é, que copia o dado quando o sinal de controle [nos contadores, normalmente chamados de *trigger* (gatilho)], muda o nível “1” para o nível “0” (Fig. 3-54). Também comum é o uso do *flip-flop JK master-slave* que tem essa característica; a saída muda na descida do sinal de controle.

A partir de contadores módulo 2, podemos obter contadores de módulo 4, 8, 16 etc., simplesmente ligando-os em cascata (Fig. 3-55). Ligando n contadores módulo 2 em cascata, obteremos um contador de módulo 2^n . Analise os gráficos da Fig. 3-55 e verifique por que os chamados de contadores binários. Verifique que, se tivermos registrado, por exemplo, o número 1011 (que representa o número decimal 11), com a chegada de um pulso em *T*, o contador passará para o estado 1100 (que representa o número decimal 12).

Existem contadores “para cima” (*up-counter*) que incrementam 1, e contadores “para baixo” (*down-counter*) que decrementam 1 com a chegada de um pulso na entrada *T*. Se, nos contadores do tipo apresentado na Fig. 3-55(a), usássemos *flip-flops* tipo *D* sensíveis à borda de subida, teríamos um contador “para baixo” (verifique). Na Fig. 3-56 encontra-se um contador “para baixo”.

Todos os contadores apresentados até agora são do tipo com propagação do “vai um” (*ripple carry counter*) às vezes chamado de “assincrono”. A justificativa desse nome é simples: se tivermos registrado no contador o número 1111, com a chegada do pulso em *T*, o primeiro

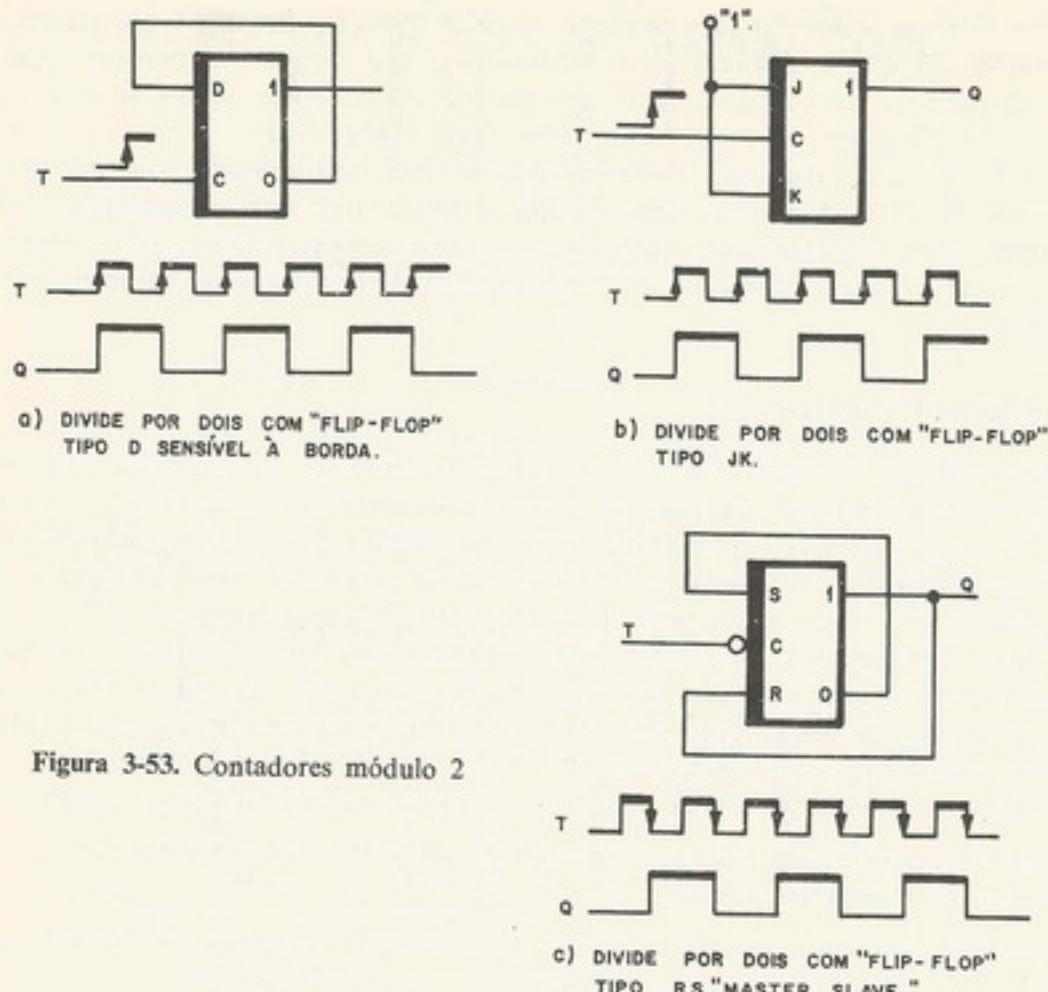


Figura 3-53. Contadores módulo 2

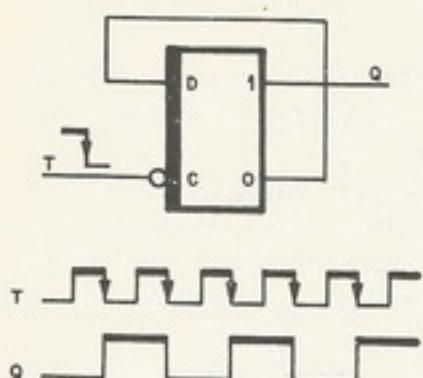
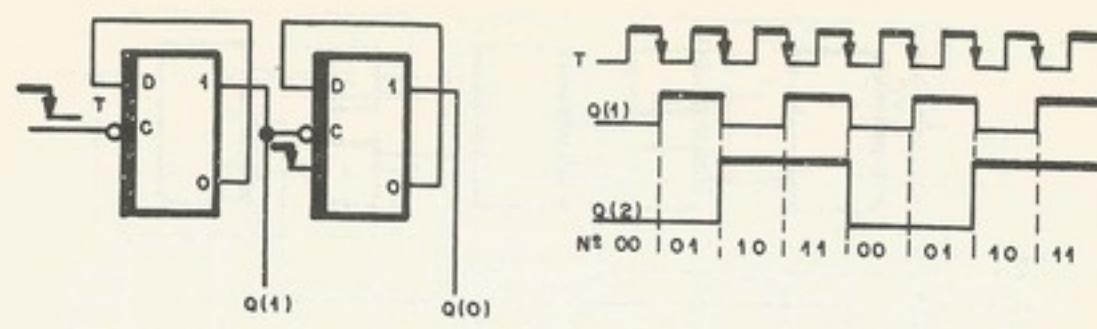


Figura 3-54. Contador módulo 2 com flip-flop tipo D sensível à borda de descida

flip-flop muda para "0" (portanto "vai um"), o que provoca uma mudança no segundo (então "vai um"), o que mudará o terceiro e assim por diante; portanto o "vai um" vai se propagando desde o flip-flop da entrada até o primeiro flip-flop que tenha registrado um "0". Esses contadores têm um inconveniente, o grande atraso de propagação. O atraso do contador será a soma dos atrasos dos flip-flops, o que limita a sua freqüência de operação.

Evita-se o atraso de propagação com estruturas do tipo apresentado na Fig. 3-57, chamado "vai um simultâneo" (*simultaneous carry counter*) às vezes chamado de "síncrono". Esse contador, com a chegada de um pulso em T , muda todos os flip-flops necessários, simultaneamente, sem que um flip-flop, em posições mais significativas, espere que outros mudem para depois mudar (observe que o controle é comum a todos os flip-flops). O inconveniente desse tipo de estrutura é o uso de portas com tanto maior *fan-in* quanto maior for o número de flip-flops usados.



A) CONTADOR BINÁRIO MÓDULO 4

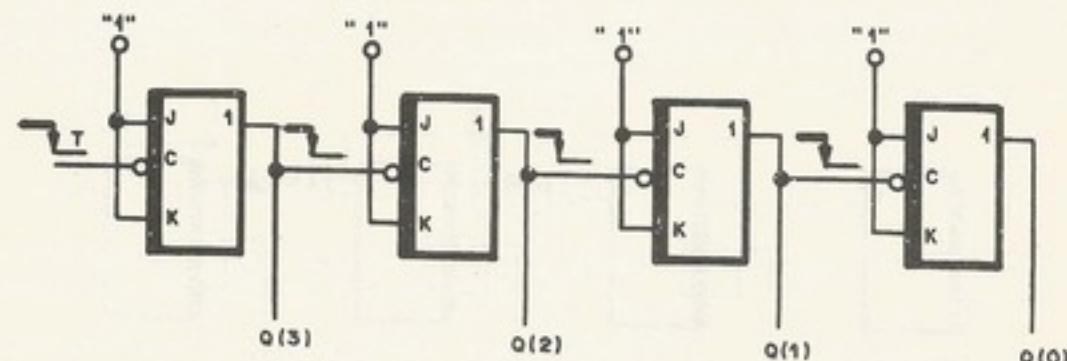
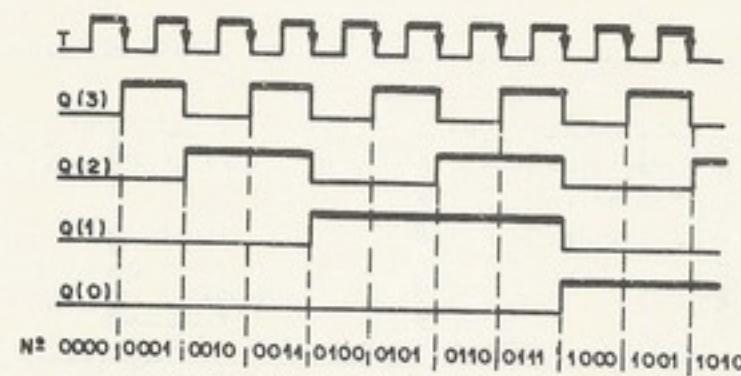


GRÁFICO DOS SINAIS



B) CONTADOR BINÁRIO MÓDULO 4

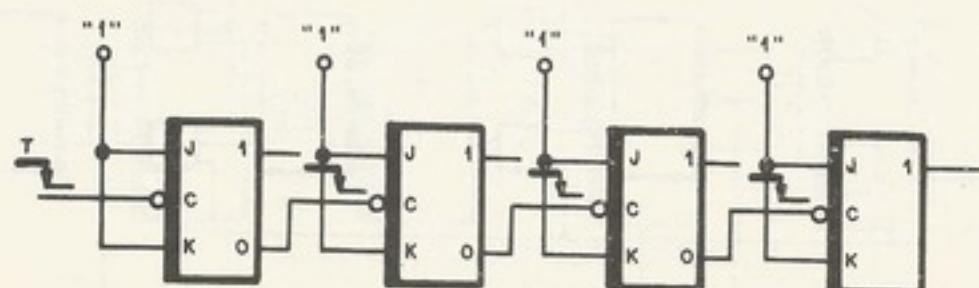
Figura 3-55. Contadores binários módulo 2ⁿ

Figura 3-56. Contador binário "para baixo"

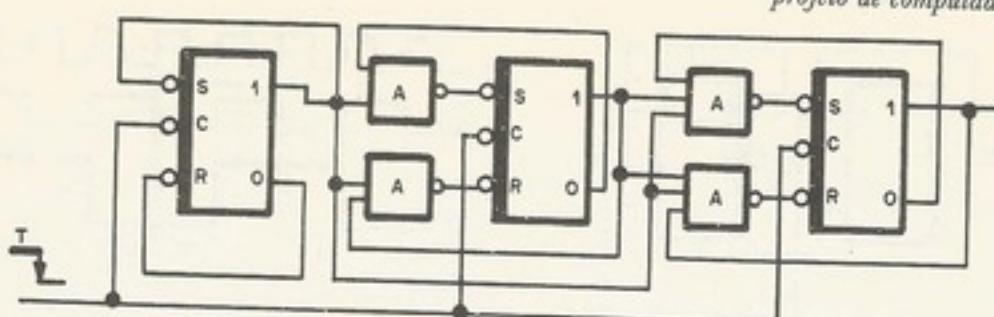


Figura 3-57. Contador binário "vai um simultâneo"; flip-flop RS, master-slave

Uma outra filosofia de contadores "vai um simultâneo" é visto na Fig. 3-58, onde vemos que o número de portas é menor e que elas não têm o *fan-in* aumentado com o aumento do número de flip-flops, contudo existe uma limitação que não existia modelo da Fig. 3-57: devido aos atrasos das portas, a distância entre dois pulsos de *trigger* tem um limite mínimo, dado pela soma dos atrasos de todas as portas.

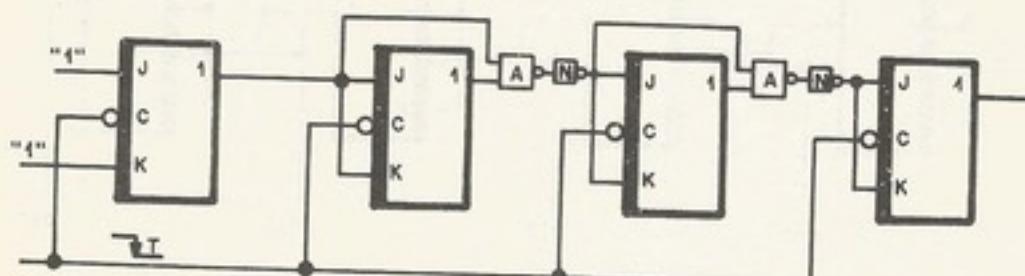


Figura 3-58. Contador binário, módulo 16, "vai um simultâneo"

A diferença funcional entre os contadores das Figs. 3-57 e 3-58 está no fato de que, no primeiro, um determinado flip-flop "olha" todos anteriores e só muda quando todos antes dele forem "1"; no segundo, um flip-flop qualquer "olha" apenas se o anterior deve mudar e se ele é igual a "1" (portanto mudará de "1" para "0").

O leitor encontrará inúmeras figuras com os flip-flops JK com as entradas J ou K abertas. Se supomos que esses flip-flops são da família TTL, significa estarmos colocando, permanentemente, nível "1" nessas entradas.

Dentro dessa mesma filosofia de contadores "vai um simultâneo", na Fig. 3-59, vemos um contador "para baixo" e "para cima", muitas vezes chamado de contador reversível.

Os circuitos das Figs. 3-57 e 3-58 podem utilizar flip-flops JK com portas já internas. Mesmo assim, pode-se tentar economizar portas usando estruturas que estão classificadas entre os tipos "propagação do vai um" e "vai um simultâneo" (Fig. 3-60)⁽⁷⁾.

Como vimos, contadores binários módulo 2ⁿ são obtidos associando-se em cascata *n* contadores de módulo 2. Para obtermos contadores de módulo diferente de potências de 2,

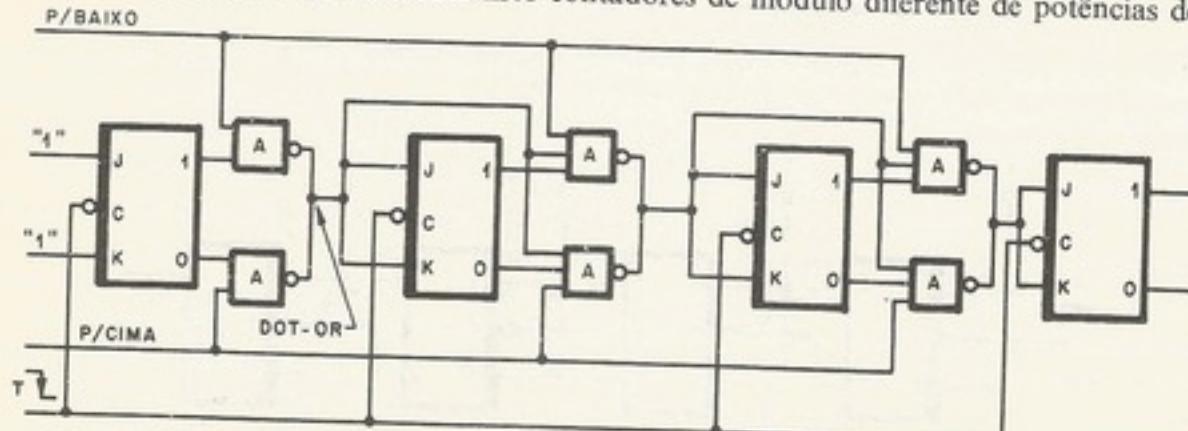
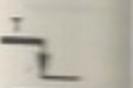


Figura 3-59. Contador binário reversível, módulo 16, "vai um simultâneo"

Figura 3-60. Contador

As entradas J e K, utilizamos técnicas obtido de um contador gamos o número 011, 100 e volta pa



RESET



RESET

Figura 3-60

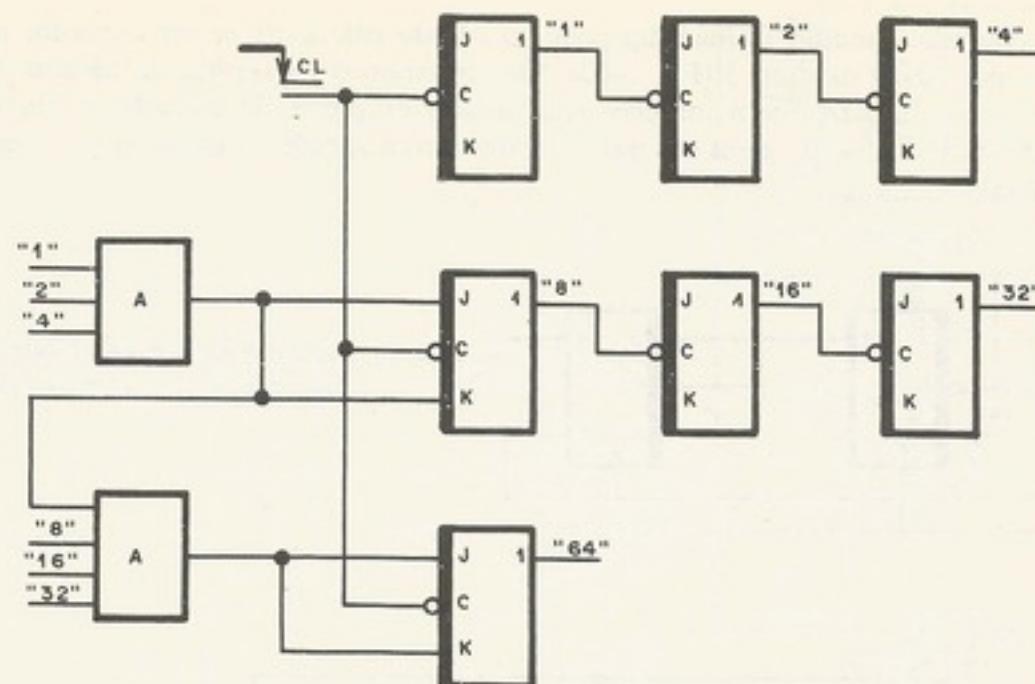


Figura 3-60. Contador binário módulo 128, do tipo "propagação do vai um" com "vai um simultâneo". As entradas J e K em aberto estão no nível "1".

utilizamos técnicas de realimentação. Por exemplo (Fig. 3-61), um contador módulo 5 é obtido de um contador módulo 8, onde, através de uma realimentação (em negrito), obrimos o número 000 a suceder o 100. Portanto ele passa por cinco estados: 000, 001, 010, 011, 100 e volta para o início.

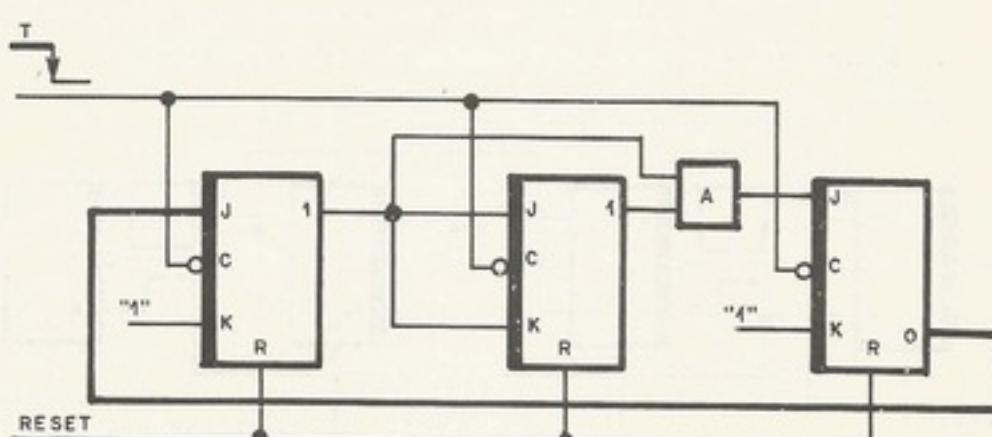


Figura 3-61. Contador binário "vai um simultâneo" módulo 5

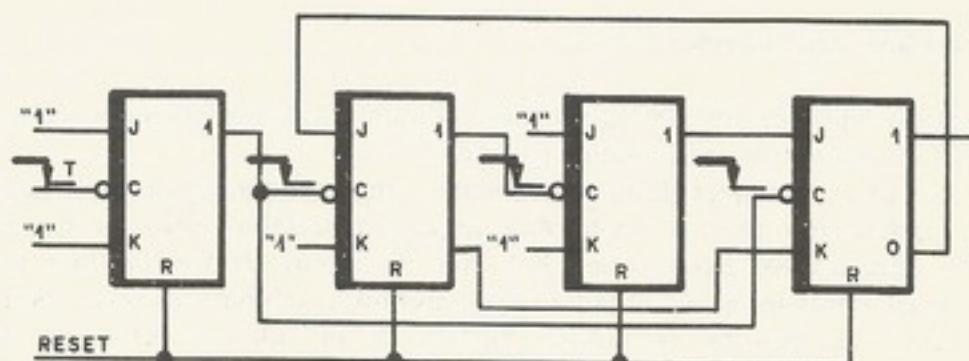


Figura 3-62. Contador binário módulo 10, com "propagação do vai um"

Um contador módulo 10 (década) pode ser obtido colocando-se um contador módulo 2 seguido por outro módulo 5 (Fig. 3-62). Não trataremos da técnica de projeto de contadores⁽⁶⁾ e nos limitaremos a fornecer mais alguns exemplos de contadores binários de módulo N , onde a filosofia geral é forçar o estado zero a suceder o estado $N - 1$ (Figs. 3-63, 3-64 e 3-65).

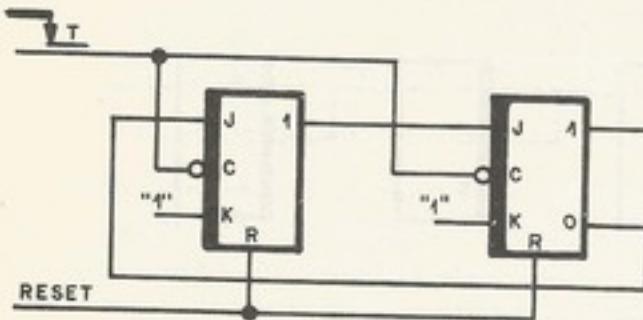


Figura 3-63. Contador binário módulo 3, "vai um simultâneo"

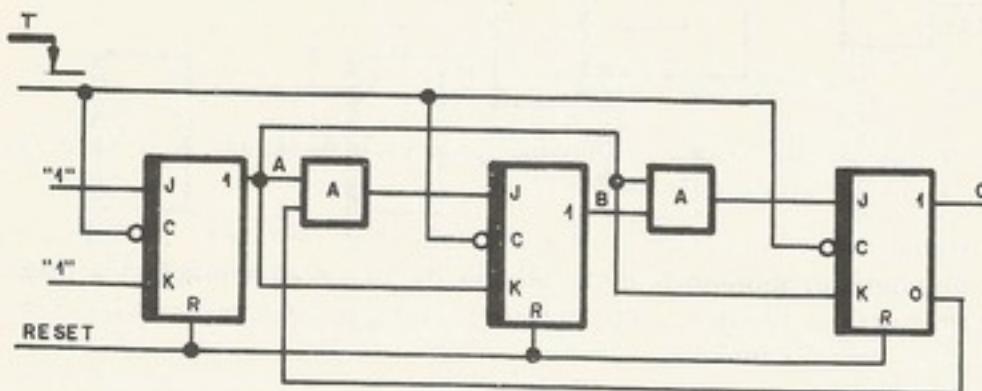


Figura 3-64. Contador binário módulo 6, "vai um simultâneo"

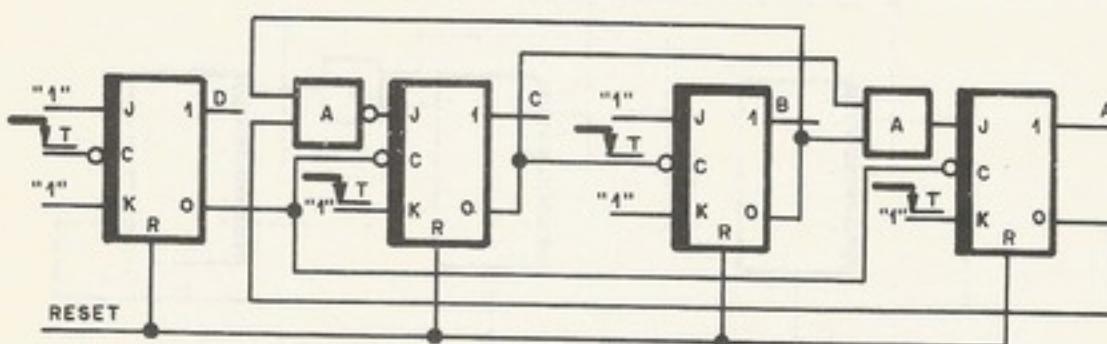


Figura 3-65. Contador binário para baixo, com "propagação do vai um" módulo 10

d. Contadores não-binários

Contadores não-binários são aqueles que contam em outros códigos, como o código de Gray, BCD excesso de três, biquinário etc.

Um meio geral e simples de se implementar um contador é visto na Fig. 3-66, onde se utilizam flip-flops tipo D sensível à borda e um circuito combinatório que, dado um estado do contador, gera o novo estado, que é colocado na entrada D dos flip-flops. O módulo e o código desses contadores são ditados pelo circuito combinatório (essa mesma técnica pode ser usada para projetar contadores binários de módulos diferentes de 2^n).

Para ilustrar a técnica, vejamos um exemplo: queremos um contador módulo 8 que conte no código de Gray. Na Fig. 3-67, vemos os estados consecutivos desse código.

Figura 3-66. Contador binário módulo 3, "vai um simultâneo"

Da Fig. 3-67, $Q(2)$ são as entradas e $D(2)$ são as saídas. Vamos o circuito

A desvantagem dos flip-flops JK, já que

Figura 3-68. Tabela de verdade da Fig. 3-67

Figura 3-66. Contador genérico, "vai um simultâneo"

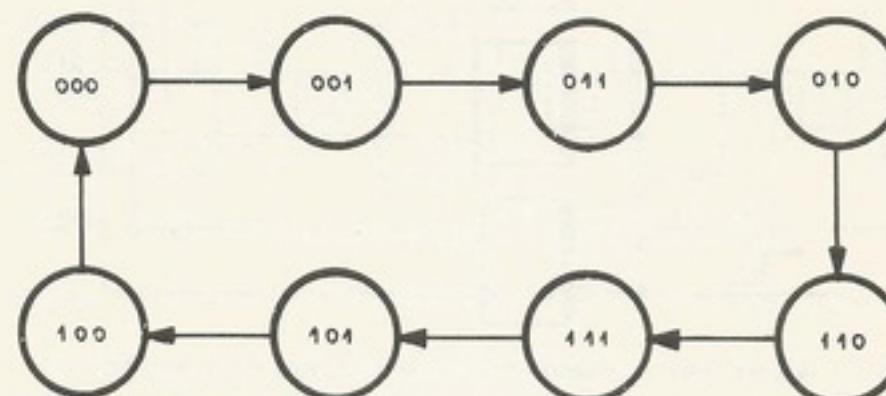
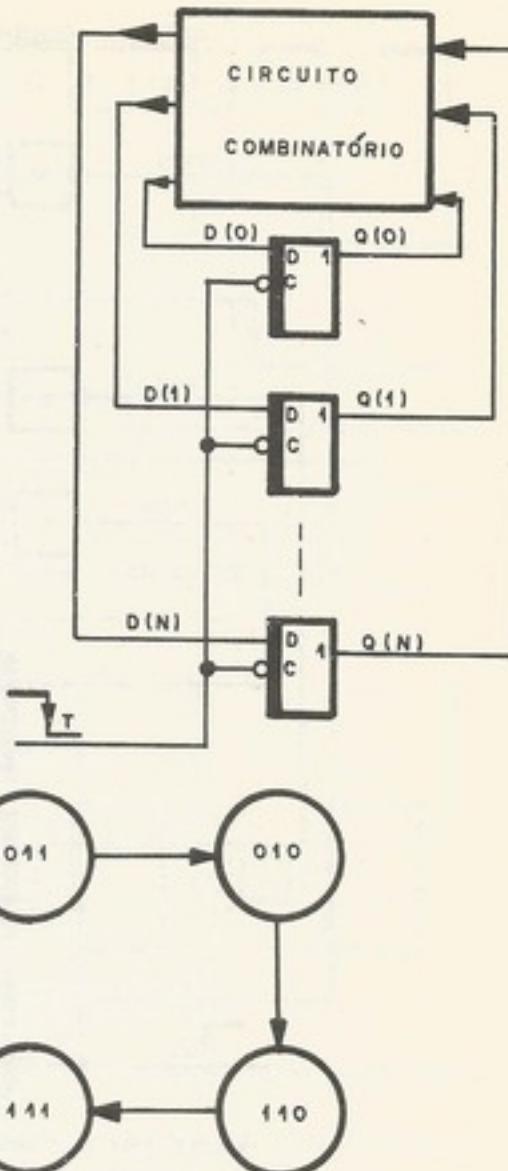


Figura 3-67. Estados consecutivos de um contador no código de Gray

Da Fig. 3-67, montamos a tabela da verdade da Fig. 3-68, onde os sinais $Q(0)$, $Q(1)$ e $Q(2)$ são as entradas do circuito combinatório (saídas dos flip-flops) e os sinais $D(0)$, $D(1)$ e $D(2)$ são as saídas do circuito combinatório (entrada dos flip-flops). Dessa tabela, projetamos o circuito combinatório da Fig. 3-69.

A desvantagem dessa técnica reside no fato de algumas vezes ser inefficiente para flip-flops JK, já que estes apresentam um modo de operação que permite a síntese de contadores com estruturas bastante simples.

ESTADO ATUAL			ESTADO SEGUINTE		
$Q(0)$	$Q(1)$	$Q(2)$	$D(0)$	$D(1)$	$D(2)$
0	0	0	0	0	1
0	0	1	0	1	1
0	1	0	1	1	0
0	1	1	0	1	0
1	0	0	0	0	0
1	0	1	1	0	0
1	1	0	1	1	1
1	1	1	1	0	1

Figura 3-68. Tabela da verdade do contador da Fig. 3-67

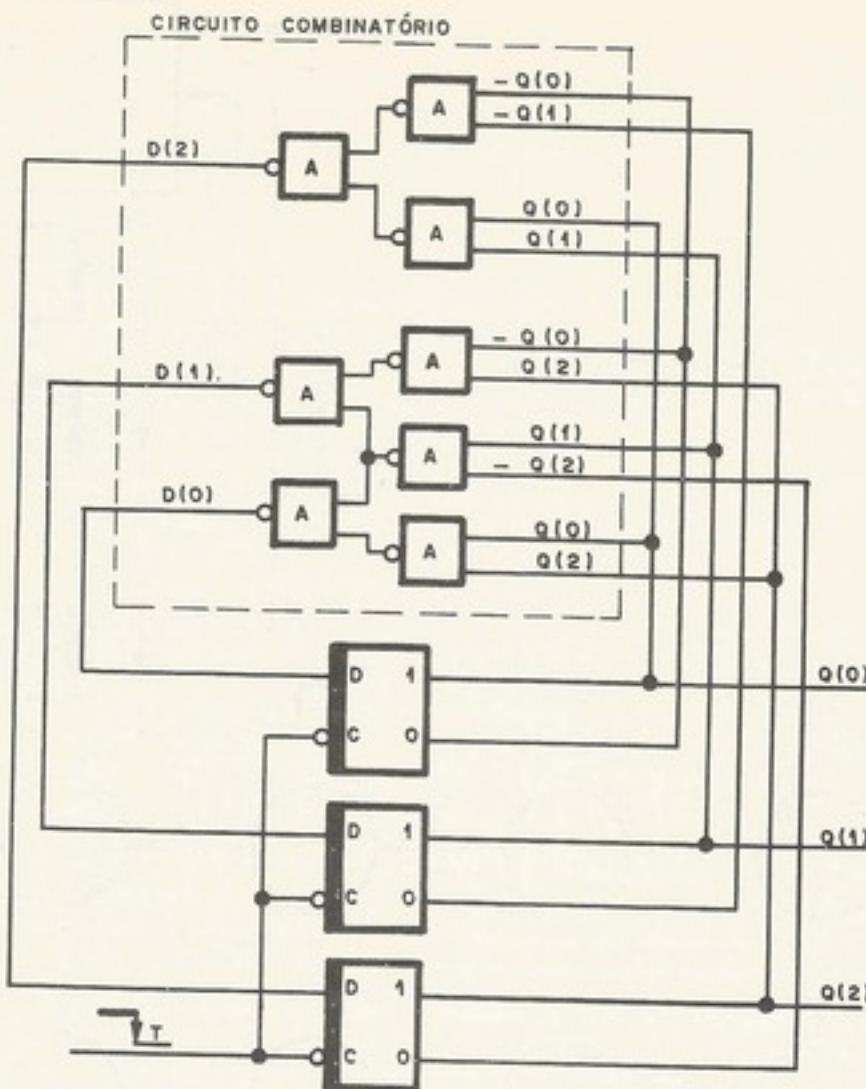


Figura 3-69. Contador em código de Gray da Fig. 3-68

Uma outra maneira de sintetizar contadores é o uso da intuição, em que se necessita, talvez, de um pouco de arte para se chegar a uma boa solução. Portanto esse processo é de difícil mecanização. Para ilustrar, tomemos o mesmo exemplo anterior, *contador no código de Gray*⁽⁷⁾. Analisemos o gráfico apresentado na Fig. 3-70, onde vemos o seguinte:

- o estágio *A* muda em toda a borda negativa do sinal de controle;
- o estágio *B* muda na borda positiva do sinal de controle quando o estágio *A* está no nível "1";
- o estágio *C* muda na borda positiva do sinal de controle quando os estágios *A* e *B* estão nos níveis "0" e "1", respectivamente;
- o estágio *D* muda na borda positiva do sinal de controle quando os estágios *A* e *B* estão nos níveis "0" e "0", respectivamente.

Com esses itens, a implementação do circuito da Fig. 3-71 é óbvio.

Como já dissemos, o nosso propósito não é analisar com detalhes a técnica de projeto de contadores. Por isso, mostraremos mais alguns exemplos que acreditamos serem úteis para ilustrar a filosofia do projeto. O leitor que necessitar de mais detalhes poderá recorrer à bibliografia do final deste capítulo. Todos os exemplos são acompanhados de tabelas e gráficos que explicam o seu funcionamento. Neles o leitor descobrirá que uma classe importante de contadores é formada por deslocadores. O contador mais comum obtido de deslocadores é o chamado "contador em anel" ou (*ring counter*), um deslocador que tem

Figura 3-71. Contador

armazenada uma posição que ficará circulando entre os circuitos decodificadores.

EXERCÍCIOS

- 3-13. Projete um
 - 3-14. Projete um
 - 3-15. Faça um gráfico
 - 3-16. Faça um diagrama
 - 3-17. Qual o módulo
 - 3-18. Qual o módulo
 - 3-19. Projete uma
- sensíveis à borda; (a) Assinale as diferenças entre os resultados obtidos para projetar um
- para projetar um
- comparar seu resultado com o resultado obtido no projeto anterior.
- 3-21. (a) Projete um
- o circuito se comporta como
- se comportar como
- quando "modo" = 1. Utilize um pulso de *clock* multiplicado por 2 para gerar a entrada paralela de

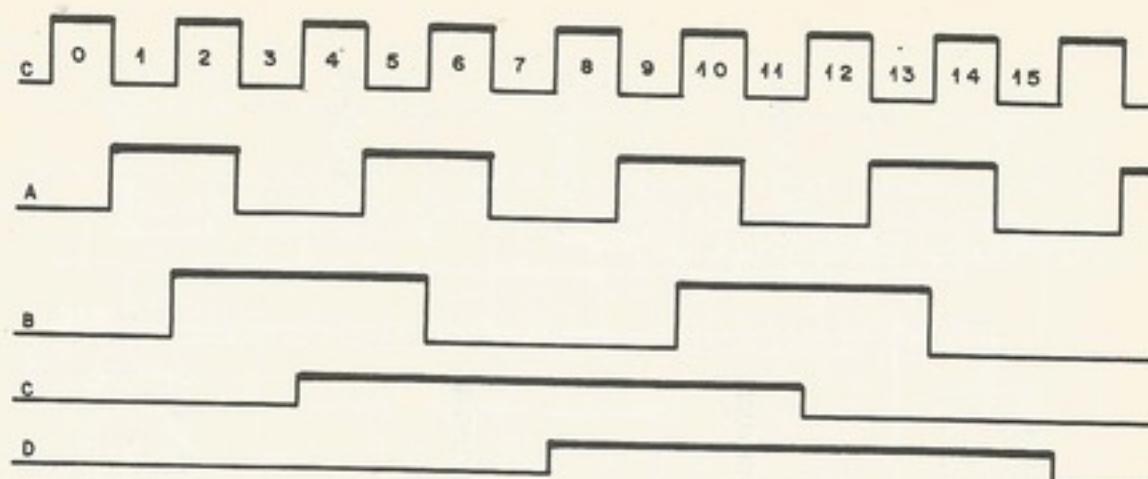
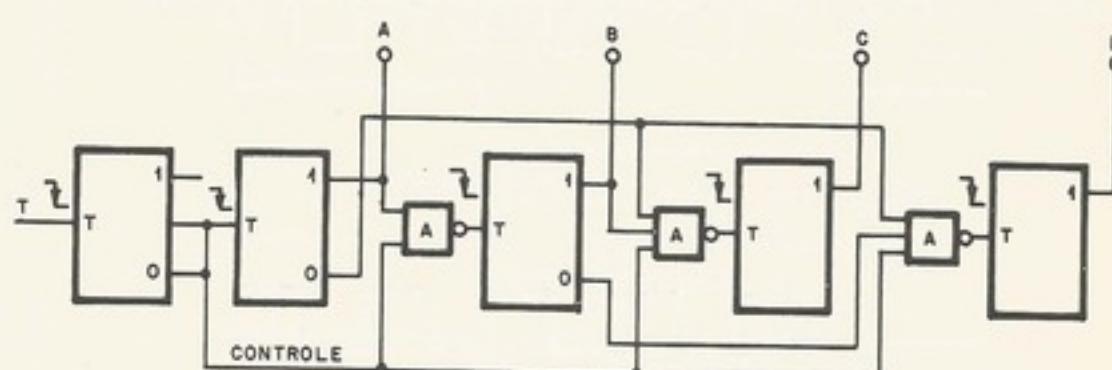
Figura 3-70. Gráfico (*timing*) dos sinais num contador de Gray

Figura 3-71. Contador no código de Gray ("propagação do vai um"). O bloco é um contador módulo 2

armazenada uma palavra apenas de zeros, com exceção de um bit que será "1". Esse "1" ficará circulando no deslocador. A vantagem desse tipo de contador é que ele dispensa circuitos decodificadores⁽²⁾.

EXERCÍCIOS

- 3-13. Projete um contador binário módulo 7 com *flip-flops* tipo JK.
- 3-14. Projete um contador binário módulo 9 com *flip-flops* tipo D sensíveis à borda.
- 3-15. Faça um gráfico de sinais (*timing*) para o contador da Fig. 3-64.
- 3-16. Faça um diagrama de estados seqüenciais para o contador da Fig. 3-65.
- 3-17. Qual o módulo de um contador em anel (*ring counter*) com n *flip-flops*?
- 3-18. Qual o módulo de um contador de Johnson (*twisted ring counter*) com n *flip-flops*?
- 3-19. Projete uma década contadora que conte segundo o código BCD usando, (a) *flip-flops* tipo D sensíveis a borda; (b) *flip-flops* tipo JK *master-slave*. Apresente os diagramas de *timing* desses circuitos. Assinale as diferenças básicas nesses diagramas.
- 3-20. Utilize uma técnica análoga à desenvolvida no texto e ilustrada nas Figs. 3-66, 3-67, 3-68 e 3-69 para projetar um contador com quatro *flip-flops* JK, contando no código BCD excesso de três. Compare seu resultado com a Fig. 3-76.
- 3-21. (a) Projete um circuito com quatro *flip-flops* e duas entradas, *clock* e "modo". Quando "modo" = 1, o circuito se comporta como um contador binário, módulo 16. Quando "modo" = 0, o circuito passa a se comportar como um deslocador, forçando zeros na posição menos significativa. Em outras palavras, quando "modo" = 1, cada pulso de *clock* incrementa um no registrador e, quando "modo" = 0, cada pulso de *clock* multiplica o seu conteúdo por dois. (b) Modifique o circuito de modo que seja possível a entrada paralela de dados. É claro que será necessária mais uma linha de controle.

3-22. Redesenhe o gráfico de sinais da Fig. 3-55(b), colocando em evidência os atrasos de propagação do "vai um". (b) Suponha que exista um bloco lógico *OR* cujas entradas são alimentadas pelas saídas $Q(1)$ e $Q(0)$ do contador. Desenhe o gráfico do sinal na saída desse *OR* junto com as curvas (onde se ressaltam os atrasos). Comente.

3-23. (a) Projete um contador "para baixo" análogo ao da Fig. 3-57. (b) Admita que o atraso de um *flip-flop* seja 20 ns e o atraso de um bloco lógico 10 ns. Calcule a máxima freqüência de operação do contador.

3-24. Repita o exercício anterior para a Fig. 3-58.

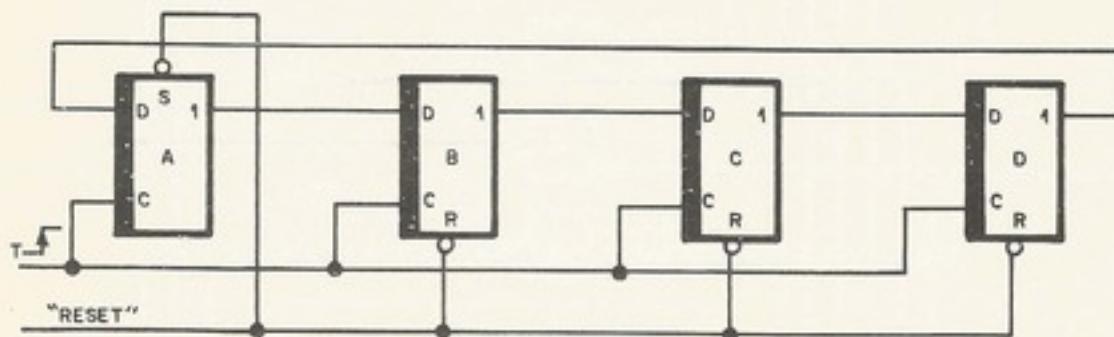


GRAFICO DOS SINAIS

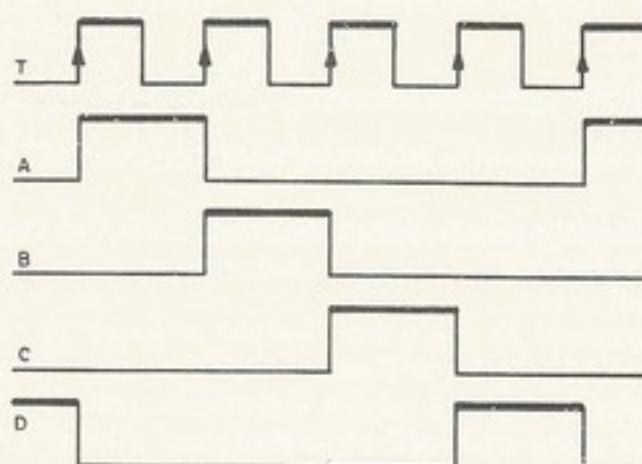
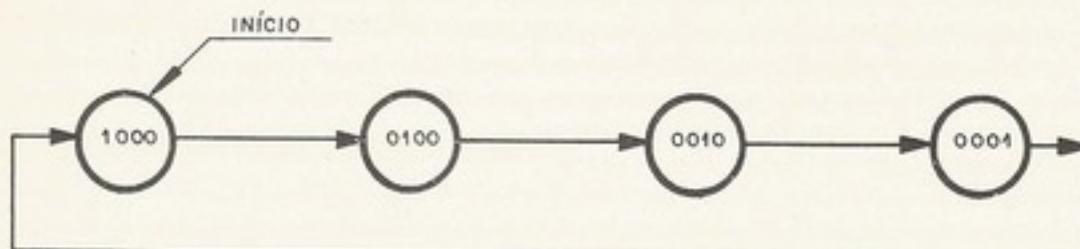


DIAGRAMA DOS ESTADOS SEQUENCIAIS

Figura 3-72. Contador em anel (*ring counter*)

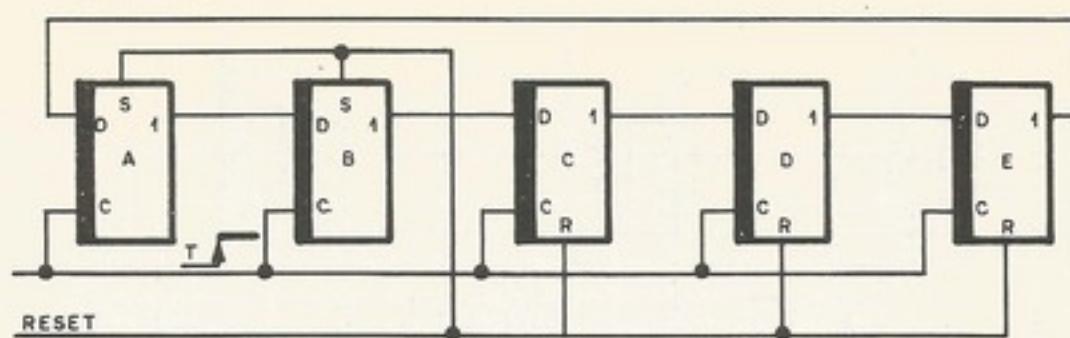


GRÁFICO DOS SINAIS

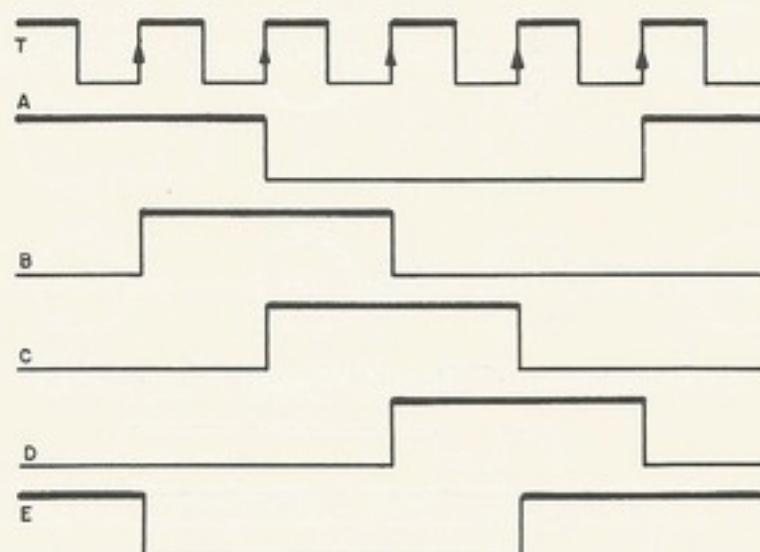


DIAGRAMA DOS ESTADOS SEQUENCIAIS

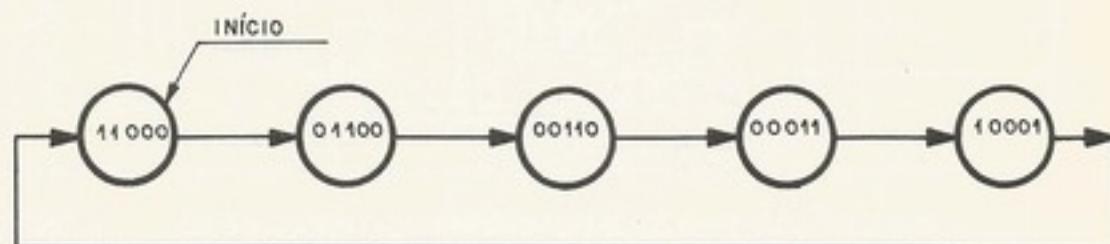


Figura 3-73. Contador em anel com superposição (overlap ring counter)

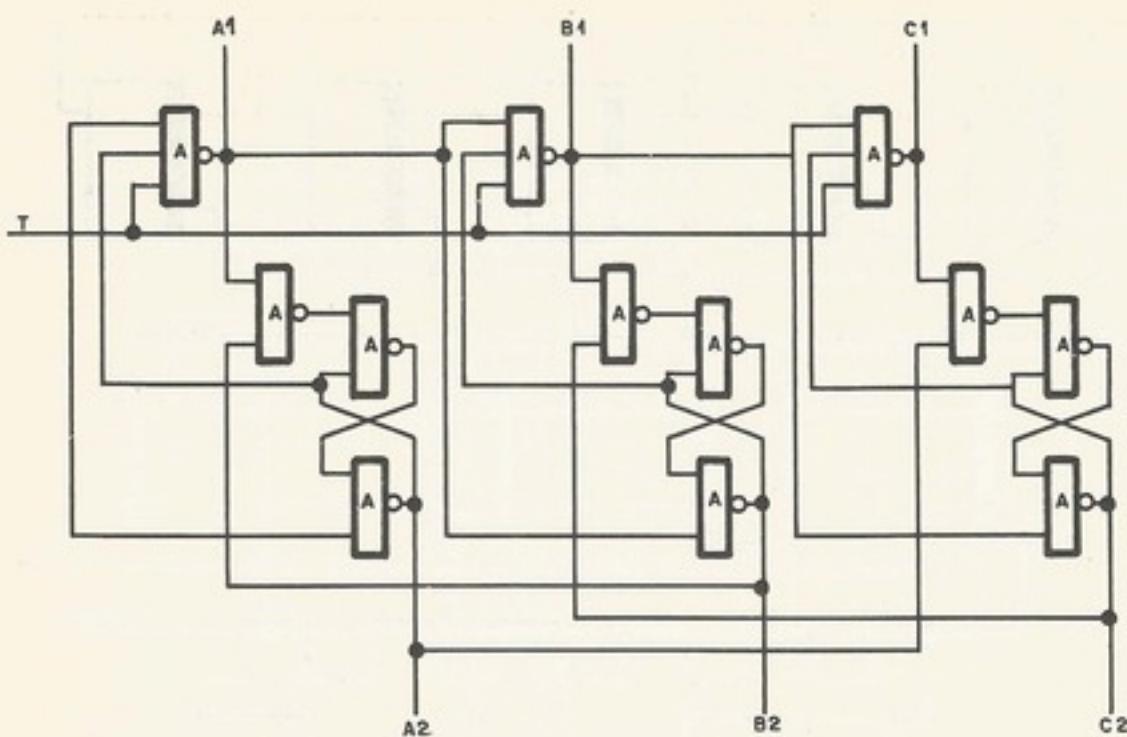
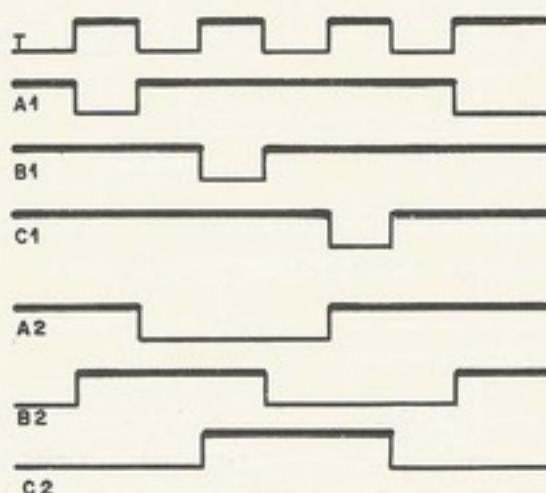


GRÁFICO DOS SINAIS

Figura 3-74. Contador em anel modificado; com portas *NAND*

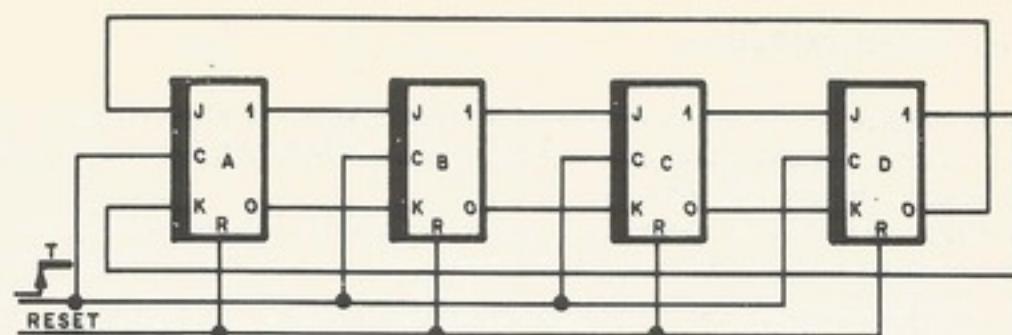


DIAGRAMA DOS ESTADOS SEQUENCIAIS

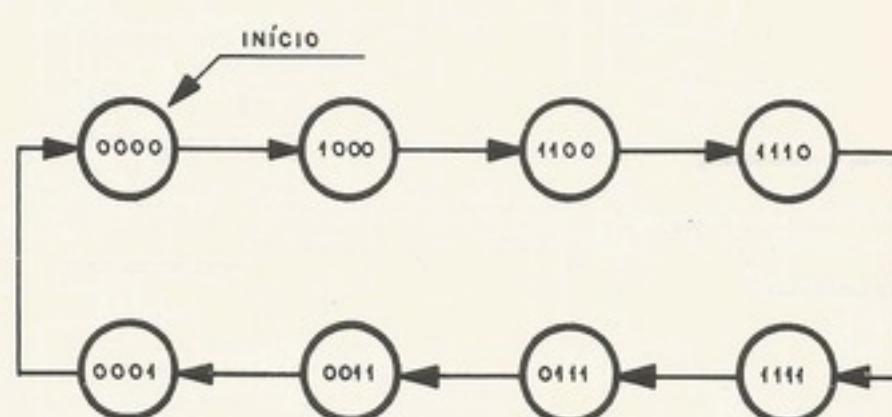
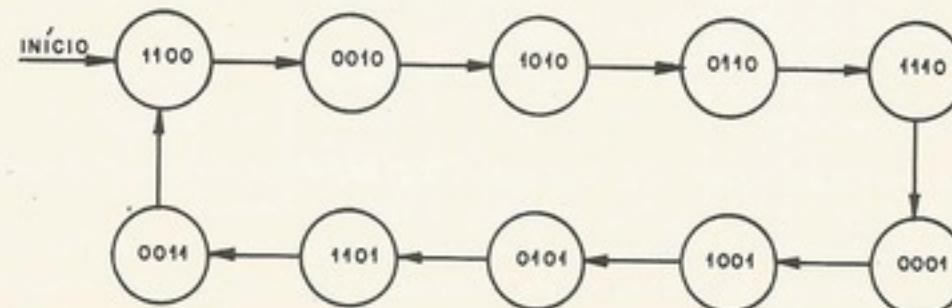
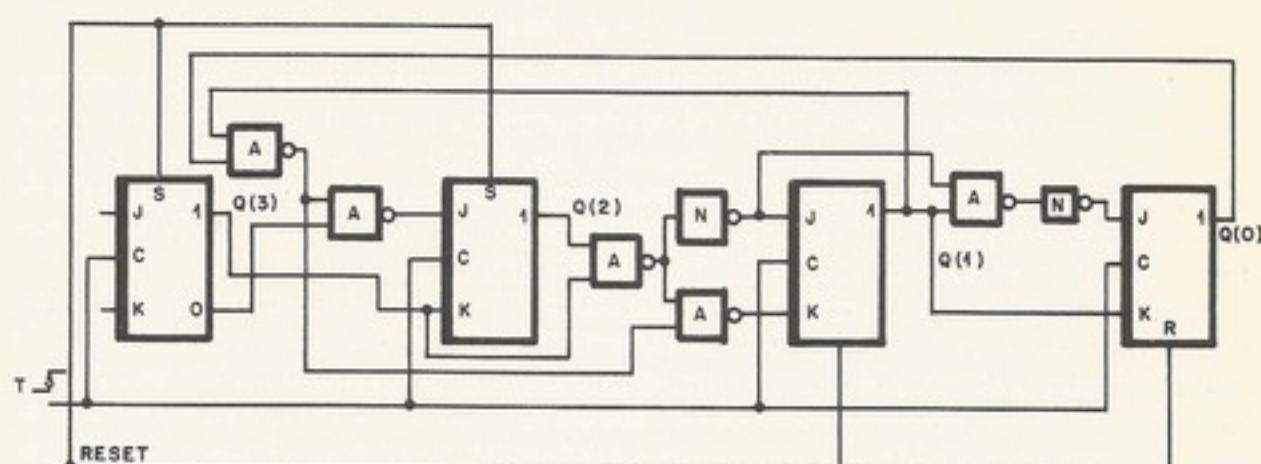
Figura 3-75. Contador de Johnson (*twisted ring counter*)

Figura 3-76. Década contadora no código BCD excesso-de-três

3.5 UNIDADES ARITMÉTICAS

Uma das partes de um sistema digital responsável pela sua *performance* é a unidade aritmética. Sistemas simples permitem unidades aritméticas simples. Sistemas sofisticados exigem unidades aritméticas sofisticadas.

A unidade aritmética é responsável pela execução de somas, subtrações, funções booleanas, comparações etc. Na Fig. 3-77, vemos um esquema simplificado de unidade aritmética, onde os operandos são os números X e Y , e o resultado é o número U . A operação é comandada pelo sinal de controle OP .

Unidades aritméticas sofisticadas podem realizar operações com números em vários códigos como, por exemplo, binário, decimal etc. O nosso estudo de unidades aritméticas será feito primeiro analisando os somadores para depois generalizar, incluindo outras operações.

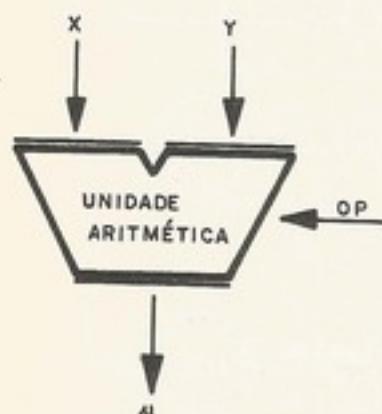


Figura 3-77. Unidade aritmética

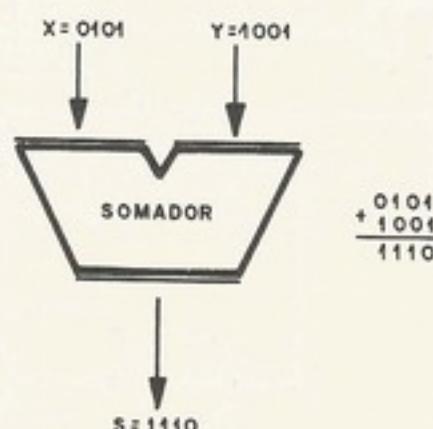
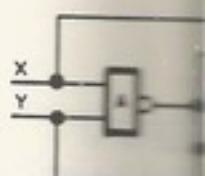


Figura 3-78. Exemplo de um somador binário



a. Somadores

Na análise de somadores, inicialmente abordaremos os somadores binários para, no final, rapidamente, estudarmos os somadores decimais.

Podemos dizer que um somador binário é um circuito onde entram duas palavras (X e Y) que são números codificados em binário e sai uma palavra (S) que é a soma binária dos dois números. Na Fig. 3-78, o leitor encontra o modelo usado para representar somadores, com um exemplo de entrada com as correspondentes saídas.

Uma primeira idéia de um somador binário seria pensarmos em um circuito combinatório que realizasse a soma: basta montarmos uma tabela da verdade para a função de múltiplas saídas (*bits* da soma) com todas as possíveis entradas e, aplicando qualquer técnica de síntese de circuitos combinatórios, obterímos como resultado o somador. Essa é a solução adotada normalmente, e o que se faz é modularizá-lo, ou seja, montar circuitos que somam cada *bit*, e associar em cascata esses somadores para se obterem somadores de vários *bits*. Os módulos que somam 1 *bit* são conhecidos como circuito "meia-soma" e circuitos "soma completa", que estudaremos a seguir.

Circuitos "meia-soma"

Os circuitos "meia-soma", *MS*, (*half-adder*) têm duas entradas (x e y) e duas saídas (s , soma módulo 2 de x e y , e v , "vai um") (Fig. 3-79).

Observe que a soma s é a função *exclusive-OR* das entradas x e y . Na Fig. 3-80, o leitor encontra várias maneiras de se implementar o circuito "meia-soma". Esse circuito não serve para executar a soma de um dos *bits* em um somador de vários *bits*, pois ele não leva em conta o "vem um", por isso é chamado de "meia-soma", ou seja, ele soma apenas dois *bits*

sem considerar o "vem um". Isso é necessário?

Circuitos "meia-soma"

É um somador de 1 bit. Na Fig. 3-81, vemos que é equivalente à Fig. 3-82 duas portas.

O modo de se implementar os circuitos "meia-soma" é mostrado na Fig. 3-83. Vemos que podemos associar as entradas x e y a portas AND e OR.

Figura 3-81.(a) Módulo de 1 bit

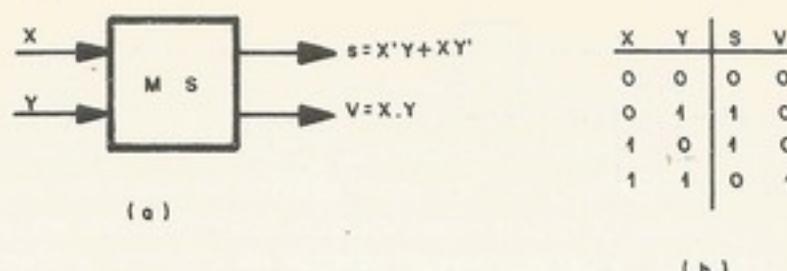


Figura 3-79. (a) Modelo do circuito "meia-soma"; (b) tabela da verdade do circuito "meia-soma"

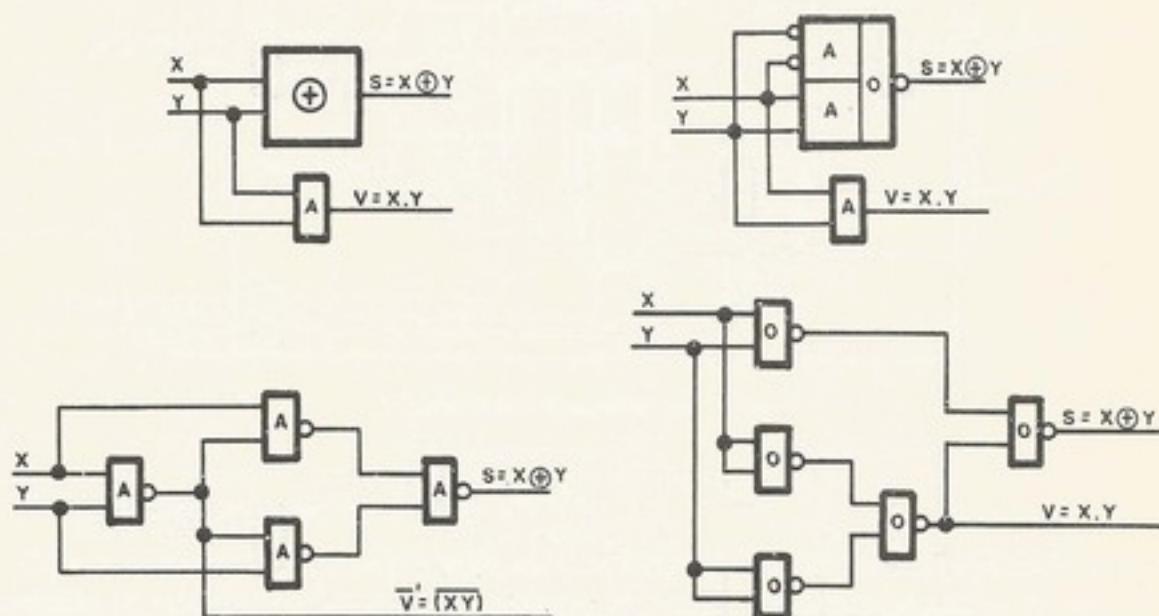


Figura 3-80. Circuitos "meia-soma"

sem considerar os bits menos significativos dos números que estão sendo somados. Por isso é necessário ampliá-lo para o seu uso em somadores em paralelo.

Circuitos "soma completa"⁽⁵⁾

É um somador de 1 bit com três entradas, x, y (bits a serem somados) e V_E ("vem um"). Na Fig. 3-81, vemos uma tabela da verdade de seu funcionamento. O leitor encontra na Fig. 3-82 duas maneiras de implementar esses somadores.

O modo de funcionamento do circuito "soma completa" (*full-adder*) mostra que podemos associá-los para obtermos somadores de vários bits. É o que veremos a seguir.

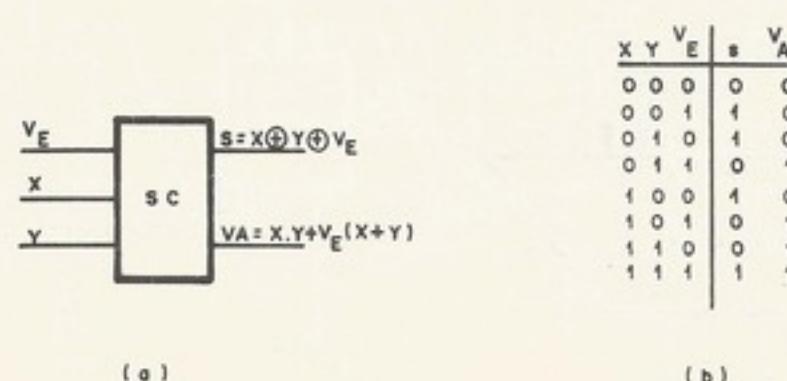


Figura 3-81.(a) Modelo do circuito "soma completa"; (b) tabela da verdade do circuito "soma completa"

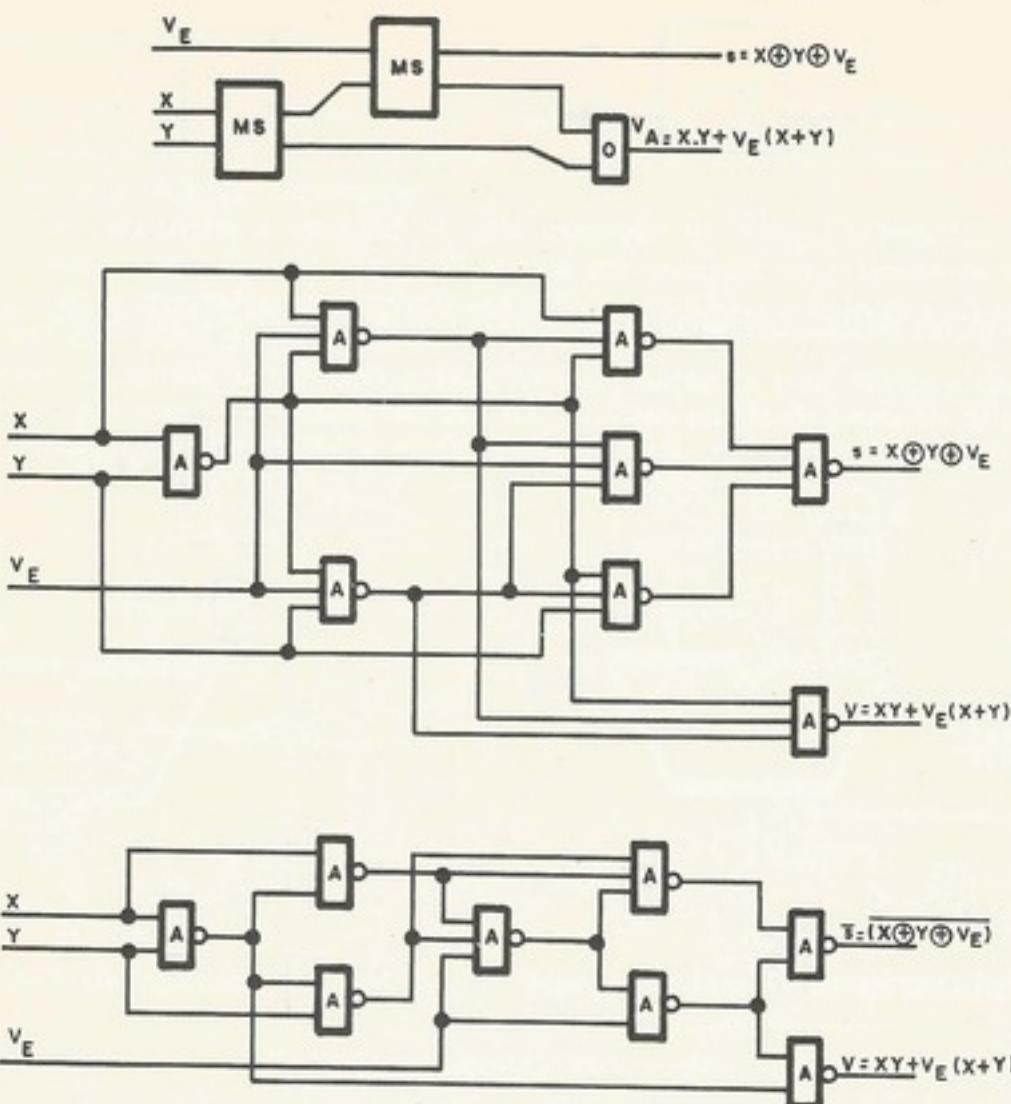


Figura 3-82. Circuitos "soma completa"

Somadores de palavras⁽²⁾

Chamamos de somadores de palavras os circuitos que somam números de vários bits. Na Fig. 3-83, vemos um modelo de um somador de palavras de 4 bits com nove entradas: 4 bits da palavra X, 4 bits da palavra Y e a entrada V_E, que serve para forçar um ("vem um"). O circuito tem cinco saídas: 4 bits da soma e a saída V_A, que pode ser interpretada como "estouro" ou transbordamento (*overflow*).

Uma das idéias bastante analisadas há algum tempo, que primava pela economia, deixando, porém, muito a desejar, dado o atraso, eram os *somadores em série*, onde as par-

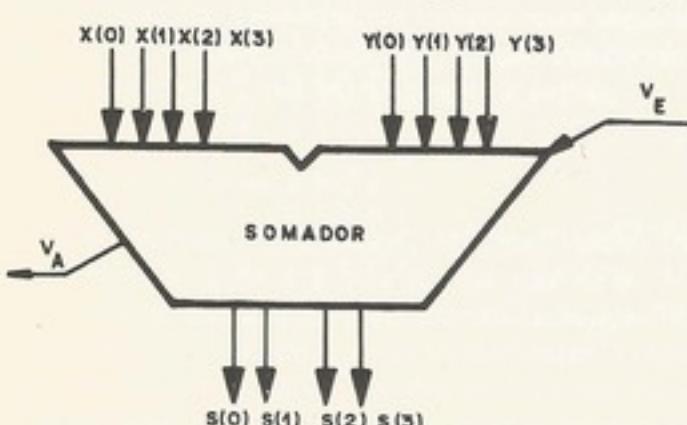


Figura 3-83. Somador de 4 bits

celas X e Y para armazenada em de clock não no flip-flop completa depreverá ser, atuais utilizan

Somador

Com a s o aumento d em série para no somador

Um tipo mas que p adder). É se vê na Fig. da saída "v for implem ximada

celas X e Y iam entrando, bit a bit, num circuito "soma completa" provido de um flip-flop para armazenar o "vai-um". Enquanto isso, a soma ia saindo do outro lado e sendo armazenada em um registrador. Na Fig. 3-84, vemos um somador em série, no qual os pulsos de clock vão deslocando as entradas X e Y e a saída S ; em cada ciclo, o "vai-um" é armazenado no flip-flop que será colocado na entrada "vem um" do ciclo seguinte. A soma estará completa depois de tantos pulsos de clock quantos forem os bits da palavra, e o período do clock deverá ser, no mínimo, o atraso do circuito "soma completa". Inúmeras das calculadoras atuais utilizam esse esquema.

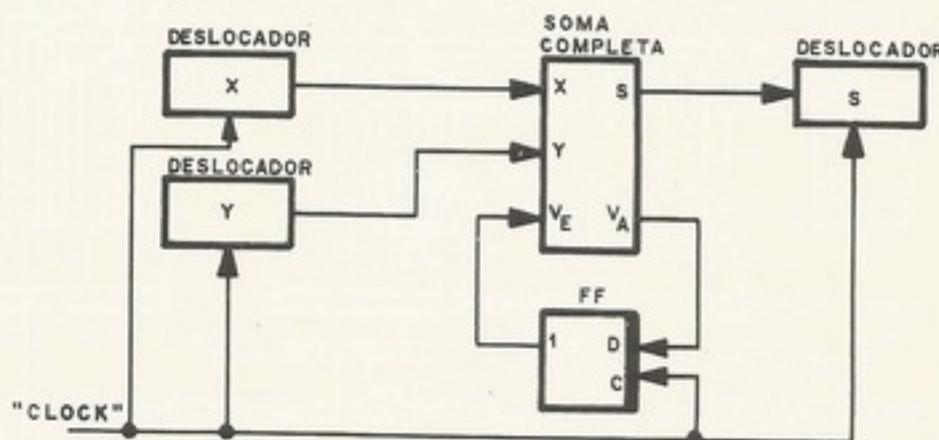


Figura 3-84. Somador em série

Somador com propagação do "vai um"⁽⁷⁾

Com a fabricação de circuitos mais baratos e com os sistemas de computação exigindo o aumento de velocidades mesmo com aumento de custo, abandonou-se a idéia de somadores em série para se passar aos somadores em paralelo, onde todos os bits das parcelas entram no somador e saem todos os bits da soma.

Um tipo de somador em paralelo bastante usado pela sua simplicidade e economia, mas que peca pelo atraso é o chamado somador com propagação do "vai um" (*ripple carry adder*). É obtido por simples associação em cascata de circuitos "soma completa", como se vê na Fig. 3-85. O atraso desse somador é, aproximadamente, igual à soma dos atrasos da saída "vai um" dos circuitos "soma completa". Por exemplo, se o somador da Fig. 3-85 for implementado com o circuito "soma completa" da Fig. 3-82, o atraso da soma será aproximadamente de doze portas.

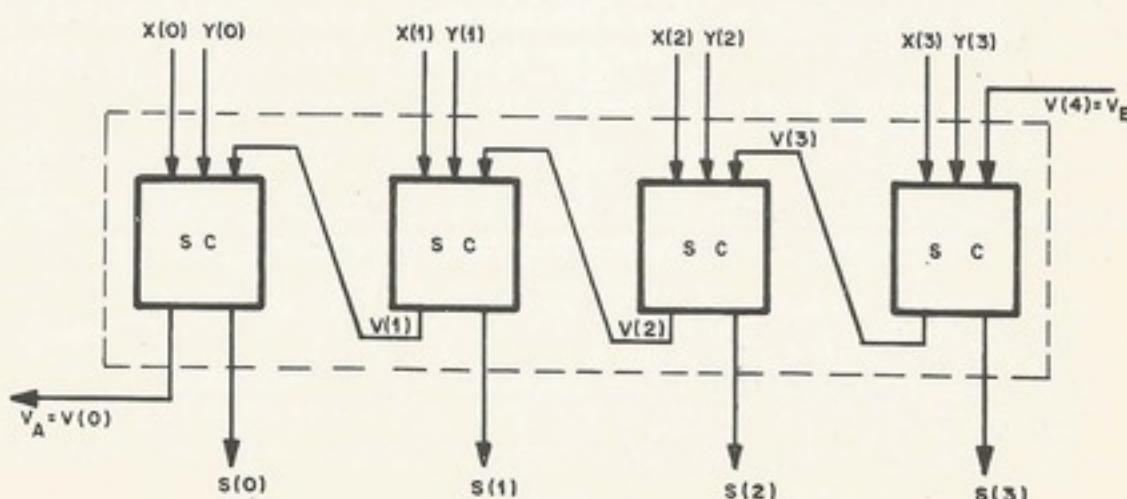


Figura 3-85. Somador de 4 bits com propagação do "vai um"

Somador com "vai um antecipado"⁽⁷⁾

Como a principal causa do atraso do somador com propagação do "vai um" era o fato de um dado bit precisar esperar que o "vai um" se propagasse desde o bit menos significativo até ele, construiu-se um somador no qual, em cada bit, existe a previsão do "vai um" sem a necessidade de se esperar que ele se propague, isto é, o somador com "vai um antecipado" (*carry lookahead adder*). Aqui, abandonamos a idéia simples da modulação para ganhar em velocidade. O leitor notará que teremos uma semimodulação, pois o circuito, que analisaremos a seguir, tem um esquema modular com módulos que aumentam conforme sua posição vai ficando mais significativa.

Olhando a Fig. 3-85, podemos escrever as expressões da soma e do "vai um" no bit i como sendo

$$\begin{aligned} S(i) &= X(i) \oplus Y(i) \oplus V(i+1), \\ \text{onde } V(i) &= G(i) + T(i) \cdot V(i+1), \\ G(i) &= X(i) \cdot Y(i) \quad \text{e} \quad T(i) = X(i) + Y(i). \end{aligned}$$

Demos um nome especial para as expressões

$$G(i) = X(i) \cdot Y(i) \quad \text{e} \quad T(i) = X(i) + Y(i)$$

porque elas podem ser interpretadas das seguintes maneiras: chama-se $G(i)$ de geração do "vai um", pois, quando $G(i) = 1$, com certeza há "vai um" saindo daquele estágio; $T(i)$ é chamado de transporte do "vai um" (às vezes também chamado de propagação do "vai um"), pois, quando $T(i) = 1$, há "vai um" saindo desse estágio se existir "vem um" entrando [alguns autores definem $T(i)$ como sendo $X(i) \oplus Y(i)$]. Vemos, portanto, a consistência da expressão

$$V(i) = G(i) + T(i) \cdot V(i+1),$$

que pode ser interpretada como: haverá "vai um" saindo do estágio i (para o estágio $i-1$) se esse "vai um" for gerado em i ou, então, se existir "vem um" e o transporte for igual a um, com

$$G(i) = X(i) \cdot Y(i) \quad \text{e} \quad T(i) = X(i) + Y(i).$$

Se tomarmos o exemplo da Fig. 3-83, poderemos escrever

$$\begin{aligned} S(3) &= X(3) \oplus Y(3) \oplus V_E, \\ V(3) &= G(3) + T(3) \cdot V_E, \quad \text{com} \quad G(3) = X(3) \cdot Y(3) \quad \text{e} \quad T(3) = X(3) + Y(3); \\ S(2) &= X(2) \oplus Y(2) \oplus V(3), \\ V(2) &= G(2) + T(2) \cdot V(3), \quad \text{com} \quad G(2) = X(2) \cdot Y(2) \quad \text{e} \quad T(2) = X(2) + Y(2); \\ S(1) &= X(1) \oplus Y(1) \oplus V(2), \\ V(1) &= G(1) + T(1) \cdot V(2), \quad \text{com} \quad G(1) = X(1) \cdot Y(1) \quad \text{e} \quad T(1) = X(1) + Y(1); \\ S(0) &= X(0) \oplus Y(0) \oplus V(1), \\ V(0) &= G(0) + T(0) \cdot V(1), \quad \text{com} \quad G(0) = X(0) \cdot Y(0) \quad \text{e} \quad T(0) = X(0) + Y(0). \end{aligned}$$

Dessas expressões de soma, eliminando as variáveis $v(i)$, obtemos as seguintes expressões para a soma:

$$\begin{aligned} S(3) &= X(3) \oplus Y(3) \oplus V_E, \\ S(2) &= X(2) \oplus Y(2) \oplus [G(3) + T(3) \cdot V_E], \\ S(1) &= X(1) \oplus Y(1) \oplus [G(2) + T(2) \cdot G(3) + T(2) \cdot T(3) \cdot V_E], \\ S(0) &= X(0) \oplus Y(0) \oplus [G(1) + T(1)G(2) + T(1) \cdot T(2) \cdot G(3) + T(1) \cdot T(2) \cdot T(3) \cdot V_E], \\ V_A &= G(0) + T(0) \cdot G(1) + T(0) \cdot T(1) \cdot G(2) + T(0) \cdot T(1) \cdot T(2) \cdot G(3) + T(0) \cdot T(1) \cdot T(2) \cdot T(3) \cdot V_E. \end{aligned}$$

Essas equações permitem a construção do circuito visto na Fig. 3-86, chamado somador com "vai um antecipado" (ou somador com previsão do "vai um"), que oferece a vantagem

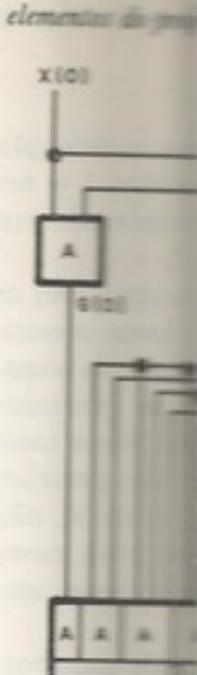


Figura 3-86

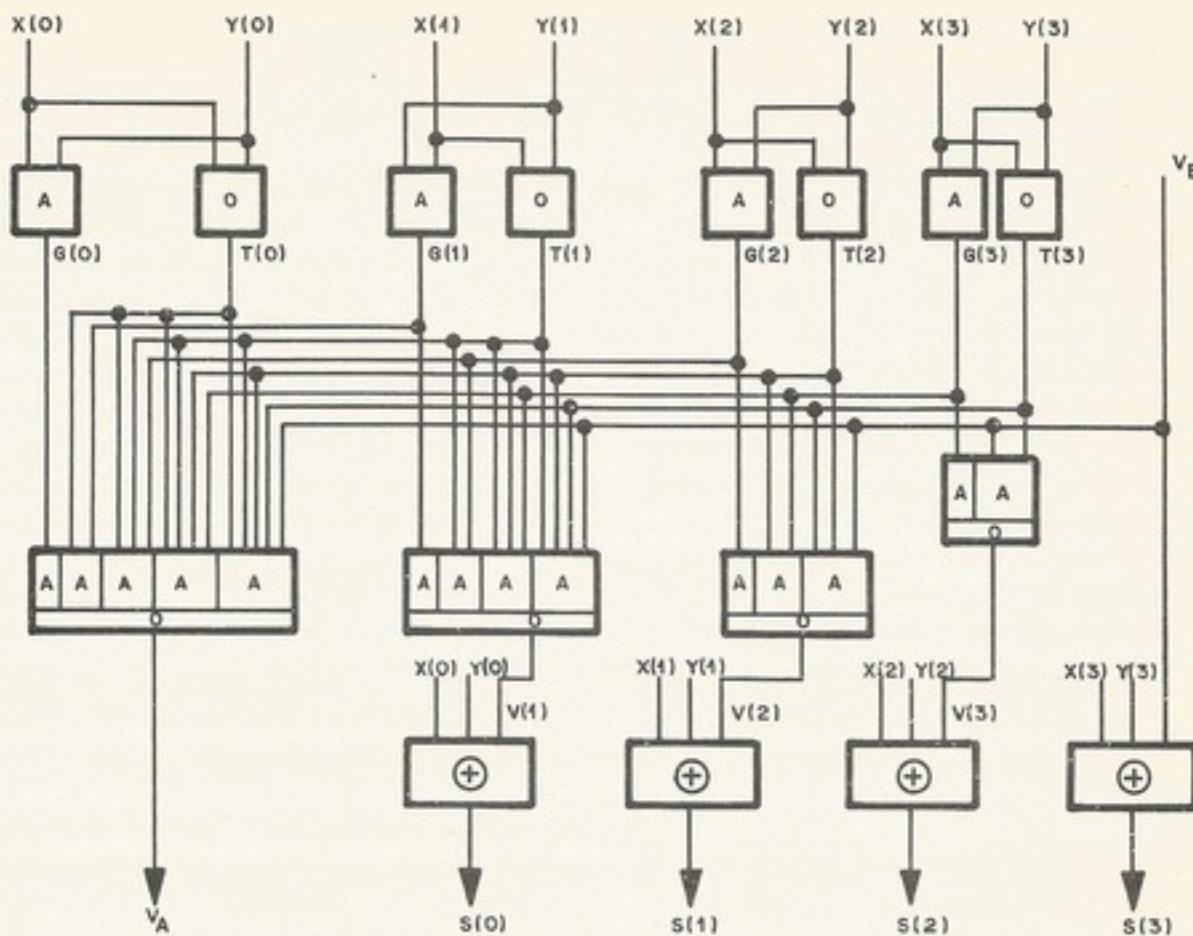


Figura 3-86. Somador de 4 bits com “vai um antecipado”

de ter um atraso bem menor que o anterior (propagação do “vai um”). O leitor deve ter notado que um inconveniente desse somador é o *fan-in* de suas portas. Quanto maior for sua capacidade em *bits*, tanto maior será o *fan-in* exigido. Por isso, quando se quer implementar somadores de muitos *bits*, associam-se em cascata somadores com “vai um” antecipado de 4 *bits* (por exemplo), formando somadores que são um misto de propagação do “vai um” com previsão do “vai um”. Esse conceito de somadores tem, normalmente, um atraso aceitável com um preço razoável (Fig. 3-87).

Na Fig. 3-87, vemos que, além dos 12 bits da soma, existe ainda a saída V_A , que, à primeira vista, interpretaríamos como transbordamento (*overflow*), isto é, a soma excedeu a capacidade dos 12 bits da palavra; ou, ainda, poderia ser interpretado como um décimo terceiro (mais significativo) bit da soma. Como veremos a seguir, o significado do sinal V_A depende do código que usamos para os números negativos.

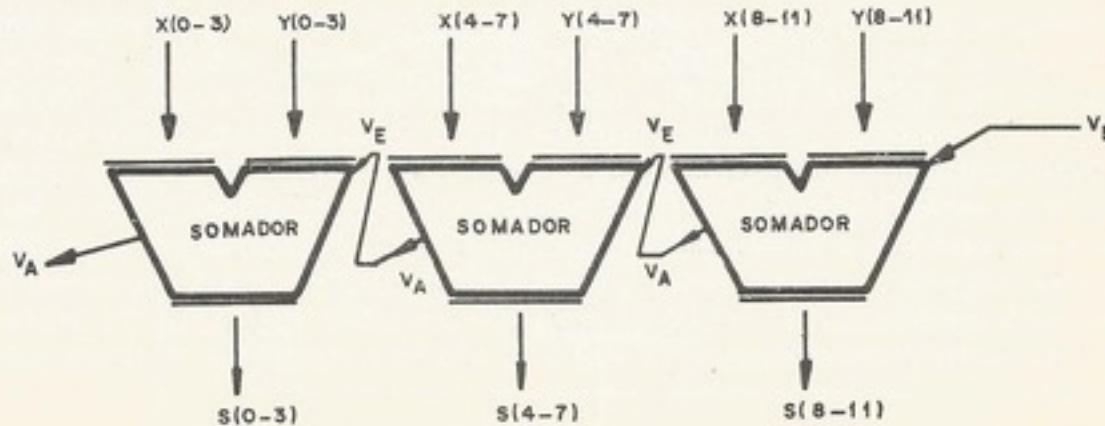


Figura 3-87. Somador de 12 bits obtidos com somadores “vai um antecipado” de 4 bits.

Somador binário em complemento de dois

Como vimos no Cap. 2, complemento de dois (*two's complement*) é uma codificação de números relativos, onde o bit mais significativo representa o sinal (0 = positivo, 1 = negativo) e os outros bits representam a amplitude do número (se positivo, é o próprio número binário e, se negativo, a amplitude está complementada e adicionada de 1).

A vantagem desse código reside no fato de que não precisamos nos preocupar com os sinais das parcelas. Usando os somadores estudados, obteremos a soma algébrica correta.

Precisamos, contudo, nos preocupar com um detalhe, o estouro da capacidade da soma. O sinal V_A ("vai um") dos somadores apresentados é útil para a situação em que queremos somar números cuja quantidade de bits é maior que a capacidade do somador. Nesse caso, somamos em duas etapas, usando o sinal V_A , obtido na soma de uma parte das parcelas, para ser forçado na entrada V_E na soma das partes seguintes. Esse sinal V_A , contudo, não implica, ao somarmos as últimas partes das parcelas, estouro da capacidade, pois, nessa última parte (mais significativa), o bit mais significativo será o sinal do número e, nesse caso, V_A perderá seu sentido.

No caso que estamos analisando (complemento de dois), a informação de transbordamento (estouro da capacidade da soma) é obtida, como já vimos no Cap. 2, pela expressão

$$T = [V(0) \oplus V(1)].$$

Com isso, o somador da Fig. 3-88 pode ser usado em sistemas que usam o código binário complemento de dois para os números.

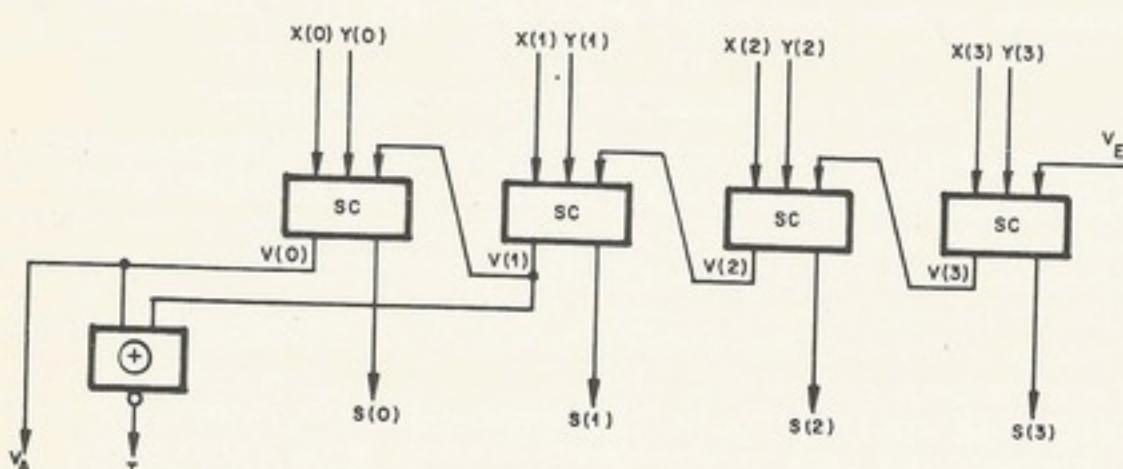


Figura 3-88. Somador de 4 bits com sinal de transbordamento para códigos complemento de dois

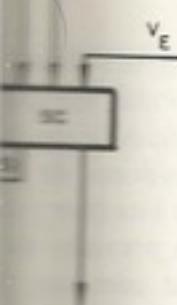
Conforme já comentamos, muitas vezes utilizamos o somador para somar parcelas maiores que a "largura" das entradas (operações em precisão múltipla). Vejamos, por exemplo, a Fig. 3-89. Se quiséssemos somar dois números de 8 bits, $X = 10110100$ e $Y = 01001101$, agiríamos da seguinte maneira: numa primeira etapa, colocaríamos os 4 bits menos significativos das parcelas X e Y no somador e, obteríamos os 4 bits menos significativos da soma ($0100 + 1101 = 0001$), teríamos em V_A a informação de que o "vai um" para a soma seguinte e o sinal T não teriam significado; na segunda etapa, colocaríamos 1 em V_E (por causa do V_A anterior) e os 4 bits mais significativos de X e Y , obteríamos então, os 4 bits mais significativos da soma ($1011 + 0100 + 1 = 0000$), teríamos em T a informação de que não houve transbordamento ($T = 0$) e V_A não teria significado. Portanto teríamos o resultado

$$10110100 + 01001101 = 00000001, \text{ isto é, } (-76) + (+77) = (+1).$$

é uma codificação
= positivo, 1 = ne-
- próprio número
de 1).
preocupar com os
algebraica correta.
idade da soma.
que queremos
mudor. Nesse caso,
nte das parcelas,
, contudo, não
nde, pois, nessa
e, nesse caso,

áculo de transbor-
2, pela expressão

código binário



plementação de dois

sumar parcelas
l. Vejamos, por
X = 10110100 e
curciamos os 4
bit mais significativos que o "vai um"
colocaríamos 1
curciamos então,
o T a informação
teríamos

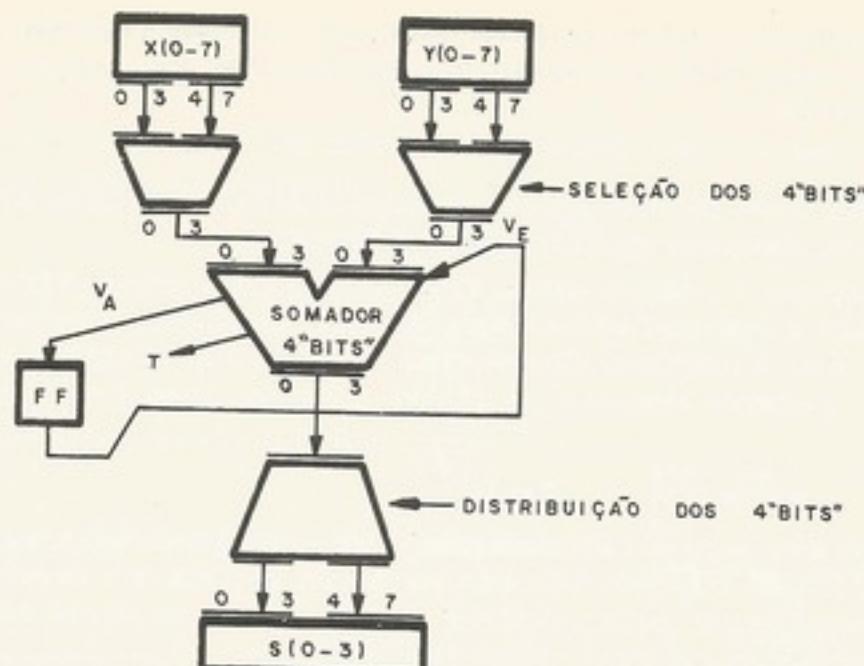


Figura 3-89. Esquema para soma de números com maior quantidade de bits que a capacidade do somador

Somador binário em complemento de um

No Cap. 2, vimos que o código binário complemento de um (*one's complement*) tem o sinal representado no bit mais significativo (0 = positivo e 1 = negativo) e os outros bits representam a amplitude; se positivo, a amplitude é o próprio número em binário e, se negativo, a amplitude está complementada.

A vantagem desse código reside no fato de que podemos, por exemplo, fazer subtrações apenas complementando, bit a bit, o subtraendo e somando com o minuendo, enquanto que, no complemento de dois, precisávamos complementar e somar um para mudar o sinal do subtraendo. A grande desvantagem desse código é que a soma, numa primeira etapa, é feita exatamente como a do código binário complemento de dois e, terminada essa etapa, somamos V_A na posição menos significativa do resultado (Fig. 3-90). Isso implica na necessidade de uma lógica especial ou de cuidados especiais na execução da soma. Além disso, o tempo exigido para uma soma, no pior caso, é o dobro do exigido pelo complemento de dois.

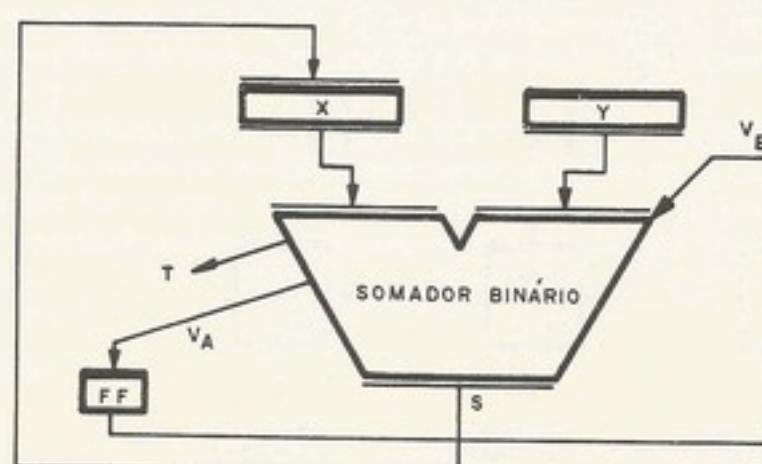


Figura 3-90. Somador em complemento de um

Com a codificação binária complemento de um, existe um problema com o código do zero, pois existem duas maneiras possíveis de representar esse número. Por exemplo, com 4 bits, teremos

$$+0 = 0000, \quad -0 = 1111.$$

O problema existe em determinadas situações onde somamos algum número com -0 . Se, por exemplo, quisermos somar -0 com -7 , teremos

$$\begin{array}{r} 1111 & (-0) \\ + 1000 & (-7) \\ \hline 0111 \\ \text{returno } 1 \\ \hline 1000 & (-7) \\ T = 1, \end{array}$$

que fornece um sinal falso de transbordamento. Isso é um grande inconveniente desse código. Porém podemos resolvê-lo incluindo, no sistema circuitos que transformam -0 em $+0$, sempre que o primeiro ocorrer. Com essa solução, teremos um sistema que usa o código binário complemento de um com o código 1111 ilegal.

Muitos autores sugerem que se realmente diretamente a saída V_A na entrada V_E , sem o uso do flip-flop e sem a necessidade da lógica coordenando os dois ciclos. Apesar de se demonstrar a possibilidade teórica de esse somador oscilar em certas circunstâncias, costuma-se aceitar que as diferenças inerentes dos atrasos dos blocos lógicos fará com que essa oscilação se amortize atingindo o equilíbrio estável rapidamente. Mesmo assim existe a possibilidade de esse circuito ser muito lento.

Somadores decimais^(2, 7)

A técnica usada em somadores BCD (decimal codificado em binário) é somarmos cada dígito decimal em somadores binários hexadecimal (4 bits) detetando o "vai um" decimal (V_{Ad}) e, quando este for "1", faremos uma correção de seis (six-correct).

O "vai um" decimal (V_{Ad}) existe sempre que ocorre o "vai um" hexadecimal (V_A), ou quando o dígito da soma é algum código decimal inválido (maior que nove). Na Fig. 3-91,

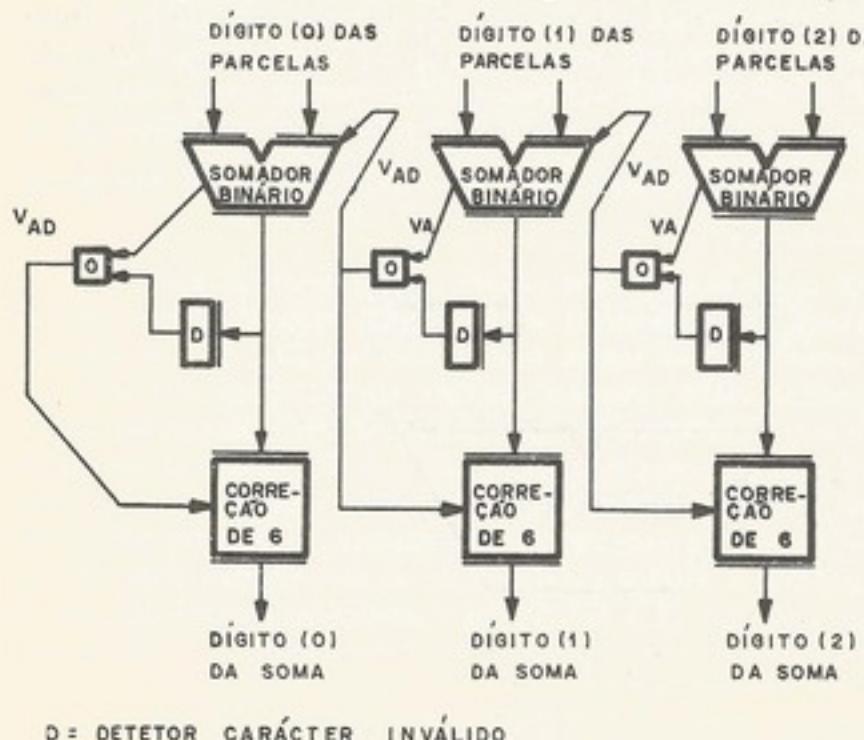


Figura 3-91. Somador decimal de 3 dígitos

vemos um esquema. A correção de... (veja o Cap. 2).

Os códigos de um dígito B

Na Fig. 3-92(a) ser feita com um anterior e, mas isto é, quando

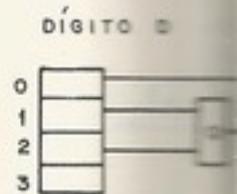


Figura 3-92(a) de código inválido

Evidentemente de seis mais baixos

O sistema com "vai um" a decisão do "vai um" fazendo-se uma fazendo-se a soma. Terminada a soma rígidos de menor filosofia, podemos

Sistemas dividida por uma soma somamos três +3 e -3), e como 1101, 1110 e 1111

A vantagem do primeiro deles é o número. O segundo, portanto, correções prévia

b. Subtração

A maior parte fazem a subtração somando com

o código
Por exemplo,
número com -0.

vemos um esquema de somador *BCD* usando somadores binários de 4 bits (hexadecimal). A correção de seis consiste em somar seis ao resultado quando existir "vai um" decimal (veja o Cap. 2).

Os códigos inválidos são vistos na Fig. 3-92(a). É fácil notar que um código inválido de um dígito *D* é dado pela expressão

$$CI = D(0) \cdot [D(1) + D(2)].$$

Na Fig. 3-92(b), vemos o circuito detetor de código inválido. A correção de seis pode ser feita com um outro somador hexadecimal onde uma das parcelas é a saída do somador anterior e, nas posições quatro e dois da outra entrada colocamos o sinal V_{Ad} (Fig. 3-93) isto é, quando $V_{Ad} = 1$, somamos seis e, quando $V_{Ad} = 0$, nada somamos.

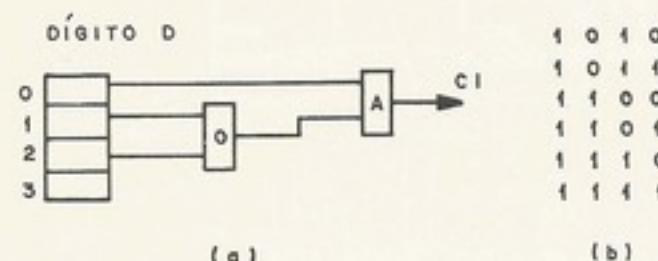


Figura 3-92.(a) Códigos *BCD* inválidos; (b) detecção de código inválido no dígito *D*

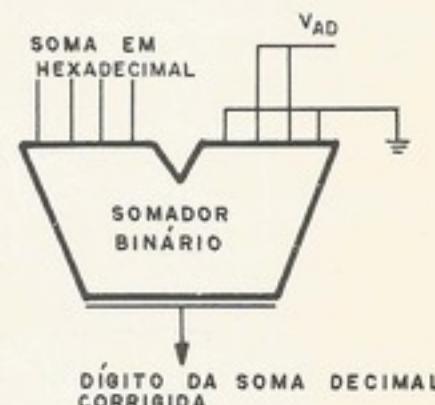


Figura 3-93. Correção de seis

Evidentemente é possível projetar-se um circuito combinatório que faça a correção de seis mais barata e, talvez, mais rápida que o apresentado na Fig. 3-93.

O sistema apresentado na Fig. 3-91 não permite a implementação de um somador com "vai um" antecipado em toda a extensão do somador, já que, entre dois dígitos, a decisão do "vai um" decimal depende da soma (carácter inválido). O problema é resolvido fazendo-se uma prévia correção de seis em todos os dígitos de uma das parcelas e, a seguir, fazendo-se a soma. Nessa situação, o "vai um" hexadecimal é igual ao "vai um" decimal. Terminada a soma, todos os dígitos da soma que não tenham criado o "vai um" serão corrigidos de menos seis (somando 1010) e o resultado obtido será a soma correta. Com essa filosofia, podemos usar a técnica "vai um" antecipado interdígitos.

Sistemas digitais que usam o código *BCD* excesso de três podem ter a sua soma realizada por uma estrutura análoga à da Fig. 3-91, com a diferença na correção: quando $V_{Ad} = 1$, somamos três ao resultado e, quando $V_{Ad} = 0$, subtraímos três do resultado (correções de +3 e -3), e como o leitor já deve saber, os códigos inválidos são diferentes (0000, 0001, 0010, 1101, 1110 e 1111).

A vantagem do código *BCD* excesso de três reside, principalmente, em dois fatos: o primeiro deles é na mudança de sinal, pois basta complementar bit a bit os dígitos do número. O segundo fato é que o "vai um" hexadecimal é o mesmo que o "vai um" decimal, sendo, portanto, possível realizar a soma com somadores do tipo "vai um" antecipado sem correções prévias.

b. Substratores

A maior parte dos sistemas digitais existentes, principalmente os de pequeno porte, fazem a subtração no próprio somador, antes mudando o sinal do subtraendo e, a seguir, somando com o minuendo.

Porém é possível construir-se unidades aritméticas que fazem a subtração diretamente, além da soma. Existem, inclusive, aquelas que têm o subtrator; a soma é feita complementando-se uma parcela e subtraindo da outra.

Circuitos "subtração completa"⁽⁵⁾

Analogamente ao que fizemos com os somadores, podemos ter circuitos unitários que fazem a subtração de 1 bit, que, unidos em paralelo, realizam a subtração de vários bits.

Na Fig. 3-94, vemos esquemas de circuitos subtração completa, onde é feita a diferença $X - Y$, o resultado é R e o empréstimo (*borrow*) de entrada E_E e o empréstimo de saída E_S .

O circuito da Fig. 3-94(d) tem a desvantagem de apresentar um atraso relativamente grande. Se quisermos um circuito mais rápido, podemos implementar em dois níveis os sinais de saída R e E_S :

$$R = x'y'E_E + xy'E'_E + E'_E + xyE_E,$$

$$E_S = x'y + x'E_E + yE_E.$$

Este esquema apresenta a desvantagem quanto ao preço do circuito.

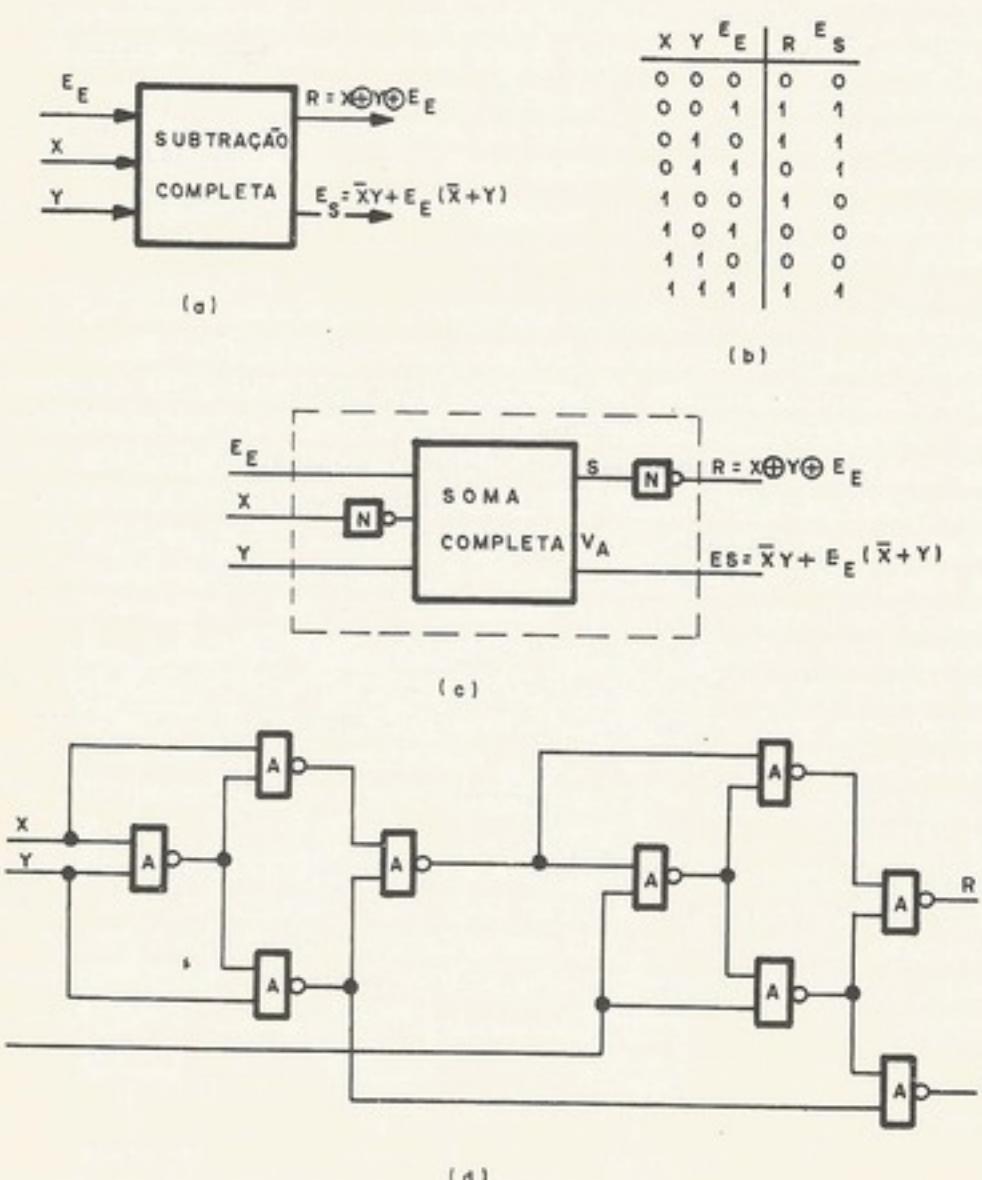


Figura 3-94. Circuito "subtração completa". (a) Modelo; (b) tabela da verdade; (c) a partir do circuito "soma completa"; (d) com *NAND*

Subtração completa
Aqui também
Fig. 3-84 e com
Subtração completa
têm as mesmas
Existem também
"vai um antecedido
fan-ins. Então, na
subtrator que se usa
empréstimo armado

c. Complemento
É comum, no
que complemento
Quando F é zero,
as entradas com
complemento

Figura 3-96. Modelador

No circuito de

e, se $F = 1$,

Portanto, podemos

Dessa expressão, pode-se obter com um pouco de

Subtratores de palavras

Aqui também podemos pensar em subtratores em série com o mesmo esquema da Fig. 3-84 e com as mesmas desvantagens do somador em série.

Subtratores em paralelo existem com *propagação do empréstimo* (Fig. 3-95) que também têm as mesmas características do somador com propagação do "vai um".

Existem também subtratores com *empréstimo antecipado*, análogos ao somador com "vai um antecipado", porém a alto custo e com o risco de se necessitar de portas com grandes *fan-ins*. Então, também nesse caso, pode-se adotar a solução mostrada na Fig. 3-87, um subtrator que se situa num ponto entre os subtratores com propagação do empréstimo e empréstimo antecipado.

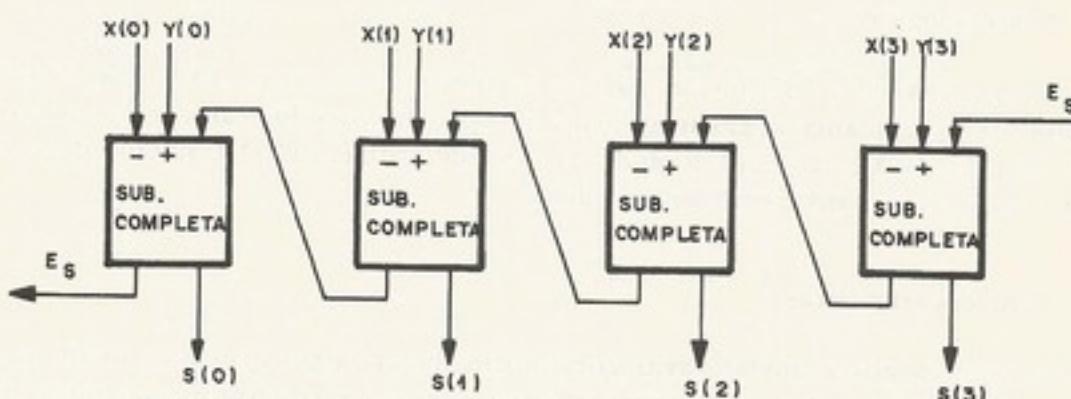
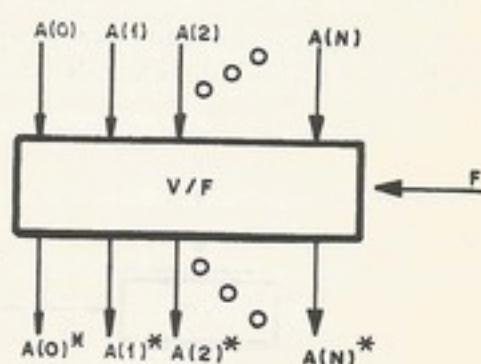


Figura 3-95. Subtrator de 4 bits com propagação do empréstimo

c. Complementação⁽⁴⁾.

É comum, as unidades aritméticas possuírem, em uma de suas entradas, um circuito que complementa todos os *bits* de uma palavra sob controle de um sinal chamado *F* (*falso*). Quando *F* é zero, não existe a complementação e, quando é 1, a saída do circuito tem todas as entradas complementadas (Fig. 3-96).

Figura 3-96. Modelo do circuito complementador



No circuito da Fig. 3-96, vemos que, se $F = 0$,

e, se $F = 1$,

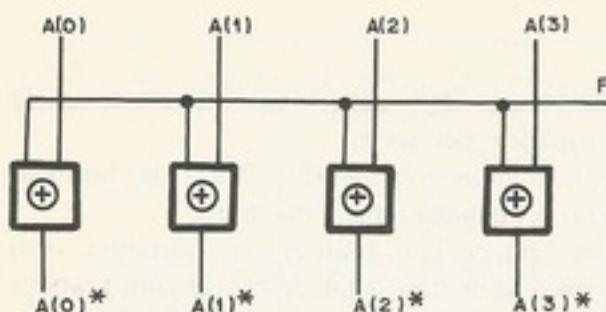
$$A(i)^* = A(i)$$

$$A(i)^* = A(i)'.$$

Portanto, podemos dizer que

$$A(i)^* = F \oplus A(i).$$

Dessa expressão, podemos concluir o circuito da Fig. 3-97, que é um complemento visto com um pouco mais de detalhes.

Figura 3-97. Circuito complementador com portas *XOR*

d. Funções lógicas

É comum chamar-se de funções lógicas as funções booleanas *AND*, *OR*, *NAND* etc. Normalmente as unidades aritméticas realizam essas funções lógicas com as palavras da entrada *bit a bit*, isto é, o *bit u(i)* de saída da unidade aritmética é o resultado dessa função booleana com os *bits x(i)* e *y(i)* das entradas.

e. Unidades aritméticas

Como já dissemos, a unidade aritmética, normalmente, é construída de modo a realizar várias operações. Dependendo da aplicação do sistema, é escolhido um conjunto de operações que a unidade aritmética precisa executar. Então, com o comando de uma terceira entrada de controle, a unidade aritmética assume a cada instante uma configuração desejada (subtrator, somador etc.).

Na Fig. 3-98, vemos um exemplo de uma unidade aritmética com potencial para somar ou subtrair. Quando o sinal *SUB* = "0", ela se comporta como um somador e, quando *SUB* = "1", ela será um subtrator (por inversão de sinal do subtraendo e somando com o minuendo). O código usado é o binário complemento de dois⁽⁴⁾.

A inclusão de operações lógicas é habitualmente feita com um cálculo em paralelo dessa função com uma porta na saída que seleciona o resultado conforme a operação. Na Fig. 3-99, um diagrama simplificado ilustra essa idéia. Evidentemente, a inclusão dessa porta aumenta o atraso da unidade aritmética.

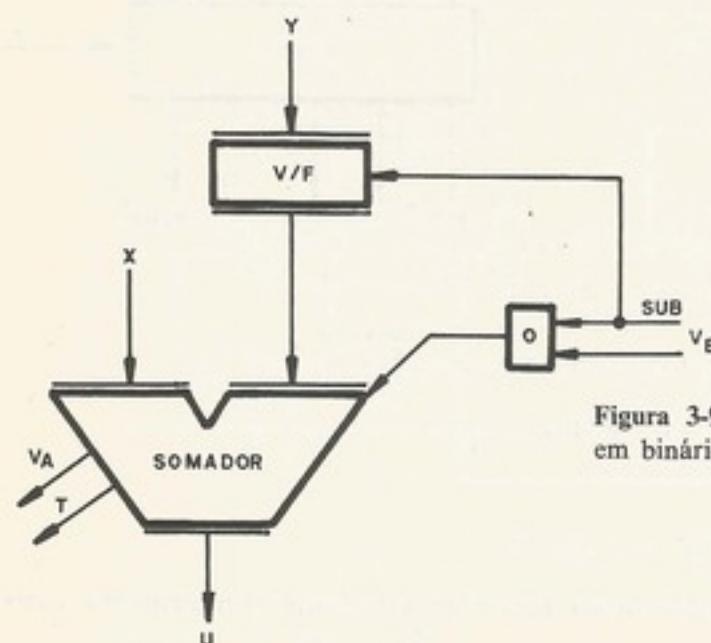


Figura 3-98. Unidade aritmética que soma e subtrai em binário complemento de dois

Figura 3-99. Unidade

realiza as operações

As idéias basicamente, fixada uma locidade e preço. Será alto, principalmente que multiplicar específicos, ou de operações sucessivas. Essas operações ou firmware (memória).

f. Exemplos de aplicações

Vamos sugerir a elaboração de 8 bits, com a unidade aritmética deverá ter 4 bits do tipo "var.

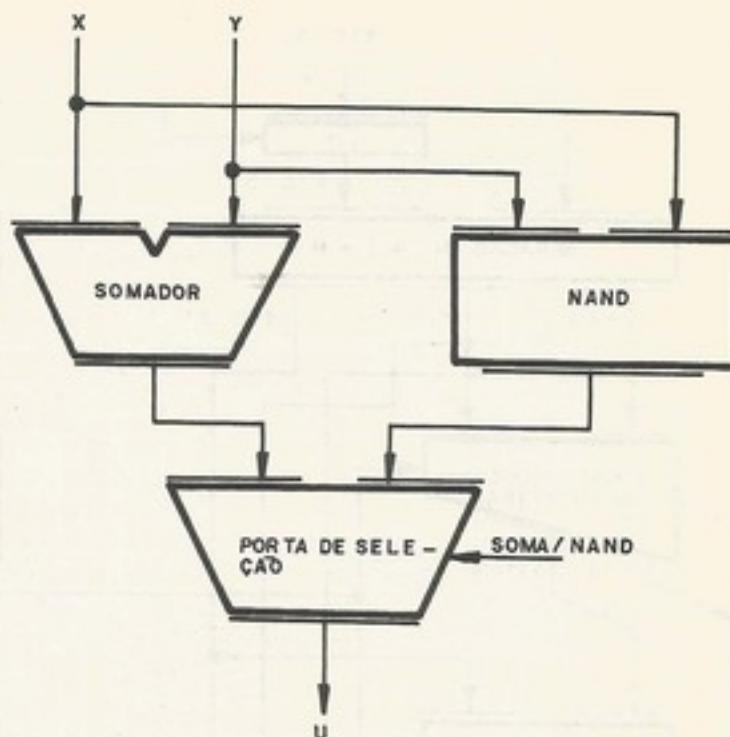
Além da soma (*X - Y*), *AND*, *OR* e *XOR*. Fig. 3-98, e as funções de um diagrama em bloco.



implementador com

OR, NAND etc.
m as palavras da
não dessa funçãomodo a realizar
conjunto de ope-
de uma terceira
informação desejadaespecial para somar
maior e, quando
comando com oem paralelo dessa
operação. Na Fig.
não dessa porta

Figura 3-99. Unidade aritmética que realiza as operações de soma e *NAND*



As idéias básicas do projeto de unidades aritméticas foram apresentadas. Evidentemente, fixada uma dada *performance* do sistema, existe o compromisso freqüente entre velocidade e preço. Se estivermos interessados em unidades aritméticas bem rápidas, seu custo será alto, principalmente com a inclusão de várias operações e códigos. Unidades aritméticas que multiplicam e dividem são caras, existindo principalmente em sistemas para usos específicos, ou de grande porte. Essas operações são realizadas com deslocamento e somas sucessivas. Essas operações, porém, na maioria dos sistemas, são realizadas por *software* ou *firmware* (microprogramas, por exemplo)^(2, 12).

f. Exemplo detalhado de uma unidade aritmética

Vamos supor que queiramos uma unidade aritmética para um sistema que tenha palavra de 8 bits, com números codificados em binário complemento de dois. A unidade aritmética deverá ter um somador composto da associação em cascata de dois módulos de 4 bits do tipo "vai um antecipado" (Fig. 3-100).

Além da soma, essa unidade aritmética deverá realizar ainda operações de subtração ($X - Y$), *AND*, *OR* e *XOR* (*exclusive OR*). Para a subtração, usaremos a idéia sugerida pela Fig. 3-98, e as funções lógicas usarão a filosofia da Fig. 3-99. Na Fig. 3-101, o leitor encontra um diagrama em blocos da unidade aritmética que estamos implementando.

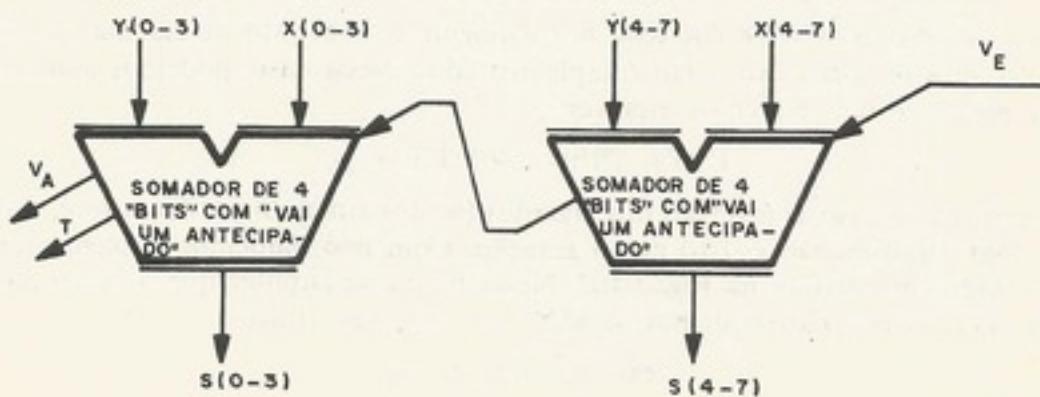


Figura 3-100. Somador de 8 bits

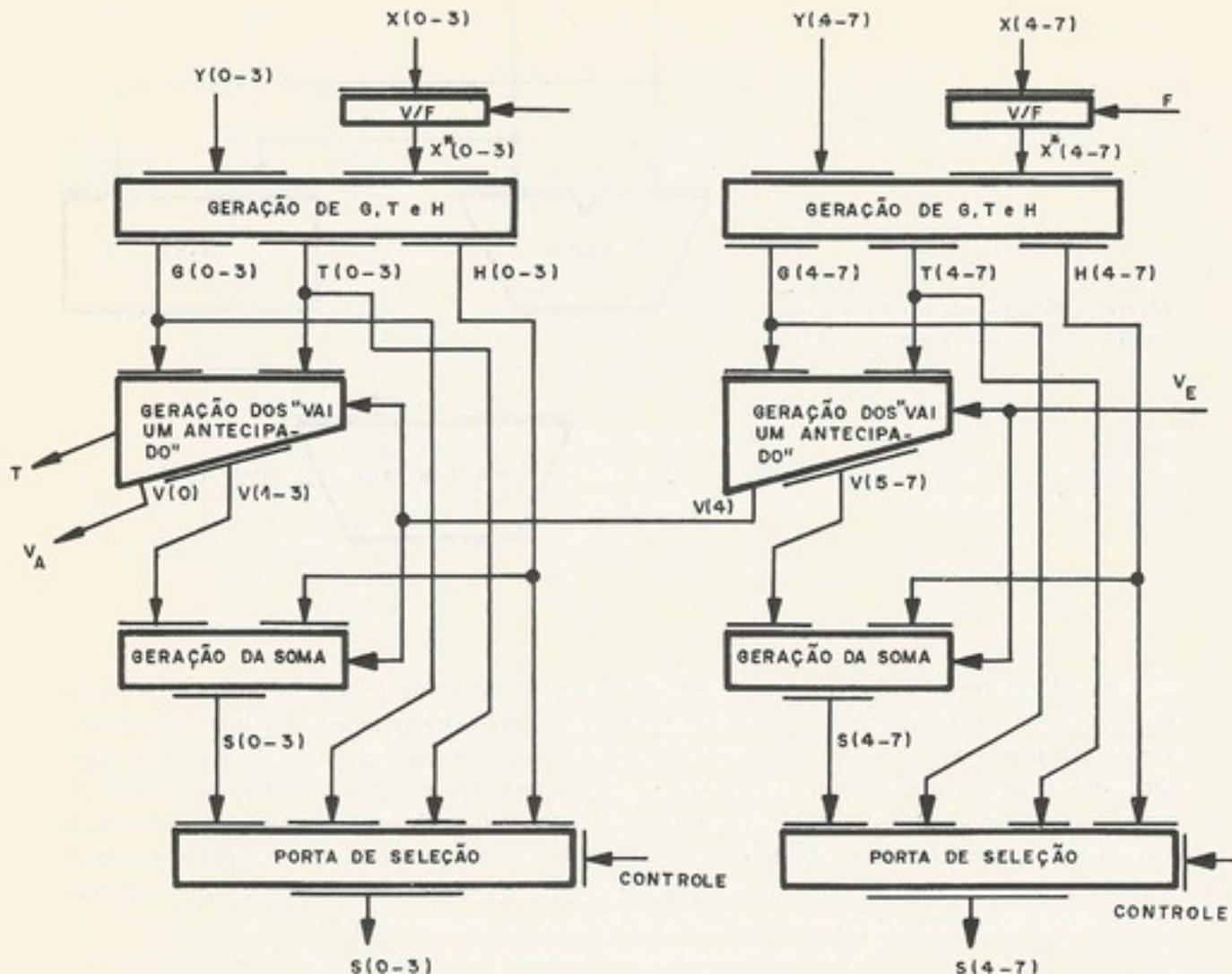


Figura 3-101. Esquema em blocos da unidade aritmética

O somador "vai um antecipado" usa os sinais

$$\begin{aligned} \text{geração do "vai um"} \quad G(i) &= X(i) \cdot Y(i), \\ \text{transmissão do "vai um"} \quad T(i) &= X(i) + Y(i); \end{aligned}$$

o "vai um" correspondente será

$$V(i) = G(i) + V(i+1) \cdot T(i)$$

e a soma correspondente será

$$S(i) = X(i) \oplus Y(i) \oplus V(i).$$

Como as funções *AND* e *OR* têm, normalmente o dobro do atraso que as *NAND* e *NOR*, geraremos os sinais $G(i)$ e $T(i)$ complementados. Nesse caso, podemos usar a álgebra booleana para ver que podemos escrever

$$V'(i) = T'(i) + G'(i) \cdot V'(i+1),$$

o que sugere, nesse caso, a inversão dos significados dos sinais $G(i)$ e $T(i)$, isto é, $G'(i)$ comporta-se como transmissão e $T'(i)$ como geração. Com isso, pudemos implementar o circuito mostrado em detalhes na Fig. 3-102. Nessa figura, se supuser que o bloco *XOR* tem atraso de duas portas ($2d$) podemos verificar os seguintes atrasos:

- atraso da saída $U : 9d$,
- atraso da saída $V_A : 6d$,
- atraso da saída $T : 8d$.

elementos do projeto lógico

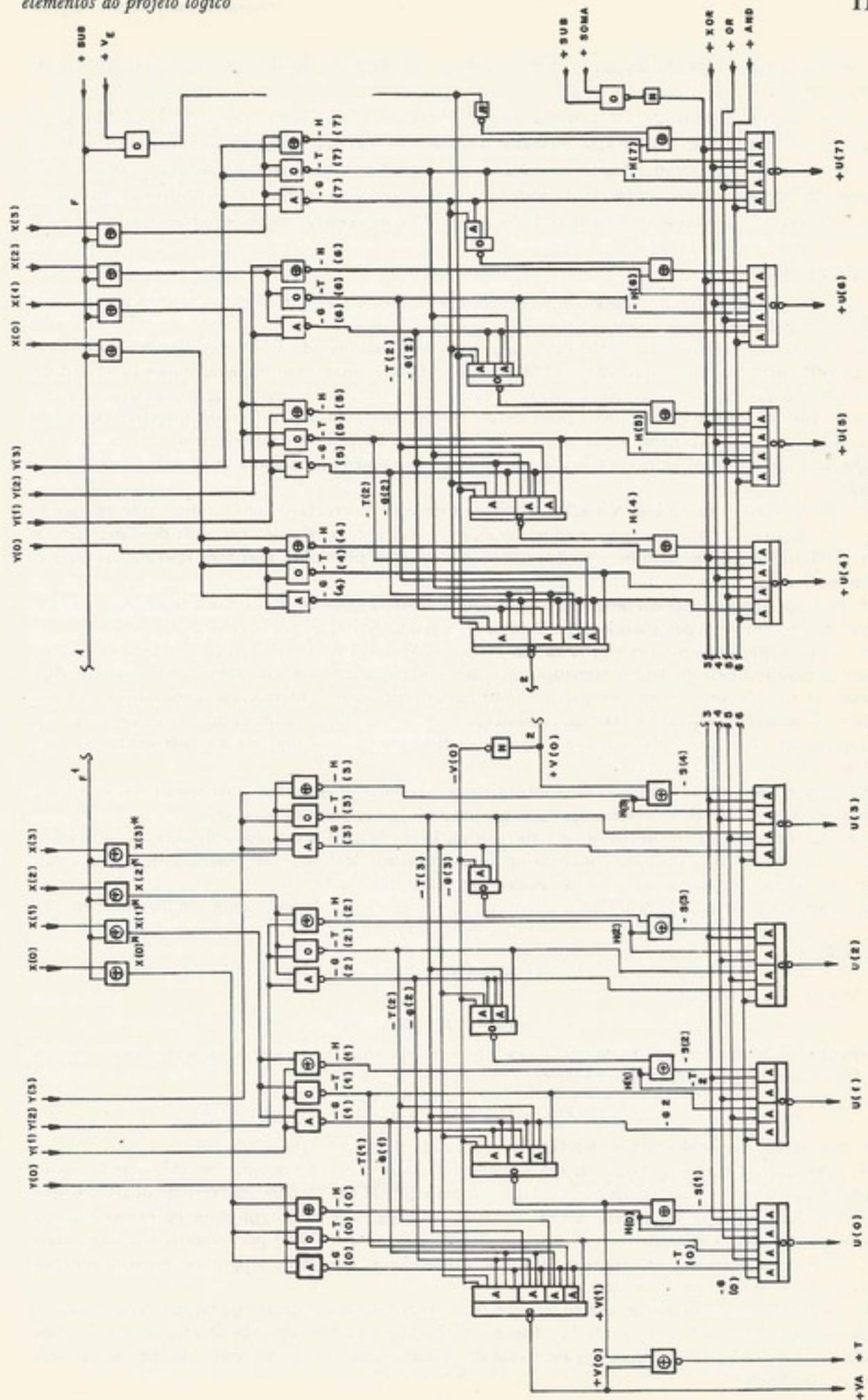


Figura 3-102. Somador de 8 bits com "vai um antecipado"

O mesmo somador implementado com a técnica de propagação do "vai um" teria um atraso para a saída U de cerca de $15d$.

Um inconveniente do circuito da Fig. 3-102 está no fato de a entrada Y ter um *fan-in* igual a 3, o que pode atrapalhar bastante no caso de ela ser alimentada, junto com outros circuitos, por uma mesma fonte de sinal. Contudo, se verificarmos que a entrada X tem atraso de $2d$ maior que a entrada Y , poderemos resolver o problema colocando, na entrada Y , dois inversores em cada bit, para reduzir o *fan-in* sem comprometer o atraso do somador.

EXERCÍCIOS

3-25. (a) Projete um circuito "soma completa" em dois níveis *AND* e *OR*. (b) Projete um circuito "subtração completa" em dois níveis *AND* e *OR*.

3-26. Projete um somador de 16 bits que usa grupos de somadores de 4 bits com ligação entre os somadores do tipo "vai um antecipado". (a) Mostre em detalhes como serão implementados os somadores de 4 bits. Provavelmente será preciso definir duas saídas extras por somador, grupo geração e grupo transmissão. (b) Mostre em detalhes como seria a geração dos "vai um" intersetores. (c) Determine os atrasos máximos e comente sobre as possibilidades de o somador ser internamente organizado com propagação do "vai um" entre os 4 bits, ao invés de ser implementado como a técnica do "vai um antecipado".

3-27. (a) Projete em dois níveis *NAND* e *NAND* um circuito "correção de seis" como o que aparece na Fig. 3-91. Leve em conta que o bit menos significativo não é alterado pelo processo de somar seis ou não. (b) Tente encontrar alguma solução mais barata para o problema, mas sem restringir o número de níveis lógicos.

3-28. (a) Faça um esquema em blocos de um somador de três dígitos *BCD* que use a técnica da correção prévia de seis em uma das parcelas. (b) Detalhe os circuitos de correção de seis e menos seis.

3-29. (a) Faça um esquema em blocos de um somador de três dígitos *BCD* excesso de três usando somadores hexadecimais. (b) Use o somador em uma unidade aritmética que some e subtraia em *BCD* excesso de três. Admita existir um quarto dígito nas parcelas que é interpretado como dígito de sinal (0000 para números positivos e 0001 para números negativos); admita também que o código usado seja "complemento de dez". (c) Inclua saídas de transbordamento e "vai um" na unidade aritmética. Comente sobre o uso dessas saídas.

3-30. Projete um subtrator binário de 4 bits que usa a técnica "emprestimo antecipado". Você terá de definir funções análogas às funções geração e transporte anteriormente analisadas.

3-31. Faça um esquema em blocos de um subtrator *BCD* de três dígitos usando subtratores hexadecimais.

3-32. Faça um esquema em blocos de uma unidade aritmética de 4 bits com potencial de somar e subtrair em binário complemento de dois, usando um subtrator de 4 bits.

3-33. Como já dissemos, podemos definir o transporte de "vai um" como sendo, indiferentemente, ou, então,

$$T(i) = X(i) + Y(i)$$

$$T(i) = X(i) \oplus Y(i).$$

(a) Mostre que a função

$$V(i) = G(i) + T(i) \cdot V(i+1)$$

e a mesma, indiferentemente da definição por nós adotada. (b) Mostre que, com a primeira definição de $T(i)$, a seguinte igualdade é verdadeira:

$$\overline{V(i)} = \overline{T(i)} + \overline{G(i)} \cdot \overline{V(i+1)}.$$

Será essa igualdade verdadeira se adotarmos a segunda definição para $T(i)$?

3-34. Faça um esquema em blocos de um somador que, sob comando de um sinal de controle, faça a soma binária com parcelas de 16 bits ou faça a soma decimal (*BCD*) com parcelas de quatro dígitos.

3-35. Faça um esquema em blocos de uma unidade aritmética que opera em números *BCD* de quatro dígitos, onde o mais significativo é o bit de sinal (0000 para positivo e 0001 para negativo). Essa unidade deve ter a capacidade de fazer soma e subtração nos códigos: (a) sinal e amplitude, (b) complemento de nove e (c) complemento de dez.

3-36. (a) Projete um somador de quatro bits que some números em complemento de um, com a saída V_A realimentada diretamente na entrada V_E , dispense o *flip-flop* intermediário. (b) Mostre em que circunstâncias o circuito oscila. (c) Sugira algum modo de usar esse circuito de tal forma que ele nunca oscile (sem o modificar).

3-37. Como você pode usar a subtração completa de dois etc.) para obter a subtração? (a) Investigue a possibilidade de implementar a subtração completa de dois etc.) para obter a subtração? (b) Associe essa subtração completa com a subtração completa de dois etc.) para obter a subtração? (c) Analise as possibilidades de implementar a subtração completa de dois etc.) para obter a subtração?

3.6 FLUXO DE DADOS

Em sistemas digitais, é comum que o fluxo de dados entre componentes seja controlado por um sinal de controle. Esse sinal de controle é gerado por um registrador e enviado para o próximo componente.

As microplaquetas contêm circuitos para gerar o sinal de controle. Estudaremos esses circuitos mais tarde. Trataremos aqui de como gerar o sinal de controle.

a. Transferência de dados

Um problema comum é transferir dados de um dispositivo para outro. Isso é feito usando, do tipo de transferência de dados.

A transferência de dados tem o inconveniente de ser mais lenta depois de se passar por um exemplo de transferência de dados em série.

A técnica de transferência de dados (Fig. 3-104), pois a operação é mais rápida.

Também pode ser realizada a transferência de dados em paralelo.



Figura 3-104

3-37. Como você deve ter notado, costuma-se escolher códigos (complemento de um, complemento de dois etc.) para que se possa operar com números positivos e negativos sem grandes problemas. Além disso, a subtração é feita complementando-se (facilmente) o subtraendo e somando-se ao minuendo.

(a) Investigue a possibilidade de se fazer um circuito que se comporte como "soma completa" ou como "subtração completa", controlado por um sinal em uma entrada extra. Isto é, o circuito deverá ter quatro entradas (dois operandos, "vem um" e a linha de controle) e duas saídas (resultado da operação e "vai um"). Quando o controle for zero, o circuito se comportará como um circuito "soma completa" e, quando for igual a um, como "subtração completa".

(b) Associe circuitos do tipo definido em cascata (com propagação do "vai um"), formando uma unidade aritmética que some ou subtraia números de 4 bits mais um bit de sinal. Analise o problema para cada um dos códigos (i) sinal e amplitude, (ii) complemento de um e (iii) complemento de dois.

(c) Analise as possibilidades de resolver o problema de maneira análoga, com a técnica de previsão do "vai um" ("vai um antecipado").

3.6 FLUXO DE SINAIS

Em sistemas digitais, habitualmente operações complexas são realizadas com a composição de operações simples, muitas vezes chamadas de microoperações, por exemplo, limpar registradores, transferir conteúdo de um registrador para outro, transferir para um registrador a soma de outros dois, etc.⁽³⁾.

As microoperações que trataremos neste item são as que envolvem a transferência de dados entre registradores, podendo ou não esses dados passarem por alguma transformação. Estudaremos, principalmente, as técnicas de transferência e controle dos dados. Esses dados são sinais elétricos armazenados nos registradores e transitando pelo sistema. Trataremos aqui do fluxo desses sinais.

a. Transferência de dados para registradores

Um problema comum em sistemas digitais é a transferência de dados de um registrador para outro. Isso pode ser realizado de várias maneiras, dependendo do tipo de flip-flop usado, do tipo de registradores envolvidos e da filosofia do ciclo básico do sistema (o timing). A transferência pode ser em série ou em paralelo.

A transferência em série, que deve ser usada com registradores deslocadores (Fig. 3-103), tem o inconveniente da velocidade de transferência, isto é, a transferência só se completa depois de n pulsos de controle, sendo n o número de bits do registrador. Na Sec. 3.5, vimos um exemplo de transferência de dados em série com modificações nesses dados (somador em série).

A técnica mais usada, apesar do custo elevado, é a da transferência em paralelo (Fig. 3-104), pois a operação termina com apenas um pulso de controle. Outra vantagem dessa técnica é que não necessitamos de registradores deslocadores para realizar a transferência.

Também podemos ter uma transferência que é um misto de série e paralelos, ou seja, transferência em série de blocos de vários bits. Isso é feito principalmente no caso em que

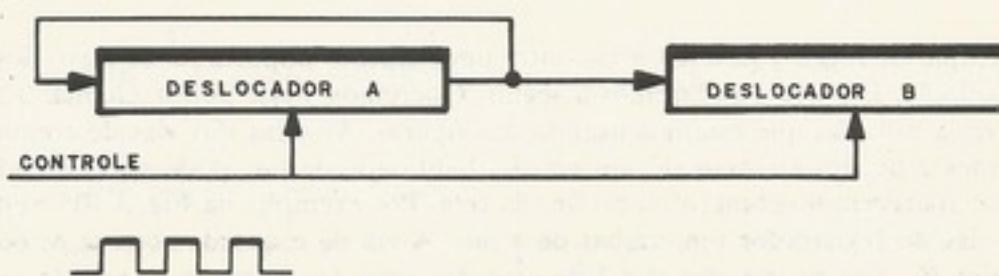


Figura 3-103. Transferência do conteúdo do deslocador A para o deslocador B.

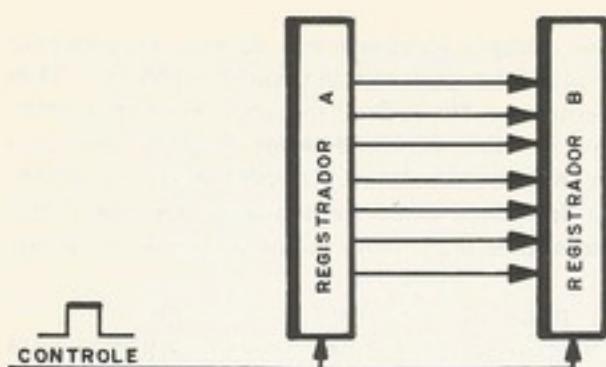


Figura 3-104. Transferência do conteúdo do registrador A para o registrador B

modificamos os dados durante a transferência e o circuito modificador não tem capacidade total do registrador. Na Fig. 3-105, vemos um exemplo dessa técnica, onde somamos 2 registradores de 8 bits em um somador de 4 bits, colocando o resultado em um terceiro registrador de 8 bits.

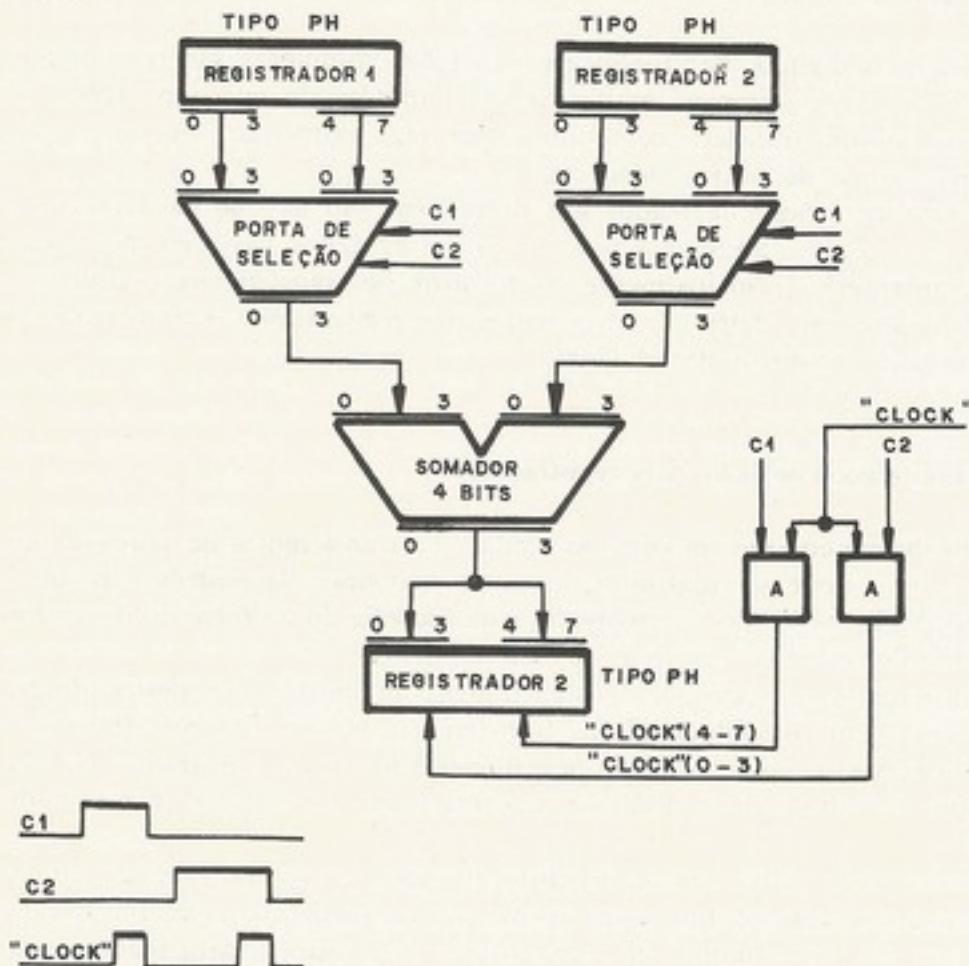


Figura 3-105. Soma de 8 bits com somador de 4 bits

No exemplo da Fig. 3-105, o leitor encontra um elemento importante do fluxo dos dados, a porta de seleção. É o que estudaremos a seguir. Queremos, neste ponto, chamar a atenção do leitor para a notação que estamos usando nas figuras. As setas são vias de comunicação entre unidades e podem ter mais de um bit. A quantidade de bits pode ser deduzida pelos números que aparecem no começo ou no fim da seta. Por exemplo, na Fig. 3-105, vemos que saem duas vias do registrador um, ambas de 4 bits. A via da esquerda conduz os bits 0 a 3 do registrador R_1 para as posições 0 a 3 das entradas da porta de seleção; a via da direita conduz os bits 4 a 7 do registrador R_1 para a outra entrada da porta de seleção.

Porta de seleção

Sempre que queremos transferir o conteúdo de um registrador para outro, vemos na Fig. 3-104 que devemos colocar na saída do registrador dois modos de transferência. Vamos analisar o esquema da Fig. 3-105, que é similar ao esquema da Fig. 3-104.

Figura 3-105

Figura 3-105

Porta de seleção

Poderíamos, por exemplo, ter uma única entrada e várias saídas para transferir a informação de um registrador para outros.

Porta de seleção⁽⁴⁾ (multiplexador)

Sempre que temos necessidade de escolher entre duas palavras para colocar na entrada de um circuito qualquer, como somador, registrador etc., usamos um circuito cujo modelo vemos na Fig. 3-106, que tem a característica de, conforme o sinal de controle (E_1/S ou E_2/S) colocar na saída uma das palavras da entrada (E_1 ou E_2). Na Fig. 3-107, o leitor encontra dois modos de implementar essas portas; apesar de ser a saída complementada, a vantagem do esquema da Fig. 3-107(a) está no atraso: tem um atraso de uma porta, enquanto que no esquema da 3-107(b) tem atraso de duas portas.

Figura 3-106. Modelo de uma porta de seleção

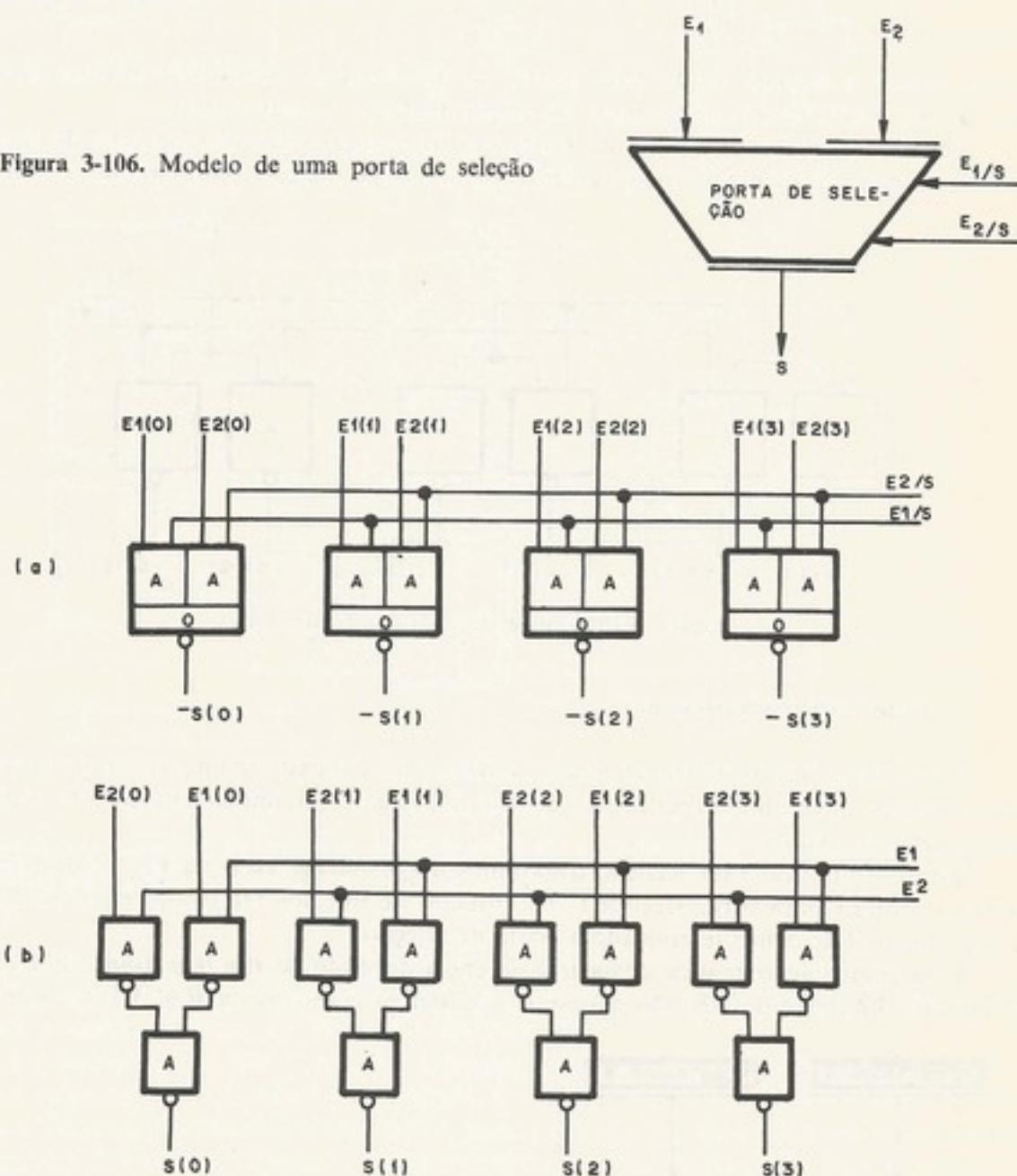


Figura 3-107. Porta de seleção de 4 bits implementada com (a) DOT-OR e (b) com NANDs

Porta de distribuição (demultiplexador)

Poderíamos dizer que é o problema inverso do anterior, pois temos uma palavra de entrada e várias de saída. Queremos, sob comando de um sinal de controle (E/S_1 ou E/S_2), transferir a entrada para uma das saídas (S_1 ou S_2), ficando as outras saídas nulas. Na Fig.

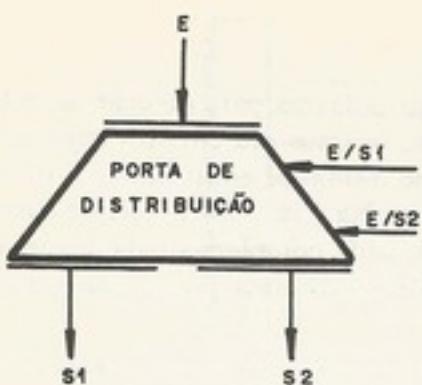


Figura 3-108. Modelo de porta de distribuição

3-108, vemos o modelo usado para portas de distribuição, no caso, com duas saídas. Na Fig. 3-109, encontra-se um exemplo de implementação dessas portas.

Uma das grandes utilidades da porta de distribuição está no aumento do *fan-out* de um sinal, mas apresenta o inconveniente de ter uma relação pino/circuito elevada.

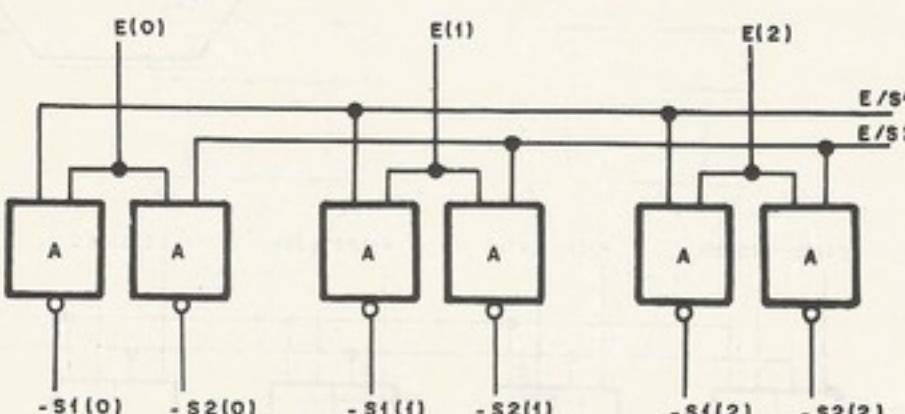


Figura 3-109. Porta de distribuição de 3 bits

b. Transferência em paralelo⁽⁷⁾

O problema básico da transferência de dados em paralelo de um registrador para outro tem soluções diferentes, dependendo do *flip-flop* usado na implementação do registrador que recebe os dados.

Para exemplificar essa técnica, trataremos do problema visto na Fig. 3-110, onde queremos transferir para o registrador *C* o conteúdo de um dos registradores *A* ou *B*, dependendo do sinal de controle aplicado à porta de seleção.

A implementação prática dessa transferência depende do *flip-flop* usado. Por exemplo, se for um *flip-flop* tipo *RS*, são necessários circuitos como os da Fig. 3-111, onde vemos,

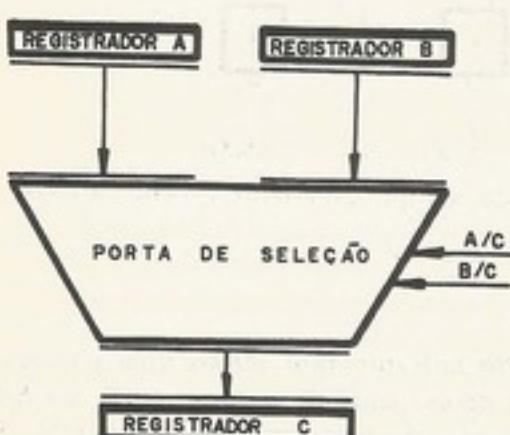


Figura 3-110. Exemplo de transferência de dados

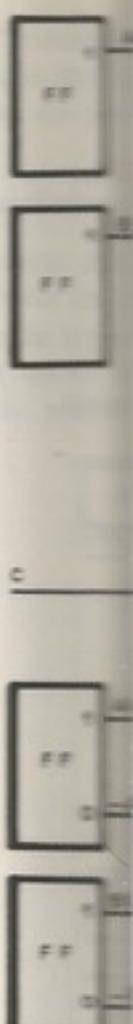


Figura 3-111. (a) Transf.

na 3-111(a), um esquema que o dispara se o controlador conforme o dado, se

Se o flip-flop é carmos a saída de con-

Com flip-flop borda, com a dife-

Usando flip-flop ligando na entra-

Dependendo do flop, como se vê no gerado X desliga o

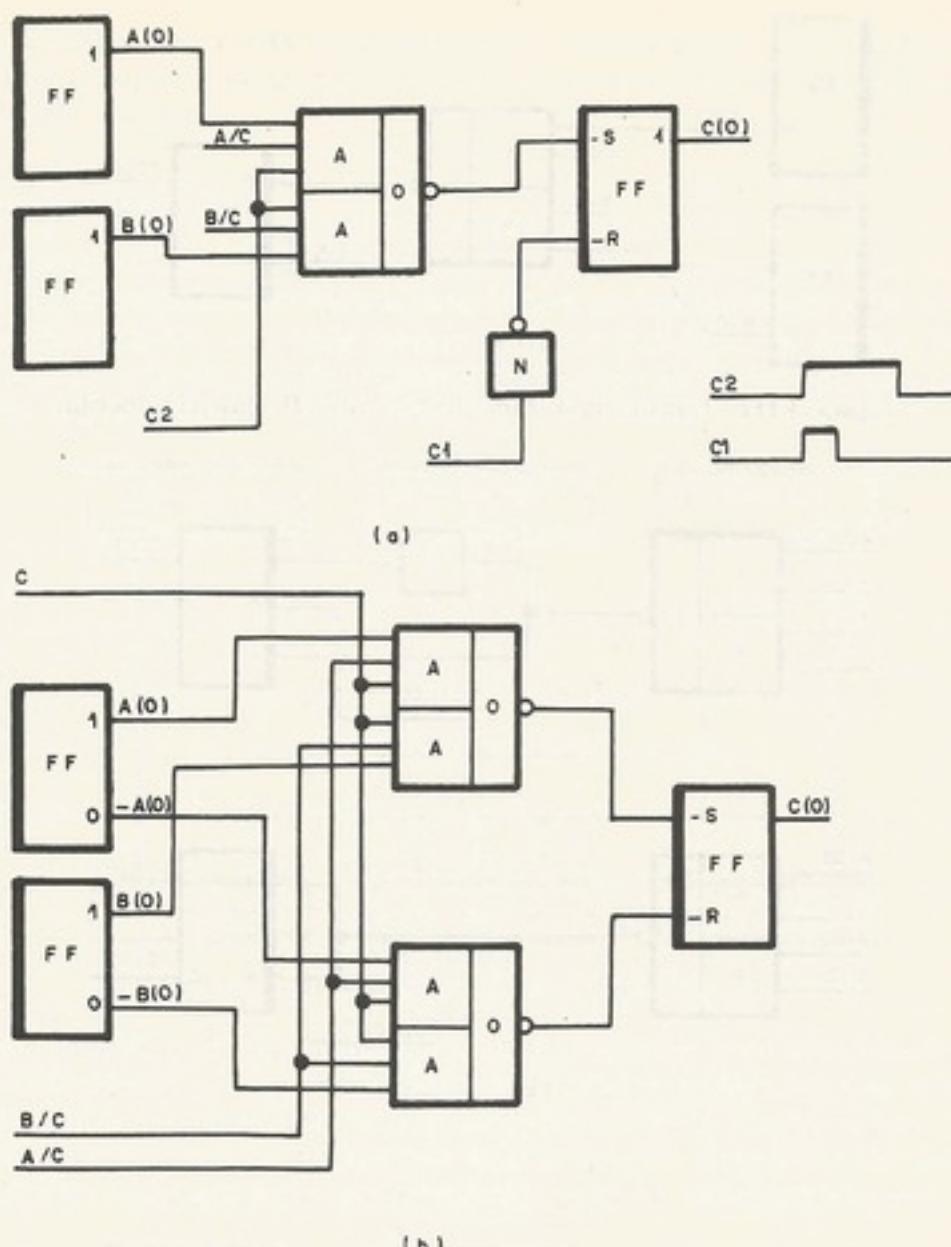


Figura 3-111. (a) Transferência com dois sinais de clock, C_1 e C_2 (overriding set) para flip-flop RS; (b) transferência para flip-flop RS com um sinal de clock (jam-transfer)

na Fig. 3-110, onde queremos A ou B , dependendo. Por exemplo, na Fig. 3-111, onde vemos,

Se o flip-flop for do tipo D sensível à borda, o problema se reduzirá, pois basta colocarmos a saída da porta de seleção na entrada D e darmos um sinal de clock (sinal de controle para disparar flip-flops). Na Fig. 3-112, vemos um circuito de 1 bit dessa transferência.

Com flip-flop tipo PH (polarity hold) é usado o mesmo esquema do tipo D sensível à borda, com a diferença de que o dado precisa ficar constante em D , já que esse flip-flop é do tipo sensível ao nível.

Usando flip-flop tipo JK , podemos transformá-lo em um tipo D invertendo o dado e ligando na entrada K . Na Fig. 3-113, encontram-se exemplos mostrando 1 bit desse tipo de transferência.

Dependendo da situação, podemos implementar a porta de seleção junto com o flip-flop, como se vê na Fig. 3-113(c) (flip-flop tipo PH). Quando chega o sinal de clock, o sinal gerado X desliga o loop de realimentação e o sinal X' abre a porta selecionada por um dos

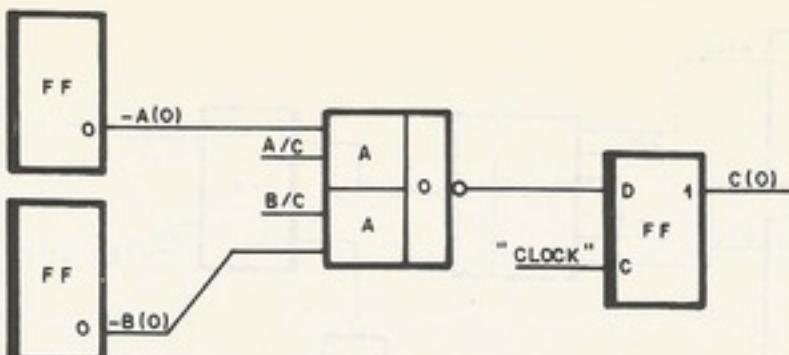
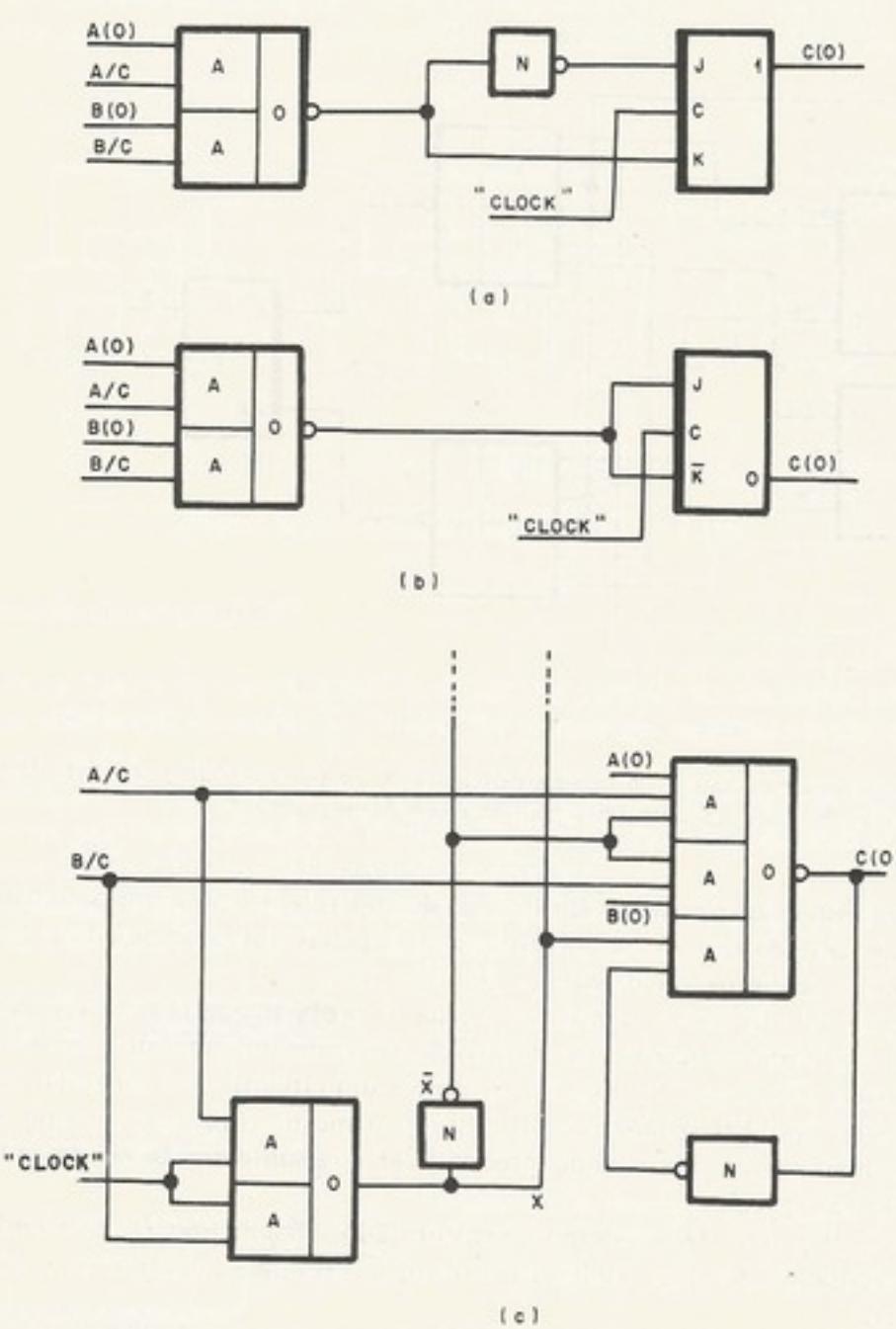


Figura 3-112. Transferência para flip-flop tipo D sensível à borda

Figura 3-113. Transferência para flip-flop (a) JK, (b) $J\bar{K}$ e (c) PH com porta de seleção implementada junto com flip-flop

sinais, A/C (comando de realimentação do flip-flop dependendo do seu estado)

c. Transferências

O conceito de transferência é a transferência de um dado para frente de cada registrador. Isso encareceria muito o pino/circuito.

Se, nesse sistema, quisermos transferir simultaneamente mais de uma informação via. Introduziremos o conceito de transferência. Esse registrador tem que ser dividido em partes para que possam ser transferidas. Essas partes são:

sinais, A/C (conteúdo de A para C) ou B/C , para a entrada do bit $A(0)$ ou $B(0)$ na malha de realimentação do flip-flop. O sinal X e X' podem ser comuns a todos os bits do registrador, dependendo do seu *fan-out*.

c. Transferências por vias

O conceito de via (*bus*)⁽³⁾, foi introduzido para resolver o problema visto na Fig. 3-114, a transferência de vários registradores para outros. A solução usual seria colocarmos à frente de cada registrador que pode receber os dados uma porta de seleção. Essa solução encareceria muito o sistema, dada a grande quantidade de portas que têm uma relação pino/circuito muito grande.

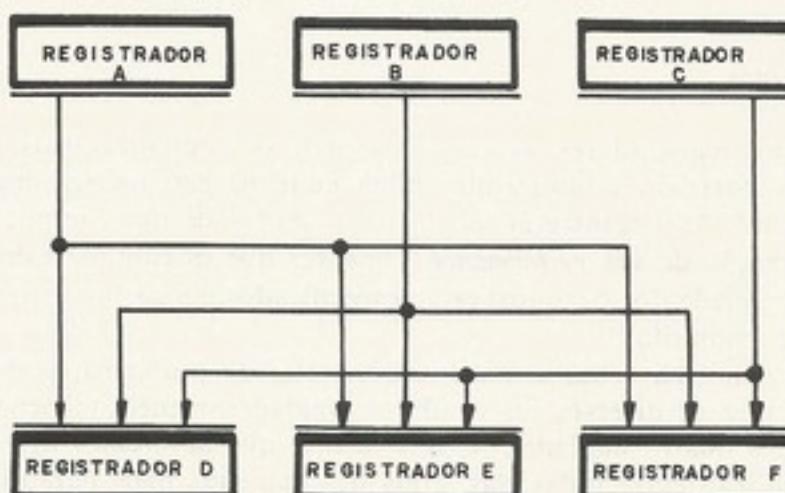


Figura 3-114. Transferência de vários registradores para outros

Se, nesse sistema, pudermos dispensar a possibilidade de fazer duas transferências distintas simultaneamente ($A \rightarrow D$ e $C \rightarrow E$, por exemplo), poderemos usar o conceito de via. Introduziremos um registrador a mais com uma porta de seleção apenas (Fig. 3-115). Esse registrador tem a função de fazer um armazenamento temporário do dado que está sendo transferido. Esse registrador (via) introduzido pode ser implementado com flip-flop RS

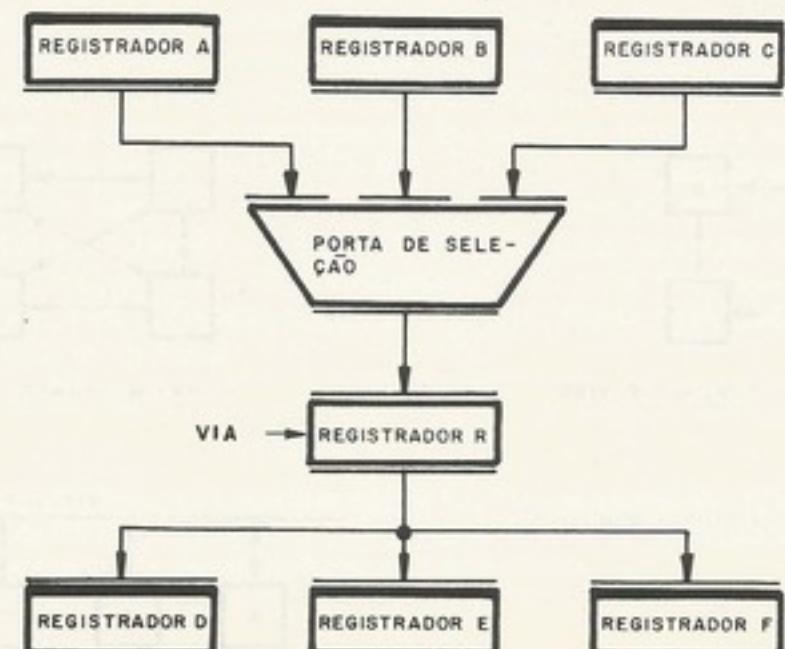


Figura 3-115. Conceito de via na transferência de dados

em esquema do tipo mostrado na Fig. 3-111(a), ligando em C_1 um sinal repetitivo, de maneira a limpar sempre o flip-flop, já que ele é usado para armazenamento temporário.

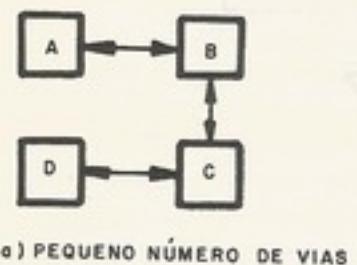
Nos capítulos seguintes você notará a grande importância do conceito de via, freqüentemente usado na maioria das unidades de tratamento de dados.

Em muitos sistemas onde se usa o conceito de via, o registrador R é dispensável. Isso é possível quando os registradores são implementados com flip-flops tipo D ou, então, quando o ciclo básico da máquina permite. Por exemplo, na Fig. 3-115, a grande utilidade do registrador R é desafogar os registradores A , B e C tão logo a porta de seleção termine seu trabalho. E, se os registradores D , E e F não puderem ser usados ainda, o dado ficará armazenado em R . Isso ficará mais claro se imaginarmos que os registradores D , E e F sejam os próprios A , B e C e, além disso, sejam sensíveis ao nível.

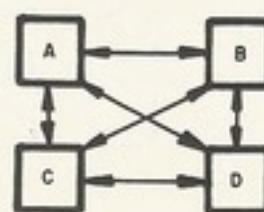
d. Fluxo de dados

O conjunto dos registradores, as vias e os circuitos combinacionais, ou seja, o fluxo de dados, são uma parte do hardware que influí muito na performance interna do sistema. Segundo Hellerman⁽²⁾, "a organização do fluxo de dados de um computador é um fator básico na determinação de sua performance. Uma vez que os caminhos do fluxo (inclusive as larguras e a velocidade dos circuitos) estão especificados e uma boa parte da performance do sistema está circunscrita".

Antes de entrarmos na nossa análise, vamos tratar do problema da definição das vias de comunicação entre os diversos registradores, unidade aritmética, memória etc. Vamos supor que tenhamos quatro unidades, A , B , C e D , e que deveremos ligá-las de modo tal que exista comunicação entre todas elas. Uma das maneiras mais baratas e que usa uma quantidade pequena de vias de ligação é vista na Fig. 3-116(a). O inconveniente é que, se A quer se comunicar com D , ele terá que fazê-lo via B e C , ou seja, mandar a mensagem para B , que deverá, então, transmiti-la a C , que, por sua vez, a mandará para D . Isso, é claro, causará alguns problemas e atrasos em certas circunstâncias, mas, se a grande maioria das comunicações ocorrer entre as unidades B e C , esse esquema poderá funcionar bem. O outro extremo seria dispormos de todas as ligações possíveis entre as unidades [Fig. 3-116(b)]. Nesse caso, não existirão mais atrasos desnecessários, mas, contudo, o preço talvez não

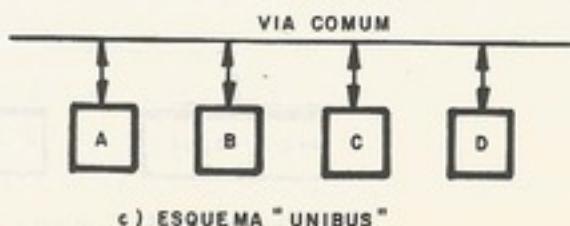


a) PEQUENO NÚMERO DE VIAS



b) MÁXIMO NÚMERO DE VIAS

Figura 3-116. Definição dos caminhos do fluxo de dados



justifique tal alternativa (unibus) [Fig. 3-116(c)]. Todas as comunicações devem passar por essa via comum, que pode a transmitir.

Em resumo, podemos considerar:

- 1) compartilhamento de dados, e pode aumentar a capacidade simultânea;
- 2) diminuição da coincidência;
- 3) aumentando a coincidência;
- 4) fazendo-se menor a coincidência, aumenta-se o custo de implementação.

É importante observar que, para uma unidade de processamento, uma unidade de comunicação é necessária para processar as informações. As microplaquetas possuem os fios ou as linhas de comunicação que são parte da unidade de processamento.

É a unidade de comunicação que faz parte do painel de controlo e das estruturas de construção. A unidade de comunicação é projetada para ser utilizada tanto como sistema interno quanto externo. É a unidade de comunicação que é utilizada para ligar o sistema interno ao sistema externo.

Atualmente existem muitos tipos de circuitos integrados lógicos (hard-wired logic) que podem ser programados. Um dos tipos mais populares é o PLD (Programmable Logic Device). Um PLD é uma tecnologia de fabricação de circuitos integrados que permite a programação de funções lógicas. Isso está detalhadamente descrito no Capítulo 10. O PLD é uma unidade de comunicação que pode ser programada para realizar funções lógicas complexas. É a unidade de comunicação que é utilizada para ligar o sistema interno ao sistema externo.

EXERCÍCIOS

- 3-38. Na Fig. 3-115, se o sinal de seleção seja dividido entre C_1 , C_2 e $clock$. Escreva a expressão para C_1 .
- 3-39. Determine, para o esquema da Fig. 3-115, os dois sinais de controle necessários para ligar o sinal de saída de cada registrador ao bloco de controle.
- 3-40. Admita que temos uma unidade central de processamento com quatro registradores.

justifique tal alternativa. Uma outra solução, que tem recebido o nome de via comum (*unibus*) [Fig. 3-116(c)], seria apenas uma via de comunicação, comum a todas unidades. Todas as comunicações seriam feitas através dessa via comum. Essa solução é apenas barata na aparência, pois o controle especial para evitar conflitos (duas unidades querendo mandar informações ao mesmo tempo) e a necessidade de circuitos especiais para ligar as unidades a essa via comum, pode tornar a solução relativamente cara. Além do quê, essa solução impede a transmissão simultânea de mais de uma mensagem entre unidades diferentes.

Em resumo, podemos dizer que na definição das vias do fluxo de dados, existirão os seguintes compromissos:

- 1) compartilhando-se caminhos para economizar, aumenta-se o atraso na transferência de dados, e pode não haver redução do custo;
- 2) diminuindo-se os caminhos, diminui-se a oportunidade de coincidência (comunicação simultânea entre diferentes unidades);
- 3) aumentando-se os caminhos diretos, aumenta-se o custo, mas pode-se aumentar a coincidência;
- 4) fazendo-se caminhos mais largos, para processar mais informações de uma vez, aumenta-se o custo desses caminhos.

É importante observar que o fluxo de dados é, num sentido, como uma unidade passiva, uma unidade que tem a capacidade de executar as *microoperações* necessárias para processar as instruções da arquitetura, mas falta o conhecimento de quando e em que sequência as *microoperações* devem ser feitas. O fluxo de dados é como uma marionete, com os fios ou as linhas de controle disponíveis para quem quiser manobrar. Em nosso caso, as linhas de controle são os sinais que abrem as portas de seleção, que indicam a operação da unidade aritmética e que controlam a entrada de dados nos registradores.

É a unidade de controle que "sabe" o que fazer. A unidade de controle recebe sinais do painel de controle, do relógio central, e também, recebe o código de operação das instruções. A unidade de controle também recebe dados de realimentação do fluxo de dados, tais como *status*, indicando se houve transbordamento no somador, se o conteúdo do acumulador é zero ou se o sinal é negativo, etc. Com essas informações, a unidade de controle é projetada para ativar as linhas de controle do fluxo de dados no tempo próprio e na ordem certa.

Atualmente existem duas técnicas de implementação da unidade de controle: por blocos lógicos (*hard-wired*) ou com o auxílio de uma memória de controle que armazena um microprograma. Um conjunto de microprogramas chama-se *microprogramação*. Em ambas as técnicas, um assunto central ao projeto é o ciclo da máquina, que regula o *timing* do sistema. Isso está detalhado no Cap. 8. A coordenação dos sinais de controle é fundamental para o correto funcionamento de um sistema. Na Fig. 3-105, vimos um somador e, da geração da sequência correta dos sinais C_1 , C_2 e *clock*, depende a soma correta; na Fig. 3-110, vimos que, para transferir, por exemplo, o conteúdo do registrador *A* para o registrador *C*, devemos antes ligar o sinal *A/C*, esperar o dado se propagar pela porta de seleção, para depois disparar o sinal de controle. A responsabilidade da geração desses sinais nos instantes corretos fica com a unidade de controle.

EXERCÍCIOS

- 3-38. Na Fig. 3-105, admitindo que o somador seja do tipo "vai um antecipado" (Fig. 3-86) e que a porta de seleção seja do tipo *DOT-OR* [Fig. 3-107(a)], determine as larguras máximas e mínimas dos sinais C_1 , C_2 e *clock*. Especifique as outras hipóteses que você adotou.
- 3-39. Determine, para os circuitos das Figs. 3-111, 3-112, 3-113 e 3-114 as larguras máximas e mínimas dos sinais de controle e *clock*.
- 3-40. Admita que tenhamos um sistema cuja saída seja uma palavra de 8 bits em paralelo (por exemplo, unidade central de processamento). Queremos transferir esses dados para um outro sistema cuja entrada

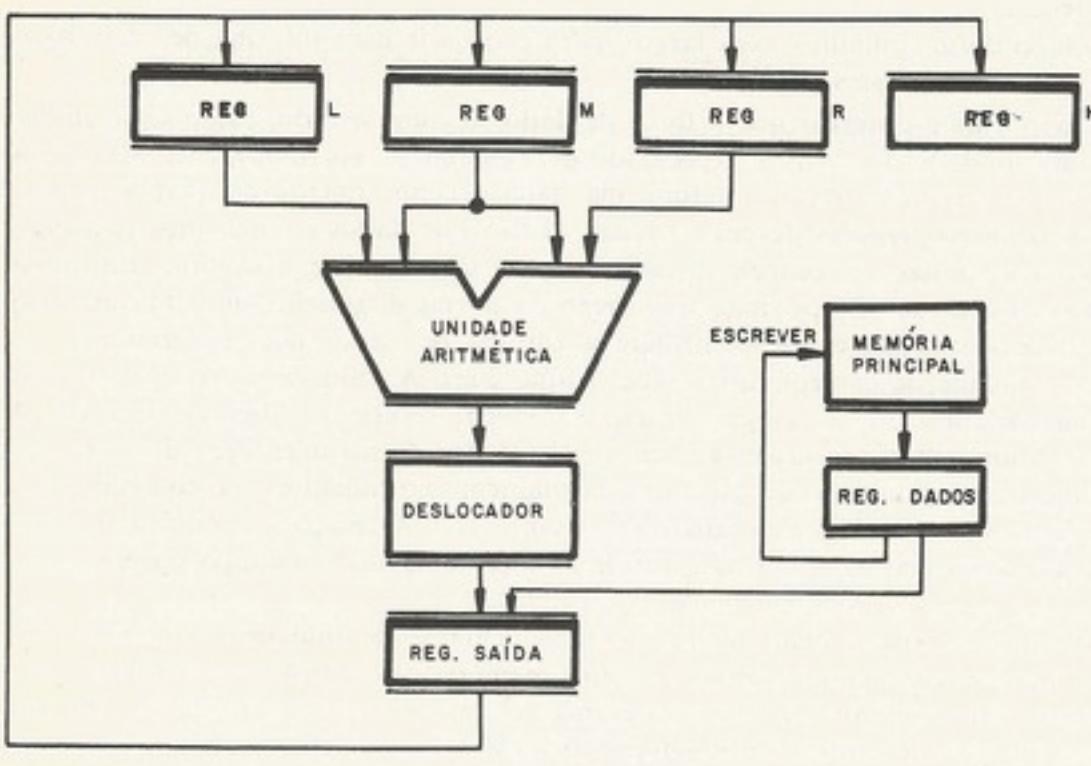
seja a série *bit a bit* (por exemplo, disco magnético). Faça um esquema de uma interface entre esses dois sistemas de modo que a transferência seja possível.

3-41. Usando o conceito de via, faça um esquema em blocos de um sistema digital com três registradores, de modo a ser possível a soma de quaisquer dois registradores e o resultado ser colocado em qualquer um dos três. Os registradores são implementados com *flip-flops* tipo *D* sensível à borda.

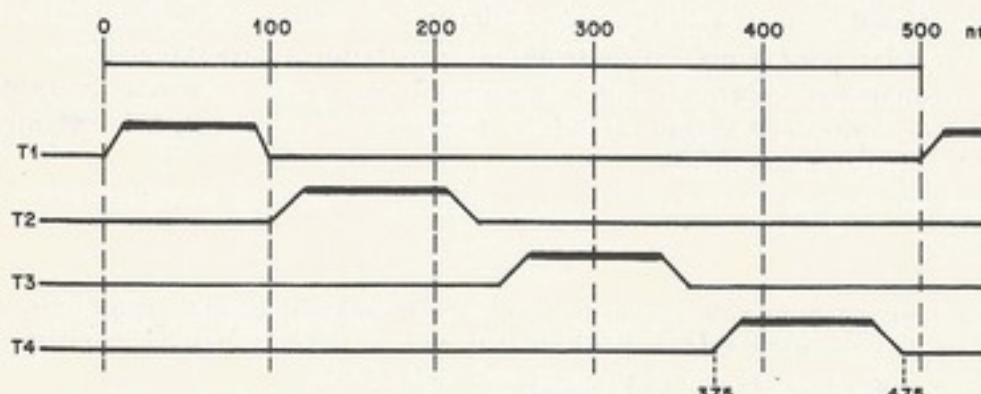
3.7 O "TIMING" E O CICLO DA MÁQUINA

a. Exemplo

Na maioria de sistemas, o ciclo da máquina é equivalente ao ciclo da memória principal ou o ciclo do fluxo de dados. Um sistema onde o ciclo da máquina é o ciclo do fluxo de dados é o sistema IBM S/360 modelo 50. O seguinte exemplo é baseado no modelo 50 (Husson⁽¹²⁾). O fluxo de dados e gráfico de *timing* do relógio central é visto na Fig. 3-117. O ciclo é de 500 ns.



a) FLUXO DE DADOS



b) SINAIS DE "TIMING" DO RELÓGIO CENTRAL

Figura 3-117. Um exemplo de ciclo de máquina baseado no modelo 50 do S/360

A unidade de implementações, cuja

Os registradores só mudam de estado quando nova informação é escrita. A queda de T_1 . O resultado (latched bus), devido ao fluxo através desse registrador ao contrário dos registradores pela microinstrução, é feita (veja Fig. 3-117). Os dois operandos estarão armazenados e cionado como resultado é de duas fases. Em circuitos combinacionais

T_4
DADOS DA M.P.
CONTROLE DE PORTAS
SINAIS DE CONTROLE

segunda fase, o resultado é enviado ao nível, é menor. A "pureza" do resultado, caso, temos os resultados recebidos a nova informação via armazenamento. Então essa informação é escrita na memória.

O modelo 50 lida na memória para a aritmética e o controlador determina a paridade dos resultados.

b. A determinação

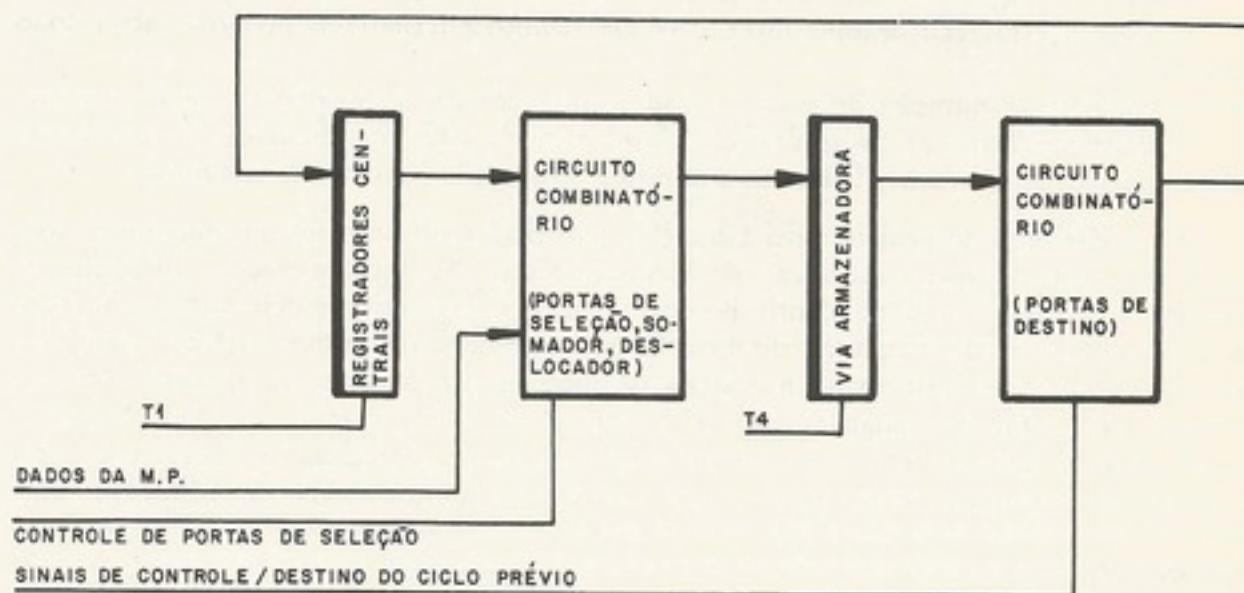
A alma da máquina é geralmente composta

entre esses dois
três registradores,
deslocado em qualquer
borda.

memória principal
é o ciclo do fluxo
no modelo 50
na Fig. 3-117.

A unidade de controle do modelo 50 é microprogramada e, como muitas dessas implementações, cada ciclo do fluxo de dados é controlado por uma microinstrução.

Os registradores *L*, *R*, *M* e *H* são os registradores centrais construídos com *flip-flop PH* e só mudam de estado durante o tempo T_1 . Isto é, se o registrador vai mudar, ele aceita a nova informação na subida do sinal T_1 e essa informação é "trancada" no registrador na queda de T_1 . O registrador *SDS* (saída do somador) age como uma "via armazenadora" (*latched bus*), ele é o distribuidor principal do fluxo de dados. Todas as transferências passam através desse registrador. Esse registrador muda de estado no T_4 de cada ciclo da máquina, ao contrário dos registradores centrais, que só mudam de estado quando são indicados pela microinstrução para receberem nova informação. O fluxo de dados, então, forma um laço (veja Fig. 3-118). Num ciclo típico, dois registradores são selecionados para fornecerem os dois operandos à unidade aritmética. Daí, o resultado, depois, talvez, de um deslocamento, estará armazenado no registrador *SDS*, no tempo T_4 . No próximo T_1 , o registrador selecionado como sendo o destino do resultado recebe-o do registro *SDS*. O sistema, então, é de duas fases. Em uma fase a informação dos registradores é transformada através de circuitos combinacionais (somador, deslocador) e armazenado na "via armazenadora" e, na



OBS. SINAIS "T1" e "T4" SÃO PARA TRANSFERÊNCIA DE DADOS AOS REGISTRADORES

Figura 3-118. Laço idealizado do fluxo de dados do modelo 50

segunda fase, o resultado é encaminhado aos registradores de destino. Com *flip-flop* sensível ao nível, é necessário usar um esquema dessa filosofia. Como em muitos sistemas práticos, a "pureza" desse esquema de *timing* é borrouda um pouco pela realidade. Em nosso caso, temos os sinais T_1 e T_2 a explicar. Acontece que, no começo de T_1 , o fluxo de dados recebe a nova microinstrução a ser executada, mas a informação de controle de destino da "via armazenadora" da microinstrução prévia precisa ser guardada até o fim do tempo T_1 . Então essa informação está guardada num registrador de controle durante o tempo T_2 .

O modelo 50 tem um esquema interno de confiabilidade. A paridade de cada palavra lida na memória principal, acompanha-a, bem como a paridade do resultado da operação aritmética e o conteúdo de cada registrador. O tempo T_3 é usado primariamente para testar a paridade dos registradores centrais, verificando-a enquanto estes estão estáveis.

b. A determinação do ciclo da máquina

A alma da unidade central de processamento é o ciclo da máquina. O relógio central geralmente consta de 4 a 40 linhas, cada linha com sua forma distinta. A duração do ciclo

depende de muitos fatores. Por exemplo, os atrasos (pior caso) do laço do fluxo de dados (veja a Fig. 3-118) não devem superar ao tempo adotado para o ciclo da máquina. Em máquinas controladas por microprogramação, o tempo de ciclo da memória de controle não deve exceder ao ciclo da máquina. Existem muitos sistemas nos quais o ciclo da máquina é sincronizado com o ciclo da memória principal, ou em que o ciclo da máquina é uma subdivisão do ciclo da memória principal. Por exemplo, no modelo 50 o ciclo da máquina é um quarto do ciclo da memória principal.

No processo de retirar instruções da memória principal e executá-las, o projetista subdivide a tarefa em certos ciclos da máquina e geralmente o ciclo consta de várias microoperações. Grosseiramente, existem duas fases principais. Na primeira, temos os *ciclos I*, em que a próxima instrução é retirada da memória principal, e, na segunda, os *ciclos E*, em que a instrução é executada. Para determinar o que acontece em qualquer ciclo da máquina, e para escalar a ordem certa dos ciclos, Ware anota os seguintes pontos⁽²³⁾:

- 1) para cada instrução, listar a seqüência de microoperações requeridas;
- 2) projetar um gráfico que escala todas as microoperações;
- 3) juntar as microoperações em comum a mais que uma instrução (para fazer economia);
- 4) fazer a escalação de microoperações em tempos alternativos, preferivelmente mais cedo no ciclo;
- 5) diminuir a duração do ciclo, tirando vantagem das coincidências onde possível;
- 6) ajustar e reajustar os gráficos de *timing* para evoluir uma escalação de microoperações para cada instrução, tentando comprimir a duração de ciclo da máquina.

No andamento do projeto, uma avaliação mais realista dos atrasos das microoperações talvez necessite de uma reescalação de algumas destas. As microoperações normalmente entram numa certa "moldura" dentro do ciclo. A meta do projetista é determinar a duração do ciclo e a largura dos caminhos do fluxo de dados de tal maneira que estes encaixem bem no ciclo da memória principal e nos ciclos da memória de controle ou memória local (se houver), para o fim de "balançar" bem o projeto. Aqui, interpretamos um "bom balanço" como sendo a situação em que nenhum componente ou especificação, obviamente, seja o ponto de estrangulamento, tanto como nenhum componente seja superdimensionado.

EXERCÍCIO

3-42. Suponha que, no caso do modelo 50, T_1 esteja presente no tempo entre 0 e 100 ns do ciclo, e T_4 presente no tempo entre 375 e 475 ns do ciclo. Suponha também que o atraso dos registradores (tipo *PH*) seja 25 ns. Tendo em vista a Fig. 3-118, calcule o atraso, no pior caso, dos circuitos combinacionais que o sistema suporta.

3.8 CIRCUITOS ESPECIAIS

Apesar da grande facilidade, da alta confiabilidade e do baixo custo dos circuitos integrados, o projetista de sistemas digitais ainda precisa recorrer aos transistores para realizar determinadas funções especiais. Da mesma forma que, apesar dos inúmeros tipos de circuitos em pastilhas *LSI* ou *MSI*, ainda necessitamos dos circuitos *SSI* com portas simples, para a implementação de inúmeros esquemas lógicos.

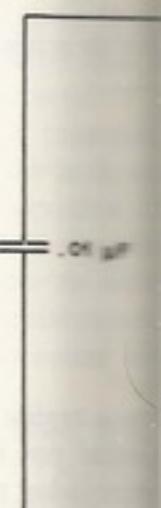
Dentre os circuitos que necessitam do transistor, ressaltamos os osciladores, *drivers*, fontes de alimentação etc. Sobre o assunto, existe uma grande quantidade de artigos em revistas, desde a década de 60 que ainda são atuais. Por isso, faremos uma breve abordagem desses assuntos, principalmente como exemplo do que dissemos. Todos os circuitos apresentados são reais.

a. Osciladores

Existem inúmeras maneiras de gerar oscilação com cristais. Vamos analisar os circuitos com transistores, que são os mais comuns. Vamos ver exemplos a seguir:

Colpits modificado

Muito usado em circuitos de exemplo.



Oscilador com cristal

A Fig. 3-120 mostra o circuito de um oscilador com cristal. Uma senóide é aplicada ao terminal de saída. O importante é que



Figura 3-120

a. Osciladores

Existem inúmeras formas de osciladores, os de grande estabilidade e precisão, geralmente com cristais ou os de estabilidade mais baixa com RC . Existem ainda os implementados com transistores e os com portas lógicas simples, como *NAND* ou *NOR*. Veja os exemplos a seguir.

Colpits modificado

Muito usado para osciladores de precisão. A frequência é dada pelo cristal. No circuito do exemplo, já existe o *driver* de saída (Fig. 3-119).

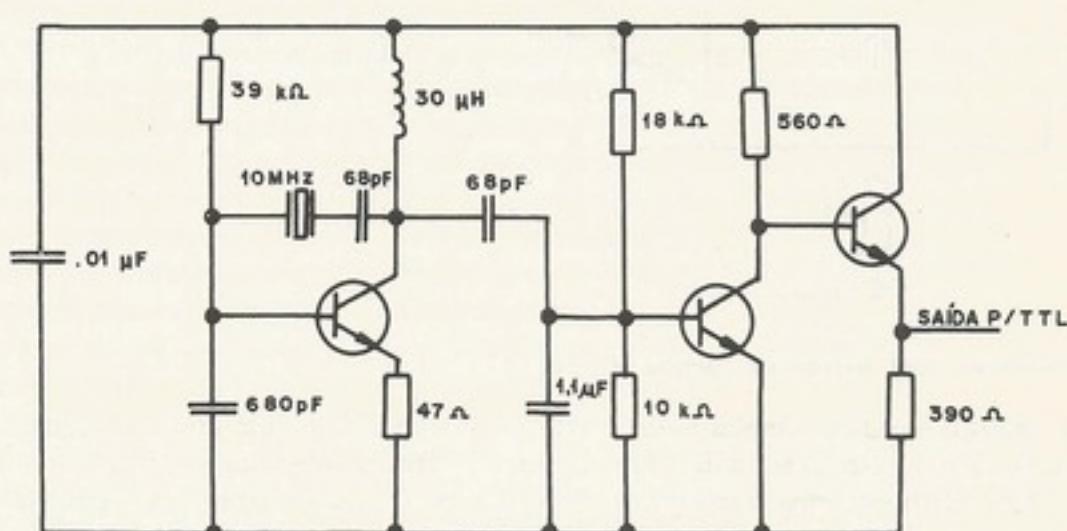


Figura 3-119. Colpits modificado

Oscilador com saída para TTL⁽¹³⁾

A Fig. 3-120 mostra um esquema de um circuito que oscila com apenas um transistor. Uma senóide é aplicada à porta do *FET* que comuta rapidamente entre o corte e a saturação. O importante é que ele se alimenta com a mesma fonte do *TTL*.

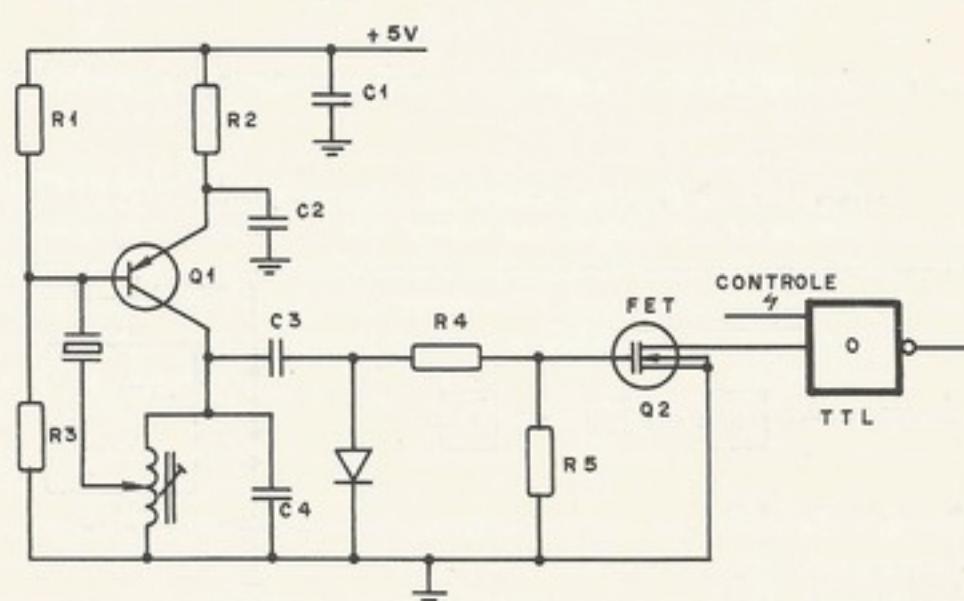


Figura 3-120. Oscilador com *FET* na saída para acionar integrados da família *TTL*

Astável de precisão⁽¹⁴⁾

A Fig. 3-121 mostra um multivibrador astável acoplado pelo emissor com um cristal para estabilizar a freqüência. É um circuito muito importante pela simplicidade e por não necessitar de indutores nem transformadores, apesar de apresentar níveis inconvenientes na saída.

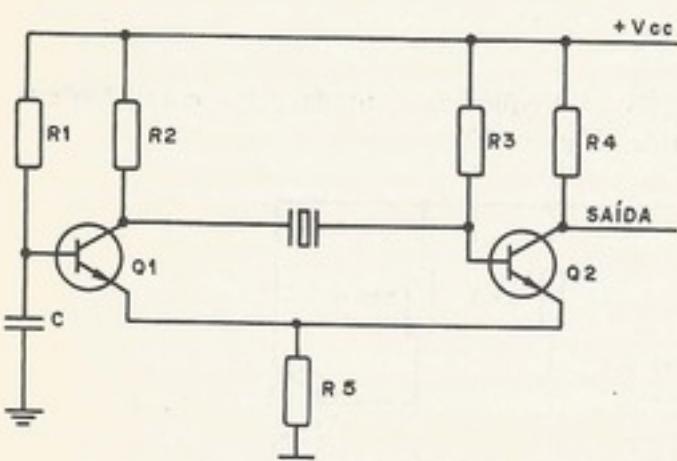


Figura 3-121. Astável de precisão

Osciladores com portas da família TTL

Um circuito bastante simples e útil é visto na Fig. 3-122⁽¹⁵⁾. Ele utiliza a região de comutação da porta NOR por oscilar. No circuito, podemos substituir o cristal por um *LC* em série. Um outro esquema que funciona em freqüências mais baixas controladas por *RC* é visto na Fig. 3-123⁽¹⁶⁾. Se fizermos $R_1 = R_2 = 470 \Omega$, variando C de 100 pF até 10 μ F, a freqüência irá variar de alguns hertz (Hz) até vários megahertz (MHz).

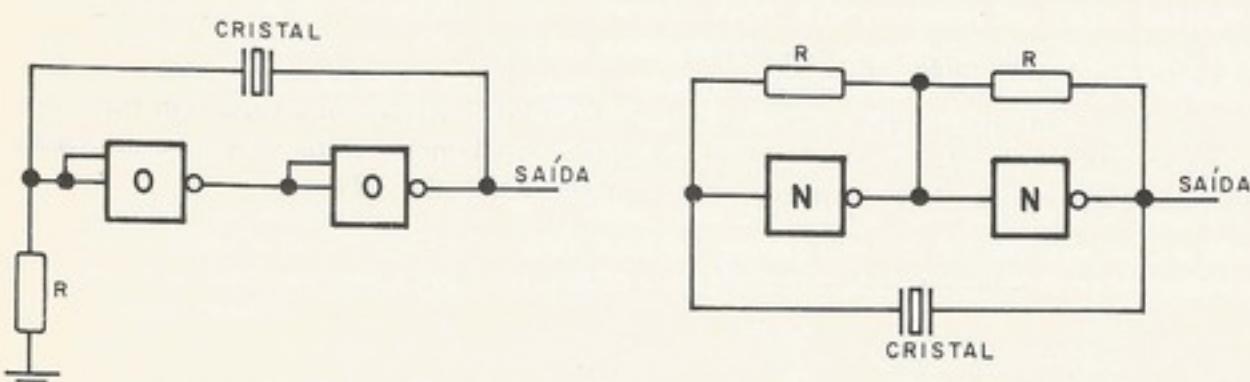
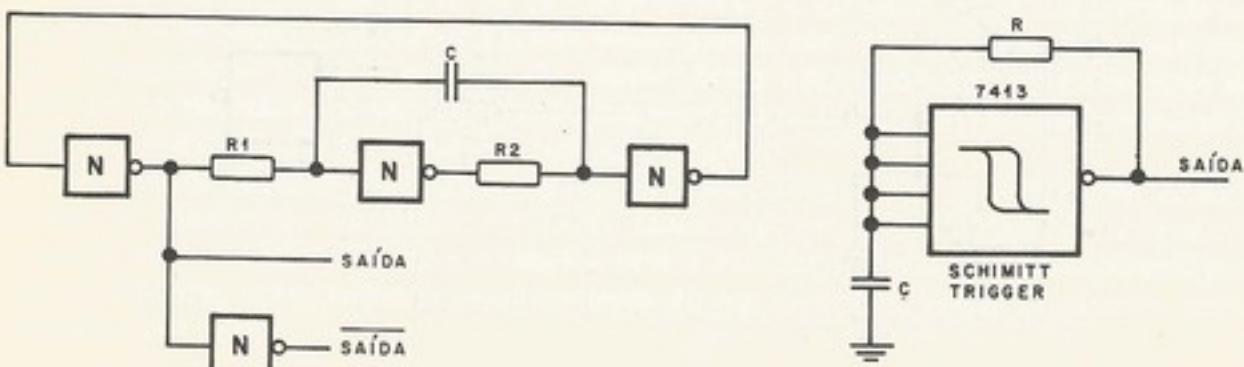


Figura 3-122. Osciladores a cristal com circuitos de família TTL

Figura 3-123. Osciladores *RC* com circuitos da família TTL

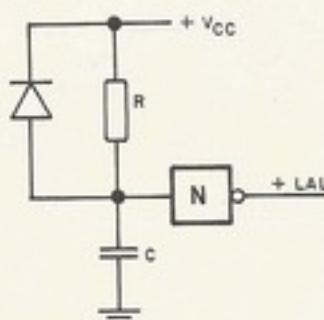
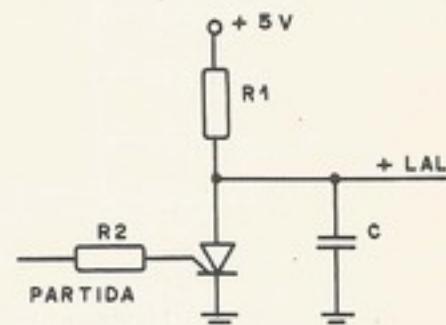
b. "Limpa ao ligar"

A maioria dos sistemas digitais relativamente sofisticados necessita de um sinal elétrico que fica no nível "1" apenas quando ligamos a máquina, e permanece no nível "0" durante todo o funcionamento. Esse sinal é usado para limpar ou disparar *flip-flops*, ou seja, é com ele que se impõe uma certa condição inicial ao sistema. Por isso, esse sinal recebe o nome de "limpa ao ligar" (*power on reset*) ou, simplesmente, *LAL* (*POR*).

Existem inúmeras maneiras de se gerar esse sinal, principalmente se o sistema tem várias fontes de alimentação que são ligadas sucessivamente, sempre numa certa ordem. Sistemas com uma só fonte de alimentação, ou com várias ligadas simultaneamente, a geração do *LAL* já é mais problemática. Apresentamos a seguir duas maneiras possíveis.

Primeira (Fig. 3-124). Quando ligamos o sistema, o capacitor C , descarregado, ainda impõe nível zero na entrada do *NOT*. À medida que o tempo passa, o capacitor se carrega e fica no nível "1" até que se desligue a máquina. Esse circuito apresenta dois inconvenientes sérios: o primeiro está na sua inércia, pois, se faltar energia elétrica por alguns instantes, o transitório da queda e retorno da força muda os *flip-flops*, e o *LAL* continua no nível zero, pois o capacitor continua carregado e, portanto, o sistema poderá iniciar um processamento caótico; o segundo inconveniente está na largura do *LAL*, que depende do capacitor. Normalmente ele seria da ordem de microssegundos, que é insuficiente para o caso de várias fontes alimentando o sistema e, atrasadas pelas cargas de seus eletrolíticos, algumas delas poderão entrar em regime alguns milissegundos depois de outras. O *LAL* não terá tempo de agir.

Segunda (Fig. 3-125). O circuito da figura já resolveu os dois problemas do esquema anterior. Para voltar a zero, ele utiliza o sinal de *partida* (*start*, *program-start* ou *run*), acionado pelo operador do sistema.

Figura 3-124. Geração do *LAL* com capacitorFigura 3-125. Geração do *LAL* com SCS

c. Entradas do sistema

As entradas do sistema que são ligadas aos dispositivos de entrada e saída são, geralmente, feitas de tal maneira que tensões acima de um certo valor (+3 V, por exemplo), são interpretadas como "1" e abaixo de outro (-3 V, por exemplo), como "0". Por isso, geralmente, nas entradas existem circuitos como o *disparador de Schmitt* (*Schmitt trigger*)⁽²²⁾ para ajustar os níveis.

d. "Driver"

Aqui, chamamos de *driver* a todo circuito que tem por finalidade aumentar a potência de um sinal elétrico pulsado, diminuindo sua impedância de saída ou aumentando a tensão. Muitas vezes, quando queremos apenas aumentar o *fan-out* de um sinal que está acionando uma grande quantidade de entradas, usamos integrados chamados *buffer*. Outras vezes, quando podemos perder um pouco de velocidade para ganhar em potência, usamos *drivers*.

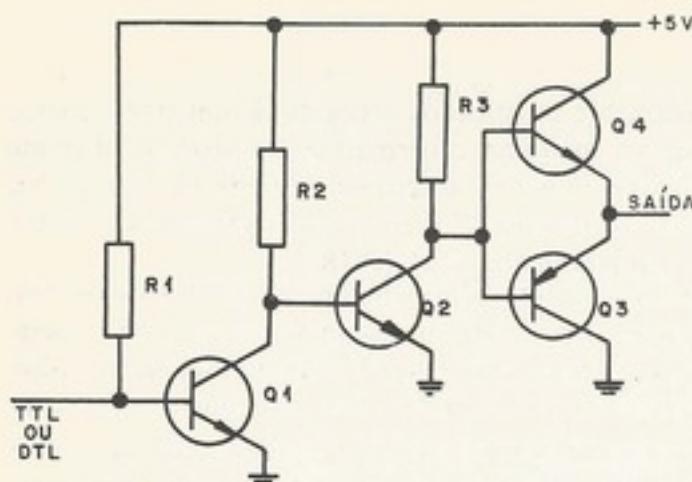


Figura 3-126. Driver do tipo push-pull

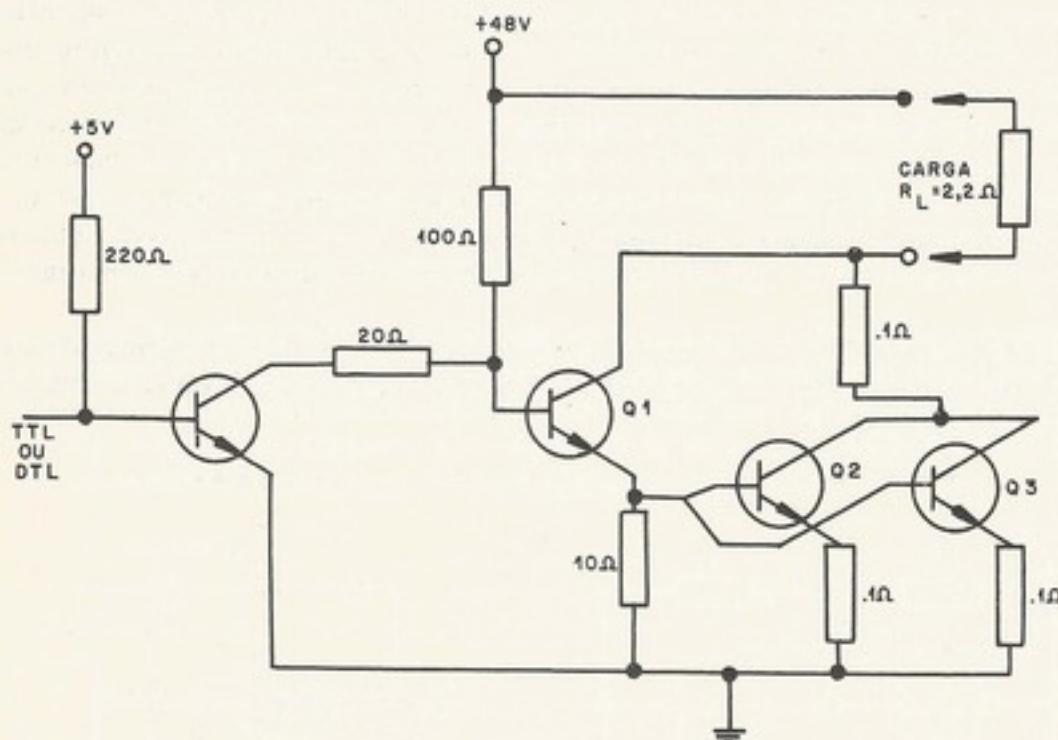


Figura 3-127. Driver de alta corrente

transistorizados. Normalmente, esses drivers são apenas estágios saturados. Outras vezes, são estágios push-pull ou totten-pole, conforme se vê na Fig. 3-126.

Para acionar núcleos magnéticos em memórias de computadores, ainda se usam drivers transistorizados, dada a alta corrente necessária. Na Fig. 3-128, mostramos o driver usado pela memória do Philips FI-21.

Quando se precisa de correntes excessivas, podem-se usar circuitos como o da Fig. 3-127⁽¹⁷⁾, que têm o inconveniente de terem o terra flutuante.

e. Fontes de alimentação

É outra situação onde o transistor é imprescindível. Principalmente na saída onde existem altas correntes e grandes dissipações de potência. Quanto à regulação, dispomos de integrados com ótima performance e enorme versatilidade.

Atualmente, para se ganhar espaço, as fontes são projetadas segundo o esquema da Fig. 3-129⁽¹⁸⁾. Retifica-se a rede diretamente e regula-se uma tensão alta (100 V, por exemplo), a "pré-regulagem"; depois, oscila-se numa freqüência de áudio (800 Hz é o mais recomendado⁽¹⁸⁾) para, a seguir, com transformadores e eletrolíticos menores, obter-se a tensão

desejada, que se consegue-se assim:

f. Outras fontes

A filosofia a considerar é:

É uma família:

⁽¹⁷⁾Marca

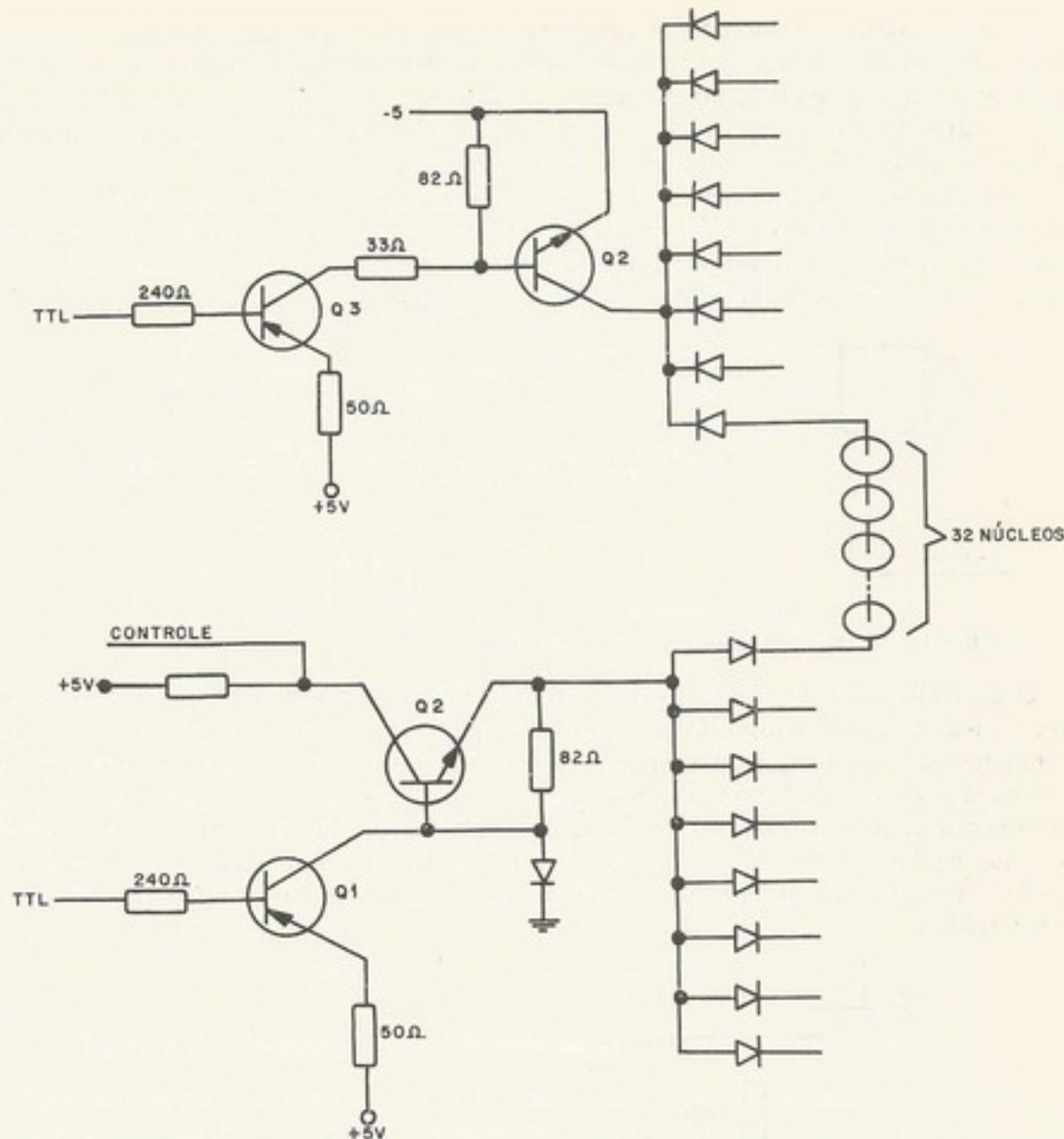


Figura 3-128. Driver de núcleos magnéticos

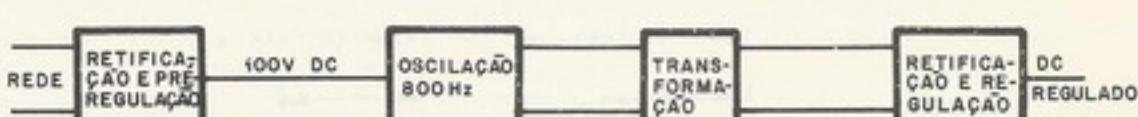


Figura 3-129. Esquema em bloco de uma fonte regulada de baixa tensão e alta corrente

desejada, que será, finalmente, regulada. Com uma eficiência de aproximadamente 75%, consegue-se uma fonte pequena, de alta regulação e isolada de ruídos da rede.

f. Outros circuitos

A filosofia de via (*bus*) e portas de seleção muda bastante quando o projetista começa a considerar a viabilidade do uso de uma família de integrados chamada *tri-state logic*⁽¹⁾. É uma família *TTL*, cuja saída, além dos estados “zero” e “um” em baixa impedância, tem

⁽¹⁾Marca registrada da National Semiconductor

um terceiro estado, considerado mesmo como aberto. Isso permite interligações de inúmeras saídas juntas e com o conveniente controle do terceiro estado, apenas uma saída fica funcionando (a que importa no momento), enquanto que as outras permanecem abertas. Essa é a filosofia da via. O terceiro estado é controlado por uma entrada específica que, quando no nível "1", torna a saída de alta impedância.

Consegue-se o mesmo efeito com circuitos integrados do tipo *expandable* da família TTL (veja a Fig. 3-130). Se ligarmos X e \bar{X} num mesmo ponto (Fig. 3-129) e se esse ponto estiver alto, o CI funcionará normalmente. Quando o ligamos à terra, a saída do CI fica aberta e temos alta impedância^[2].

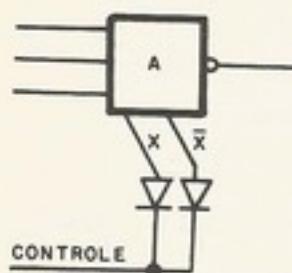


Figura 3-130. Circuito *expandable* ligado como *tri-state logic*

g. Circuitos de interface com contatos mecânicos

O problema surgido com o uso de contatos metálicos como entradas de um sistema digital existe há muito tempo. O problema é o "pulo" do contato móvel (*contact bounce*) de um potencial a outro, que dura o tempo de transição (*bounce time*). Normalmente o tempo de transição persiste alguns microsegundos, bem mais que um ciclo básico da máquina. Um exemplo de pulo do contato e sinal ruidoso que resulta durante o tempo de transição são vistos na Fig. 3-131. O resistor de 180Ω provê um caminho para a terra, necessário quando o contato está aberto. Antes de tudo, desejamos suprimir o ruído para "limpar" o sinal ruidoso.

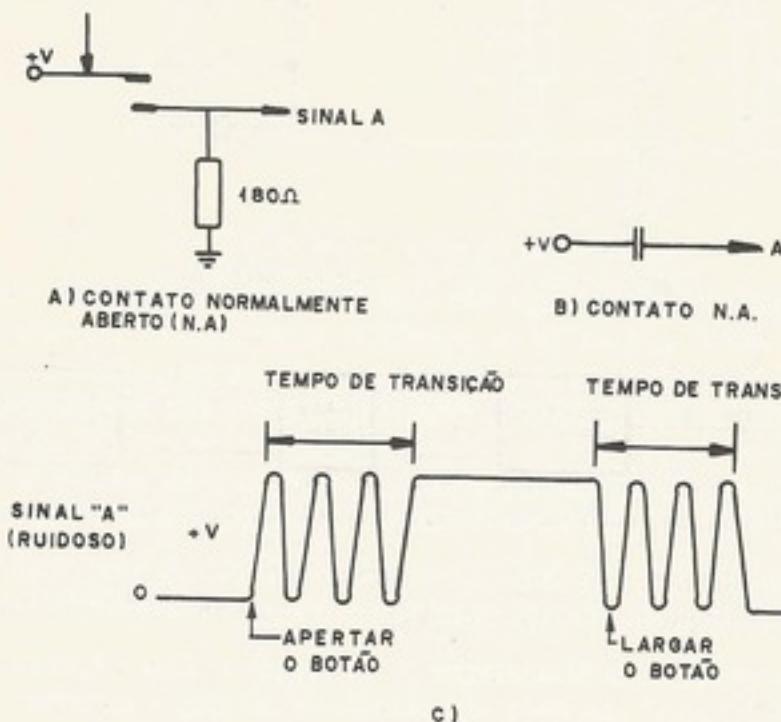


Figura 3-131. (a) Contato mecânico N.A.; (b) modelo do contato N.A.; (c) gráfico de *timing* do sinal ruidoso na borda de ataque e na borda de fuga

^[2]Essa aplicação do *expandable* foi concebida pelo engenheiro Célio Y. Ikeda, da Escola Politécnica da USP

Uma solução é usar o terminal X do CI (que é o terminal $N.A.$) e outro terminal para o antípulo). Uma conexão direta entre os dois é ruidosa (veja a Fig. 3-132).

Figura 3-132

Faltando os detalhes, Teremos de avaliar o efeito de atraso ou que o pior caso de atraso é o gráfico de *timing*.

O G. Fontaine, quando a tecnologia

O primeiro circuito, quando fornecerá pulsos de atraso do circuito.

Uma vez que o cronismo deste circuito é chamado *TB* (tempo de bate), passar um pulso não pode ser nem menor, nem maior. O circuito usa flip-flop de "limpar".

Um outra maneira é usar um relógio central, que gera o pulso à borda de ataque.

configurações de inúmeras saídas fica comuns abertas. Isso especifica que,

disponível da família TTL e se esse ponto de saída do CI fica

on-state logic

de um sistema de um sistema (contact bounce) normalmente o tempo da máquina. tempo de transição a terra, necessário para "limpar"

Uma solução simples, se os contatos são do tipo de dois pólos, um normalmente aberto (N.A.) e outro normalmente fechado (N.F.), é usar um flip-flop (chamado de *debounce* ou antipulo). Uma característica de contatos de dois pólos é que só a borda de ataque do sinal é ruidosa (veja a Fig. 3-132).

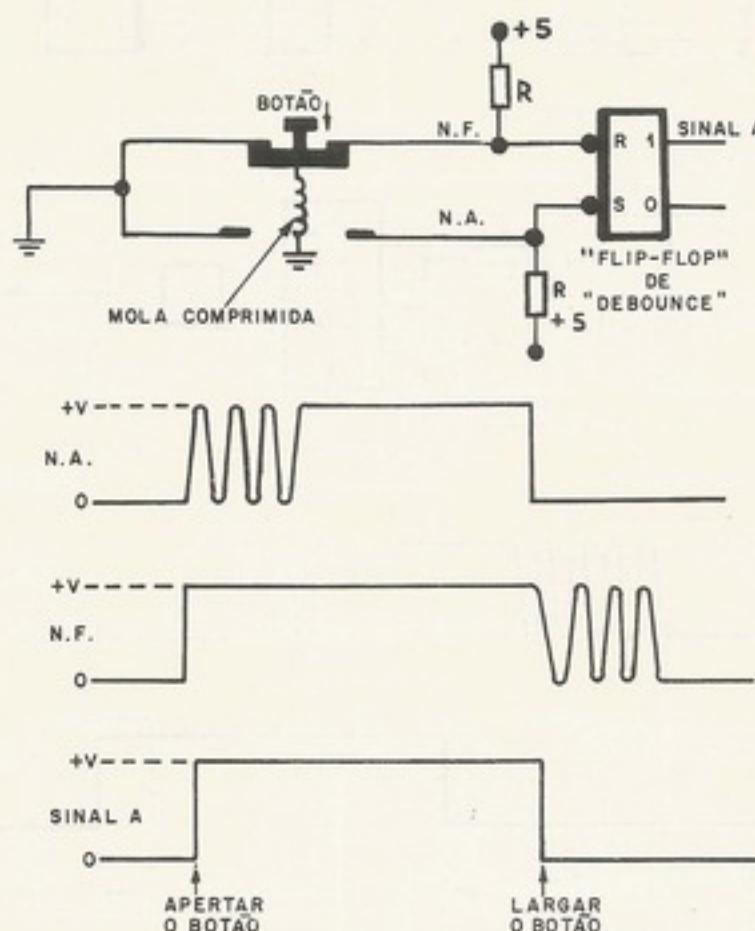


Figura 3-132. Limpando o ruído de sinais de contato com flip-flop de debounce

Faltando os dois contatos N.A. e N.F., não haverá solução elegante para o problema. Teremos de avaliar o pior caso da duração do transitório (*bounce time*) e prover um elemento de atraso ou um monoestável para esperar o fim do ruído. Por exemplo, supomos que o pior caso de um transitório é 5 ms. Então, na Fig. 3-133, aparecem duas soluções com o gráfico de *timing* (o segundo circuito é de Wood⁽¹⁹⁾).

O G. Fontaine⁽²⁰⁾, mostra dois circuitos para suprimir o ruído do "pulo" de contatos, quando a tecnologia é de TTL.

O primeiro circuito fornece pulsos positivos (como no caso de contatos N.A.) e o segundo fornece pulsos negativos (como no caso de contatos N.F.). Veja a Fig. 3-134. O tempo de atraso do circuito é da ordem de 20 ms.

Uma vez que o sinal proveniente do contato está limpo, um outro problema é o sincronismo deste com o ciclo da máquina. Vamos supor que exista um sinal do relógio central chamado *TB* (tempo *B*), que dura 250 ns. Cada vez que o contato *A* é acionado, queremos passar um pulso simples de 250 ns, sincronizado com *TB*. A duração do pulso não deve ser nem menor, nem maior que 250 ns. O circuito da Fig. 3-135, de Sepe⁽²¹⁾, faz essa tarefa. O circuito usa flip-flop tipo *JK*, sensível à borda de fuga, com uma entrada sensível ao nível de "limpar".

Um outra maneira de obter-se um sinal por um ciclo simples necessita dois sinais do relógio central, que chamaremos de *T₁* e *T₂*. Tem dois flip-flops auxiliares, tipo *D*, sensíveis à borda de ataque, "passo 1" e "passo 2". Para o circuito e o *timing*, veja a Fig. 3-136.

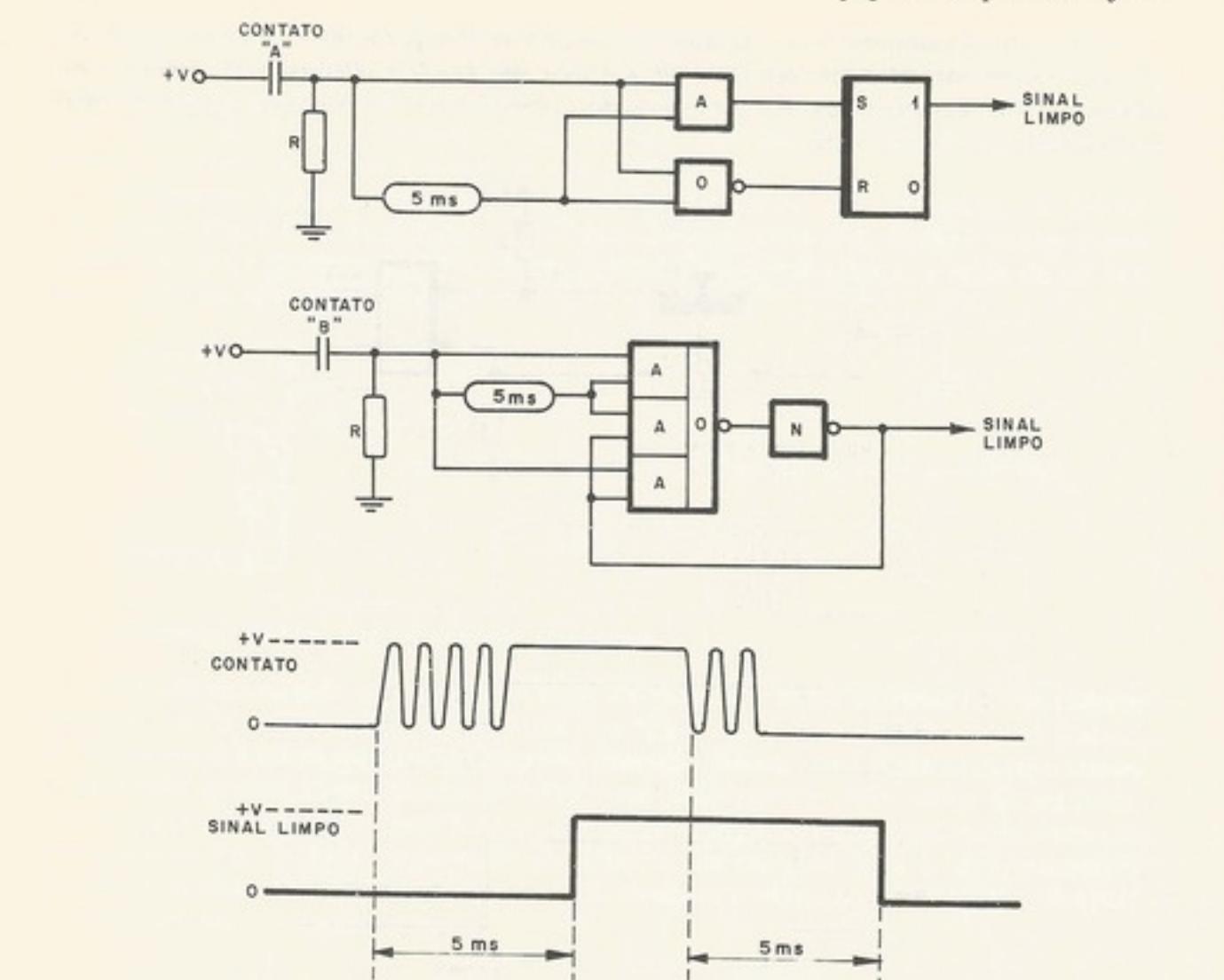


Figura 3-133. A aplicação de atrasos e flip-flops para suprimir ruído

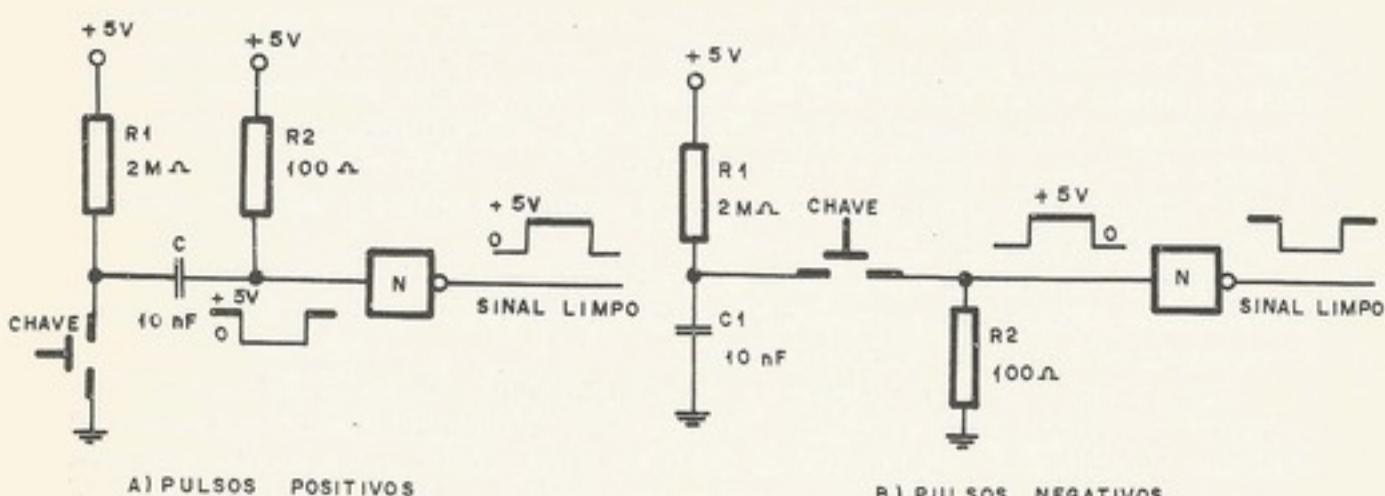
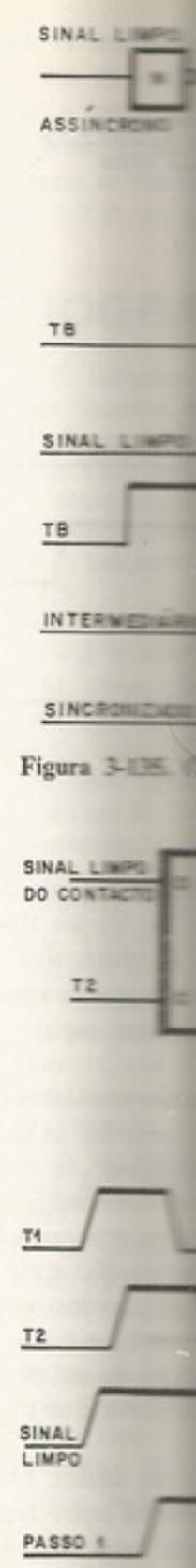


Figura 3-134. Circuitos RC para suprimir ruidos de contato

Um outro circuito dessa espécie⁽⁵⁾, chamado *gated oscillator*, deixa passar pulsos completos desde que o contato esteja fechado, ao contrário dos circuitos anteriores, em que só passa o primeiro pulso (veja a Fig. 3-137). Uma propriedade importante do circuito é que os pulsos de saída não são curtos e nem compridos demais.



Figura

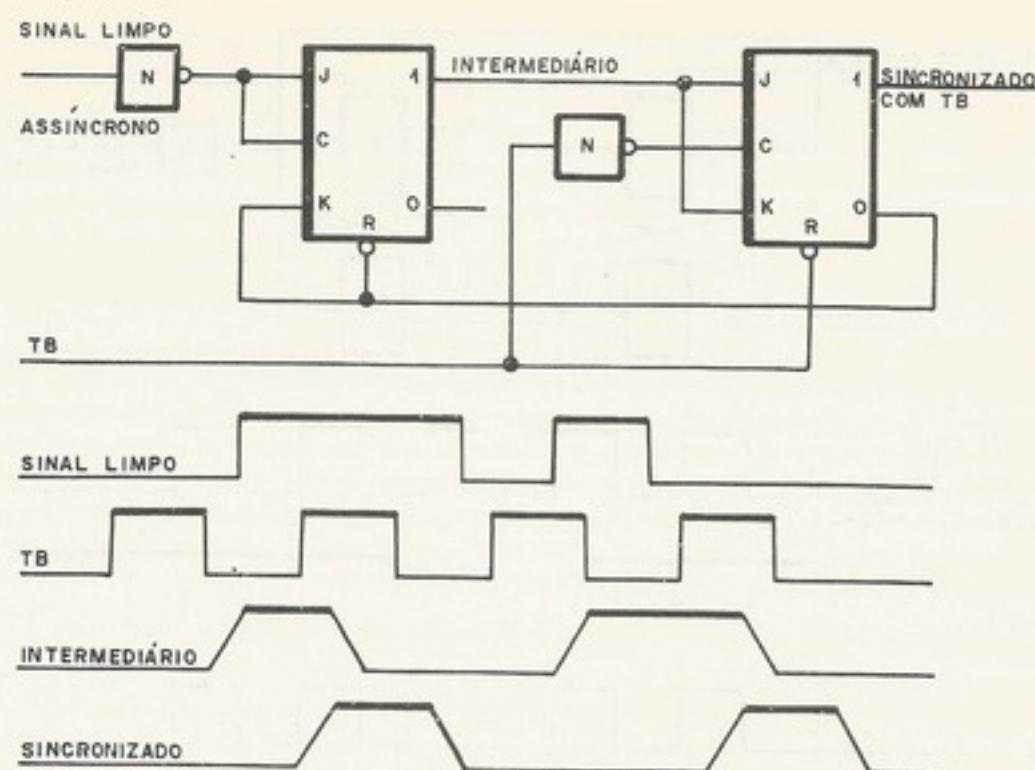


Figura 3-135. Circuito para sincronizar um sinal assíncrono com um sinal periódico

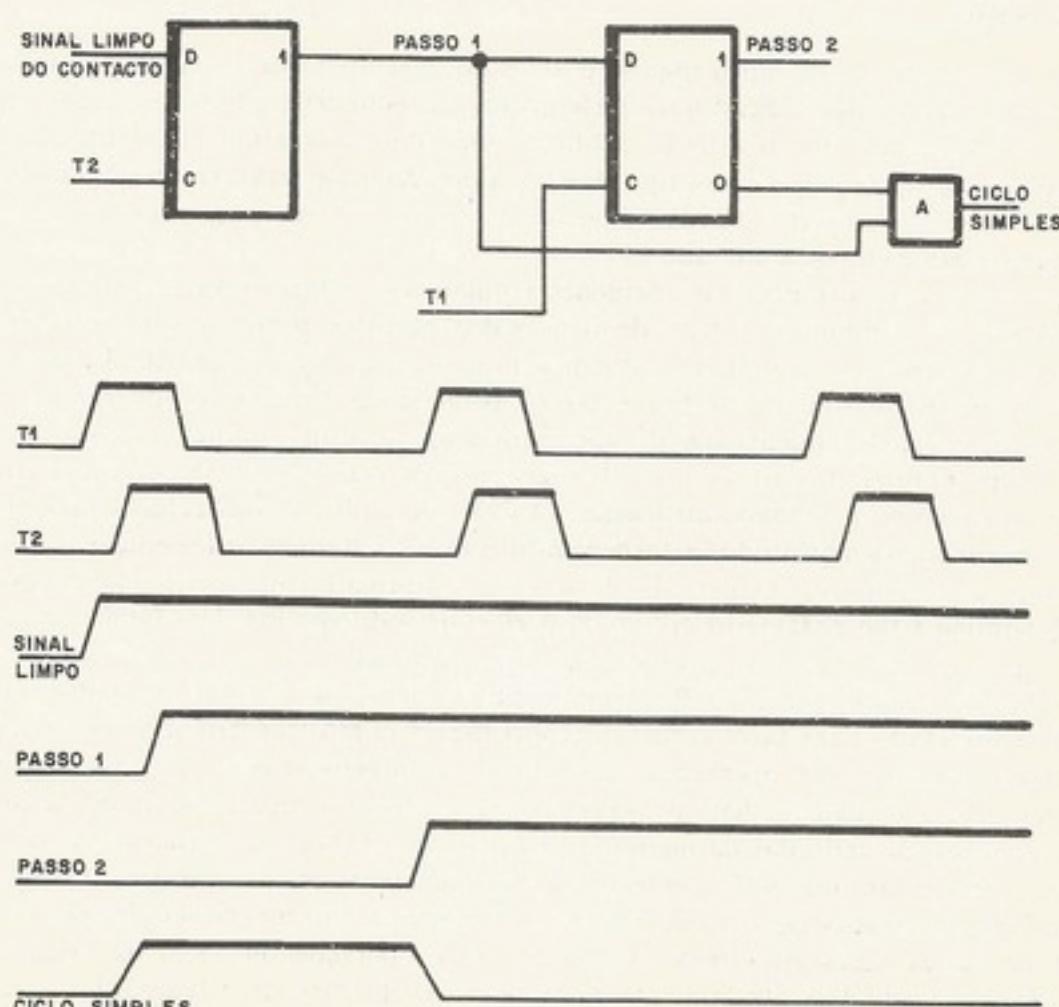
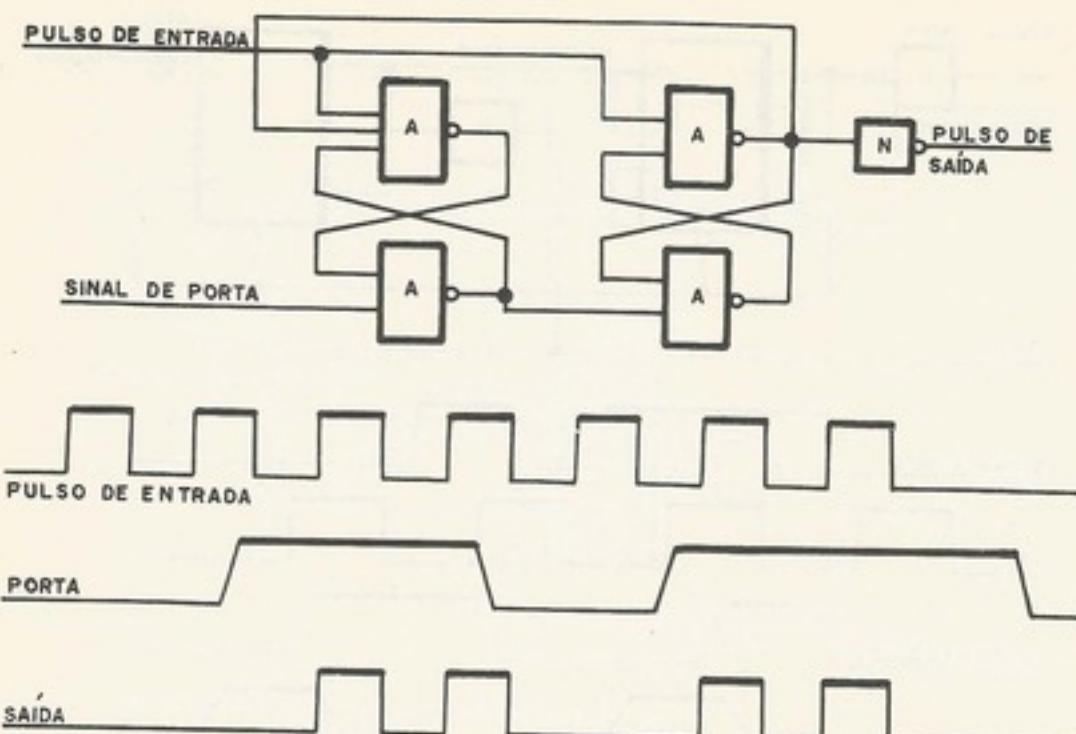


Figura 3-136. Um circuito para gerar um sinal de ciclo simples

Figura 3-137. O circuito *gated oscillator*

3.9 SUMÁRIO

Foram estudados os circuitos lógicos básicos do engenheiro de sistemas digitais, alguns sem armazenamento (decodificadores) e elementos de memória, como os vários tipos de *flip-flop*. É importante que o artesão conheça bem suas ferramentas, e vimos que existe uma variedade de *flip-flops* (*set-reset*, tipo *D*, *PH*, e tipo *JK*) tanto como os modos de operação (sensível ao nível e sensível à borda). Com os *flip-flops*, é possível projetar registradores, registradores deslocadores e contadores.

Ressaltamos a importância das operações aritméticas e cobrimos várias implementações de somadores. Para diminuir o efeito de atrasos dos circuitos, foram mostradas as técnicas de "vai um antecipado". A unidade aritmética normalmente tem a capacidade de executar as microoperações de somar, subtrair e fazer operações booleanas nos operandos.

A transferência de informações de um registrador ao outro é uma microoperação de fluxo de dados. O fluxo de dados consta de registradores, vias e portas de seleção, e circuitos combinacionais como a unidade aritmética. O fluxo de dados é construído como um laço em que tipicamente os registradores fornecem operandos à unidade aritmética e o resultado é devolvido ao registrador. O período desse ciclo é normalmente o ciclo da máquina. O ciclo da máquina é um parâmetro crítico da *performance* do sistema. Um fator que também influiu no ciclo da máquina é o ciclo da memória principal.

O processo de projetar a *UCP* normalmente começa com o fluxo de dados. O fluxo de dados é projetado para fazer certas microoperações como transferências entre registradores, deslocamentos e as operações aritméticas. A implementação de uma arquitetura através desse fluxo de dados é feita pelo processo de ordenar as microoperações, de maneira que as instruções são retiradas da memória principal (ciclos *I*) e executadas (ciclos *E*). Cada código de operação da instrução controla uma seqüência de microoperações. Essas microoperações são encaixadas no ciclo da máquina. O relógio central é projetado para fornecer os sinais de *timing* necessários para ordenar as microoperações no ciclo da máquina. Os atrasos (pior caso) no laço do fluxo de dados não devem superar o tempo desse ciclo.

O fluxo de dados, uma parte controlada como as marionetes, precisa receber nos seus fios de controle as formas de ondas corretas. Esses sinais são provenientes da unidade de

controle. A implementação pode ser feita ou por um microprocessador ou por um microcontrolador.

Além dos circuitos de controle, a unidade de processamento não poderia se comunicar com as unidades de memória e de interface da *UCP*, ou vice-versa.

BIBLIOGRAFIA

- (1) Glen G. Langdon, Jr., "Digital Computer Progress National Bureau of Standards," *Review of Research*, N.º 1, p. 121, novembro de 1969.
- (2) H. Hellerman, "Digital Computer Progress," *Review of Research*, N.º 1, p. 121, novembro de 1969.
- (3) Yachan Chu, *Microprocessor Handbook*, McGraw-Hill, 1970.
- (4) G. A. Maley, *Microprocessor Handbook*, McGraw-Hill, 1970.
- (5) G. A. Maley, *Microprocessor Handbook*, McGraw-Hill, 1970.
- (6) Glen G. Langdon, Jr., "Digital Computer Progress," *Review of Research*, N.º 1, p. 121, novembro de 1970.
- (7) Glen G. Langdon, Jr., "Digital Computer Progress," *Review of Research*, N.º 1, p. 121, novembro de 1970.
- (8) A. J. M. Dorn, "Digital Computer Progress," *Review of Research*, N.º 1, p. 121, novembro de 1970.
- (9) Glen G. Langdon, Jr., "Digital Computer Progress," *Review of Research*, N.º 1, p. 121, novembro de 1970.
- (10) F. F. Sellers, Jr., "Digital Computer Progress," *Review of Research*, N.º 1, p. 121, novembro de 1970.
- (11) L. Naselsky, *IBM System/360*, IBM, 1967.
- (12) S. S. Husson, *IBM System/360*, IBM, 1967.
- (13) C. S. L. Keay e R. E. Moore, "IBM System/360," *Computer*, fevereiro de 1968.
- (14) G. W. Harrison, "Digital Computer Progress," *Review of Research*, N.º 1, p. 121, novembro de 1970.
- (15) M. Murata, et al., "Digital Computer Progress," *Review of Research*, N.º 1, p. 121, novembro de 1970.
- (16) O. Q. Flinck Jr., "Digital Computer Progress," *Review of Research*, N.º 1, p. 121, novembro de 1970.
- (17) Lynn S. Bell, "Digital Computer Progress," *Review of Research*, N.º 1, p. 121, novembro de 1968.
- (18) R. D. Grawford, "Digital Computer Progress," *Review of Research*, N.º 1, p. 121, novembro de 1968.
- (19) P. Wood, *Swing Electronics*, 1968.
- (20) G. Fontaine, "Digital Computer Progress," *Review of Research*, N.º 1, p. 121, novembro de 1968.
- (21) R. Sepe, "How to Design a Microprocessor," *Computer*, junho de 1968.
- (22) J. Millman e E. Well, "Digital Computer Progress," *Review of Research*, N.º 1, p. 121, novembro de 1968.
- (23) W. Ware, *Digital Computer Progress*, N.º 1, p. 121, novembro de 1968.

controle. A implementação dessa unidade pode ser ou em *hard-wire* (isto é, em circuitos) ou por um microprograma armazenado numa memória de controle. Para melhor compreender o processo de projeto lógico, estudaremos um exemplo no Cap. 5.

Além dos circuitos lógicos da *UCP*, existem vários circuitos especiais, sem o que a *UCP* não poderia se comunicar com o mundo fora de si. Encontram-se esses circuitos nas interfaces da *UCP*, ou em subsistemas e dispositivos periféricos.

BIBLIOGRAFIA

- (1) Glen G. Langdon, Jr., "Conseqüências da tecnologia de *LSI* no projeto lógico", Anais do IV Congresso Nacional de Processamento de Dados, 11-15, São Paulo, outubro de 1971
- (2) H. Hellerman, *Digital Computer System Principles*, McGraw-Hill, New York, 1967
- (3) Yachan Chu, *Introduction to Computer Organization*, Prentice-Hall, Inc., New Jersey, 1970
- (4) G. A. Maley, et al., *Introduction to Digital Computers*, Prentice-Hall, New Jersey, 1968
- (5) G. A. Maley, *Manual of Logic Circuits*, Prentice-Hall, Inc., New Jersey, 1970
- (6) Glen G. Langdon, Jr., "A Survey of Counter Design Techniques", projeto de computador, outubro de 1970
- (7) Glen G. Langdon, Jr., "Some Definitions and Practices in Logic Design", SDD technical report, IBM, TR 01-497, Endicott, 1969
- (8) A. J. M. Dorn, "Decades and other Counters using Integrated Circuits FCJ 101 and FCJ 111", Philips, application notes 846, 1969
- (9) Glen G. Langdon, Jr., "Logic Design", SDD technical report, IBM, TR-01-1396, TR-01-1370 e TR-01-1469, Endicott, 1971
- (10) F. F. Sellers, Jr., et al., *Error Detecting Logic for Digital Computers*, McGraw-Hill, New York, 1968
- (11) L. Nashelsky, *Digital Computer Theory*, John Wiley, Série "Electronic Engineering Technology", 1967
- (12) S. S. Husson, *Microprogramming, Principles and Practices*, Prentice-Hall, New Jersey, 1970
- (13) C. S. L. Keay e I. C. Graham, "Fet links oscillator to TTL Circuit", *Electronics*, Vol. 42, n.º 4, p. 97, fevereiro de 1969
- (14) G. W. Harrison, "Crystal Gives Precision to a Stable Multivibrator", *Electronics*, Vol. 41, n.º 43, p. 121, novembro de 1969
- (15) M. Murata, et al., "Analysis of an Oscillator consisting of Digital Integrated Circuits", *IEEE Journal of Solid State Circuits*, Vol. SC-5, n.º 4, pp. 165-168, agosto de 1970
- (16) O. Q. Flint Jr., "Free Running Multivibrator IS Made With a Nand Gate", *Electronics*, Vol. 42, n.º 1, p. 95, janeiro de 1969
- (17) Lynn S. Bell, "High Current Switch is Driven by an IC", *Electronics*, Vol. 41, n.º 23, p. 120, novembro de 1968
- (18) R. D. Grawford e G. Justice, "A Bantam Power Supply for a Minicomputer", *Hewlett-Packard Journal*, pp. 13-15, outubro de 1971
- (19) P. Wood, *Switching Theory*, McGraw-Hill, pp. 358-361, 1968
- (20) G. Fontaine, "Gate Suppresses Pulses from Switch Contact Bounces", *Electronics*, Vol. 44, n.º 6, p. 76, março de 1971
- (21) R. Sepe, "How to Synchronize Pulses with only two Flip-Flops", *The Electronics Engineer*, pp. 58-60, junho de 1968
- (22) J. Millman e H. Taub, "Pulse Digital and Switching Waveforms", McGraw-Hill, pp. 389-394, 1965
- (23) W. Ware, *Digital Computer Technology and Design*, Vol. II, John Wiley and Sons, New York, 1963

capítulo 4

MEMÓRIA E ARMAZENAMENTO

4.1 INTRODUÇÃO

A unidade central de processamento retira e executa instruções da memória principal. O volante real do sistema é o programa na memória principal. Os operandos das instruções são identificados pela própria instrução, tipicamente através de um campo de endereço que vem junto com a instrução. Os operandos são guardados na mesma memória que as instruções. Assim, instruções podem operar nas outras instruções como se fossem operandos. O primeiro computador cuja arquitetura permitia isso foi o computador *IAS*, construído no Institute for Advanced Study, Princeton, New Jersey. Por isso, computadores em que instruções e dados podem ser confundidos são da classe Princeton. A maioria das arquiteturas da segunda e terceira gerações pertencem a classe Princeton. No primeiro computador, o Harvard Mark I (também chamado de IBM Selective Controlled Calculator), as instruções foram armazenadas em fitas perfuradas e os operandos foram armazenados numa matriz de registradores de relé. Era impossível confundir instruções com dados. Máquinas desse tipo são da classe Harvard.

R. M. Mead⁽¹⁾ (como outros) observou que existem duas fronteiras ou interfaces naturais num sistema de computação. A primeira é a interface homem-máquina. Ao lado da máquina, temos acesso eletrônico ao programa, mas, no outro lado, fica a informação, que só o homem pode fornecer. Geralmente, o homem fornece essa informação à memória principal através de um terminal ou cartões perfurados. Também, relativamente à saída verdadeira de informações do sistema, é normalmente o homem quem recebe e faz alguma coisa com a informação. (Como exceção, temos saídas que controlam um processo). Então, informações transferidas por essa fronteira podem ser chamadas de "entrada/saída verdadeira". A segunda interface natural é interna ao sistema. De um lado está a memória primária (*MP*), cujo conteúdo pode ser endereçado através do endereço efetivo da instrução, que a *UCP* interpreta. Do outro lado dessa interface está o armazenamento secundário (*AS*), ou memória auxiliar (geralmente dispositivos eletromecânicos) cujas informações têm de ser transferidas à memória primária antes que possam ser processadas. Essa transferência normalmente toma lugar através de comandos específicos de entrada/saída, e a transferência fica sob o controle de rotinas dos sistema operacional ou supervisor.

A memória principal (primária) é, então, vista como o ponto focal do sistema. As entradas e saídas verdadeiras passam pela memória principal. Também, antes de serem utilizadas, informações (tanto dados como programa) residentes no armazenamento secundário, são transferidas à memória primária. É da memória primária que a *UCP* recebe as instruções e operandos. O diagrama do sistema é mostrado na Fig. 4-1.

4.2 O "BANDWIDTH" E SUAS IMPLICAÇÕES

Dois parâmetros importantes relativos à memória principal são o *bandwidth*⁽³⁾ e o *tempo de acesso*. O *bandwidth* é a medida da quantidade de palavras (ou bytes) que pode

⁽³⁾O sentido de *bandwidth* aqui não é "largura da faixa", que se encontra em comunicações, mas é uma medida da velocidade de transmissão de informações

ARMAZENAMENTO
SECUNDÁRIO
(AS)

ser transferida em um determinado tempo, de ciclo de 25 ns.
800 000 bytes por segundo.

Em geral o acesso é mais lento que o de memória primária.
Existe uma grande diferença entre o tempo de acesso da memória primária e o de memória auxiliar. Normalmente a memória auxiliar é mais rápida que a memória primária. O armazenamento secundário é usado para armazenar informações demoradas que não precisam ser processadas imediatamente.

O acesso à memória auxiliar é mais lento que à memória primária. Se os dados forem transferidos de uma memória auxiliar para a memória primária, o tempo de acesso é menor. O bloco de dados é transferido para a memória primária a uma velocidade de 1 byte por segundo. Isso também é verdadeiro para a memória auxiliar, o acesso é mais lento que a memória primária, mas o tempo de acesso é menor.

Em sistemas de processamento de dados, a memória auxiliar é usada para armazenar dados que são utilizados frequentemente. Enquanto o sistema opera, os dados são transferidos para a memória primária, que é mais rápida. Quando os dados são transferidos de volta para a memória auxiliar, o tempo de acesso é maior, mas o custo é menor.

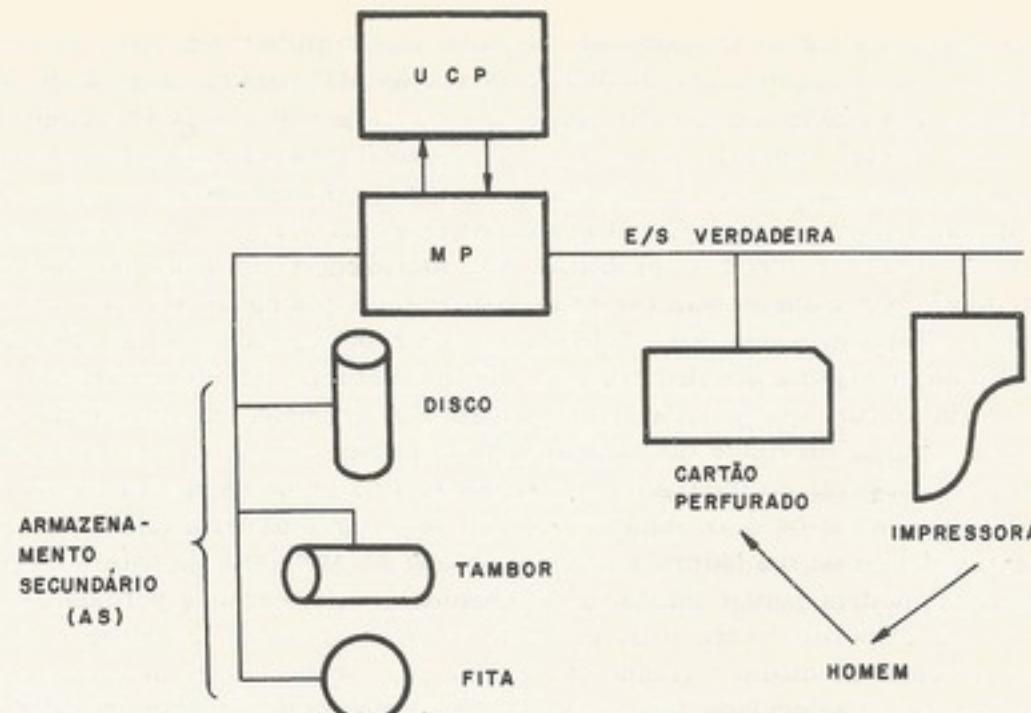


Figura 4-1. Armazenamento num sistema de computação

ser transferida através da interface num certo prazo. Por exemplo, uma memória principal de ciclo de $2,5 \mu s$ e largura de 2 bytes, possui um *bandwidth* de 2 bytes por $2,5 \mu s$, ou seja, 800 000 bytes por segundo. O tempo de acesso à memória pode variar muito entre sistemas. Em geral o acesso à memória principal é da ordem de 200 ns a $2 \mu s$.

Existe uma variedade de dispositivos para armazenamento secundário como, por exemplo, fita magnética, disco de cabeçote móvel, e tambor (ou disco de cabeçote fixo). Normalmente o dispositivo com o mínimo custo por bit é o mais lento. Então a maioria da informação no armazenamento secundário é encontrada em veículos mais econômicos. O armazenamento secundário é freqüentemente visto como uma *hierarquia*, na qual as informações dos níveis mais econômicos são colocadas nos níveis de acesso mais veloz, para processamento.

O acesso a informações no armazenamento secundário é geralmente mais complicado do que à memória primária, pois a informação tem de atravessar uma fronteira natural. Se os dados desejados não estão na memória primária, então eles não são acessíveis às instruções, e só se pode ter acesso à informação após um tempo longo comparado com a velocidade de execução de instruções. O acesso geralmente é feito através de programas especializados para entrada/saída. Para ler, tipicamente o usuário tem espaço chamado *buffer*, na memória principal, reservado para receber um "bloco" de palavras do dispositivo. O "tempo de transporte" consta do tempo de acesso ao dispositivo e do tempo de transferir o bloco. O termo *bandwidth*, quando aplicado a um dispositivo da memória auxiliar, indica a velocidade de transferência de bytes em seguida, uma vez que o acesso foi feito ao primeiro byte. Isso também é chamado de *velocidade de dados (data rate)*. Por exemplo, para fita magnética, o acesso ao primeiro byte de um registro é tipicamente de 5 ms, devido ao intervalo entre registros (*interrecord gap*), mas o *bandwidth* está tipicamente entre 30 000 e 96 000 bytes por segundo.

Em sistemas simples, o acesso aos dispositivos é feito através de instruções, freqüentemente a uma palavra por vez. Com essas instruções, todo o processamento na UCP pára enquanto o sistema aguarda a palavra. Para evitar esse tempo ocioso da UCP, desenvolveram-se esquemas, chamados canais, em que a entrada/saída poderia operar assincronamente com o processamento. Com esquemas desse tipo, é possível sobrepor (*overlap*) processamento com acesso ao armazenamento secundário.

Normalmente, a *UCP* dá um comando ao canal para transferir um bloco de um certo número de palavras, começando em um dado endereço da *MP*, para tal dispositivo. Quando o canal termina a transferência, ele interrompe a *UCP* (veja a Fig. 4-2). Interrompendo-se a *UCP*, o canal “rouba” um ciclo de memória (*cycle-steal*) para a transferência da palavra.

Com um ou mais canais, a *UCP* não é mais a única utilizadora da *MP* porque, durante operações de saída (ou entrada), o canal precisa de ciclos da *MP* para retirar (ou guardar) palavras. Então pode acontecer o problema de “interferência” (*memory interference*), situações em que a *UCP* e um ou mais canais desejam um ciclo da memória ao mesmo tempo. Normalmente o dispositivo tem prioridade sobre a *UCP* por causa do perigo do problema chamado *overrun*. A maioria dos dispositivos é eletromecânica, exigindo, então, movimento mecânico de um veículo que guarda a informação. Por exemplo, uma fita magnética ou disco magnético passa em frente do cabeçote leitura/escrita com uma certa velocidade, e só pode parar ou acelerar no intervalo entre registros. Isso implica que as palavras provenientes do registro na fita ou disco durante a leitura vêm com uma certa velocidade. Se uma dessas palavras, depois de sua leitura, não for guardada na *MP* antes da leitura da palavra seguinte, o canal poderá perder informações. Quando a informação é perdida assim, ou seja, quando um dispositivo precisa de um ciclo de memória e não o recebe, é chamado *overrun*. Em sistemas de médio e grande portes, isso representa uma limitação ao número de dispositivos de alta velocidade, ligados ao sistema, e que podem transferir dados simultaneamente.

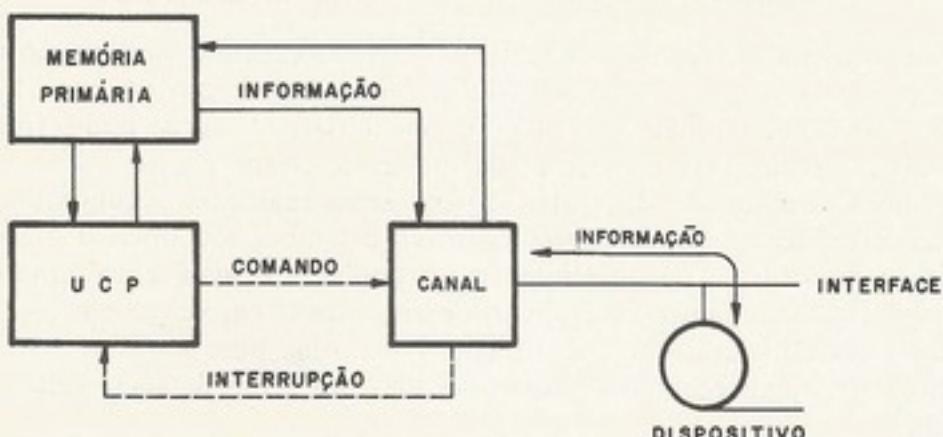


Figura 4-2. O funcionamento do canal independente da UCP para sobrepor processamento com entrada-saída

O "tráfego" que passa através da interface *MP-AS* é um parâmetro importante da *performance* do sistema. Heineck e Kronian⁽²⁾ citam estudos de sistemas de IBM 7090, para mostrar que mais ou menos dez instruções são executadas para cada byte (8 bits) de informação que atravessa essa interface. Assim, o projetista do sistema da *E/S* pode avaliar grosseiramente quantos bytes por segundo, em média o esquema de *E/S* vai transferir. Usando-se essa figura empírica no IBM S/360 modelo 30, que, segundo Brooks e Iverson⁽³⁾, executa uma instrução (em média) cada 30 μ s, o modelo 30 deve tratar com um byte de *E/S* cada 300 μ s.

4.3 MEMÓRIA PRIMÁRIA

A memória primária é o coração de um computador. É a memória primária que armazena os programas que dirigem o sistema. Sem um programa, o sistema nada faria. Também as características da memória primária, o tempo de ciclo e a capacidade, são parâmetros importantes na *performance* do sistema inteiro.

Na maioria dos sistemas, as memórias primária e principal se confundem. A memória primária é aquela *endereçada* pela UCP, enquanto que a principal (contida na primária) é a parte da memória primária de acesso direto e rápido.

memória e esquecimento

A unidade da palavra da memória é uma palavra: um endereço é a localização de uma palavra.

O endereço é de informação, ou informação, ou

Na forma maiuscula de endereços (RE), Os bits da palavra de dados (RD), P

Hoje em dia, foram usadas também é um apagará. Memória veículos magnet

4.4 MEMÓRIA

A tecnologia dos computadores é no sentido broad como sendo "o" de Fig. 4-3.

1

Para se escapar deixa o núcleo para ler o comando no estado "V" (+E) estiver no estado "A" de sentido. Esse sensor e armazena de ler, o bit que os núcleos são do tipo

de um certo tempo. Quando se compõe-se da palavra, durante esse tempo (ou guardar) *inference*, si mesmo tempo. do problema de movimento magnética ou de velocidade, e palavras provem. Se uma parte da palavra é assim, ou é chamado ao número de dados simul-

A unidade básica de informação é o *bit*. Um conjunto de *bits* forma uma palavra. A palavra da memória principal é geralmente a quantidade de *bits* retirada por acesso. O acesso a uma palavra armazenada na memória principal é efetuada através do seu endereço. O endereço é a localização da informação.

O endereço também é um conjunto de *bits*, e não deve ser confundido com a palavra de informação, isto é, a localização da informação não deve ser confundida com a própria informação, ou seja, com o conteúdo da localização.

Na forma mais simples, a memória principal é endereçada através de um registrador de endereços (*RE*), que é decodificado para atuar os próprios circuitos *drivers* da memória. Os *bits* da palavra lida são amplificados por circuitos *sense amps* e colocados no registrador de dados (*RD*). Para escrever na memória, a palavra a ser escrita é antes guardada no *RD*.

Hoje em dia, a memória principal é estática, mas, na primeira geração de computadores, foram usadas memórias dinâmicas como linhas de atraso ou retardo. A linha de atraso também é um exemplo de uma memória "volátil", isto é, se faltar energia, a memória se apagará. Memórias monolíticas também são voláteis. Memórias de núcleos de ferrite e veículos magnéticos como fita, disco etc. são exemplos de memórias não-voláteis.

4.4 MEMÓRIA DE NÚCLEO DE FERRITE

A tecnologia para memória principal que dominou a segunda e terceira gerações de computadores foi a memória de núcleos. O núcleo pode ser magnetizado em duas direções, no sentido horário e no anti-horário. Portanto dois sentidos que podemos convencionar como sendo "0" e "1". O núcleo, com a sua curva magnetização-corrente é mostrado na Fig. 4-3.

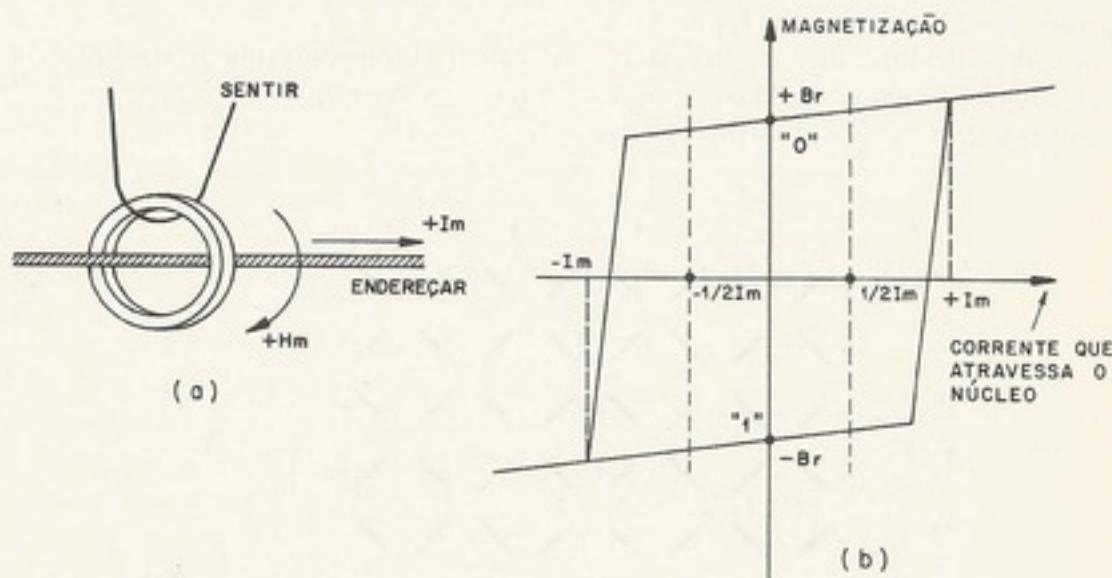


Figura 4-3. (a) O núcleo de ferrite e (b) sua curva magnetização-corrente

Para se escrever "0" no núcleo, basta passar a corrente $+I_m$ no fio de endereço, que deixará o núcleo magnetizado no estado $+B_r$. O processo de escrever "0" é a técnica usada para ler o conteúdo do núcleo. Durante a aplicação da corrente $+I_m$, se o núcleo estiver no estado "0" ($+B_r$), pouca corrente será induzida no fio "sentir". Caso contrário, se o núcleo estiver no estado "1", o núcleo passará de $-B_r$ a $+B_r$, induzindo bastante corrente no fio de sentido. Essa corrente é amplificada e descoberta através de um circuito amplificador sensor e armazenada num *flip-flop*. Para não perder o conteúdo do núcleo após o processo de ler, o *bit* que foi armazenado no *flip-flop* deve ser reescrito. Por isso, as memórias de núcleos são do tipo "ler-destrutivo".

Em vez de prover um fio e, então, um *driver* para cada endereço na memória, a maioria dos sistemas usam a técnica de "corrente coincidente". Os núcleos são montados na forma de uma matriz com os fios de endereçamento formando um reticulado com um núcleo em cada interseção. O esquema é mostrado na Fig. 4-4.

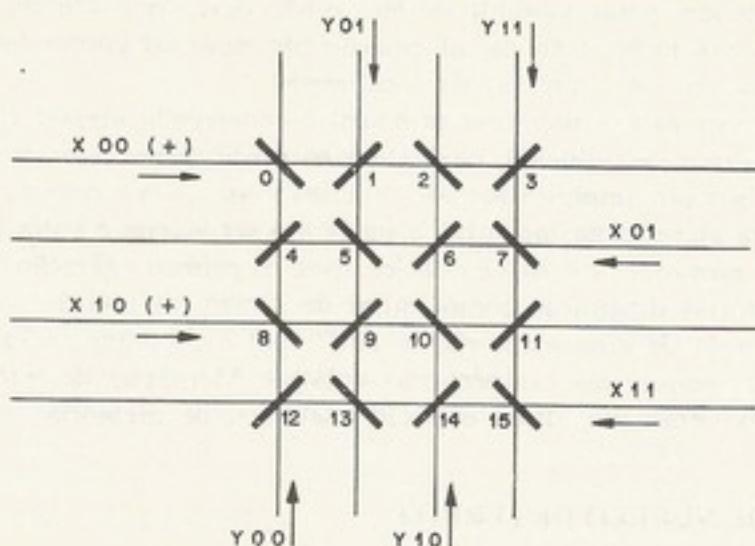


Figura 4-4. Fiação de endereçamento de corrente coincidente

Aplicando-se $1/2 I_m$ ao fio X_{10} e $+1/2 I_m$ ao fio Y_{01} , só o núcleo nove recebe $+I_m$, e só o núcleo nove é selecionado. Os núcleos 1, 5 e 13 recebem $+1/2 I_m$ (*half select*) do Y_{01} e núcleos 8, 10 e 11 só recebem $+1/2 I_m$ do X_{10} . Esses núcleos são perturbados um pouco, mas não o suficiente para mudarem de estado. Agora, para escrever "1" no núcleo 9, basta aplicarmos $-1/2 I_m$ no X_{10} e Y_{01} .

Para uma memória de 4 bits na palavra, é comum termos um reticulado X-Y para cada bit então quatro reticulados. Mas, para economizar nos *drivers*, o *driver* X_{01} , por exemplo, passa através de todos os quatro reticulados, como na Fig. 4-5.

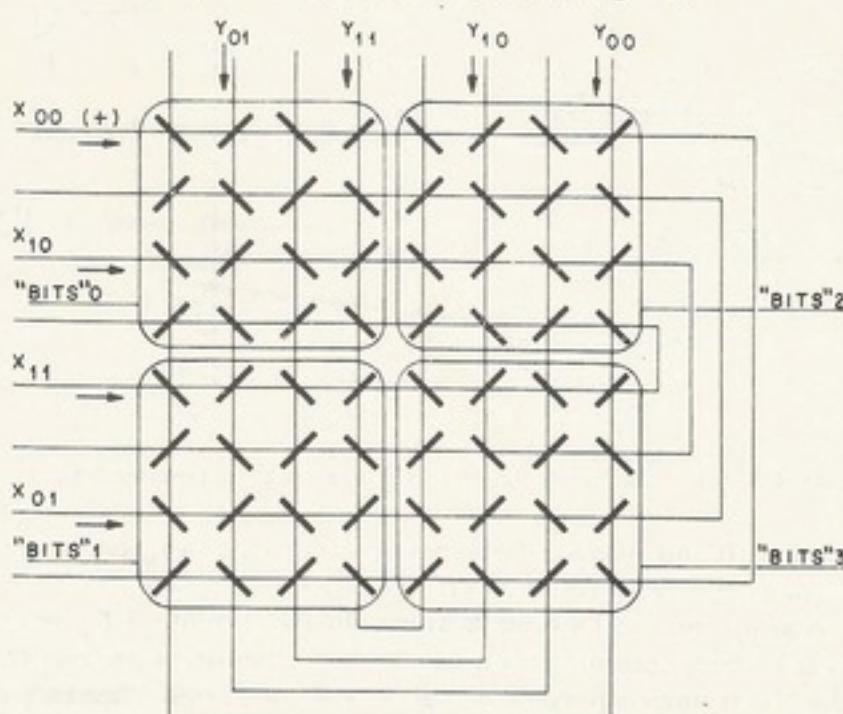


Figura 4-5. Memória de núcleos de 16 palavras de 4 bits

Agora o processo de reescrever a palavra lida é problemático. Aplicando-se $+1/2 I_m$ e $-1/2 I_m$ respectivamente aos fios de endereçamento só podemos escrever palavras "0000"

ou "1111", chamado *inhibit*, mostrada na

Figura 4-6.

Cada memória tem que, dada uma palavra de 4 bits, "0110", isto é, os drivers X_{01} e Y_{01} de endereço 9, devemos aplicarmos $+1/2 I_m$ para mudar para "0000" na Fig. 4-7. Cada

Podemos ler algo da memória e transferindo os dados da memória para restaurar o conteúdo.

memória, a maioria
estudos na forma
de um núcleo em

ou "1111", respectivamente. A solução do problema é colocarmos um quarto fio no núcleo, chamado *inhibit*, que tem a finalidade de anular uma das correntes. A fiação do *inhibit* é mostrada na Fig. 4-6.

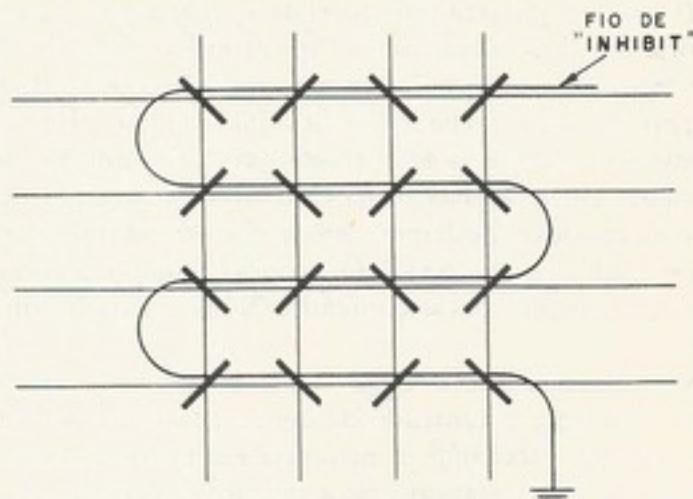


Figura 4-6. O fio *inhibit*

Cada reticulado tem seu próprio fio *inhibit*, como tem seu próprio fio sensor. Suponhamos que, durante o processo de ler o endereço 9, descobramos que estava guardando "0110", isto é, que bits 0 e 3 eram "0" e bits 1 e 2 eram "1". Para reescrever a palavra, os drivers X_{10} e Y_{01} irão colocar $-1/2 I_m$ cada para um total de $-I_m$ em todos os quatro núcleos de endereço 9, tentando escrever "1111". Para cancelar o efeito de $-I_m$ nos bits 0 e 3, colocaremos $+1/2 I_m$ no fio *inhibit* desses bits. Agora só bits 1 e 2 recebem o fluxo necessário para mudar para $-B_r$. O gráfico do ciclo ler-escrever de uma memória desse tipo aparece na Fig. 4-7. Como se vê, a operação consta de dois meios-ciclos.

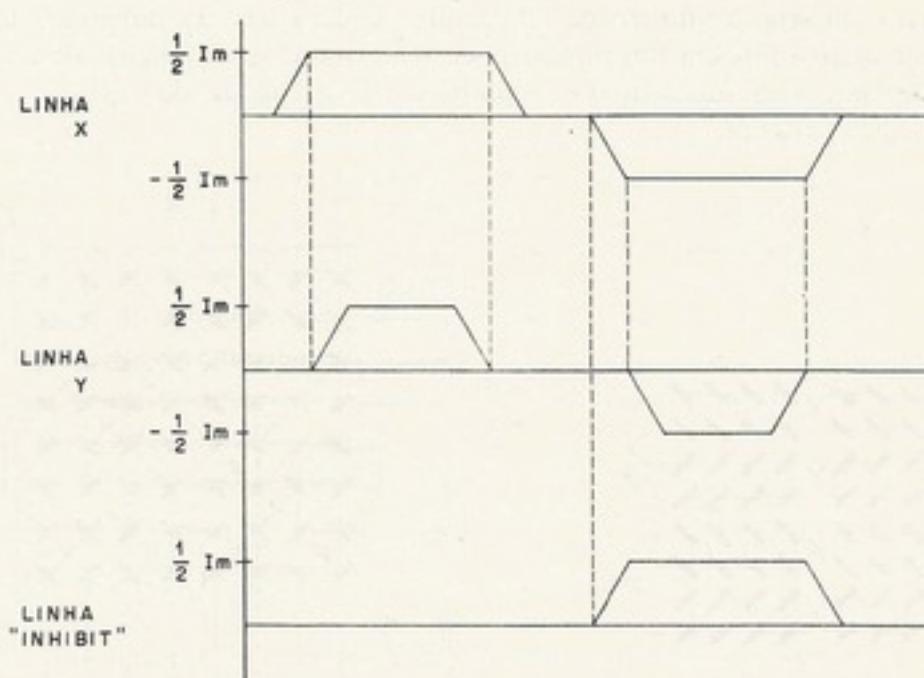


Figura 4-7. O ciclo completo de memória corrente coincidente

Podemos operar a memória em várias formas diferentes. Algumas vezes, queremos *ler* algo da memória. Nesse caso, devemos iniciar lendo o conteúdo da posição endereçada e transferindo esse conteúdo para um registrador externo, que chamaremos de "registrador de dados da memória", ou simplesmente, "registrador de dados" (RD). Após isso, devemos restaurar o conteúdo da posição lida. Outras vezes, nós operamos a memória para escrever

$+1/2 I_m$
palavras "0000"

alguma palavra anteriormente armazenada no RD. Isso é conseguido "simulando-se" uma leitura da posição endereçada (para torná-la zero), sem transferir para RD o conteúdo lido. Deixamos esse conteúdo se perder. Na hora de restaurar a memória, escrevemos o conteúdo de RD, que é a palavra que queríamos guardar na memória. Note, portanto, que o ciclo da memória em dois meios-ciclos, um primeiro ciclo, que é "ler-limpar", e um segundo ciclo, que é "restaurar-escrever". Por isso, muitos módulos de memória são projetados para operar na forma de "ciclo rachado" (*split-cycle*), onde o próprio módulo de memória gera os sinais de controle e os pulsos de corrente para cada um dos meios-ciclos, e a unidade de controle do sistema envia apenas ordens do tipo "meio-ciclo de ler", "meio-ciclo de escrever", etc. Em contraposição, podemos pensar em um módulo de memória que receba apenas ordens do tipo "leia" ou "escreva". O próprio módulo geraria os meios-ciclos convenientes para a execução correta desse comando. Nesse caso, diríamos que a memória opera em "ciclo completo".

Suponhamos que a UCP tenha de ler uma determinada palavra da memória, fazer a soma com ela e o conteúdo de um registrador e colocar o resultado na localização da primeira palavra. Operando a memória em ciclos completos, a operação precisa de um ciclo completo de "ler-restaurar" para retirar o operando e somá-lo com o registrador, e de um segundo ciclo completo de "limpar-escrever" para guardar o resultado, num total de quatro meios-ciclos. Operando em "ciclo rachado", a operação só leva três meios-ciclos, um meio-ciclo de leitura, um meio-ciclo ocioso para a memória em que a soma dos operandos é feita, e um meio-ciclo de escrita para guardar o resultado. O "ciclo rachado" é uma técnica geralmente utilizada pelas memórias do tipo "ler-destrutivo", pois existe a restrição de que, antes de escrever, a palavra endereçada deve ser limpa.

A estrutura de fiação dos núcleos magnéticos com os fios *X* e *Y*, *sense* e *inhibit* que acabamos de descrever é chamada 3D de quatro fios [Fig. 4-8(a)]. A notação 3D vem das três dimensões da memória, *X* e *Y* formando um reticulado de duas dimensões, e o número de bits por palavra, ou seja, o número de reticulados sendo a terceira dimensão. O problema de fazer passar quatro fios em um núcleo pode ser evitado se as funções de sentir e inibir forem combinadas em um único fio. O resultado é um sistema 3D de três fios. A Fig. 4-8(b) mostra esse outro sistema.

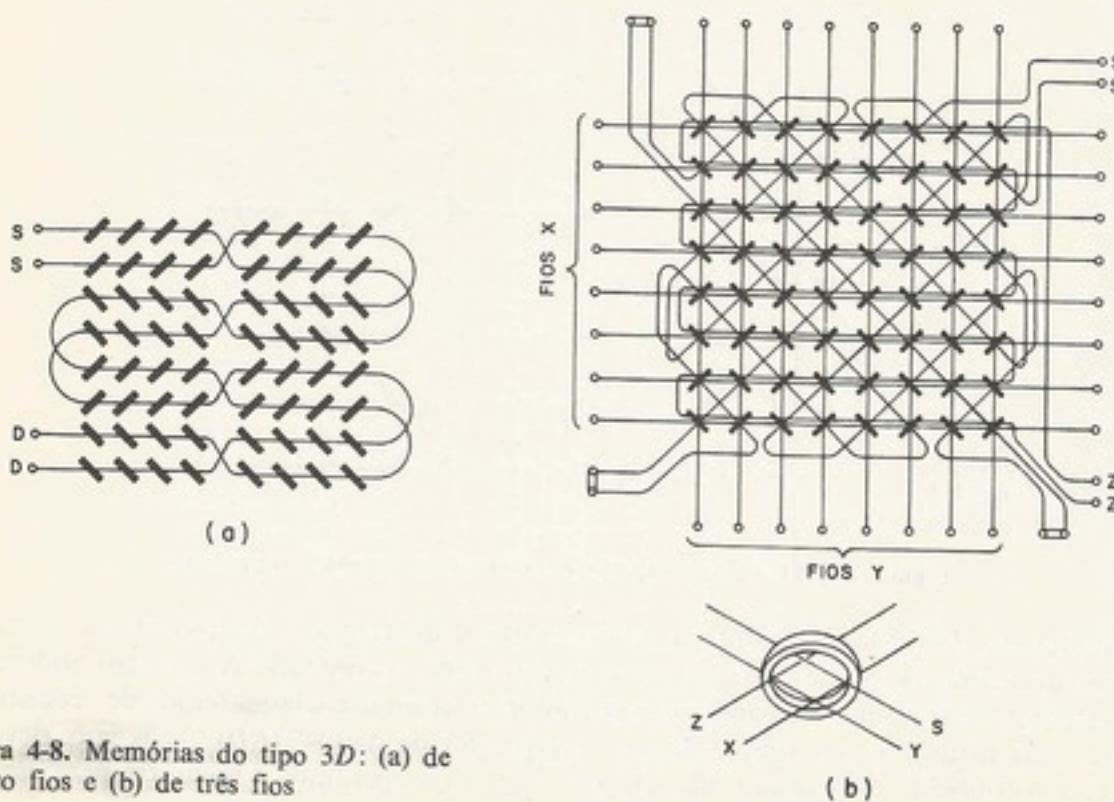


Figura 4-8. Memórias do tipo 3D: (a) de quatro fios e (b) de três fios

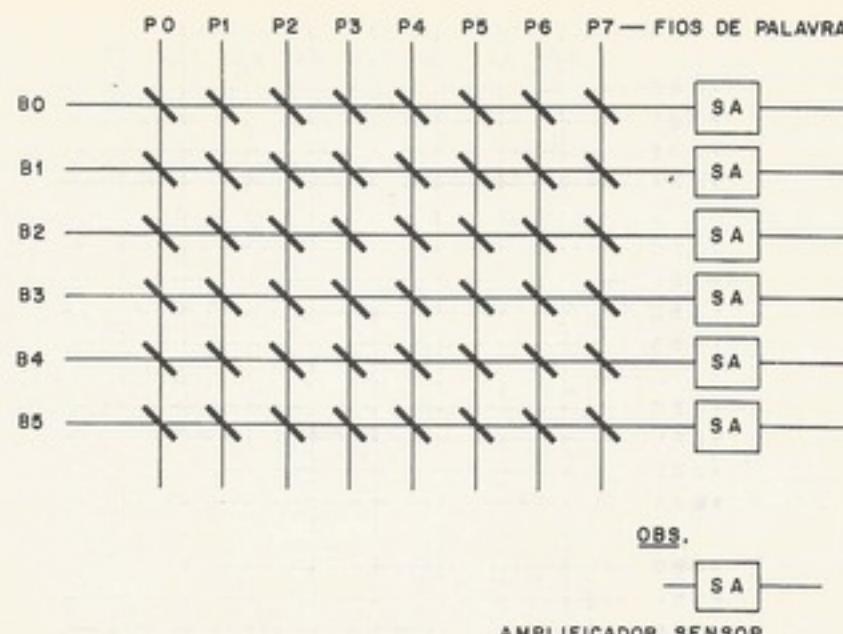


Figura 4-9. Memórias de dois fios tipo 2D, de oito palavras de 5 bits

Uma memória do tipo 2D é mostrada na Fig. 4-8. O método da seleção do endereço é linear em vez de corrente coincidente.

A memória tipo 2D tem um *driver* para cada palavra. Cada bit tem um fio que é usado como *sense* nos meios-ciclos de leitura; é como *inhibit* nos meios-ciclos de "escrever".

Entre as organizações 2D e 3D, existe uma intermediária chamada $2\frac{1}{2}D$, vista na Fig. 4-10(a). O inconveniente na organização 2D é que as memórias de muitas palavras têm muitos *drivers* (um *driver* por posição de memória) e o diagrama pareceria ser muito comprido em relação à largura. Pode-se pensar em um outro esquema onde o número de palavras (p) é dividido em K blocos, entre os quais se usa a técnica de corrente coincidente. Nesse caso, o diagrama ficaria com Kb fios, onde b é o número de bits por palavra, na direção dos bits horizontais e $p \div K$ fios na direção das palavras verticais. No meio-ciclo de leitura um fio na direção das palavras x_i recebe $+1/2 I_m$, e b fios do grupo correspondente à direção dos bits recebem $+1/2 I_m$. Os fios de *sense* [Fig. 4-10(a)], um para cada bit da palavra, ligam em série todos os bits correspondentes de cada grupo. Portanto existem b amplificadores de sentir. Talvez um exemplo esclareça melhor: para ler a palavra X2Y3 na Fig. 4-10(a), o fio X2, na direção das palavras, recebe $+1/2 I_m$ e os fios Y3B0, Y3B1, Y3B2 e Y3B3, na direção dos bits, recebem $+1/2 I_m$. Para escrever "0110" na palavra X2Y3 da Fig. 4-10(a), primeiro limpamos o conteúdo dessa posição (lendo) e depois fazemos passar $-1/2 I_m$ em X2 Y3B1 e Y3B2.

Assim, os núcleos X2Y3B1 e X2Y3B2 são atravessados pela corrente $-I_m$ necessária para levá-los ao estado "1". Existem também organizações tipo $2\frac{1}{2}D$ com apenas dois fios atravessando cada núcleo. Nesse esquema, os fios na direção dos bits horizontais são usados como *sense*, no meio-ciclo de "ler", ou para conduzirem correntes de $+1/2 I_m$ ou $-1/2 I_m$ no meio-ciclo de "escrever". Portanto é um esquema mais lento. O processo de seleção dos bits a serem lidos depende da polaridade da corrente na direção dos bits [veja a Fig. 4-10(b)].

Se queremos ler o conteúdo dos dois bits da palavra X0Y1, é necessário passar uma corrente de $1/2 I_m$ no fio X0 e correntes também de $1/2 I_m$ nos pares de fios cujos terminais tenham o nome Y0B0 e Y1B1. A corrente entra no circuito pelo terminal Y1B1 e sai pelo terminal Y0B1 (se estivéssemos lendo a palavra X0Y0, a corrente entraria pelo terminal Y0B1). Portanto, no momento da leitura, haverá correntes de $1/2 I_m$ da esquerda para a direita nos quatro fios superiores do esquema da Fig. 4-10(b), e corrente de $1/2 I_m$, de cima para baixo, no fio X0. Portanto os únicos núcleos que são atravessados por uma corrente total de I_m são os núcleos marcados com os números 2 e 4 (os 2 bits da palavra X0Y1). Observe

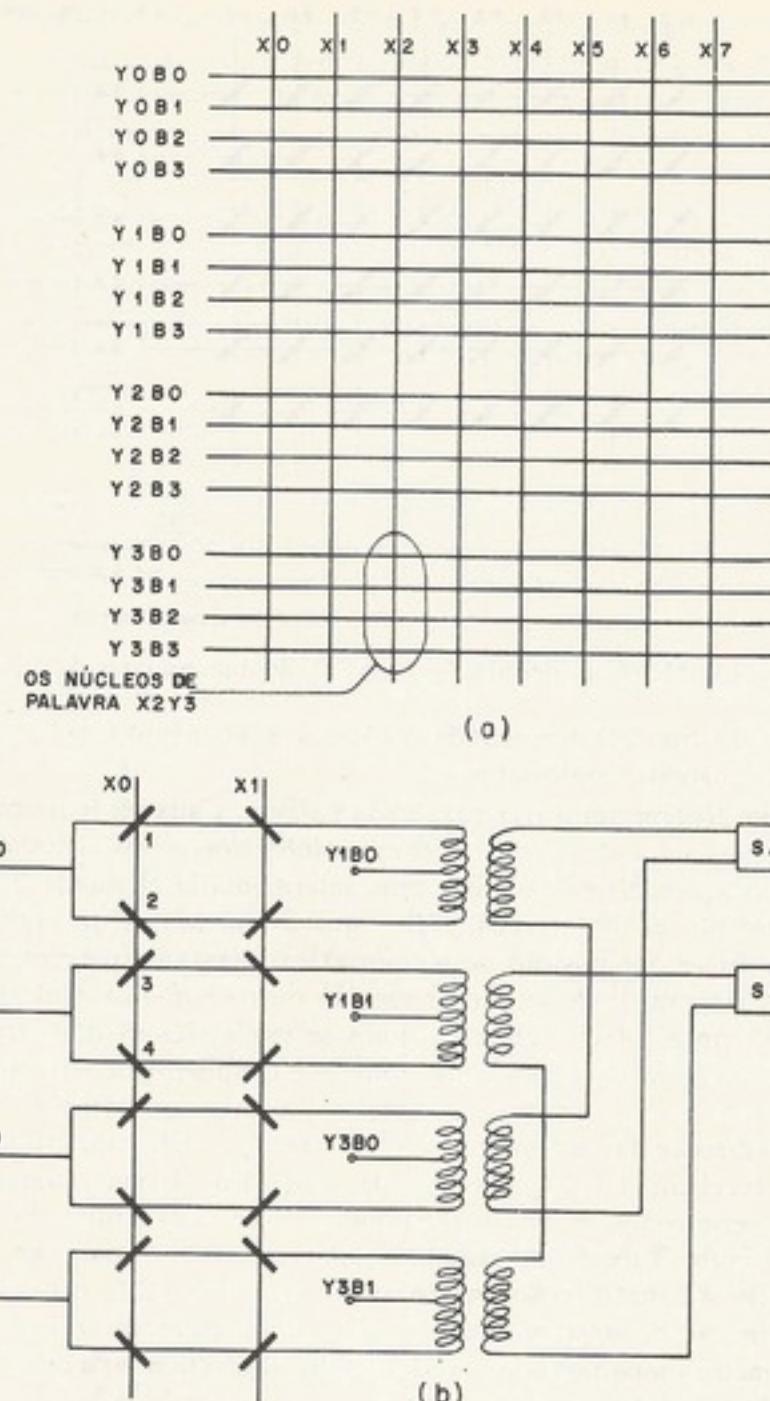


Figura 4-10.(a) Memória de três fios tipo $2\frac{1}{2}D$, de 32 palavras de 4 bits, com $K = 4$ (o fio de sense não é mostrado); (b) memória de dois fios tipo $2\frac{1}{2}D$ de oito palavras de 2 bits

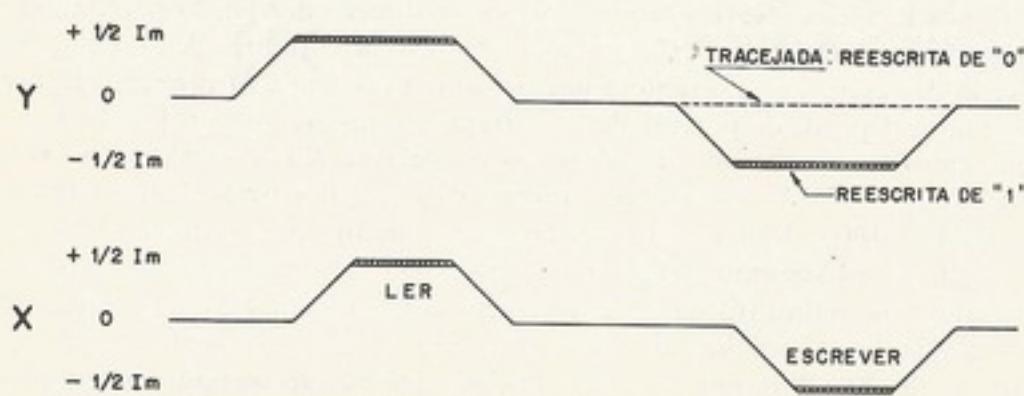


Figura 4-11. Ciclo completo de memória tipo $2\frac{1}{2}D$

memória e armazena-

que os núcleos de

as correntes são e

O ciclo de "le-

tudo de reescrita"

A capacidade

($1K \cong 1024 = 2^{10}$)



Figura 4-12. A orga-

nizado; RE, regis-

o método de co-

decodificador X

na direção X. Na

4-12, e a orga-

4.5 MEMÓRIAS

A aplicação

oferece uma em

Uma das primei-

que os núcleos 1 e 3 são cruzados por dois fios que conduzem correntes de $1/2 I_m$, porém as correntes são em sentidos opostos, o que faz com que elas se cancelem.

O ciclo de "ler-escrever" da memória de Fig. 4-10(b) é mostrado na Fig. 4-11. O método de reescrever é o mesmo método usado no esquema da Fig. 4-10(a).

A capacidade da memória principal é normalmente medida em termos de K palavras ($1K \cong 1024 = 2^{10}$). Com 10 bits, por exemplo, podemos endereçar 1024 palavras. Usando

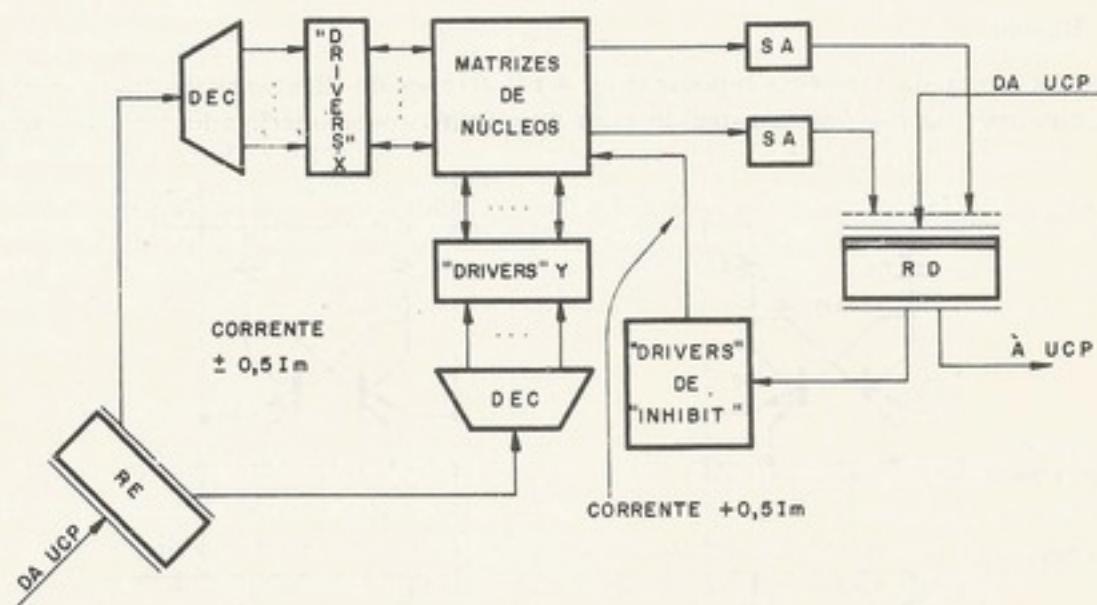


Figura 4.12. A organização geral de uma memória tipo 3D. DEC, decodificador; SA, amplificador sensor; RE, registrador de endereço; RD, registrador de dados

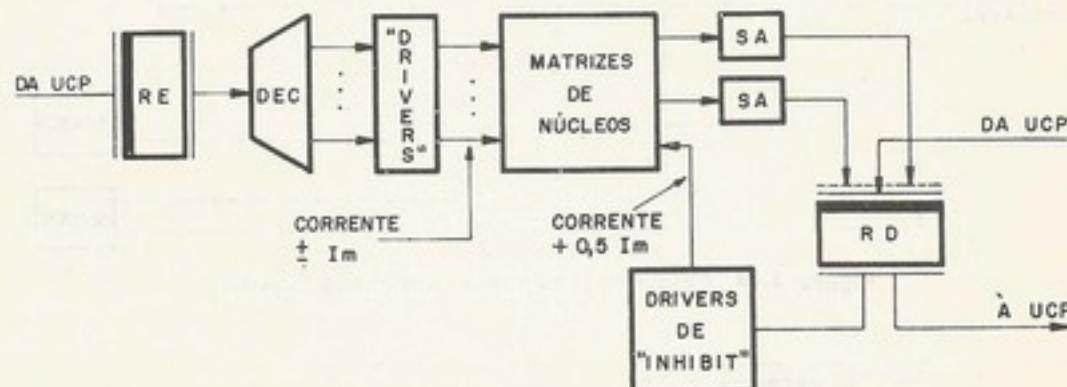


Figura 4-13. A organização de uma memória tipo 2D

o método de corrente coincidente, o endereço seria dividido em duas partes, 5 bits para o decodificador X e 32 drivers ($2^5 = 32$) na direção X ; 5 bits para o decodificador Y e 32 drivers na direção Y . A organização em blocos funcionais de uma memória 3D é mostrado na Fig. 4-12, e a organização 2D é mostrada na Fig. 4-13.

4.5 MEMÓRIA MONOLÍTICA

A aplicação da tecnologia de LSI (*large-scale integration*) para a função da memória oferece uma estrutura regular de interligações na pastilha e uma boa relação pinos/circuitos. Uma das primeiras aplicações em grande escala de memória monolítica foi o armazenador

da memória principal (o *cache*) do sistema IBM S/360 modelo 85, com uma capacidade de 2M palavras de 72 bits⁽⁴⁾. Dois dos primeiros sistemas a usarem a memória monolítica para a própria memória principal foram os sistemas Illiac IV, da University of Illinois, e o IBM S/370 modelo 145⁽⁵⁾.

Existem várias técnicas de implementação de memória monolítica. As mais comuns são o bipolar, o *MOS* (metal-oxide semiconductor), estático e o *MOS* dinâmico.

a. Bipolar

A célula típica da memória bipolar (Fig. 4-14) através do diagrama de quatro células, é nada mais que um *flip-flop* construído com dois transistores interligados (*cross-coupled*).

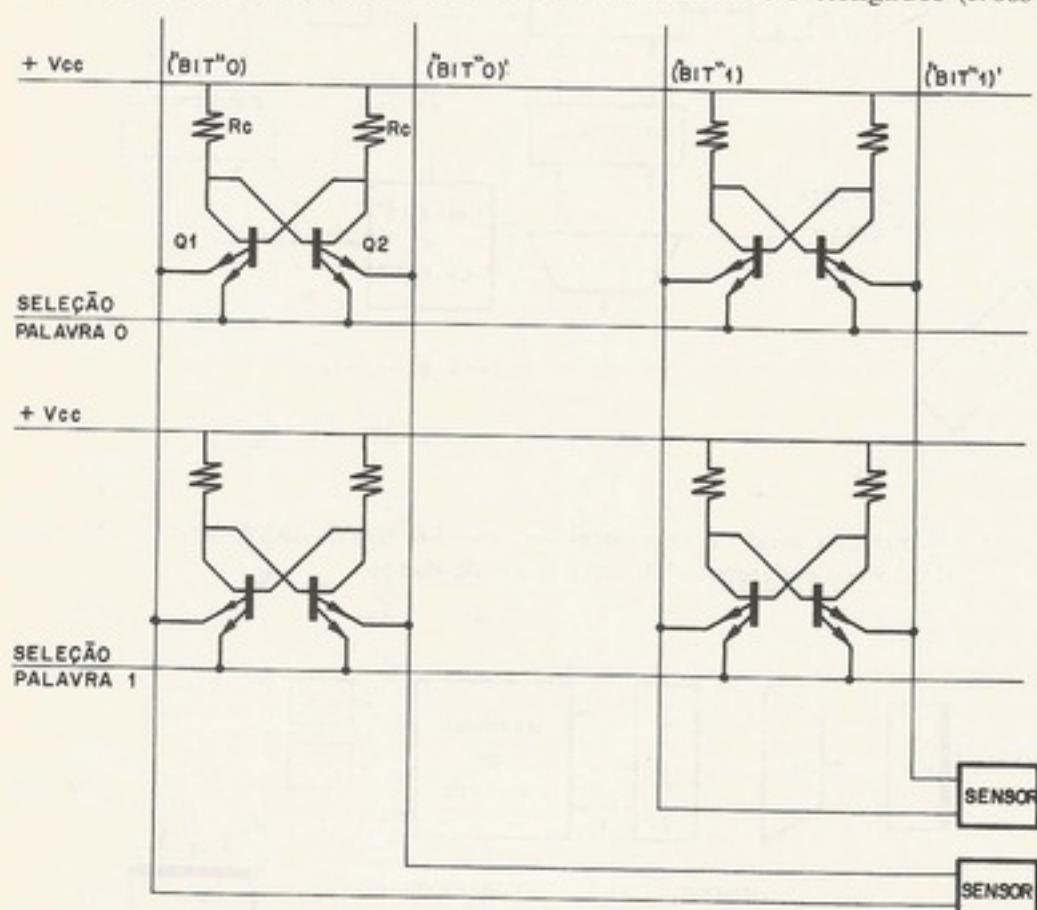


Figura 4-14. Células de memória monolítica bipolar



Figura 4-15. Organização da pastilha de matriz de células

A aparência da pastilha bipolar é a de uma matriz dessas células junto com um decodificador de endereço e um circuito sensor, como visto na Fig. 4-15.

Para entendermos o funcionamento da célula, devemos observar o bit 0 da palavra 0 da Fig. 4-14. Suponhamos o transistor Q_1 conduzindo, cortando o transistor Q_2 , o que significa estar a célula no estado "1". Normalmente o sinal "seleção palavra 0" está na tensão baixa, recebendo a corrente do transistor 1. Para leitura, a tensão desse sinal é elevada a um potencial mais alto que o dos sinais do bit, forçando a corrente no emissor "bit 0" do transistor 1, que é descoberto através do sensor. Se a célula estivesse armazenando "0", seria o sinal

(bit 0)' que receberia a tensão elevada. O sinal "bit 0" é lido quando a tensão é baixa, e o sinal "bit 0"' é lido quando a tensão é alta.

Por motivos de simplicidade, o diagrama mostra apenas duas células. Na realidade, a matriz tem 128 linhas e 128 colunas. O decodificador é responsável por gerar os 128 sinais de seleção de linha e coluna. As pastilhas de memória bipolar são normalmente feitas com um tipo de estrutura de célula que não é mostrada aqui.

b) Memória MOS

A célula de memória MOS é baseada no efeito de campo. Os transistores Q_5 e Q_6 são análogos ao flip-flop bipolar, mas controlados através de um sinal de seleção de célula. Para "leitura", o sinal "seleção célula" é elevado para nível alto, permitindo que o sinal "seleção palavra" entre no flip-flop no estado desejado.

(bit 0)' que receberia essa corrente. Depois que o estado é descoberto, o sinal ("seleção palavra 0" volta ao nível anterior e o transistor 1 da célula continua conduzindo. O processo de leitura não modifica o estado da célula e é então chamado "leitura não-destrutiva". Para se escrever "0" na célula, os sinais "seleção palavra 0" e "bit 0" são colocados na alta tensão, enquanto (bit 0)' continua no nível normal e o transistor 2 tem de conduzir a corrente. O transistor 2 continua conduzindo depois que os sinais "seleção palavra 0" e "bit 0" voltam ao seu nível normal.

Por motivos de confiabilidade e para melhor localizar erros, é costume organizar uma pastilha de 128 células (por exemplo) em 128 "palavras" de um bit cada. Nesse exemplo, o decodificador é de 7 bits. Agora o defeito em uma pastilha não influí mais que um bit. Além das pastilhas da matriz (array chips) um sistema de memória monolítica necessita de um outro tipo de pastilha, chamado pastilha de apoio. As pastilhas de apoio (há vários tipos) normalmente desempenham três funções: (1) decodificar os bits mais significativos do endereço para selecionar as próprias pastilhas da matriz, (2) prover uma via para ligar cada pastilha de matriz com uma posição do registrador de dados (RD), e (3) fornecer os sinais de timing.

b) Memória MOS

A célula do tipo *MOS* é mostrado na Fig. 4-16, onde os transistores são do tipo *FET* (efeito de campo) e as polaridades são para dispositivos canal *P* (*P-channel*). Os transistores Q_5 e Q_6 são análogos ao resistor de coletor R_c (a carga) da célula bipolar. O *flip-flop* é controlado através de transistores Q_3 e Q_4 , funcionando analogicamente como os emissores de seleção da célula bipolar. A operação da célula é como no caso bipolar. Para leitura, o sinal "seleção palavra" liga Q_3 e Q_4 , transferindo o bit do *flip-flop* para os fios (bit) e (bit)'. Para "escrita", o sinal "seleção palavra" liga Q_3 e Q_4 , enquanto os níveis para colocar o *flip-flop* no estado desejado são estabelecidos nos fios (bit) e (bit)'.

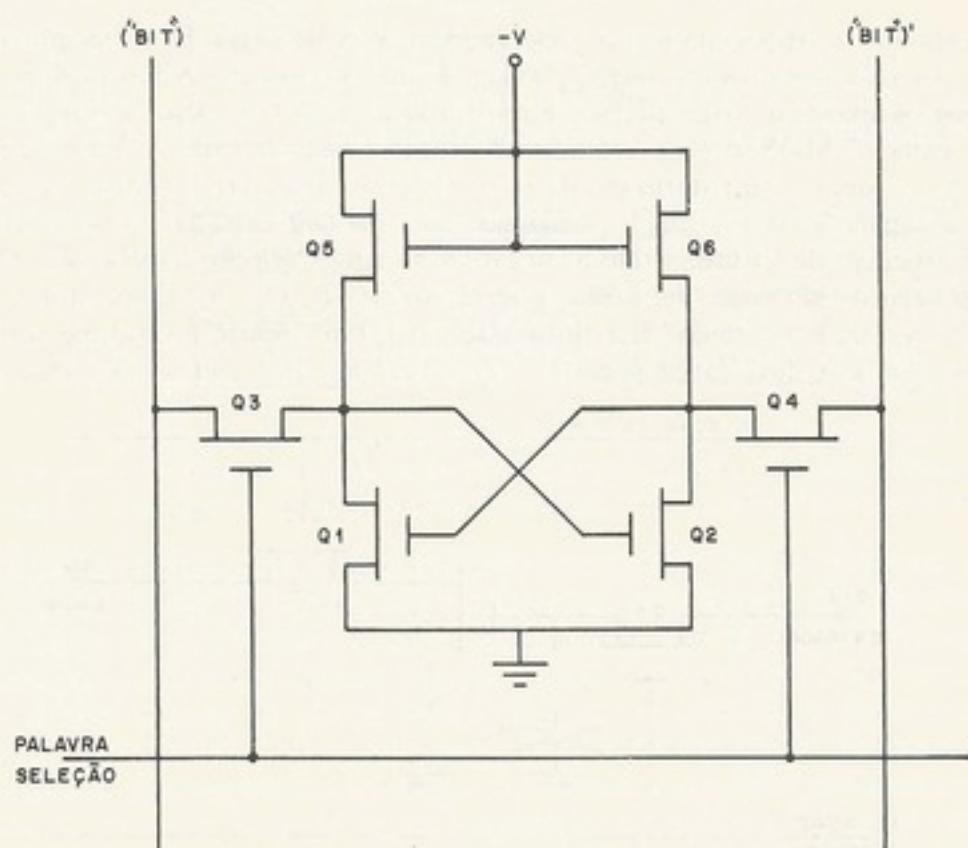


Figura 4-16. Célula para memória monolítica tipo *MOS* estática

A idéia de memória dinâmica de *MOS*, como na Fig. 4-17, é eliminar os *FET* de carga (Q_5 e Q_6 da Fig. 4-16) e usar a carga armazenada em Q_1 ou Q_2 (*gate capacitance*) para lembrar o estado do *flip-flop*. Infelizmente, a carga elétrica escapa (*leakage*) e cada célula tem de ser regenerada periodicamente para não perder informação. Para leitura, o fio “seleção palavra” é usado para ligar Q_3 e Q_4 , e corrente é sentido no fio (*bit*) ou (*bit*)'. Por exemplo, se Q_2 está conduzindo, no fio (*bit*)' passa a corrente, que é sentida pelo correspondente amplificador. Ao mesmo tempo, nesse caso, a carga da capacitância da porta de Q_1 está reabastecida. Então, para “regenerar” a célula, basta só fazer a operação de leitura, inclusive todas as células na pastilha podem ser regeneradas ao mesmo tempo, colocando-se todos os fios “seleção palavra” no nível negativo de uma vez. O modo de escrever essa célula (Fig. 4-17) é o mesmo da célula estática (Fig. 4-16).

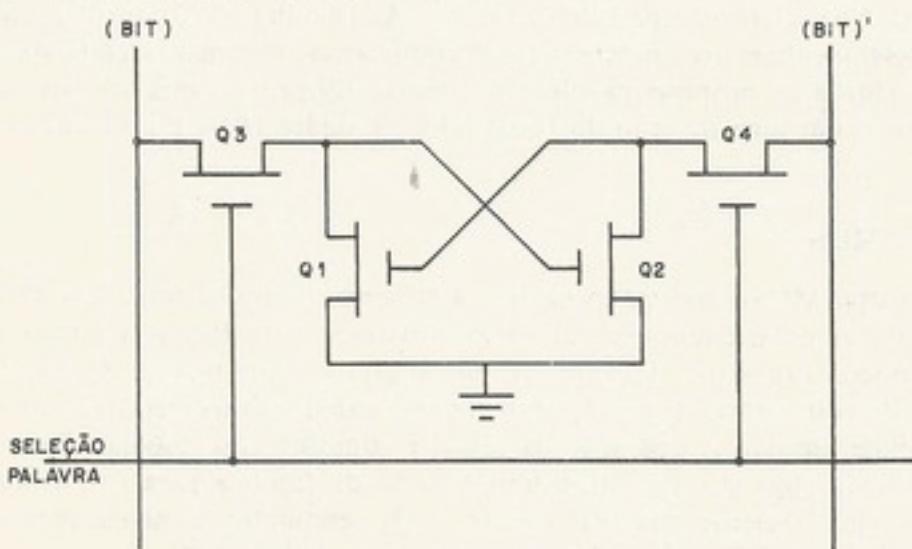


Figura 4-17. Célula de memória dinâmica de *MOS*

Na realidade, o armazenamento é efetuado através de carga na capacitância da porta do *FET* e, por isso, dos dois *flip-flop*, Q_1 e Q_2 , só um é necessário. A nova célula, que só usa um *FET* para armazenar carga elétrica, é mostrado na Fig. 4-18. Aqui, em termos das polaridades de canal *P MOS*, o *bit* é armazenado como tensão negativa (“1”) ou como tensão nula (“0”) no capacitor parasitário (C) da porta de Q_1 . Para escrita, o sinal “seleção escrita” liga Q_3 , e a capacitância C segue a polaridade do sinal “*bit* entrada”, ou negativo ou terra. Durante o processo de leitura, a tensão negativa no sinal “seleção leitura” liga Q_2 e acopla a voltagem negativa do sinal “*bit* saída” à terra, através de Q_1 , se o capacitor C é negativo (“1”); se não, o sinal “*bit* saída” fica flutuando (“0”). Para sentir, é costume, antes de ligar (*turn-on*) Q_2 , colocar uma carga negativa (pré-carga) no sinal “*bit* saída”. Daí, quando Q_2

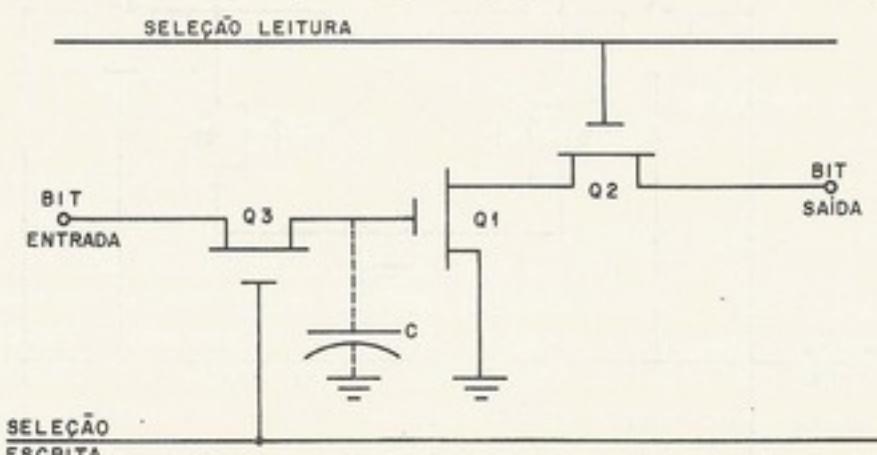


Figura 4-18. Célula dinâmica *MOS* de 3 *FETs*

conduz, “bit saída” é a tensão negativa da variação de leitura num bit numa célula de regeneração grande. A taxa de regeneração é grande, de fios “bit entradas” da linha da matriz é de 1000 bits por segundo. Um comando periódico de 100 ciclos de regeneração de 1%.

Com pastilhas de *MOS* ou *MOS*. Dispositivos A solução foram apoiados de bipolar: decodificar (além disso) (em blocos) de uma é de 1024 bits (32x32). decodificador em cada que seleciona a matriz.

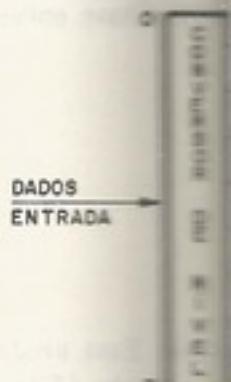


Figura 4-19. Diagrama de driver com relé.

O registrador é de 1000 bits. Os bits nas células vizinhas. Memória é de 1024 bits. Lendo a passagem de volta os bits das células para parar, mas, com a tensão.

Um problema é a queda da força, isto é, a tensão principal, pode ser escrita, o requerimento

um FET de carga para lembrar a célula tem de ser o fio "seleção escrita". Por exemplo, correspondente amostra Q_2 está reabastecendo, inclusive todas as células todos os fios da matriz (Fig. 4-17)

conduz, "bit saída" descarrega à terra só se estiver guardando "1". Com essa célula, a operação de leitura não regenera o bit guardado. Para regenerar o bit, é necessário colocar o bit numa outra célula (chamada célula de regeneração) e, depois, reescrever o bit. O processo de regeneração precisa de um ciclo de leitura e, em seguida, um ciclo de escrita. A célula de regeneração é partilhada entre as colunas da matriz, isto é, cada coluna partilha o par de fios "bit entrada" e "bit saída", junto com uma célula e amplificador de regeneração. Cada linha da matriz é selecionada para regeneração à razão de uma em cada um ou dois milisegundos. Um contador de endereço guarda o endereço a ser regenerado e um relógio inicia periodicamente o ciclo de regeneração. Se o ciclo da memória dinâmica for 500 ns, e 32 ciclos de regeneração forem precisos a cada 2 ms, a interferência de regeneração será menor de 1%.

Com pastilhas de matriz na tecnologia *MOS*, as pastilhas de apoio podem ser bipolar ou *MOS*. Dispositivos *MOS*, sendo de alta impedância, têm capacidade limitada para *drive*. A solução foram os sistemas híbridos com pastilhas de memória de *MOS* e pastilhas de apoio de bipolar; isto é, as pastilhas bipolares fazem as operações de *drive*, de sentir e de decodificar (além do decodificador nas pastilhas). Na Fig. 4-19 mostra-se uma organização (em blocos) de uma memória *MOS* dinâmica de 4K palavras de 8 bits, onde cada pastilha é de 1 024 bits (32 linhas por 32 colunas). Dos 12 bits de endereço, 10 bits são enviados ao decodificador em cada pastilha, e dois são decodificados para formar o sinal *chip-enable* que seleciona a pastilha.

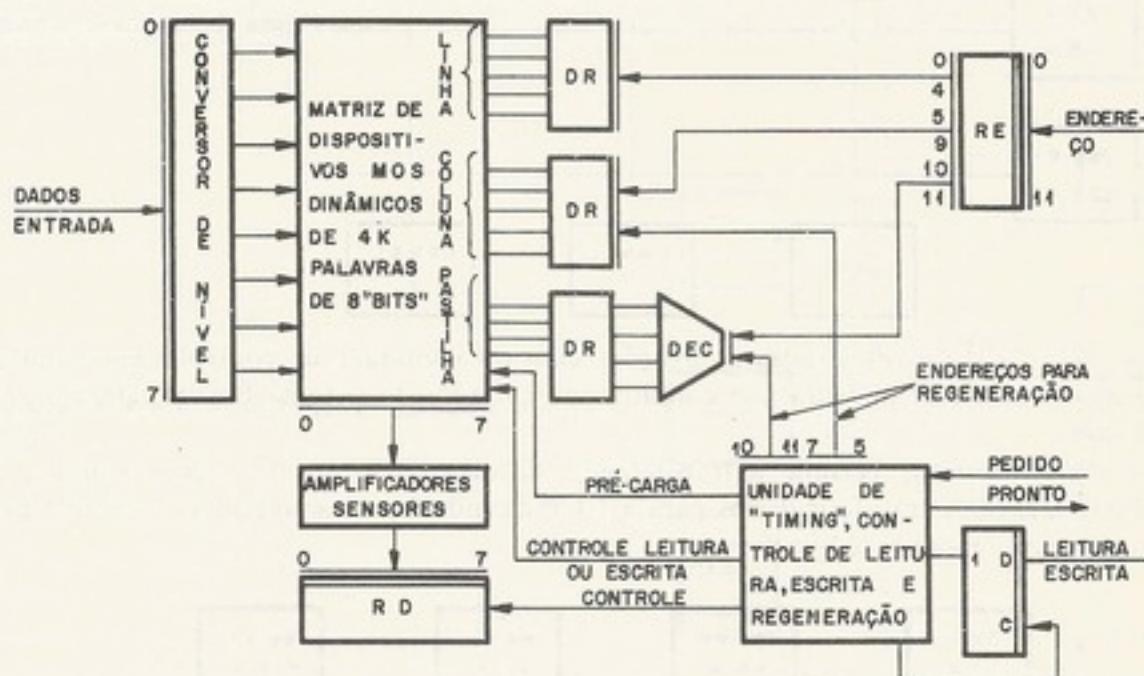


Figura 4-19. Diagrama em blocos da organização de uma memória *MOS* dinâmica. [Observação. DR é driver com relógio; conversor de nível é de TTL à voltagem exigida pela tecnologia *MOS*]

O registrador deslocador é uma organização que resolve bem o problema de regeneração. Os bits nas células dinâmicas, em vez de serem regeneradas, são passadas à célula vizinha. Memórias assim, geralmente usam duas células por bit. Um sinal do relógio controlando a passagem dos bits das células pares às células ímpares e um outro sinal passando os bits das células ímpares às células pares. Com células estáticas, o deslocamento pode parar, mas, com células dinâmicas, tem uma velocidade mínima de deslocamento.

Um problema com memória monolítica é o fato de a informação perder-se com a queda da força, isto é, a memória é volátil. Se houver prejuízo em perder a informação da memória principal, pode ser projetada uma bateria em stand-by. Quando a memória não está sendo escrita, o requerimento para potência é bastante menor. Em compensação, devemos re-

lembra que o processo de leitura não-destrutiva prescinde da restauração da informação através de um ciclo de escrever.

4.6 SISTEMAS DE MEMÓRIA PRIMÁRIA

Em sistemas não muito complicados, a memória primária é um simples módulo de memória, ligado ao *UCP* através de uma interface. Alguns sistemas, para aumentar o *bandwidth*, tem dividido a memória em módulos que operam independentemente e, para esse fim, cada memória tem seu registrador de dados (*RD*) e seu registrador de endereço (*RE*). Sistemas assim precisam de uma unidade para controlar a ligação das memórias com a *UCP* e os canais de *E/S*. Um método de efetuar essa ligação é a chave *cross-bar* mostrado

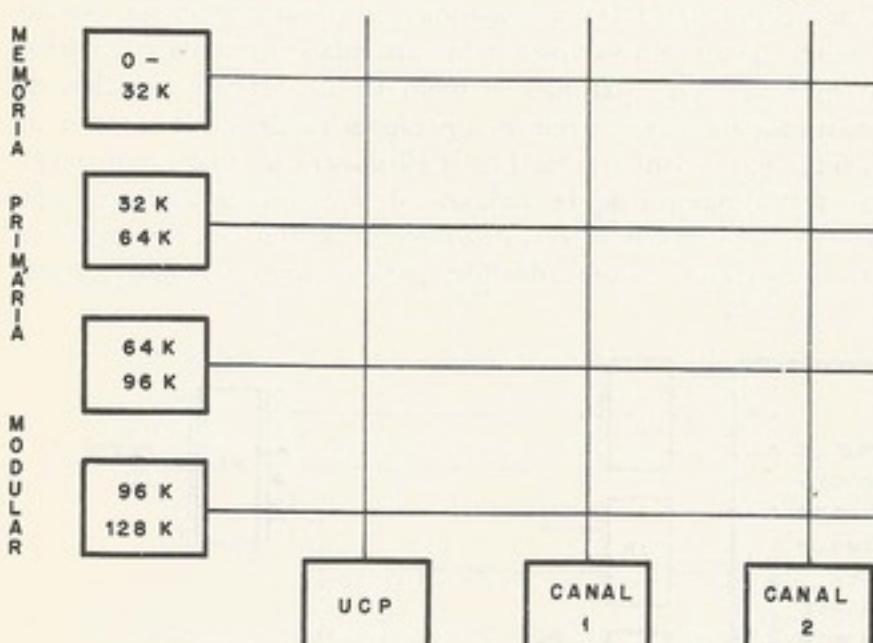


Figura 4-20. Chave *cross-bar*

na Fig. 4-20. Não são mostradas as ligações com uma unidade de controle. Essa unidade efetua as conexões, de maneira que conflitos são evitados. O sistema *Bourroughs 5500* usa essa técnica.

A chave *cross-bar* permite a transferência de dados, em paralelo, por exemplo, um módulo 0-32K pode transferir dados para a *UCP* quando, ao mesmo tempo, o canal 1 pode ser ligado com módulo 64-96K.

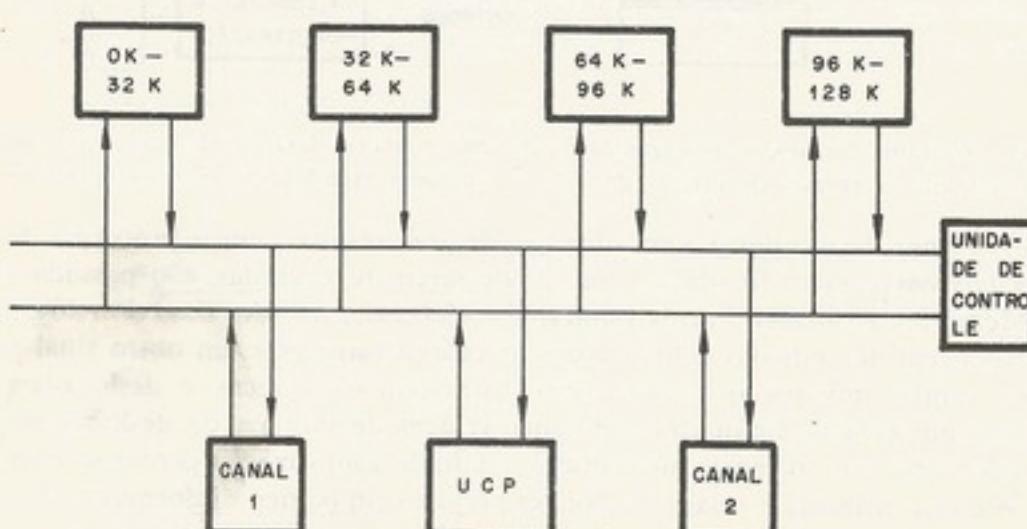


Figura 4-21. Sistema de memória primária modular com via da memória

O sistema também tem a Fig. 4-21. Também tem controle, que regula

Esse esquema mostra um módulo pode ter mais de uma via não é muito é lento, em relação a uma via.

O sistema modular é feito da memória física em módulo 0-32K. No princípio, os dois são o motivo do esquema. Os menos significativos módulos, os endereços e os endereços impre

Nos sistemas de próprio *RE* e *RD*, a unidade *RE* ou *RD*. Esse tipo é usado no *CDC 3600*. Um dígito significa que cada módulo

Figura 4-22. Unidade de controle de um porto

Normalmente, os dados via é sincronizado com a generalmente a *UCP* e sincronamente com os canais. É muito mais econômico. O programador vai querer de lotação de memória.

A memória primária (*large capacity main memory*) é como a memória primária é como a

O sistema também pode ser controlado através de uma via de memória, conforme mostra a Fig. 4-21. Também não são mostrados os sinais de controle provenientes da unidade de controle, que regula as transferências através da via.

Esse esquema não permite ligações em paralelo como no *cross-bar*; ao contrário, só um módulo pode transferir dados de uma vez. O problema do "ponto de estrangulamento" da via não é muito sério, levando-se em conta o fato de que o ciclo da memória em geral é lento, em relação ao tempo necessário para transferir dados eletronicamente através de uma via.

O sistema modular da Fig. 4-21 é chamado de *particionado*. Observar que a seleção do módulo é feita pelos bits mais significativos do endereço. Se a atividade dos usuários da memória fica em módulos diferentes (por exemplo, a UCP está executando instruções em módulo 0-32K quando o canal 1 está transferindo dados para o módulo 32-64K), em princípio, os dois módulos podem operar simultaneamente, aumentando o *bandwidth* efetivo do esquema. Uma outra maneira de organizar uma memória modular é usar os bits menos significativos para selecionar o módulo. Por exemplo, numa memória de dois módulos, os endereços pares (o bit menos significativo é igual a zero) pertencem a um módulo e os endereços ímpares pertencem ao outro.

Nos sistemas modulares (Figs. 4-20 e 4-21) supõe-se que cada módulo tivesse o seu próprio RE e RD. Agora, vamos supor que cada módulo tenha dois (ou mais) conjuntos de RE ou RD. Esse tipo de organização é chamado *multiporto* (*multiport*) e é usado no sistema *CDC 3600*. Um diagrama é mostrado na Fig. 4-22, onde cada módulo tem três portos, que significa que cada módulo tem 3 RE, 3RD, e um circuito de prioridade para resolver conflitos.

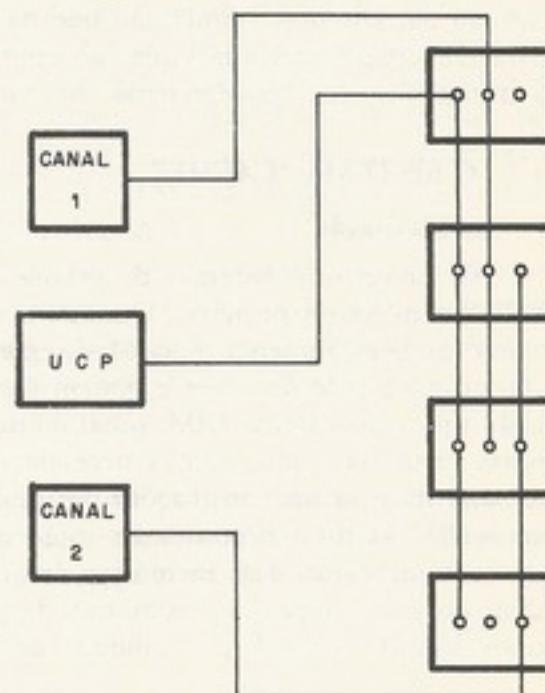


Figura 4-22. Uma organização de memória tipo multiporto

Normalmente, os módulos de memória ligados à UCP são do mesmo tipo e o ciclo da via é sincronizado com o ciclo da memória. Em caso de conflito com uma unidade usuária, geralmente a UCP tem de aguardar a vez. Então, vamos supor que um dos módulos (geralmente com os endereços mais altos) seja mais lento que os outros (e, em comparação, muito mais econômico). Quando isso acontece, a memória não é mais homogênea e o programador vai querer tratar os módulos de maneira diferente, complicando o desempenho de lotação de memória.

A memória mais lenta que é colocada na via de memória primária, é chamada *LCS* (*large capacity store*) ou *ECS* (*extended core storage*). Com *LCS*, a aparência da memória primária é como na Fig. 4-23. A Fig. 4-23 também é um exemplo de como distinguir entre

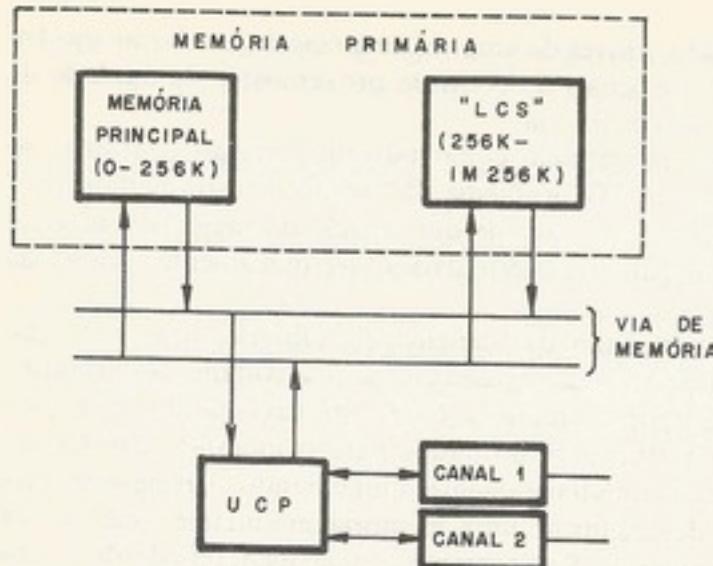


Figura 4-23. Um sistema com LCS

a memória principal e a memória primária. Em muitos sistemas a memória principal e a memória primária são a mesma. Mas, quando o sistema de memória endereçável por instrução consta de componentes não-homogêneos como o *LCS* e como no esquema *cache-backing* (descrição a seguir), facilita-se a distinção do sistema em geral com a memória primária.

A Fig. 4-23 também mostra uma técnica em que a *UCP* é a única que se comunica com a memória primária. Assim, o sistema faz entrada/saída através da *UCP*. Essa técnica é comum em sistemas "mini", em que não existe uma unidade especial (canal) em hardware para controlar a entrada/saída; ao contrário, a unidade central é interrompida para fazer o tratamento das transferências dos lados entre a *MP* e a *E/S*.

4.7 O SISTEMA "CACHE"

a. Motivação

No projeto de sistemas de grande porte, surgiu um problema grave na interface da *UCP* e a memória primária. Quando a velocidade da *UCP* aumenta, a capacidade da memória também aumenta, pois os sistemas mais poderosos podem ter mais programas multiprogramados de uma vez e podem resolver problemas maiores. Por exemplo, a configuração típica do sistema IBM S/360 modelo 85 usa entre um e dois milhões de bytes de memória primária. Junto com a necessidade de bastante memória principal, uma *UCP* que executa duas ou mais instruções por microsegundo exige uma memória veloz e de grande *bandwidth*. As duas necessidades estão em conflito, pois as memórias de alta *performance* são bastante caras, e as memórias de grande capacidade normalmente são as mais lentas. Além do mais, acesso às memórias de grande capacidade têm de suportar os atrasos em cabos ligando a *UCP* e a unidade de controle da memória principal.

b. O sistema "cache-backing"

A solução desse problema no IBM S/360 modelo 85 foi a hierarquização da memória principal. O elemento-chave da organização do modelo 85 é a memória monolítica de alta velocidade, chamado *cache*, que age como armazenador ou *buffer* entre a *UCP* e a memória principal (lenta) agora chamado o *Backing store* (veja a Fig. 4-24).

c. Localidade de referência

Por simulação, verificou-se que a *performance* da memória primária *cache-backing store* do modelo 85 é mais ou menos 80% da *performance* de uma memória principal de

Figura 4-

ciclo de 80 ns. I
memória guarda
mas a idéia é q
que certos end
descobriu-se q
de endereços q
variarem dimen
lação à execuçã
antecipação d
A antecipação
um bloco de 8
8 bytes. A reutilizaç
mente instruções
Conti⁽⁶⁾ atuaçõe

A reutilização
a leitura e escrita
o cache e o bus
16 acessos no

d. A "perf

A perfoma
se pode pr
Chamamo
cache. Por pr
por várias tare
trace tapes, form
o projeto e ava
descobrir a po
Além das f
que o sistema

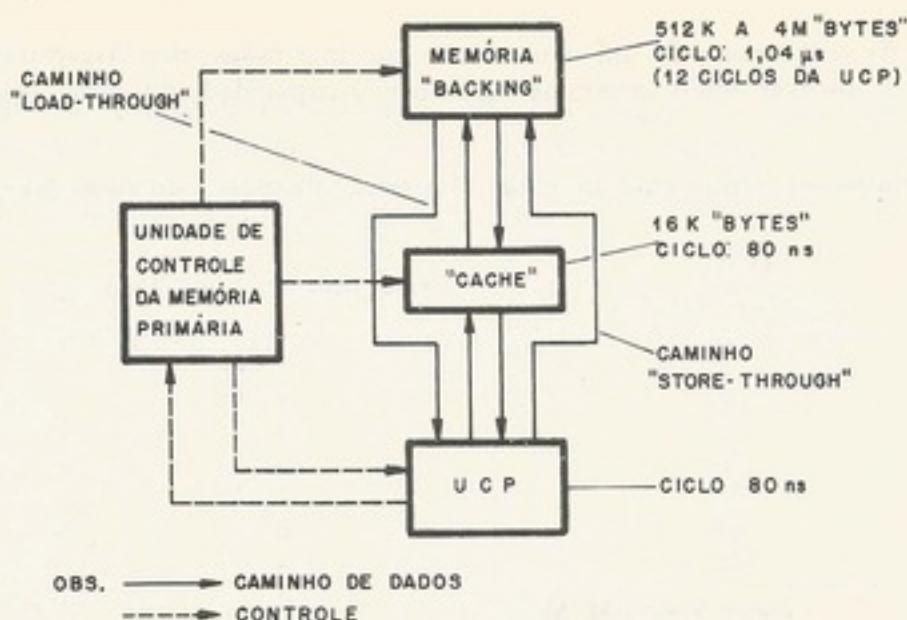


Figura 4-24. Diagrama em blocos da memória primária do IBM S/360, modelo 85

ciclo de 80 ns. Uma razão para o êxito do *cache* é resumida pela frase “vinte por cento da memória guarda oitenta por cento da informação”. Essa frase não está completamente certa, mas a idéia é que muitas informações na memória não são retiradas freqüentemente, isto é, que certos endereços têm maior probabilidade de aparecer e outros menos. Por pesquisa, descobriu-se que os endereços gerados pela *UCP* tendem a agrupar-se. Esse agrupamento de endereços é chamado de “localidade de referência”. Apesar dos endereços mais referidos variarem dinamicamente durante a execução de programas, essa variação é lenta em relação à execução de instruções. Podemos aproveitar esse fenômeno de duas maneiras: pela *antecipação* da informação e pela *reutilização* de informação retirada do *backing store* (*reuse*). A antecipação acontece no modelo 85 porque uma transferência do *backing store* é sempre um bloco de 64 bytes, enquanto que a informação requerida pela *UCP* é, no máximo, de 8 bytes. A técnica de antecipação funciona bem na seqüência de instruções. A técnica de reutilização também funciona nas instruções quando a *UCP* está executando reiterativamente instruções de um laço ou uma malha (*loop*). O fenômeno é também ilustrado por Conti⁽⁶⁾ através do exemplo que segue. O problema é somar dois vetores,

$$A_i + B_i \rightarrow A_i \quad (i = 1, 128).$$

A reutilização acontece porque cada elemento do vetor *A* é utilizado duas vezes, uma para a leitura e outra para a escrita. Também a antecipação é mostrada com transferência entre o *cache* e o *backing store* em blocos de 64 bytes; o exemplo exige 384 acessos ao *cache* e só 16 acessos ao *backing store*.

d. A “performance” do esquema

A *performance* de um sistema *cache-backing* dificilmente pode ser avaliada porque não se pode prever o tempo de acesso efetivo à memória principal.

Chamamos de *falta* ao acontecimento de não encontrar a informação procurada no *cache*. Por programação, é possível gravar uma fita com a seqüência dos endereços gerados por várias tarefas típicas rodadas num computador. Essas fitas, que Conti chamou de *address trace tapes*, foram a base de simulação de várias configurações de parâmetros do *cache* para o projeto e avaliação de *performance* do modelo 85. Assim, o programa simulador pode descobrir a porcentagem de falhas.

Além das falhas, a *performance* depende do tempo de acesso ao *backing*. Conti indica que o sistema hierárquico de um *cache* de ciclo de 80 ns com um *backing* de ciclo 1 μ s pode

atingir 80% da *performance* de um sistema que usa uma única memória principal de ciclo de 80 ns. Para observar como variam as falhas com a capacidade do *cache*, veja a Tab. 4-1.

Tabela 4-1. Distribuição de falhas (%) versus capacidade do *cache* (bytes)

Cache, capacidade (K bytes)	Faixa de distribuição de falhas (%)
2	4 - 20
4	2 - 10
8	1,8 - 5
16	1,3 - 3
32	1 - 2
64	0,6 - 1
128	0,4 - 0,5

Fonte: Robert M. Meade⁽¹⁾

É também interessante mostrar como a *performance* piora quando o tempo de acesso ao *backing* aumenta (veja a Tab. 4-2).

Tabela 4-2. Queda de *performance* com mais atraso do *backing*

Falhas (%)	Queda de <i>performance</i> (%)		
	Acesso de 6 ciclos	Acesso de 11 ciclos	Acesso de 18 ciclos
0	0	0	0
2	6	12	15
4	11	22	30
6	16	26	36
8	22	35	48

A primeira aplicação de *cache-backing* "transparente" foi o sistema Atlas, por volta de 1960. O *backing* do Atlas foi um tambor de acesso medido em milissegundos, o que representa uma ordem de mil vezes o acesso ao *cache*. Com essa diferença entre o acesso aos dois níveis, a *performance* do sistema está mais vinculada ao acesso do *backing* do que ao *cache* (ou, como chamado no Atlas, o *look-a-side*). Para aproximar a *performance* do *cache* quando o acesso ao *backing* é muito lento, a porcentagem das falhas deve ser da ordem de 0,000001%. Por exemplo, Meade⁽¹⁾ indica que, com uma hierarquia de um *cache* com ciclo de 2 µs e capacidade de 64K bytes, com um *backing* de 8 ms de acesso, então a *performance* é mais ou menos 1% da *performance* com uma única memória com ciclo de 2 µs.

e) A viabilidade do "cache"

Conti⁽⁶⁾ indica três desenvolvimentos que contribuíram para a praticabilidade da hierarquia do modelo 85: a memória monolítica de alta *performance*; técnicas de avaliação de *performance*; e algoritmos eficientes para se controlar o fluxo de informação entre o *cache* e o *backing store*.

Além da "localidade de referência" e da praticabilidade, para ser viável, a hierarquia de dois níveis precisa ser mais econômica que uma memória de um simples nível da mesma *performance* capacidade. Meade⁽¹⁾ observa que a hierarquia de memória existe por motivos de custo, pois, se a memória mais veloz fosse também a mais econômica, existiriam poucos incentivos de desenvolvimento do sistema *cache-backing store*.

f. A administração de cache

Também existe um endereço de cache que gera um endereço de cache e, se estiver em um determinado lugar. Essa



Conti⁽⁶⁾ explica que o sistema é mais efetivo, levando-o a aqui de conjunto com blocos de 64 bytes. Também a memória com 6 bits menos significa o próprio bloco. Para "duplas palavras" de 32 bits, a UCP do cache da memória. Os blocos estão agrupados pelo endereço do bloco.

Então o bloco 1, o bloco 2, ..., que o bloco 1, o bloco 2, ...

Os bits mais significativos do endereço de 24 bits dividem o bloco.

Em um cache com associatividade, exemplo, a associação entre cache e *backing store* é mostrada na figura.

principal de ciclo
veja a Tab. 4-1.

(bytes)

tempo de acesso

de 18 ciclos

adias, por volta
undos, o que re-
ce o acesso aos
lém do que ao
mance do cache
da ordem de
cache com ciclo
a performance
de 2 μ s.

idade da hie-
de avaliação
entre o cache

a hierarquia
vel da mesma
por motivos
poucos

f. A administração do "cache"

Também existe o problema do controle ou administração do *cache*. Quando a *UCP* gera um endereço de uma palavra, é necessário determinar, primeiro, se a palavra está no *cache* e, se estiver, segundo, descobrir onde; se não estiver, colocá-la no *cache* num determinado lugar. Essa tarefa é mostrada na Fig. 4-25.

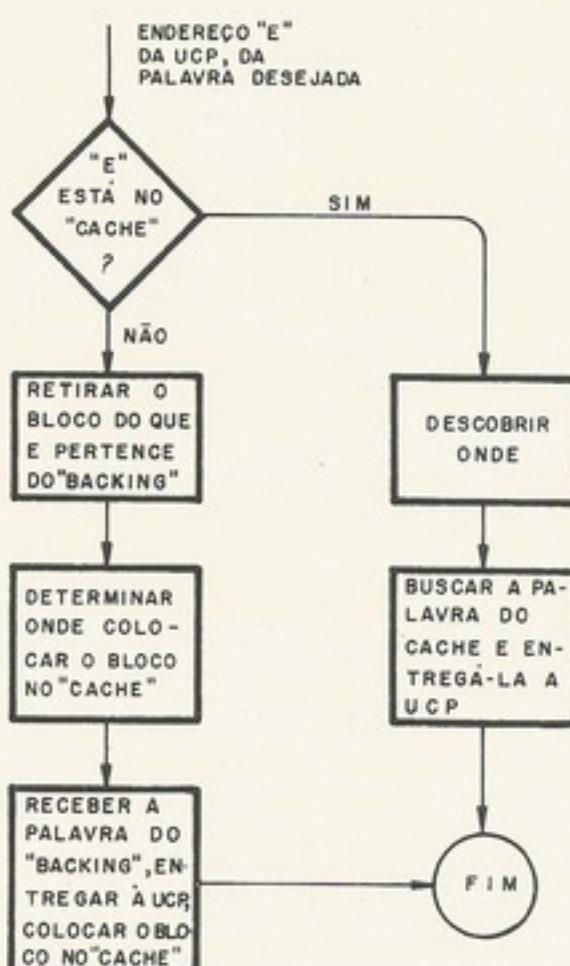


Figura 4-25. Tarefa de controle do cache

Conti⁽⁶⁾ explica vários esquemas para controlar o *cache*. Ele conclui que o esquema mais efetivo, levando-se em conta a atual tecnologia, é o do *set associative*, que chamaremos aqui de *conjunto associativo*. No exemplo seguinte (de Conti), o tamanho do *cache* é de 128 blocos de 64 bytes cada. (Segundo Conti, 64 bytes é ótimo para a arquitetura IBM S/360.) Também a memória principal é dividida em blocos de 64 bytes cada. Assim, o campo de 6 bits menos significativos do endereço indica o byte do bloco, e não é usado para endereçar o próprio bloco. Para facilitar o fluxo de bytes, o bloco de 64 bytes é subdividido em oito "duplas palavras" de 8 bytes cada, sendo 8 bytes a unidade de transferência de informação à *UCP* do *cache* do modelo 85. Os bits mais significativos do endereço indicam então o bloco. Os blocos estão agrupados em 64 conjuntos, conforme os 6 bits menos significativos do endereço do bloco.

Então o bloco "0", bloco 64, bloco 128 etc. pertencem ao mesmo conjunto, enquanto que o bloco 1, o bloco 65, o bloco 129 etc. pertencem ao mesmo conjunto.

Os bits mais significativos do endereço constituem o *tag*. A interpretação de um endereço de 24 bits dividido em quatro campos é mostrado na Fig. 4-26.

Em um *cache* com 128 blocos de capacidade cabem 2 blocos de cada conjunto. Nesse exemplo, a associatividade é dois. Assim, a relação entre os blocos do *cache* e do *backing store* é mostrada na Fig. 4-27. Essa relação também é chamada de "congruência".

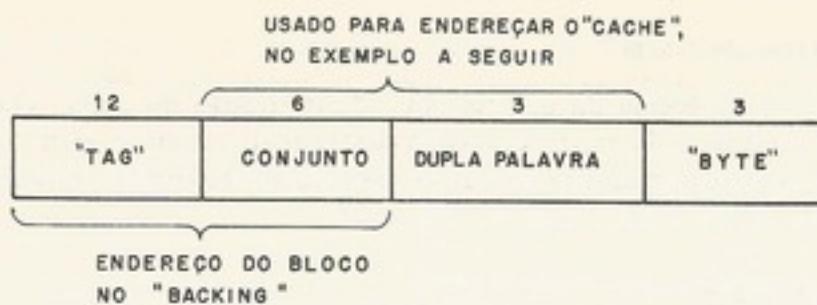


Figura 4-26. Interpretação do endereço fornecido pela UCP

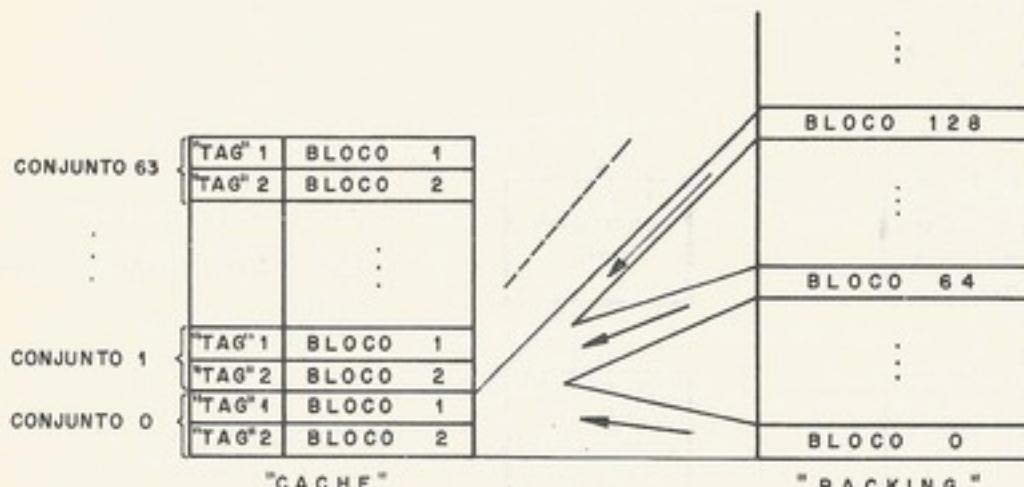


Figura 4-27. A seleção de blocos entre cache e backing

A vantagem do esquema do conjunto associativo é que, conhecendo-se o conjunto (que se sabe através do próprio campo do endereço), torna-se mais fácil descobrir se o bloco procurado está no *cache*; no exemplo, só é preciso procurar em dois lugares. (No esquema completamente associativo, o bloco procurado pode estar em qualquer lugar no *cache*). Supondo associatividade dois, o *cache* pode ser organizado para descobrir rapidamente onde fica a palavra desejada. Com uma interface de 8 bytes (ou uma "dupla palavra" segundo a arquitetura S/360) de informação, o bloco de 64 bytes pode ser dividido em oito unidades ou "módulos", a unidade do byte 0 a 7, do byte 8 a 15 etc. até a "dupla palavra" de bytes 56 a 63. A largura de acesso do módulo é de duas "duplas palavras" dois *tags*. Então a leitura do módulo retira as duas "duplas palavras" do *cache* do conjunto (segundo os bits do endereço) da "dupla palavra" procurada, mais os *tags*. O "tag" do endereço é comparado a os dois *tags* no *cache* simultaneamente. Uma comparação afirmativa seleciona a própria "dupla palavra" e a destina à UCP. Caso contrário, uma comparação negativa necessita um acesso ao *Backing store* e uma decisão sobre qual dos dois blocos no *cache* será substituído pelo bloco endereçado. A idéia é mostrada graficamente na Fig. 4-28.

g. Assuntos relacionados ao "cache"

Quando a UCP quer escrever uma palavra, existem duas filosofias; o *store-through* e a "troca". A idéia do *store-through* é sempre armazenar a palavra no *cache* (se o bloco for para lá) e no *Backing store*. Assim, uma cópia atual está sempre no *Backing store*. Na filosofia de "troca", a palavra é escrita só no *cache* (se o bloco for para lá) e, portanto, a cópia no *Backing store* pode ser desatualizada. Enquanto na filosofia de "troca", isto é, substituição de um bloco no *cache*, se o bloco a ser substituído for modificado (através do processo de escrita, ele terá

de ser escrito no
do *Backing store*, mas
um único *bus*!
Uma outra
qual dos blocos
sistema entre os

4.8 ARMAZENAMENTO

a. Componentes

Armazenamento
trole do sistema
armazenamento
conjunto de regis-
que é o estudo dos
gistros de ativida-
arquivo principal
subcampos, que
Um exemplo de
a Fig. 4-29.

Figura 4-29. C...

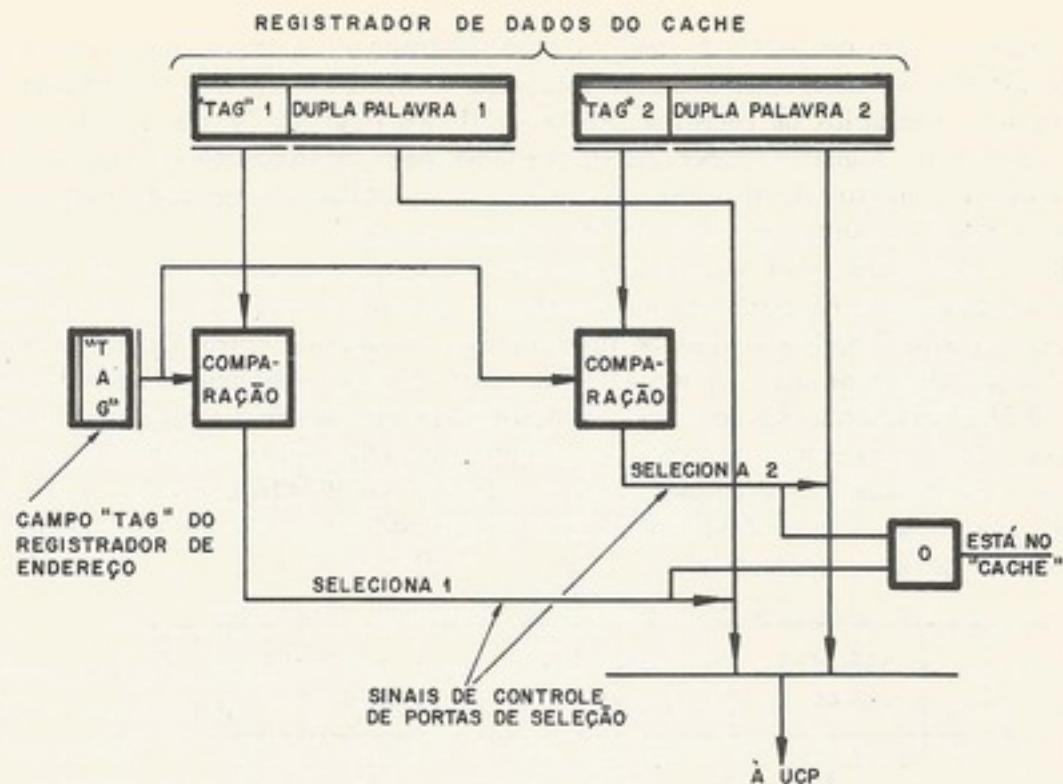


Figura 4-28. Retirando informação do cache

de ser escrito no *backing*. O método de “troca” é mais eficiente, pois diminui a atividade do *backing*, mas esse método complica a configuração em que mais de uma *UCP* partilha um único *backing*, enquanto cada *UCP* tem seu próprio *cache*.

Uma outra consideração é a do algoritmo de substituição ou reposição que determina qual dos blocos do conjunto será substituído. Felizmente as diferenças na *performance* do sistema entre os vários métodos (inclusive aleatório) são poucos.

4.8 ARMAZENAMENTO SECUNDÁRIO

a. Composição de arquivos

Armazenamento secundário (*AS*), fisicamente, consta de dispositivos que, sob o controle do sistema, podem fornecer ou aceitar dados da memória primária. Logicamente o armazenamento secundário parece um conjunto de arquivos. Um arquivo consta de um conjunto de registros. Exemplos de tipos de arquivos são o arquivo principal (*master file*), que é o estado atual (*status*) de uma lista de itens, arquivo de transações que consta de registros de atividade, usados para atualizar um arquivo principal ou arquivo histórico (um arquivo principal obsoleto). Os componentes básicos de um arquivo são os campos e os subcampos, que são posições (caracteres) vizinhas e significam uma unidade de informação. Um exemplo de um campo pode ser o de data, com subcampos para o dia, mês e ano (veja a Fig. 4-29).

Figura 4-29. Campos e subcampos



Um registro é um conjunto de campos num determinado formato relacionado a um único identificador. O campo de controle comporta o identificador, que unicamente identifica o registro. Exemplos de registros são os de dados (registro de informação ou *data record*, a maioria dos registros), *label* (registros usados para identificação do arquivo e para confiabilidade) e registros de transbordamentos (para guardar informações além do que consta do registro principal, gerados só em casos especiais).

Sendo armazenados num veículo magnético como, por exemplo, fita, ou superfície com disco ou tambor, e necessitando de movimento mecânico para sua leitura ou escrita, o bloco ou "registro físico" é a unidade normalmente transferida entre *MP* e *AS* de uma vez, e os blocos são separados por um intervalo entre registros chamados *interrecord gap* (ou *IRG*). O *IRG* representa espaço ocioso e é usado para, no caso de fita magnética, acelerar e desacelerar a fita. Para conservar espaço, economizando *IRG*, surgiu a técnica de *blocking*, em que uma quantidade, por exemplo 10, de registros "lógicos" são agrupados num bloco com só um *IRG*; veja as Figs. 4-30 a 4-31. Se os registros lógicos são lidos e escritos individualmente, chamam-se *unblocked*.



Figura 4-30. Registros em formato *unblocked*



Figura 4-31. Registros de dados em formato *blocked*

b. A técnica de "blocking"

A escolha do tamanho do bloco é importante, dependendo da quantidade de memória primária disponível para os armazenadores (*buffers*). Em muitos casos, os programadores usam três áreas de armazenador para processar registros de arquivos: armazenador de entrada, área de rascunho e armazenador de saída. O processo é mostrado na Fig. 4-32. Os registros físicos (blocos) lidos são colocados num armazenador ou área de entrada, onde o programa transfere, um por um, os registros lógicos à área do rascunho. Depois do processamento, os registros lógicos são montados num registro físico armazenador de saída. Quando a área de saída está completa, o bloco é transferido ao arquivo *AS*. Quando todos os registros lógicos da área de entrada são processados, um novo bloco é transferido ao armazenador. A técnica de desmontar o registro físico (bloco) em registros lógicos para processamento um por um chama-se *deblocking*, e a idéia de se montarem registros lógicos em blocos é chamar-se *blocking*.

c. O sistema IOCS

Normalmente o sistema de programação de um computador tem rotinas especializadas para cuidar do acionamento dos dispositivos do *AS*, dos *buffers* e do *blocking* e *de-*

blocking. O sistema IOCS, o trabalho é feito pelo programador *open endereço* do programa. Com isso, quaisquer registros (*recording*). A menor nível ao programador dispositivo, fita e disco coloca um registro transferidos sob condições comuns (rados) do versão Flores⁽⁷⁾.

Os dispositivos fita magnética se ao "próximo" isto é, com a cada registros desejados de "procura" que tenha um argumento (*search argument*)

d. Exemplo

A utilização do supor que temos de identificação. Temos também nados conforme senta um cliente gistro do argumento o programa atua no novo arquivo para procura no argumento o programa, am-

relacionado a um unicamente identificação ou *data* do arquivo e para funções além do que

fita, ou superfície de leitura ou escrita, MP e AS de uma interrecord gap magnética, acelerar a técnica de *blocking*, agrupados num bloco lidos e escritos indi-

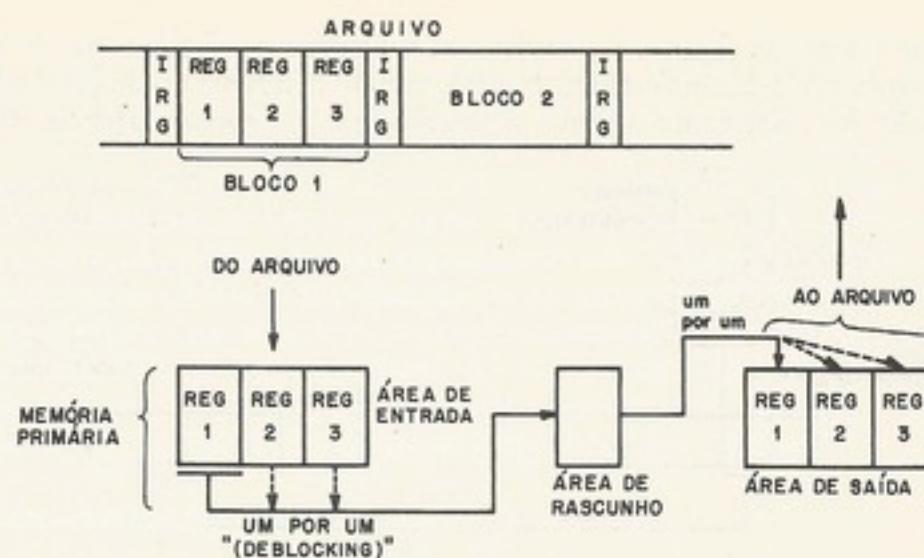


Figura 4-32. O conceito de buffering e blocking-deblocking

blocking. O nome genérico para essas rotinas é *IOCS* (input-output control system). Com *IOCS*, o trabalho do programador é bastante facilitado. Nos sistemas mais sofisticados, o programador tem disponíveis, pelo menos, quatro macroinstruções, *open*, *close*, *put* e *get*. *open* endereça o arquivo e *IOCS* o procura, verifica o *label*, e faz o arquivo disponível ao programa. Com *close*, o arquivo não é mais disponível ao programa, depois da escrita de quaisquer registros ainda não escritos, e certos comandos de controle (por exemplo, *recording*). A macroinstrução *get* retira um registro do arquivo, isto é, faz o registro disponível ao programa, ou num buffer de entrada ou numa área de rascunho. O *get* aciona o dispositivo, faz o *deblocking* e reconhece erros e condições como fim de arquivo, etc. O *put* coloca um registro no arquivo, isto é, ele faz com que dados da memória primária sejam transferidos ao *AS*. Para isso, o *put* aciona dispositivos, faz *blocking*, e reconhece erros e condições como o fim físico (no caso de fita magnética) ou falta (no caso de cartões perfurados) do veículo. Uma explicação mais detalhada de *IOCS* aparece nos Caps. 9 e 10 de Flores⁽⁷⁾.

Os dispositivos do *AS* normalmente são endereçáveis, seqüencial ou diretamente. A fita magnética só pode ser endereçada seqüencialmente, isto é, o programa só pode ter acesso ao "próximo" registro físico da fita. No caso de tambor e disco, o endereçamento é direto, isto é, com a especificação da trilha (e cilindro, no caso do disco), é possível ter acesso aos registros desejados sem ler todos os registros anteriores. Em alguns discos, existe comando de "procura" (*search*), que "verifica" todos os registros numa área até encontrar um registro que tenha um certo campo de "chave" (*key*) que compara com um dado argumento de procura (*search argument*).

d. Exemplo

A utilização de "procura" (*search*) é evidente no seguinte exemplo simples. Vamos supor que tenhamos um arquivo principal velho no disco cujos registros tenham um campo de identificação, e que o arquivo seja ordenado conforme com o campo de identificação. Temos também um arquivo de transações numa fita magnética cujos registros são ordenados conforme com o campo de identificação. Cada registro do arquivo principal representa um cliente de um banco e o campo de identificação é o número da conta. Cada registro do arquivo de transações representa uma transação durante o dia. Nesse exemplo, o programa atualiza o registro da conta do cliente, conforme as transações, formando um novo arquivo principal. O programa, começando com o primeiro registro de transações, procura no arquivo principal o registro com o mesmo campo de identificação. Achando o programa, atualiza o registro do arquivo principal. Agora o programa continua esse

processo com o seguinte registro do arquivo de transações, até que todos os registros do arquivo de transações estejam tratados. Nesse exemplo, o argumento de procura é o campo de identificação do registro do arquivo de transações. O processo aparece na Fig. 4.32.

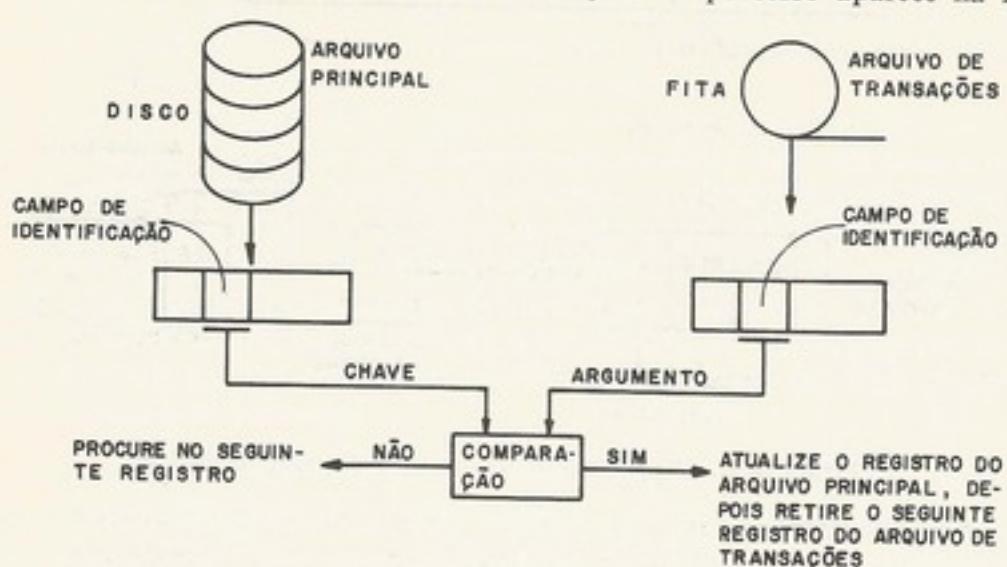


Figura 4-33. Exemplos da aplicação da função procura.

4.9 MEMÓRIA VIRTUAL

a. Motivació

Desde a primeira geração de computadores, cabe ao programador cuidar da memória disponível a ele. Quando a totalidade de informações e programas excedia a capacidade da memória principal, o programador tinha de dividir o programa em segmentos e aplicar a técnica de *overlay*, ou seja, substituir um segmento pelo outro. O problema de locação de armazenamento é o de determinar como a informação deve ser distribuída. Esse problema pertence tanto ao programador usuário quanto ao sistema operacional.

A memória virtual serve para resolver o problema de se administrar a memória principal física, enquanto aumenta o espaço que pode ser endereçado. Blocos adjacentes na memória virtual não precisam ser adjacentes na memória real (física). O problema de *overlay*, tal como o de relocação, é feito automaticamente pelo mecanismo da memória virtual. O sistema lota a memória conforme com as necessidades dinâmicas e não na base de quanta memória o programador pensa que seu programa vai precisar. Em meados da década de 60, computadores de grande porte, como o IBM S/360 modelo 67, CDC 7600, Burroughs B6500 e GE 645, tinham aplicado idéias para atingir a memória virtual.

b. Transcodificação dinâmica de endereço

A memória virtual tem semelhanças com o esquema *cache-backing*. Segundo Conti⁽⁶⁾, as razões do êxito do *cache-backing* do modelo 85 também se aplicam ao problema de armazenamento em geral. O mecanismo de *transcodificação dinâmica do endereço (dynamic address translation)* para implementar a chamada “memória virtual” aproveita o fenômeno de localidade de referência, embora não exista uma memória específica chamada *backing*. Ambas as técnicas, *cache-backing* e transcodificação dinâmica de endereço, são “transparentes”, no sentido de que o mecanismo é “escondido” do programador usuário. A memória primária parece que é de um único nível (*single-level storage*).

Para entender a memória virtual, é necessário definir, relativamente à memória primária, o “espaço de endereçamento” ou “espaço de nomes” em contraposição ao “espaço de memória”, ou seja, as localizações físicas da memória. É a transcodificação dinâmica

de endereço que permite essa distinção. Por exemplo, a arquitetura do IBM S/360 dispõe de 24 bits de endereço, podendo endereçar 16M bytes de memória primária, enquanto que nenhum dos modelos usam tanta memória. (A máxima capacidade no modelo 40, por exemplo, é de 256K bytes). Em sistemas sem memória virtual, se um programa tentar endereçar localizações, na memória primária, que não existem, isso será tratado como falha e o programa será terminado por interrupção. Com memória virtual, o "nome" é o endereço virtual e não é necessário que esses identificadores de informação (os endereços efetivos das instruções) correspondam a uma localização na memória real. Assim, se a arquitetura permitir endereçamento até 16M byte, o programador poderá escrever os seus programas para uma memória virtual desse tamanho, isto é, como se tivesse 16M bytes disponíveis.

Aqui encontramos uma diferença com *cache-backing*, pois, no sistema *cache-backing*, os endereços têm de corresponder às localizações físicas do *backing*.

c. O mecanismo de transcodificação

É o mecanismo que permite a existência da memória virtual, e que faz dinamicamente a transcodificação do endereço efetivo à localização física. O processo é também conhecido como "relocação dinâmica". Basicamente, temos o espaço de endereço ou "nomes" (N), com endereços $N = (0, 1, \dots, n - 1)$ e o espaço físico de localizações da memória $M = (0, 1, \dots, m - 1)$ onde $m \leq n$. Notamos que cada nome não precisa representar alguma coisa. A função f , que relaciona os nomes com o espaço físico da memória, é geralmente implementada na arquitetura através de *table look-up*, isto é, através de acesso a uma tabela.

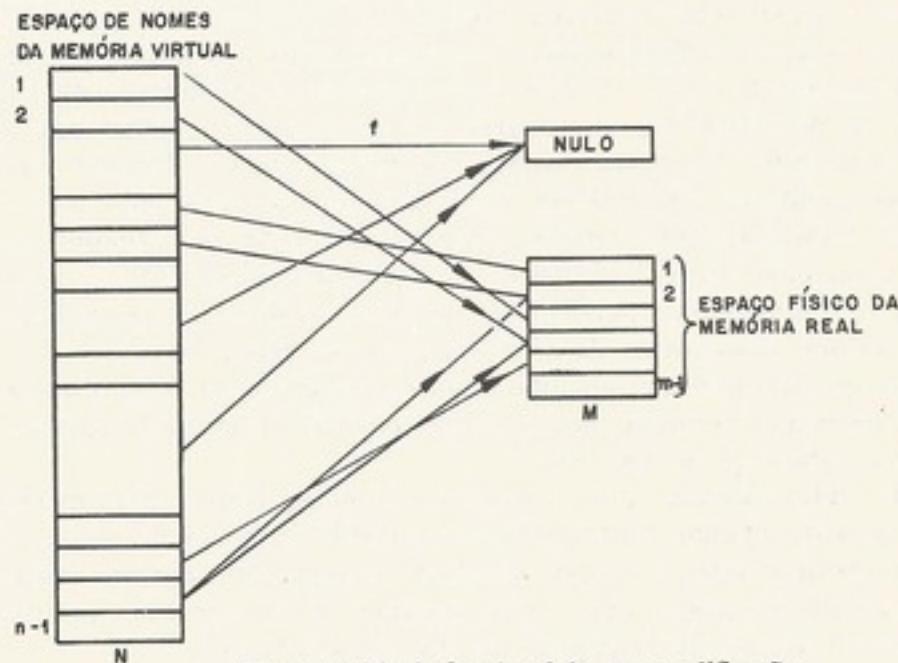


Figura 4-34. A função f de transcodificação

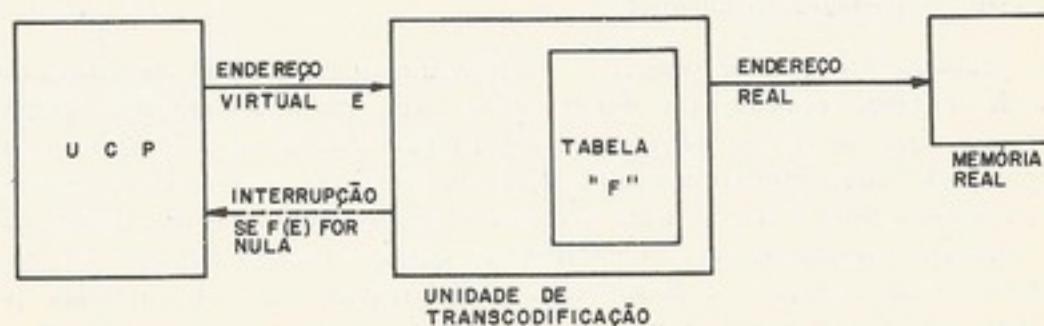


Figura 4-35. Dispositivo de transcodificação

Também acontece que, na memória M , não cabem todos os "nomes" e, então, certos endereços não terão localização correspondente na memória real. Nesse caso, a função é nula. Graficamente, a função f é como na Fig. 4-34. Em linhas gerais, a função é implementada como na Fig. 4-35.

Se o endereço virtual não estiver na memória real (uma falha), haverá uma interrupção no programa supervisor. Daí, o programa supervisor descobre qual endereço virtual provocou a falha. Com essa informação, o supervisor procura numa outra tabela, para descobrir a localização da informação desejada, isto é, o endereço virtual correspondente a um lugar no armazenamento secundário de acesso direto (disco ou tambor). A tarefa do supervisor é trazer a informação do armazenamento secundário à memória primária. Aqui encontramos uma outra diferença importante entre o esquema *cache-backing* e memória virtual. No caso de *cache-backing*, uma falha não necessita de uma procura para localizar a informação desejada (o endereço já é do *backing*) e, mais importante ainda, não necessita de um cruzamento da fronteira entre memória primária e armazenamento secundário. A semelhança com *cache backing* é que agora o supervisor tem de decidir onde colocar a nova informação que vem entrando na memória primária; o algoritmo que resolve isso é o algoritmo de reposição (*replacement algorithm*).

d. O tamanho dos blocos ou páginas

Para facilitar esse movimento de informação entre a memória primária e o armazenamento secundário, a memória virtual é dividida em blocos de endereços contíguos, chamados páginas, e a memória primária é dividida em "molduras" de páginas (*page frames*). A página é uma unidade de informação e transmissão entre a memória primária e o armazenamento secundário. Por isso, essa técnica de implementar memória virtual será chamada de "paginação de demanda" (*demand paging*) ou, simplesmente, "paginação". Para fins de aproveitar eficientemente a procura e acesso do armazenamento secundário, o tamanho da página deve ser grande, na ordem de 4K bytes ou mais. Por outro lado, para melhor aproveitar e utilizar a memória real, a página deve ser menor. Se, por exemplo, entra uma página de 4K bytes na memória real, é possível que o programa só necessite de 2K bytes daquela página. Assim, se o tamanho da página fosse de 256 bytes ou 512 bytes, poderia ter tomado só 2K bytes (ou um pouco mais) da memória real em vez de 4K bytes.

Uma coisa que favorece um tamanho pequeno é a chamada "fragmentação". Um programa de 10K bytes, por exemplo, precisa de três páginas de 4K bytes cada ou um total de 12K bytes, um desperdício de 2K bytes.

Não existe muita pesquisa sobre o tamanho ótimo da página, para melhor aproveitar a memória real, mas, segundo Denning⁽⁸⁾, esse tamanho é menos de 256 bytes. Por outro lado, para melhor aproveitar o tempo de acesso ao armazenamento secundário, Denning indica que o tamanho da página deve ser, no mínimo, 2K bytes, e que é por isso que os sistemas de memória virtual usam páginas de 2K ou 4K bytes.

e. Tabelas de paginação de um nível

Uma primeira tentativa de projetar a arquitetura do mecanismo da transcodificação dinâmica de endereço (relocalização dinâmica) foi usar uma única tabela de páginas, conhecida como tabela de página de um nível. O sistema Atlas foi assim⁽⁹⁾. Num sistema de multiprogramação, cada programa usuário tem sua tabela particular, isto é, cada usuário tem sua memória virtual. Um problema com isso é que os programas não podem partilhar páginas. Também, programas como compiladores são sensíveis ao tamanho do programa a ser compilado; se fosse para se deixar lugar para a maior "tabela de símbolos" possível, deixar-se-iam buracos na tabela de páginas para a maioria dos usuários. Além do mais, a tabela de páginas necessita de blocos contíguos da memória primária. Uma memória virtual

de 16M bytes, página necessária páginas de 4

f. Tabelas

Esses problemas uma tabela de nível é muito útil. O segmento associado (veja Denning).

Para cada nível o "nome" da página de "segmentação" (*modular programação*) permite que a memória multiprogramada seja segmentada.



de 16M bytes, com páginas de 4K bytes, necessita 16K bytes para essa tabela porque cada página necessita 4 bytes da tabela e todas as 4 096 páginas possíveis são utilizadas (4 096 páginas de 4 bytes cada faz 16K bytes).

f. Tabelas de paginação de dois níveis

Esses problemas são resolvidos pela técnica de tabelas de página de dois níveis, ou seja, uma tabela de "segmentos" que indica a própria tabela de páginas. A idéia de segmentos é muito útil. O espaço da memória virtual é visto como um conjunto de segmentos, cada segmento sendo de tamanho variável e constando de uma seqüência de endereços em seguida (veja Dennis⁽¹⁰⁾).

Para endereçar informação num espaço de segmentos, basta fornecer dois componentes, o "nome" do segmento e o endereço da palavra desejada dentro do segmento. A técnica de "segmentação" facilita a técnica de organizar programas como um conjunto de módulos (*modular programming*); em cada módulo fica um segmento. A segmentação também permite que a informação partilhada entre dois programas *A* e *B*, rodando num sistema multiprogramado, bastando que programas *A* e *B* tenham um segmento em comum. A segmentação também facilita o tratamento de tabelas de tamanho arbitrário, como a tabela

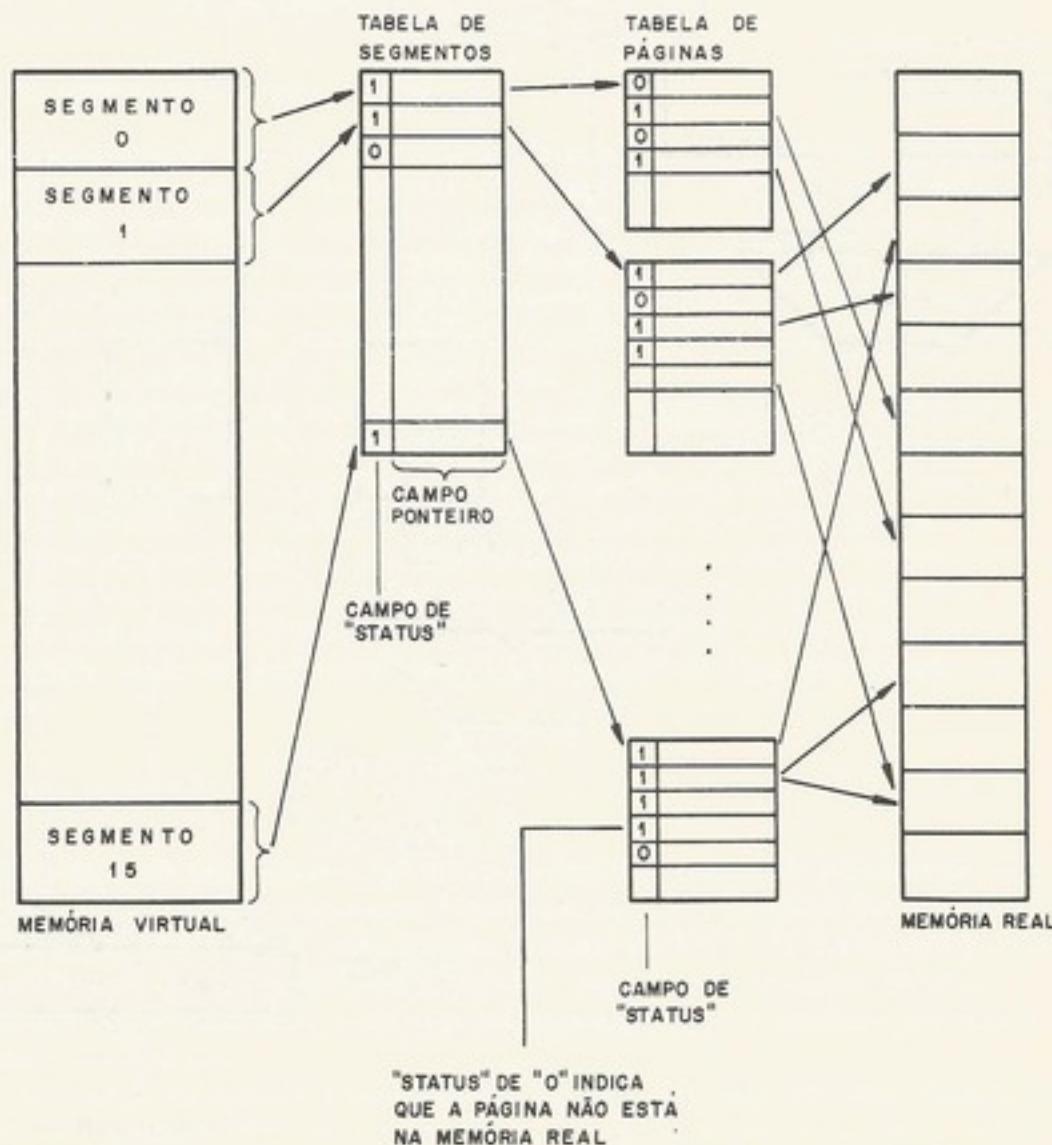


Figura 4-36. Estrutura de tabela de páginas de dois níveis

de símbolos (*symbol table*) de um programa compilador. As tabelas de páginas de dois níveis podem ser vistas como um casamento entre segmentação e paginação de um simples nível. A memória virtual consta de um certo número de segmentos dentro do qual o endereçamento é linear, mas dividido em páginas. Para a transcodificação do endereço de dois componentes da memória virtual, é preciso usar o primeiro componente (nome do segmento) para descobrir a localização da tabela de páginas para aquele segmento. Depois, o segundo componente é usado na tabela de páginas para descobrir o endereço na memória real. O endereçamento, assim que divide a memória em um conjunto de espaços lineares, é chamado de "multilinear". O processo de transcodificação de endereços, é visto na Fig. 4-36.

Como nas tabelas de um nível só, o relacionamento para o segmento pode ser "nulo", indicado através de um campo de *status* ou um *tag* na tabela. Em tal caso, uma interrupção no programa supervisor é provocada. No S/360 modelo 67, o endereço virtual é de 24 bits dividido em três campos: segmento de 4 bits, página de 8 bits e byte de 12 bits. Então a memória virtual de cada usuário tem, no máximo, 16 segmentos e cada segmento tem, no máximo, 256 páginas. A página é de 4K bytes. (veja a Fig. 4-37).

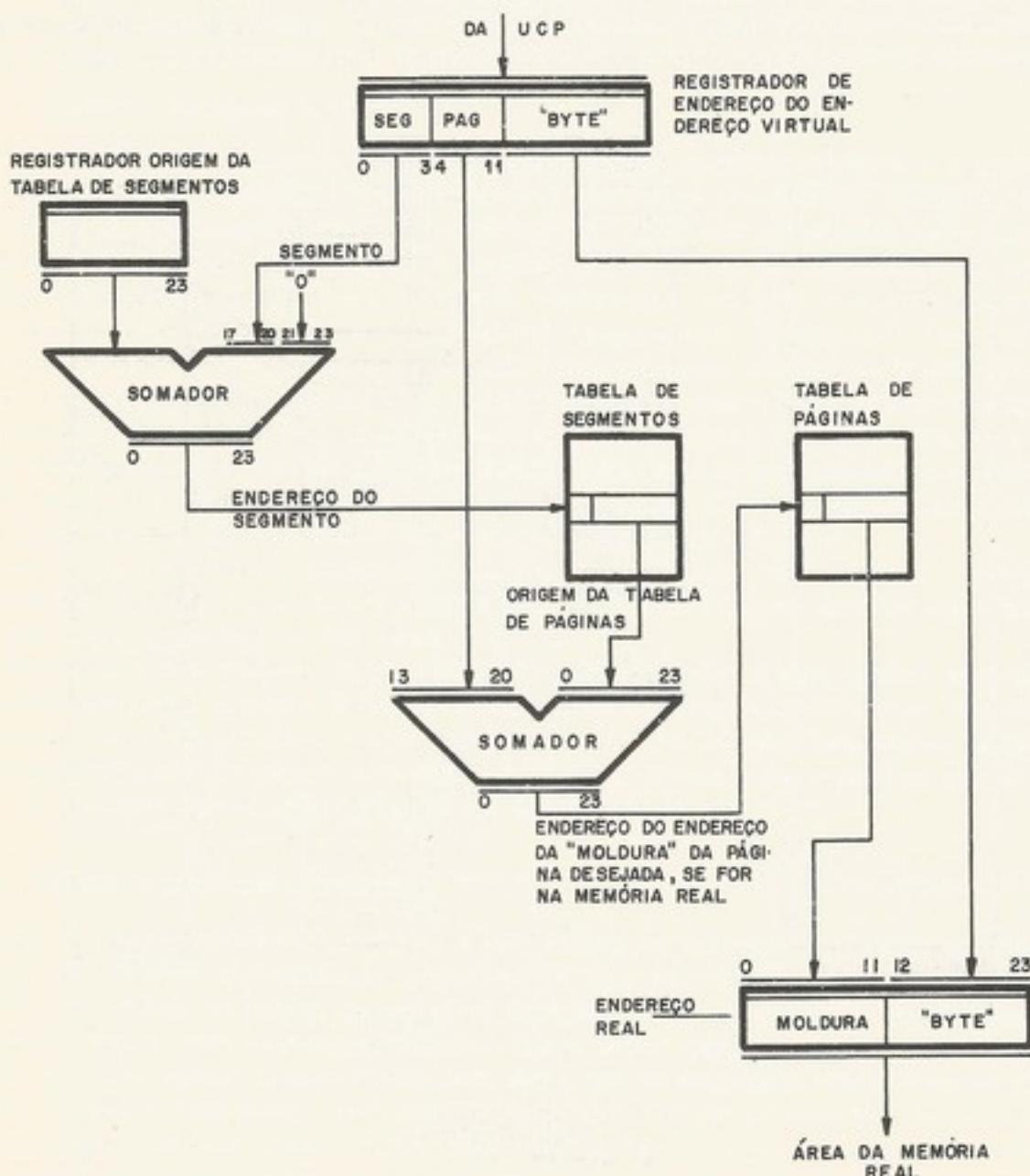


Figura 4-37. Descobrindo o endereço real, arquitetura de relocalização dinâmica de S/360 modelo 67

memória e comunicação

g. Implementação

Como o problema de armazenamento na UCP (como os endereços tirar a informação de uma página; e, tentar obter a performance mais baixa, comparado ao que foi resolvido no modelo 67, na unidade das oito páginas da unidade, os 12 bits com os 12 bits usual), o registro de memória real. As palavras não são há falha na memória principal para ter as

h. Experiências

Denning disse que é de aplicar o mecanismo de paginação entra no software. Não ao comportamento de localidade de que utilize a memória.

Uma das aplicações é o *sharing*. Segundo Denning bem e o conceito de memória com a mesma quando e por que o programa escaladou do sistema operacional xível, também pode ser da tabela.

4.10 SUMÁRIO

Neste capítulo, o armazenamento de fitas e, na parte cima para servir é a interface entre armazenamento e processamento.

A tecnologia de diferenças em sua organização da memória não é sempre usada.

g. Implementação de relocalização dinâmica no S/360 modelo 67

Como o processo de transcodificação de endereço é dinâmico e como as tabelas estão armazenadas na memória primária, é possível pensar-se que cada endereço gerado pela UCP (como os endereços dos operandos) levará três ciclos da memória primária para retirar a informação desejada: primeiro, ler a tabela de segmento; segundo, ler a tabela da página; e, terceiro ler a informação desejada (processo mostrado na Fig. 4-37). Se fosse isso, a performance interna da UCP (em termos de instruções executadas por segundo) seria muito baixa, comparada com um sistema sem relocalização dinâmica. O problema de performance foi resolvido tirando-se vantagem do princípio de localidade de referência. No S/360 modelo 67, na unidade de relocalização dinâmica, há, armazenada em registradores, uma tabela das oito páginas mais recentemente transcodificadas. Quando a UCP fornece um endereço à unidade, os 12 bits mais significativos (os campos de segmento e página) são comparados com os 12 bits correspondentes dos oito registradores. Se não houver falta (a situação usual), o registrador que comparou certo já fornece o endereço da própria "moldura" na memória real. Essa pequena memória (de 8 palavras) é do tipo "associativo", porque as palavras não são endereçadas por sua localização, mas na base de seu conteúdo. É só quando há falha na memória associativa que a unidade precisa fazer os três ciclos da memória principal para ter acesso à informação desejada.

h. Experiência com memória virtual

Denning distingue entre o mecanismo de implementar memória virtual e a política de aplicar o mecanismo. No nosso caso, o mecanismo é a relocalização dinâmica e a política entra no algoritmo de reposição das páginas e em outros aspectos administrativos do software. Na realidade, o comportamento do sistema de memória virtual é muito sensível ao comportamento de programas. Um programador que não dá importância ao princípio de localidade de referência pode facilmente escrever um programa não bem organizado e que utilize a memória virtual inefficientemente.

Uma das aplicações da memória virtual é em sistemas de tempo compartilhado (*time sharing*). Segundo Lett e Konigsford⁽¹¹⁾ o princípio de relocalização dinâmica funciona bem e o conceito é bom. O problema existe geralmente não tanto com a memória virtual, mas com a estratégia de escalada (*scheduling*), ou seja, fixar quais programas vão rodar, quando e por quanto tempo. É geralmente concedido ao algoritmo do *scheduler*, ou programa escalador, responder dinamicamente à atual situação e, por esse meio, o *scheduler* do sistema operacional TSS/360 é dirigido por uma tabela. O algoritmo, além de ser flexível, também pode ser facilmente modificado através de uma modificação do conteúdo da tabela.

4.10 SUMÁRIO

Neste capítulo estudamos a estrutura de armazenamento de sistemas de computação. O armazenamento é visto como uma hierarquia, na parte de baixo ficam as bibliotecas de fitas e, na parte de cima, os registradores centrais da UCP. As informações passam para cima para serem processadas. Essa hierarquia possui duas barreiras naturais; a primeira é a interface entre o homem e o sistema, e a segunda é entre a memória primária e o armazenamento secundário.

A tecnologia da memória primária foi estudada em mais detalhes, iluminando as diferenças em aplicação entre memórias de núcleos e memórias monolíticas. As técnicas de organização da memória primária foram mostradas, e notou-se que a memória primária não é sempre uma simples memória principal. Esse último ponto foi demonstrado com a

descrição de sistemas com *LCS* (memória em grande quantidade muito mais lenta) e com o sistema *cache-backing*. A importância da "localidade de referência" dos endereços foi acentuada.

Os conceitos básicos da organização de arquivos foram dados, bem como os métodos de acesso ao armazenamento secundário. Uma ponte através da barreira natural entre memória primária e armazenamento secundário, do ponto de vista do programador, é a relocação dinâmica e os esquemas de segmentação e paginação. Essas técnicas gozam das vantagens de "localidade de referência", mas as diferenças com *cache-backing* foram expressadas também.

EXERCÍCIOS

- 4-1. Explicar a diferença (se existir) entre memória principal e memória primária.
- 4-2. Qual é o *bandwidth* de uma fita magnética com largura de um byte, densidade de 800 bits por polegada, e velocidade de fita de 120 polegadas por segundo?
- 4-3. Suponha que um sistema dispõe de três canais (chamados 0, 1 e 2) de E/S que podem operar simultaneamente através de *cycle steal*, e uma memória principal de largura um byte e ciclo de 2 μ s (500 000 bytes por segundo). Cada canal tem armazenador (*buffer*) de 1 byte. Os canais 0 e 2 não têm dispositivos com velocidade maior que 25 000 bytes por segundo. (a) Se nenhum canal tiver prioridade sobre o outro (o primeiro canal a pedir interrupção para transferir e o aceito), poderemos colocar um disco com velocidade de 450 000 bytes por segundo no canal 1 sem perigo de *overrun* no canal 1? (b) Repetir (a) se o canal 1 tiver prioridade sobre os outros. (c) Repetir (a) (sem prioridade) se cada canal tiver 2 bytes de armazenador.
- 4-4. Discutir a sentença: "não há sentido em operar-se uma memória monolítica de uma maneira que não o ciclo rachado porque a memória é do tipo 'ler' não-destrutivo".
- 4-5. Suponha que tenhamos disponíveis pastilhas de matriz de 1 bit por 1 024 palavras. Mostre em blocos como organizar uma memória de 4K palavras de 8 bits por palavra. Inclua *RE* e *RD*.
- 4-6. Por que não é econômico o esquema *cache-backing* em sistemas de baixo porte?
- 4-7. Num sistema *cache-backing*, o *cache* é de 1K bytes, organizado em 64 palavras de 16 bytes cada. A interface da *UCP* é de 8 bytes. O ciclo do *backing* é 10 ciclos do *cache*. Tendo em vista a economia, projete o tamanho do bloco, a largura da interface entre o *cache* e o *backing*, e os demais detalhes para o esquema conjunto associativo.
- 4-8. Qual é a diferença entre o registro físico e o registro lógico?
- 4-9. Quais são as diferenças e semelhanças entre a memória virtual e o esquema *cache-backing*.
- 4-10. Quais são as vantagens das tabelas de relocação de dois níveis sobre um nível?
- 4-11. Suponha que uma arquitetura disponha de 16 bits de endereço. Será que vale a pena implementar memória virtual nessa arquitetura? Discutir.
- 4-12. A maioria de sistemas com paginação tem páginas de 2K a 4K bytes. Quais seriam as vantagens e desvantagens de um tamanho de página de 8K bytes?

BIBLIOGRAFIA

- (1) Robert M. Meade, "On Memory System Design", 1970 Fall Joint Computer Conference, *Proc. AFIPS*, Vol. 37, pp. 33-43
- (2) A. W. Heineck e R. R. Kronian, "Integrated System Throughput Design for High-Speed Processors", Fall Joint Computer Conference 1967, *Proc. AFIPS*, Vol. 31. (Também relatório TROO.1568 de 2 de junho de 1967, IBM Laboratory, Poughkeepsie, New York)
- (3) F. P. Brooks e K. E. Iverson, *Automatic Data Processing*, segunda edição, John Wiley, 1970
- (4) J. S. Liptay, "The Cache", *IBM Systems Journal*, Vol. 7, n.º 1, pp. 15-21, 1968
- (5) L. L. Vadasz, "Semi-conductor Random Access Memories", *IEEE Spectrum*, Vol. 8 pp. 40-48, maio de 1971
- (6) C. J. Conti, "Concepts for Buffer Storage", *Computer Group News*, Vol. 2, n.º 8, pp. 9-13, março de 1969

- memória e armazenamento
- (7) Ivan Flores
 - (8) Peter J. Denning, 1970
 - (9) T. Kilburn et al., pp. 225-236
 - (10) J. B. Dennis, *of the ACM*
 - (11) A. S. Lett et al., *Conference*

- (7) Ivan Flores, *Computer Software*, Prentice-Hall, 1965
- (8) Peter J. Denning, "Virtual Memory", *Computing Surveys*, Vol. 2, n.º 3, pp. 153-189, setembro de 1970
- (9) T. Kilburn *et al.*, "One-level Storage System", *IRE Transactions on Computers*, Vol. EC-11, n.º 82, pp. 223-235, abril de 1962
- (10) J. B. Dennis, "Segmentation and the Design of Multiprogrammed Computer Systems", *Journal of the ACM*, Vol. 12, n.º 4, pp. 589-602, outubro de 1965
- (11) A. S. Lett e W. L. Konigsford, "TSS/360, a time-shared operating system", 1968 Fall Joint Computer Conference, *Proc. AFIPS*, Vol. 33, pp. 15-28

capítulo 5

EXEMPLO DE UM MINICOMPUTADOR

5.1 INTRODUÇÃO

Para melhor explicar como se projetam sistemas digitais, neste capítulo usaremos como veículo didático um minicomputador cuja arquitetura foi definida, no primeiro semestre de 1971, pelos estudantes de pós-graduação do curso PEL-727, "Projeto de sistemas digitais", da Escola Politécnica da USP, projetado e implementado no Laboratório de Sistemas Digitais do Departamento de Engenharia de Eletricidade. Este capítulo também tem por finalidade dar uma introdução aos vários assuntos de arquitetura, tais como o formato e os campos da instrução, o formato dos dados e o endereçamento dos operandos pelas instruções. Será examinado em detalhes o projeto da arquitetura geral, o conjunto de instruções, o fluxo de dados, a unidade aritmética, o ciclo da máquina, as fases, a unidade de controle, o esquema de interrupção e os controles manuais.

5.2 ESBOÇO DA ARQUITETURA

Um vínculo do projeto foi a memória principal, o modelo FI-21 da Philips, já disponível. A memória tem ciclo de 1,6 µs, largura de 8 bits, e capacidade em quantidades de 1K (1 024) palavras. Assim, o tamanho da palavra ficou 8 bits ou seja, 1 byte. Para muitos fins, o tamanho de 1 byte é razoável. Como 8 bits já é um carácter de código *ASCII*, um processador de 8 bits serve para receber, guardar e retransmitir mensagens entre terminais do teclado impressora ou entre computadores. Chama-se essa aplicação de chaveamento de mensagens (*message switching*). O tamanho de 8 bits também serve para concentrador de dados (*data concentrator*). O concentrador regula o fluxo de mensagens em tempo real entre um computador e um conjunto de terminais, usando o computador em tempo partilhado (*time-sharing*). Uma outra aplicação que usa 8 bits é o teclado à fita (*key-to-tape*), em que informações para um computador são montadas diretamente de um teclado a uma fita magnética.

A palavra, então, ficou de 8 bits. Para fins aritméticos, foi definida a representação de negativos como a complementação de dois. O formato da instrução é do tipo *endereço simples*, isto é, a instrução só tem um campo de endereço. Quando a operação (definida pelo código da operação na instrução) precisa de dois operandos, como soma, por exemplo, o outro operando é um registrador central chamado *acumulador*. O acumulador é de 8 bits, com mais um *flip-flop* chamado *V* para guardar o último "vai um transbordo".

Com 8 bits, só podemos representar números de 0 a 255 ou, no caso de representação de negativos, de -128 a +127. Para facilitar o aumento da precisão, a palavra "1" da memória foi designada extensão do acumulador, e existe uma instrução para trocar o acumulador e a extensão. Assim, sub-rotinas podem ser programadas para executar aritmética em dupla precisão de 16 bits.

No inicio do projeto, tentou-se utilizar no máximo 1K (1 024) bytes da memória principal, que só necessita de 10 bits de endereço, mas ficou evidente que era viável prover um

campo de endereço de 11 bits, que é o que é usado.

A técnica de endereçamento é a mesma usada no projeto do operando com 16 bits. Modificando o campo de endereço de 11 bits para 12 bits, é possível aumentar o conteúdo da memória para 4 096 bytes. As palavras, o endereço e o campo de endereço são divididos em 8 bits cada. A arquitetura, o endereçamento e o campo de endereço devem ser de 8 bits, enquanto o campo de endereço é dividido em 11 bits. A técnica de endereçamento é a mesma usada no projeto do operando com 16 bits. Modificando o campo de endereço de 11 bits para 12 bits, é possível aumentar o conteúdo da memória para 4 096 bytes. As palavras, o endereço e o campo de endereço são divididos em 8 bits cada. A arquitetura, o endereçamento e o campo de endereço devem ser de 8 bits, enquanto o campo de endereço é dividido em 11 bits.

Na Fig. 5-1, é mostrado o fluxo de dados entre o processador e os registradores de endereço. O concentrador de dados é implementado com apenas um byte.

campo de endereço de 12 bits, já que fora resolvido usarem-se instruções longas de duas palavras. Com 16 bits disponíveis, quatro foram usados como código de instrução e doze de campo de endereço e, assim, a capacidade da memória principal passou para 4K bytes.

A técnica de indexação é, basicamente, a formação do endereço efetivo (o endereço real do operando) como a soma do campo de endereço e o conteúdo de um registrador indexador. Modificando o conteúdo do indexador, favorece a técnica de laços (*loops*) no programa. Um outro tipo de endereçamento chama-se *indireto*; o endereço efetivo do operando é o conteúdo da localização, indicado pelo campo de endereço da instrução. Em outras palavras, o endereço que vem junto com a instrução indica o ponteiro para o operando. Nessa arquitetura, o endereçamento indireto é complicado, devido ao fato de o conteúdo da palavra ser de 8 bits, enquanto o endereço precisa ser de 12 bits. Selecionou-se um sistema de endereçamento indexado. O esquema é restrito, no sentido de ser o índice de apenas 8 bits. Assim, a técnica do uso do indexador como ponteiro para qualquer palavra na memória não funciona porque, com 8 bits, só podemos indicar uma de 256 posições. Para economizar em registradores centrais, o registrador indexador ficou na posição "0" da memória. Assim, as instruções para carregar ou retirar palavras da memória também são usadas para carregar o indexador.

Na Fig. 5-1, o leitor encontra um diagrama simplificado da estrutura interna, ou seja, o fluxo de dados da UCP, ressaltando apenas os caminhos e paradas importantes. Dois registradores trabalham com a memória principal, o registrador de dados (RD) e o registrador de endereço (RE). O registrador da instrução (RI) guarda as instruções curtas de apenas um byte, ou o primeiro byte das instruções longas, isso para controlar as suas exe-

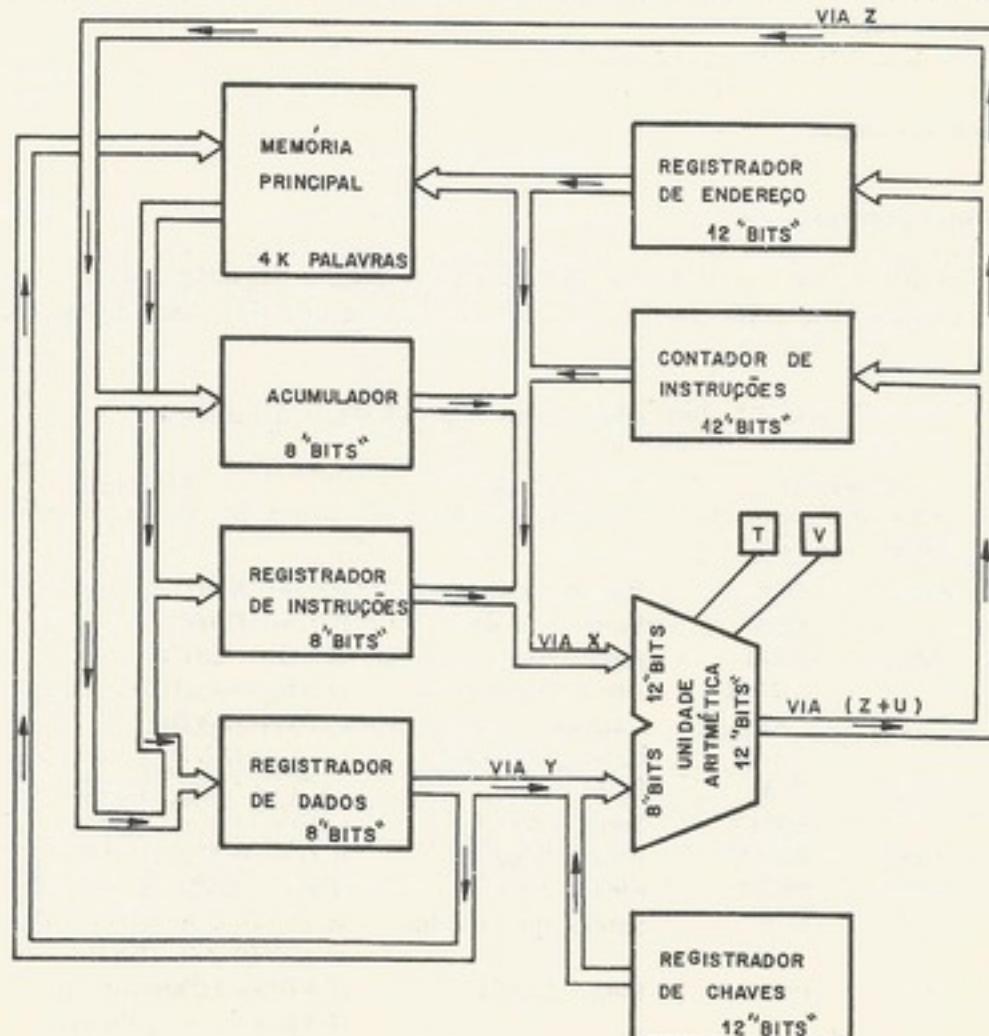


Figura 5-1. Fluxo de dados simplificado

ções, pois é no primeiro byte que está o código de operação. O conteúdo do contador de instrução (*CI*) é o ponteiro que mostra a localização, na memória, da próxima instrução. O acumulador (*AC*) é o registrador cujo conteúdo é controlado pelo programador através das instruções.

5.3 INSTRUÇÕES PRINCIPAIS

Utilizando uma memória com palavra de 8 bits, ficou definida a largura das vias e circuitos tratadores de dados (somador, portas de seleção, registradores etc.). Escolheu-se a representação dos números como o código complemento de dois, dadas as suas vantagens na realização das operações aritméticas. Fizeram-se as vias de endereçamento dos 12 bits necessários para o endereçamento direto das 4K palavras da memória.

As instruções com referência à memória são do tipo endereço simples, com o segundo operando operando no acumulador (registrador de 8 bits). Essas instruções são longas, ou seja, utilizam duas palavras (16 bits) já que se necessita de 12 bits para endereçar-se o primeiro operando. A maioria das instruções sem referência à memória é de largura de uma palavra (8 bits), pois não se precisa de campo de endereço.

Durante a fase de projeto lógico utilizou-se um conjunto de siglas mnemônicas para especificar as instruções. Terminando o projeto, tendo em vista a preparação dos programas ao *assembler*, mudaram-se as siglas mnemônicas para outros códigos melhores. Por isso, ver-se-á, na descrição a seguir, para cada instrução, duas siglas mnemônicas: a da primeira coluna (inicial), que aparecerá nos gráficos dos circuitos é a sigla usada na fase inicial do projeto; e a segunda coluna (definitivo) é a sigla definida para utilização pelo programador da máquina, na sua versão final.

a. Instruções longas

a.1. Grupo principal

Instruções longas são aquelas que têm o comprimento de duas palavras. A maioria delas tem o formato mostrado na Fig. 5-2. As instruções que têm esse formato são vistas na Tab. 5-1.

Tabela 5-1. Instruções longas com referência à memória

Código	Mnemônico		Descrição	Operação
	Inicial	Definitivo		
0000	PLI	PLA	Pulo incondicional	$(CI) \leftarrow END$
0001		PLAX	Pulo indexado	$(CI) \leftarrow END_{ef}$
0010	ARA	ARM	Armazena	$(END) \leftarrow (AC)$
0011		ARMX	Armazena indexado	$(END_{ef}) \leftarrow (AC)$
0100	CAC	CAR	Carrega	$(AC) \leftarrow (END)$
0101		CARX	Carrega indexado	$(AC) \leftarrow (END_{ef})$
0110	SOM	SOM	Soma	$(AC) \leftarrow (AC) + (END)$
0111		SOMX	Soma indexada	$(AC) \leftarrow (AC) + (END_{ef})$
1010	PAN	PLAN	Pula se negativo	$(CI) \leftarrow (END)$ se $AC < 0$
1011	PAZ	PLAZ	Pula se zero	$(CI) \leftarrow (END)$ se $AC = 0$
1110	SUS	SUS	Subtrai um ou salta	se $(END) = 0$: $(CI) \leftarrow (CI) + 2$ se $(END) \neq 0$: $(END) \leftarrow (END) - 1$
1111	PUG	PUG	Pula e guarda	$(END) \leftarrow 0000(CI(0-3))$ $(END + 1) \leftarrow (CI(4-11))$ $(CI) \leftarrow (END) + 2$

Figura 5-2. Formato de instruções longas

As instruções longas podem ser indexadas ou não. Com elas pode ser manipuladas na implementação "funcionais", dependendo de "salta" (*SUS*) é quando o contador executa a operação de um laço (que não volta ao começo na memória) e quando "conta" (quando "conta")

LADO ANTES
CONTA PELA
ZERO

A instrução é dividida em rotinas. Isso permite que o leitor saiba o que ele. Vamos dividir

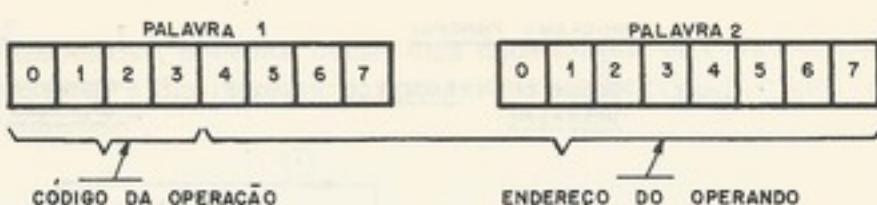
O leitor já deve ter percebido que a "multiplicação" é feita por software isto é, fazemos vários multiplicadores em vários pontos da máquina. Vários programas só rotina de multiplicação queira multiplicar, voltamos com o resultado. Surge um problema: é útil a instrução de multiplicação?

Com a multiplicação, podemos obter o conteúdo do CI. Se a palavra da sub-máquina onde será encontrada, está ilustrada na figura.

a.2. Grupo de instruções de armazenamento

Ainda como convém discutir

Figura 5-2. Formato das instruções longas



As instruções "pulo incondicional", "armazena", "carrega" e "soma", são as únicas que podem ser indexadas. Nessas instruções, o bit 3 do código da operação indica se é indexado ou não. Com essas quatro, julga-se que tabelas de dados ou cadeias de caracteres podem ser manipuladas sem grande dificuldade. O "pulo incondicional" indexado tem aplicação na implementação de desvios de tipo "multi-rumo" (*many-way-branch*). Os "pulos condicionais", dependendo do acumulador, são usados para testá-lo. A instrução "subtrai um ou salta" (SUS) é interessante, pois permite que qualquer palavra da memória possa servir como contador: o conteúdo do endereço efetivo, se não for zero, é decrementado e o computador executa a próxima instrução, que provavelmente é um pulo incondicional para o começo de um laço; quando o contador se anula, a próxima instrução é saltada, e o programa não volta ao começo do laço (veja a Fig. 5-3). Neste exemplo, "conta" (uma certa posição na memória) é carregada com "9", então o laço será percorrido dez vezes até que termine (quando "conta" = 0) e saia do laço.

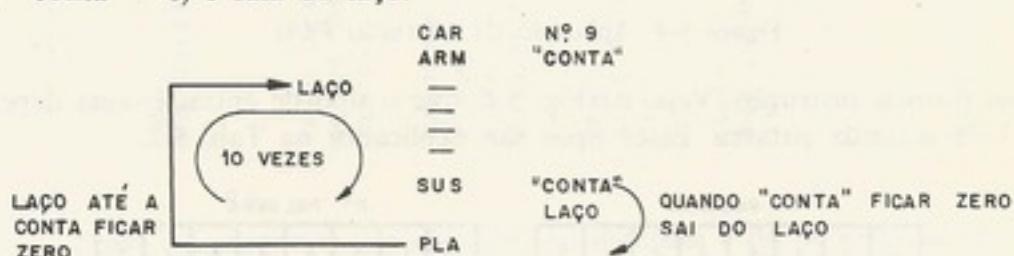


Figura 5-3. Exemplo da aplicação de instrução SUS

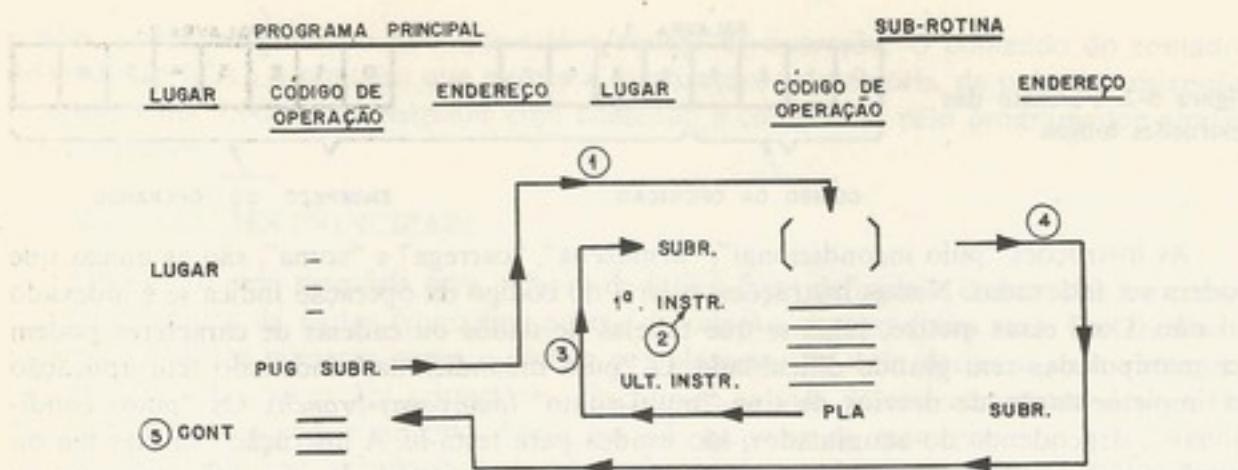
A instrução de "pula e guarda" (PUG) é um desvio que permite o uso de técnica de sub-rotinas. Isso porque o PUG deixa uma indicação do ponto de saída, para permitir regresso a ele. Vamos dizer algumas palavras sobre essa técnica.

O leitor já deve ter notado que a arquitetura desse minicomputador não tem a instrução "multiplicação". Portanto, se quisermos multiplicar dois números, devemos fazê-lo por software isto é, fazer um trecho de programa que, utilizando somas e deslocamentos sucessivos, multiplique dois números. Admita agora que tenhamos um programa principal onde, em vários pontos, faça-se uma multiplicação. Uma solução para o problema seria inserirmos vários programas de multiplicação no programa principal. Outra solução seria termos uma só rotina de multiplicação (sub-rotina) e, em cada ponto do programa principal onde se queira multiplicar, desviamos o processamento para essa sub-rotina. No final da sub-rotina, voltamos com o processamento para o ponto do programa principal de onde saímos. Aqui surge um problema: como voltar ao ponto de saída, já que são vários os possíveis? Ai torna-se útil a instrução de PUG.

Com a instrução de PUG, endereçamos o começo da sub-rotina, onde é, automaticamente, montado um PLA para a volta ao programa principal (esse ponto de volta era o conteúdo do CI durante a fase de fetch de PUG) e o processamento é indicado na terceira palavra da sub-rotina. Terminada a sub-rotina existe um desvio (PLA) para o seu começo, onde será encontrado outro PLA para a volta ao programa principal. A aplicação do PUG está ilustrada na Fig. 5-4.

a.2. Grupo entrada/saída

Ainda como instruções longas, temos as instruções de entrada/saída num novo formato. Convém dizermos que esse sistema admite até dezenas de dispositivos de entrada/saída ende-



1. PUG: o CI que indica *CONT* está guardado no inicio da sub-rotina
2. O computador executa a primeira instrução da sub-rotina
3. Depois de executar a última instrução da sub-rotina, o computador executa um pulo para o lugar (*SUBR*) onde foi guardado o *CI*
4. No lugar *SUBR* o computador encontra um pulo incondicional para o lugar *CONT*
5. O computador executa a próxima instrução do programa principal

Figura 5-4. Aplicação da instrução *PUG*

reçáveis na própria instrução. Veja, na Fig. 5-5, que o tipo de entrada/saída depende dos bits 0 a 3 da segunda palavra. Esses tipos são explicados na Tab. 5-2.



Figura 5-5. Formato das instruções longas do grupo entrada-saída

Tabela 5-2. Instruções do grupo entrada e saída

Mnemônico	Descrição
Inicial	Definitivo
<i>SAEDS</i>	Saída de dados, comando <i>c</i> para dispositivo <i>n</i>
<i>SAEDE</i>	Entrada de dados, comando <i>c</i> do dispositivo <i>n</i>
<i>SAESA</i>	Salto, tipo <i>c</i> para o dispositivo <i>n</i>
<i>SAEFU</i>	Função tipo <i>c</i> para o dispositivo <i>n</i>

As técnicas de entrada/saída serão ressaltadas em detalhes no Cap. 7, mas acreditamos que aqui seja necessária uma breve descrição das duas últimas instruções.

SAL, n, c. Para cada dispositivo, dos dezesseis possíveis, podemos testar alguma condição (por exemplo, se o dispositivo está ocupado) previamente estabelecida por hardware e codificada no campo "comando" (portanto até dezesseis condições por dispositivo) e, no caso dessa condição testada ser satisfeita saltamos a instrução seguinte.

FNC, n. c. Para cada dispositivo, podemos iniciar uma dada função, específica de cada um deles (por exemplo, reenrolar a fita magnética), previamente estabelecida por *hardware* e codificada no campo "comando".

a.3. Grupo das imediatas

São aquelas em cujo campo de endereço existe o próprio operando. Com um campo de código das imediatas de 4 bits, poder-se-iam implementar até dezesseis instruções nesse grupo. Contudo bastaram quatro delas: "soma", *NAND*, *exclusive OR* e "carrega acumulador". Aproveitou-se então para escolherem-se os códigos de modo a simplificar o decodificador. E, ainda, colocou-se, junto com essas instruções, a instrução de deslocamento que não é imediata. Veja, na Fig. 5-6(a), o formato das instruções imediatas e, na 5-6(b), o das instruções de deslocamento.

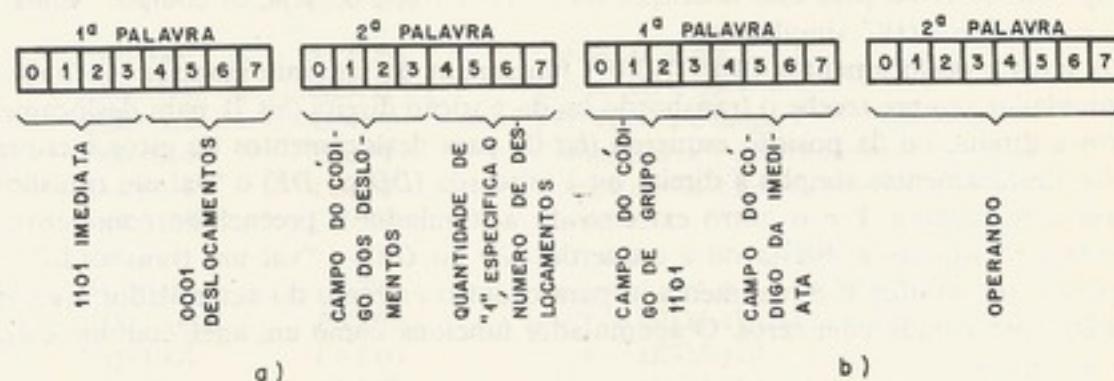


Figura 5-6. Formatos das instruções (a) imediatas e (b) de deslocamentos

Tabela 5-3.(a) Instruções imediatas

Código da imediata	Mnemônico		Descrição	Operação
	Inicial	Definitivo		
1000	IMEADD	SOMI	Soma imediata	$(AC) \leftarrow (AC) + \text{Operando}$
0100	IMENAND	NAND	<i>NAND</i> imediato	$(AC) \leftarrow ((AC) \wedge \text{Operando})'$
0010	IMEXOR	XOR	<i>Exclusive-OR</i> imediato	$(AC) \leftarrow (AC) \oplus \text{Operando}$
1010	IMECAC	CARI	Carrega imediato	$(AC) \leftarrow \text{Operando}$

Tabela 5-3.(b) Instruções de deslocamento

Código do deslocamento	Mnemônico		Descrição
	Inicial	Definitivo	
1000	SR + DUPSI	DDS	Desloca à direita com duplicação do sinal
0000	SR	DD	Desloca à direita
0100	SL	DE	Desloca à esquerda
0001	SR C/T	DDV	Desloca à direita com V
0101	SL C/T	DEV	Desloca à esquerda com V
0010	SR C/G	GD	Giro à direita
0110	SL C/G	GE	Giro à esquerda
0011	SR C/TG	GDV	Giro à direita com V
0111	SL C/TG	GEV	Giro à esquerda com V

Nessa arquitetura, existe uma certa economia no uso das instruções imediatas, pois o endereço de um byte precisa mais bits que o próprio byte. Então, para instruções em que um operando é uma constante (por exemplo, para somar de 1 ao acumulador), é mais econômico que o operando seja o próprio campo de endereço da instrução. Julgou-se também que, nas ocasiões em que se usam operações booleanas, fazemo-lo, freqüentemente, com constantes. A operação *NAND* é útil, pois, com ela, conseguimos executar as operações de *AND* e *OR*, com o auxílio de complementação. Suponha que queiramos fazer *AND* do acumulador com "11110000". Basta fazermos o *NAND* imediato com "11110000" e depois complementar o acumulador. Agora suponha que queiramos fazer *OR* do acumulador com "00001111". Podemos primeiro complementar o acumulador e depois fazer *NAND* com "11110000" (o complemento bit a bit dá "00001111").

Depois de definida a arquitetura, durante o projeto lógico desse computador, percebeu-se que era muito simples e barato criar-se a instrução de "carrega imediato" (*CARI*). O código que se criou para essa instrução foi o "11011010", ou seja, os códigos "soma imediata" e "exclusive OR" simultâneos.

Os giros e deslocamentos [Tab. 5-3(b)] funcionam da seguinte maneira: o *flip-flop V* do acumulador sempre recebe o transbordo ou da posição direita (bit 7), para deslocamentos ou giros à direita, ou da posição esquerda (bit 0), para deslocamentos ou giros à esquerda.

Nos deslocamentos simples à direita ou à esquerda (*DD* ou *DE*) o "vai um transbordo" vai para o registrador *V* e o outro extremo do acumulador é preenchido com zeros.

Com giro simples à direita ou à esquerda (*GD* ou *GE*), o "vai um transbordo", além de ir para o registrador *V*, é realimentado para o outro extremo do acumulador, que, nesse caso, não é preenchido com zeros. O acumulador funciona como um anel, com um extremo ligado ao outro.

As instruções *DDV* e *GDV* são iguais. O mesmo se diz dos *DEV* e *GEV*. Nessas instruções, o acumulador se fecha, também como um anel, tendo o registrador *V* intercalado entre os extremos dele. E o deslocamento se processa em giro, à esquerda ou à direita.

Na instrução particular *DDS* (desloca à direita com duplicação de sinal), o bit "0" (sinal do acumulador) não muda e ocorre um deslocamento à direita. Resulta então uma duplicação desse sinal para dentro do acumulador, conforme os deslocamentos vão ocorrendo.

b. Instruções curtas

As instruções curtas são instruções de apenas uma palavra. O computador distingue as instruções curtas das longas pelos três primeiros bits, que, nas curtas, devem ser 100. Qualquer instrução cujos três primeiros bits da primeira palavra não formem o número 100 é uma instrução longa. Nessa arquitetura, a instrução curta recebeu o nome de *microinstrução*. É importante ressaltar que esse nome nada tem a ver com microprogramas e foi dado apenas porque é uma instrução *curta*. Essas instruções curtas são divididas em dois microgrupos, conforme segue.

b.1. Microgrupo 1 (MICG1)

As microinstruções desse grupo controlam um ciclo em que o acumulador passa pela entrada *X* do somador e o resultado é devolvido ao acumulador. O formato da instrução é visto na Fig. 5-7, e a descrição é vista na Tab. 5-4.

Os bits 4 a 7 dessa microinstrução agem da seguinte maneira: o bit 4 deixa passar o estado das chaves do painel para a entrada *Y* do somador. Se o acumulador não passa pela entrada *X* do somador (bits 5, 6 e 7 da microinstrução no estado "0"), a condição das chaves do painel é colocada no acumulador. Esse é um modo pelo qual o operador pode se comunicar com o programa. O bit 5, no estado "1", permite a passagem do acumulador para a entrada *X* do somador e o bit 6, no estado "1", permite a passagem do acumulador complementado pela entrada *X* do somador. Com os bits 5 e 6 dessa instrução simultaneamente

no estado "1" ("vem um que Para incrementar 5 = 1) com basta passar Será um

b.2. Micr

Essas micr

e, se for necess

O formu

Figura 5-8. ○ instruções, gru

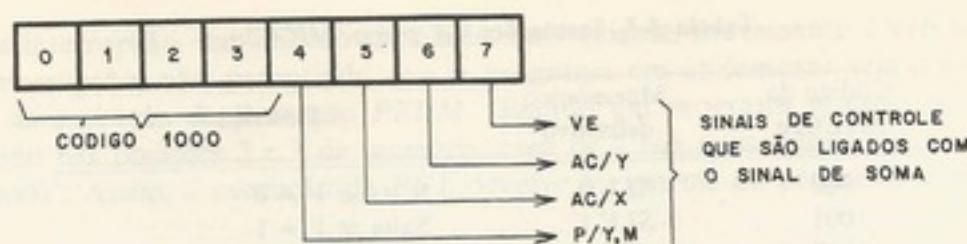
exemplo de um minicomputador

Figura 5-7. Formato da instrução curta, grupo 1

Tabela 5-4. Relação das microinstruções do grupo 1

Instrução	Mnemônico	Descrição
	Definitivo	
1000 0000	LIMP, 0	Limpa AC e faz $V = 0$
1000 0111	LIMP, 1	Limpa AC e faz $V = 1$
1000 0001	UM	Faz $AC = 1$ e limpa V
1000 0010	CMP1	Complementa 1 no acumulador e limpa V
1000 0011	CMP2	Complementa 2 no acumulador e limpa V
1000 0100	LIMV	Limpa V
1000 0101	INC	Incrementa o acumulador
1000 0110	UNEG	Faz acumulador $= -1$ e limpa T
1000 1000	PNL(0)	$(AC) \leftarrow (RC(4-11)), (V) \leftarrow 0$
1000 1001	PNL(1)	$(AC) \leftarrow (RC(4-11)) + 1$
1000 1010	PNL(2)	$(AC) \leftarrow (RC(4-11)) - (AC) - 1$
1000 1011	PNL(3)	$(AC) \leftarrow (RC(4-11)) - (AC)$
1000 1100	PNL(4)	$(AC) \leftarrow (RC(4-11)) + (AC)$
1000 1101	PNL(5)	$(AC) \leftarrow (RC(4-11)) + (AC) + 1$
1000 1110	PNL(6)	$(AC) \leftarrow (RC(4-11)) - 1$
1000 1111	PNL(7)	$(AC) \leftarrow (RC(4-11)), (V) \leftarrow 1$

no estado “1”, a entrada X do somador fica “1111 1111”. O bit 7 controla a entrada VUQ (“vem um quente”) do acumulador. Com o bit 7 no estado “1”, a quantidade “1” é somada. Para incrementar 1 ao acumulador, então basta passar o acumulador pela entrada X (bit 5 = 1) com VUQ (bit 7 = 1) do somador. Para fazer o complemento de 2 do acumulador, basta passar o complemento do acumulador com VUQ .

Será um bom exercício analisar todas essas combinações e testar o resultado.

b.2. Microgrupo 2a (MICG2a)

Essas microinstruções são usadas como salto, ou seja, é testada uma dada condição e, se for verdadeira, salta sobre a próxima instrução, supondo que ela seja longa.

O formato dessas instruções é visto na Fig. 5-8 e a descrição é mostrada na Tab. 5-5.

Figura 5-8. O formato das microinstruções, grupo 2

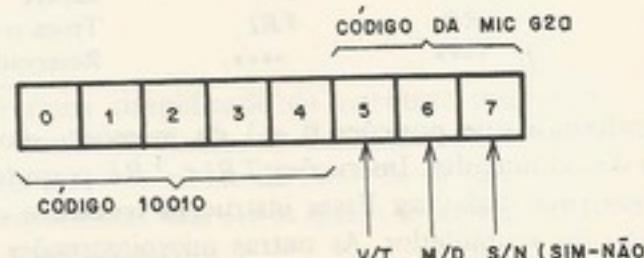


Tabela 5-5. Instruções do grupo *MICG2a*

Código da <i>MICG2a</i>	Mnemônico definitivo	Descrição
000	<i>SLV,0</i>	Salta se <i>V</i> = 0
001	<i>SLV,1</i>	Salta se <i>V</i> = 1
010	<i>SLVM,0</i>	Se <i>V</i> = 0, salta e faz <i>V</i> = 1
011	<i>SLVM,1</i>	Se <i>V</i> = 1, salta e faz <i>V</i> = 0
100	<i>SLT,0</i>	Salta se <i>T</i> = 0
101	<i>SLT,1</i>	Salta se <i>T</i> = 1
110	<i>SLTM,0</i>	Se <i>T</i> = 0, salta e faz <i>T</i> = 1
111	<i>SLTM,1</i>	Se <i>T</i> = 1, salta e faz <i>T</i> = 0

As microinstruções do grupo 2a são usadas para testar o estado dos *flip-flops V* ("vai um" do somador e o último bit de derrame dos deslocamentos ou giros) e *T* (transbordo-estouro das instruções aritméticas). O bit 5 seleciona qual *flip-flop* será testado pela microinstrução, ou *V* ou *T*. O bit 6, quando no sentido "muda", provê a capacidade de disparar ou limpar os *flip-flops*. Esse bit, no sentido "muda", só realizará a mudança (de "1" para "0" ou "0" para "1") quando o computador saltar; caso não salte, ele deixará o *flip-flop* como está. Assim, o mnemônico *SLTM,1* sempre deixa o *flip-flop T* limpo; se *T* for "0", o computador não saltará e deixará *T* limpo; caso contrário, se *T* for "1" o computador saltará e mudará *T* para "0". Os saltos são executados (incluído o ciclo *I*) em um ciclo da máquina apenas. Essas instruções, seguidas de um pulo incondicional, equivalem a *pulos condicionais*, testando os *flip-flops V* e *T*.

b.3. Microgrupo 2b (*MICG2b*)

O grupo 2b consta dos comandos especiais. O formato é visto na Fig. 5-9 e a descrição dos subcódigos é mostrada na Tab. 5-6.

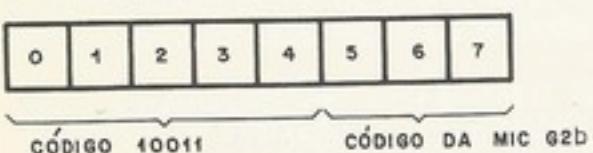


Figura 5-9. Formato das microinstruções grupo 2b

Tabela 5-6. Instruções do grupo *MICG2b*

Código da <i>MICG2b</i>	Inicial	Definitivo	Descrição
000	<i>PUL</i>	<i>PUL</i>	Pula para a posição 2 e limpa interrupção
001	<i>TRE</i>	<i>TRE</i>	Troca o acumulador com extensão
010	<i>INI</i>	<i>INIB</i>	Inibe interrupção
011	<i>PER</i>	<i>PERM</i>	Permite interrupção
100	<i>PAR-D</i>	<i>PARE</i>	Pare
101	<i>ESP-D</i>	<i>ESP</i>	Espere
110	<i>TRO</i>	<i>TRI</i>	Troca o acumulador com registrador de índice
111	****	****	Reservado para uso futuro

Ressaltamos que posições 0 e 1 da memória são, respectivamente, o indexador e a extensão do acumulador. Instruções *TRI* e *TRE* permitem os meios econômicos para retirar ou carregar essas palavras. Essas instruções trocam o conteúdo da respectiva palavra com o conteúdo do acumulador. As outras microinstruções de grupo 2b são vinculadas com o

exemplo de um mimo

esquema de interrupção. O tema de interrupção é abordado mais tarde, mas é importante mencionar que a interrupção é realizada quando o bit 6 da microinstrução é alterado de 0 para 1. Isso ocorre quando a interrupção é acionada ou quando a microinstrução é executada. A interrupção é cancelada quando o bit 6 é alterado de 1 para 0.

A instrução *SOM* é usada para somar o conteúdo do acumulador com o conteúdo da extensão. Ela é executada quando o bit 6 da microinstrução é alterado de 0 para 1. A instrução *SOMX* é usada para somar o conteúdo do acumulador com o conteúdo da extensão, mas o resultado é armazenado no acumulador. A instrução *SOMI* é usada para somar o conteúdo do acumulador com o conteúdo da extensão, mas o resultado é armazenado na extensão.

O leitor deve ter em mente que as instruções *SOM*, *SOMX* e *SOMI* são executadas quando o bit 6 da microinstrução é alterado de 0 para 1.

Mnemônico	Descrição
<i>SOM</i>	Soma o conteúdo do acumulador com o conteúdo da extensão.
<i>SOMX</i>	Soma o conteúdo do acumulador com o conteúdo da extensão, e o resultado é armazenado no acumulador.
<i>SOMI</i>	Soma o conteúdo do acumulador com o conteúdo da extensão, e o resultado é armazenado na extensão.
<i>ARM</i>	Armazena o conteúdo do acumulador na extensão.
<i>ARMX</i>	Armazena o conteúdo do acumulador na extensão, e o resultado é armazenado no acumulador.
<i>CAR</i>	Carrega o conteúdo da extensão para o acumulador.
<i>CARX</i>	Carrega o conteúdo da extensão para o acumulador, e o resultado é armazenado na extensão.
<i>CARI</i>	Carrega o conteúdo da extensão para o acumulador, e o resultado é armazenado na extensão.
<i>PUG</i>	Pula para a posição 2 e limpa interrupção.
<i>SUS</i>	Suspõe o processamento.
<i>NAND</i>	Faz a operação lógica AND complementada entre o conteúdo do acumulador e a extensão.
<i>XOR</i>	Faz a operação lógica XOR entre o conteúdo do acumulador e a extensão.
<i>PLA</i>	Faz a operação lógica PLA entre o conteúdo do acumulador e a extensão.
<i>PLAX</i>	Faz a operação lógica PLAX entre o conteúdo do acumulador e a extensão.
<i>PLAN</i>	Faz a operação lógica PLAN entre o conteúdo do acumulador e a extensão.
<i>PLAZ</i>	Faz a operação lógica PLAZ entre o conteúdo do acumulador e a extensão.
<i>TRE</i>	Troca o conteúdo do acumulador com a extensão.
<i>TRI</i>	Troca o conteúdo do acumulador com o registrador de índice.
<i>PUL</i>	Pula para a posição 2 e limpa interrupção.
<i>PARE</i>	Pare o processamento.
<i>ESP</i>	Espere.
<i>INIB</i>	Inibe interrupção.
<i>PERM</i>	Permite interrupção.
<i>DDS</i>	Desabilita o deslocamento.
<i>DD</i>	Desabilita o deslocamento.
<i>DE</i>	Desabilita o deslocamento.
<i>DET</i>	Desabilita o deslocamento.

5.4 FLUXO DE DADOS

Na Fig. 5-10, vemos o fluxo de dados entre o processador e a memória. O processador tem uma interface central de dados (bus de dados) que conecta ao sistema de memória. O sistema de memória consiste em uma memória principal e uma memória auxiliar. A memória principal é composta por módulos de memória RAM e ROM. A memória auxiliar é composta por módulos de memória cache. O fluxo de dados é controlado por um controlador de memória que gerencia a transferência de dados entre o processador e a memória.

esquema de interrupção, um assunto abordado mais adiante. Brevemente, *INIB* inibe o sistema de interrupção, não permitindo que o programa em andamento seja interrompido, até depois da execução da instrução *PERM*. Quando um programa é interrompido, o *CI* fica guardado nas posições 2 e 3 da memória, com os 4 bits mais significativos da palavra 2 sendo "0000". Assim, a execução de *PUL* devolve o controle ao programa que foi interrompido.

A instrução *ESP* coloca o computador num estado de "espera" até que seja acionado o botão de partida do painel ou que uma interrupção aconteça. A instrução *PARE* é igual à *ESP*, com exceção de que *PARE* não aceita interrupção, o computador só continua com intervenção pelo painel.

O leitor encontra na Tab. 5-7 uma relação das principais instruções desse minicomputador.

Tabela 5-7. Instruções principais

Mnemônico	Descrição	Mnemônico	Descrição
<i>SOM</i>	Soma	<i>GD</i>	Giro à direita
<i>SOMX</i>	Soma indexado	<i>GE</i>	Giro à esquerda
<i>SOMI</i>	Soma imediato	<i>GDT</i>	Giro à direita com <i>T</i>
<i>ARM</i>	Armazena	<i>GET</i>	Giro à esquerda com <i>T</i>
<i>ARMX</i>	Armazena indexado	<i>SLT</i>	Salta se <i>T</i> igual operando
<i>CAR</i>	Carrega	<i>SLO</i>	Salta se <i>OVF</i> igual operando
<i>CARX</i>	Carrega indexado	<i>SLTM</i>	Salta se <i>T</i> = operando e muda <i>T</i>
<i>CARI</i>	Carrega imediato	<i>SLOM</i>	Salta se <i>OVF</i> = operando/muda <i>OVF</i>
<i>PUG</i>	Pula e guarda	<i>LIMP</i>	Limpa <i>AC</i> e faz <i>T</i> = operando
<i>SUS</i>	Subtrai um ou salta	<i>CMP1</i>	Comp. de 1 o <i>AC</i> e limpa <i>T</i>
<i>NAND</i>	<i>NAND</i>	<i>CMP2</i>	Complementa de 2 o <i>AC</i>
<i>XOR</i>	<i>Exclusive-OR</i>	<i>INC</i>	Incrementa <i>AC</i>
<i>PLA</i>	Pulo incondicional	<i>UM</i>	Faz <i>AC</i> = 1 e limpa <i>T</i>
<i>PLAX</i>	Pulo indexado	<i>UNEG</i>	Faz <i>AC</i> = -1 e limpa <i>T</i>
<i>PLAN</i>	Pula se negativo	<i>LIMT</i>	Limpa <i>T</i>
<i>PLAZ</i>	Pula se zero	<i>PNL(0)</i>	$(AC) \leftarrow (RC(4-11)), (T) \leftarrow 0$
<i>TRE</i>	Troca com extensão	<i>PNL(1)</i>	$(AC) \leftarrow (RC(4-11)) + 1$
<i>TRI</i>	Troca com índice	<i>PNL(2)</i>	$(AC) \leftarrow (RC(4-11)) - (AC) - 1$
<i>PUL</i>	Pula e limpa	<i>PNL(3)</i>	$(AC) \leftarrow (RC(4-11)) - (AC)$
<i>PARE</i>	Pare	<i>PNL(4)</i>	$(AC) \leftarrow (RC(4-11)) + (AC)$
<i>ESP</i>	Espere	<i>PNL(5)</i>	$(AC) \leftarrow (RC(4-11)) + (AC) + 1$
<i>INIB</i>	Inibe interrupção	<i>PNL(6)</i>	$(AC) \leftarrow (RC(4-11)) - 1$
<i>PERM</i>	Permite interrupção	<i>PNL(7)</i>	$(AC) \leftarrow (RC(4-11)), (T) \leftarrow 1$
<i>DDS</i>	Desloca à direita c/dupl. sinal	<i>SAI, n, c</i>	Saída de dados p/dispositivo <i>n</i>
<i>DD</i>	Desloca à direita	<i>ENT, n, c</i>	Entr. de dados p/dispositivo <i>n</i>
<i>DE</i>	Desloca à esquerda	<i>SAL, n, c</i>	Salto tipo I p/dispositivo <i>n</i>
<i>DDT</i>	Desloca à direita com <i>T</i>	<i>FNC, n, c</i>	Função tipo I p/dispositivo <i>n</i>
<i>DET</i>	Desloca à esquerda com <i>T</i>		

5.4 FLUXO DE DADOS

Na Fig. 5-1, encontra-se um diagrama simplificado da estrutura interna da unidade central de processamento (*UCP*) do minicomputador do LSD (Laboratório de Sistemas Digitais). Esse diagrama, normalmente, recebe o nome de fluxo de dados, pois ele ressalta apenas os caminhos e paradas mais importantes dos dados que o sistema processa. Na Fig. 5-10, vemos o fluxo de dados mais detalhadamente.

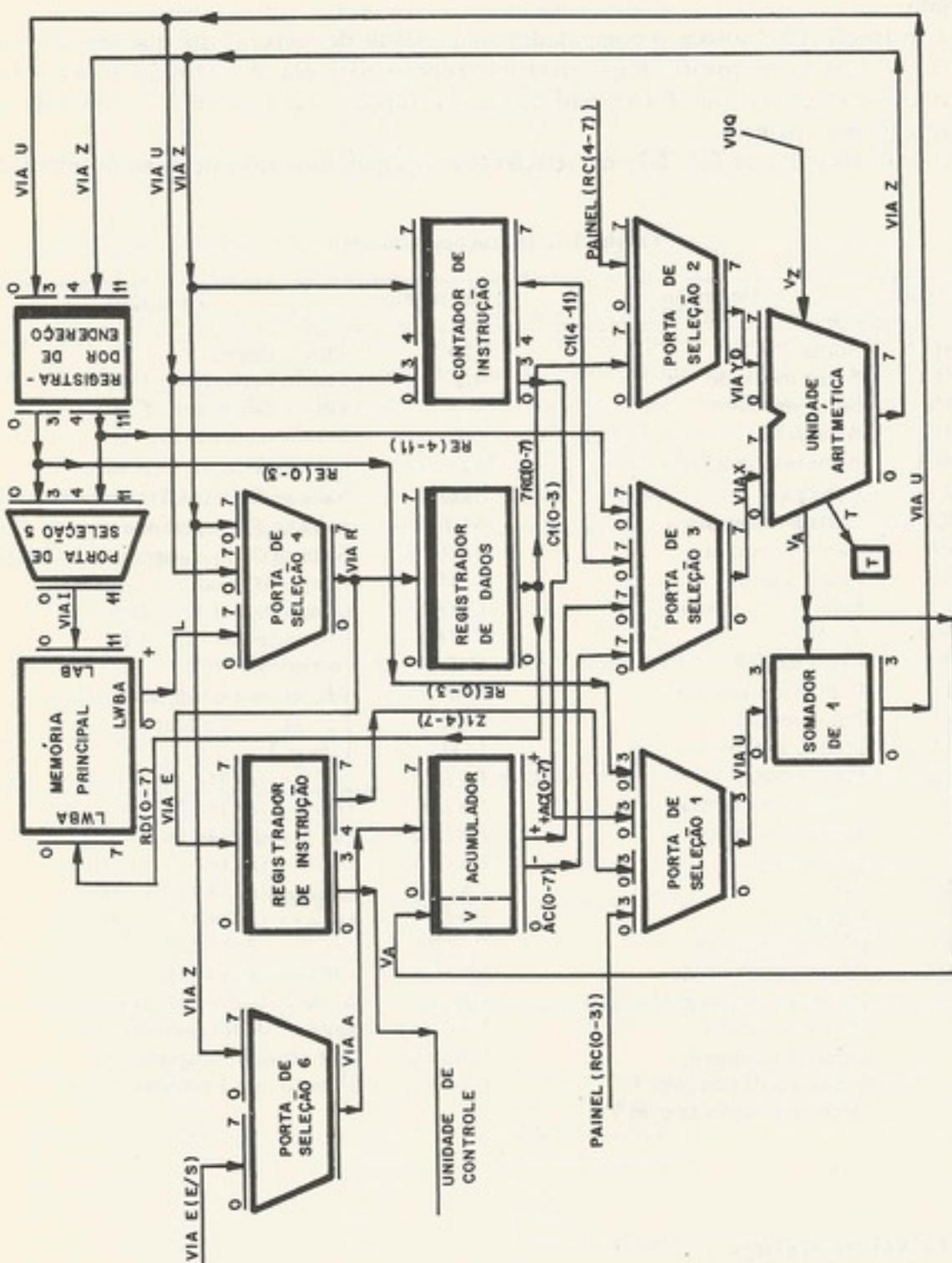


Figura 5-10. Fluxo de dados mais detalhado

a. Análise de circuitos

a.1. Memória

A memória principal, organizada em palavras de 8 bits num total de 4096 palavras, é uma memória de núcleos de ferrite, numa montagem de três fios por núcleo (FI 21, Philips). Dos sete modos de funcionamento, ressaltar-se-ão os três mais usados.

Ciclo completo

“Ler e restaurar” (*M*4): duração total, 1,6 μ s; tempo de acesso, 0,4 μ s.

Ciclo rachado

“Ler” (*M*1): duração total, 0,6 μ s; tempo de acesso, 0,4 μ s.

“Escrever” (*M*3): duração total, 1,0 μ s.

Sob o ponto de vista da unidade de controle, é preciso lembrar que, durante todo o ciclo, o endereço deve ficar constante. Existem inúmeros sinais elétricos necessários para controlar a memória e, por isso, projetou-se um circuito de interface de tal forma que, quando a unidade de controle precisar ler ou escrever algo na memória, ela envia apenas os sinais *M*4 ou *M*1 ou *M*3 que esse circuito gera os controles necessários.

As posições 0 e 1 da memória são reservadas como índice e extensão do acumulador, respectivamente.

Na Fig. 5-10, distinguem-se os seguintes blocos funcionais: memória, registradores, unidade aritmética e portas de seleção.

a.2. Registradores

Os cinco registradores do fluxo de dados podem ser agrupados em duas classes: os registradores de dados (*RD*, *AC*, *RI*), de 8 bits, com a finalidade de armazenar uma palavra de memória; e os registradores de endereço (*RE* e *CI*), com o objetivo de guardar endereços. Todos os registradores foram implementados com *flip-flops* tipo *D* sensíveis à borda. Segue-se uma breve descrição desses registradores.

Registrador de endereço (*RE*)

É o registrador que tem a função de segurar o endereço de memória e mantê-lo fixo durante todo o ciclo. Sua entrada são vias *Z* e *U*, saídas da unidade aritmética, e sua saída alimenta a memória através da porta de seleção 5. Esse registrador tem uma linha de controle (*set-reset*) para forçar o endereço 2, utilizado quando chega um pedido de interrupção. A Fig. 5-11 mostra alguns detalhes desse circuito.

Contador de instruções (*CI*)

É outro registrador de 12 bits com a entrada ligada à saída da unidade aritmética. Sua função é armazenar o endereço da instrução seguinte (note-se que, nessa arquitetura, o registrador faz parte do fluxo de dados e não da unidade de controle). Para se incrementar um ao endereço, utiliza-se a unidade aritmética, fazendo-se a soma com o *VUQ* (“vai um quente” ou “vem um”). A saída do *CI* vai direto à entrada da unidade aritmética.

Acumulador (*AC*)

É um registrador deslocador de 8 bits, com a opção de colocar o *flip-flop V* (“vai um”) em série com os 8 bits, como se fosse um nono bit do acumulador (Fig. 5-12). Esse registrador é, talvez, o núcleo da arquitetura do computador; todas as operações aritméticas e lógicas são feitas tendo o conteúdo do acumulador como um dos operandos, sendo o resultado armazenado nele; todos os desvios condicionais são feitos a partir de testes em seu conteúdo; a entrada e a saída de dados são feitas através dele.

Figura 5-10. Fluxo de dados mais detalhado



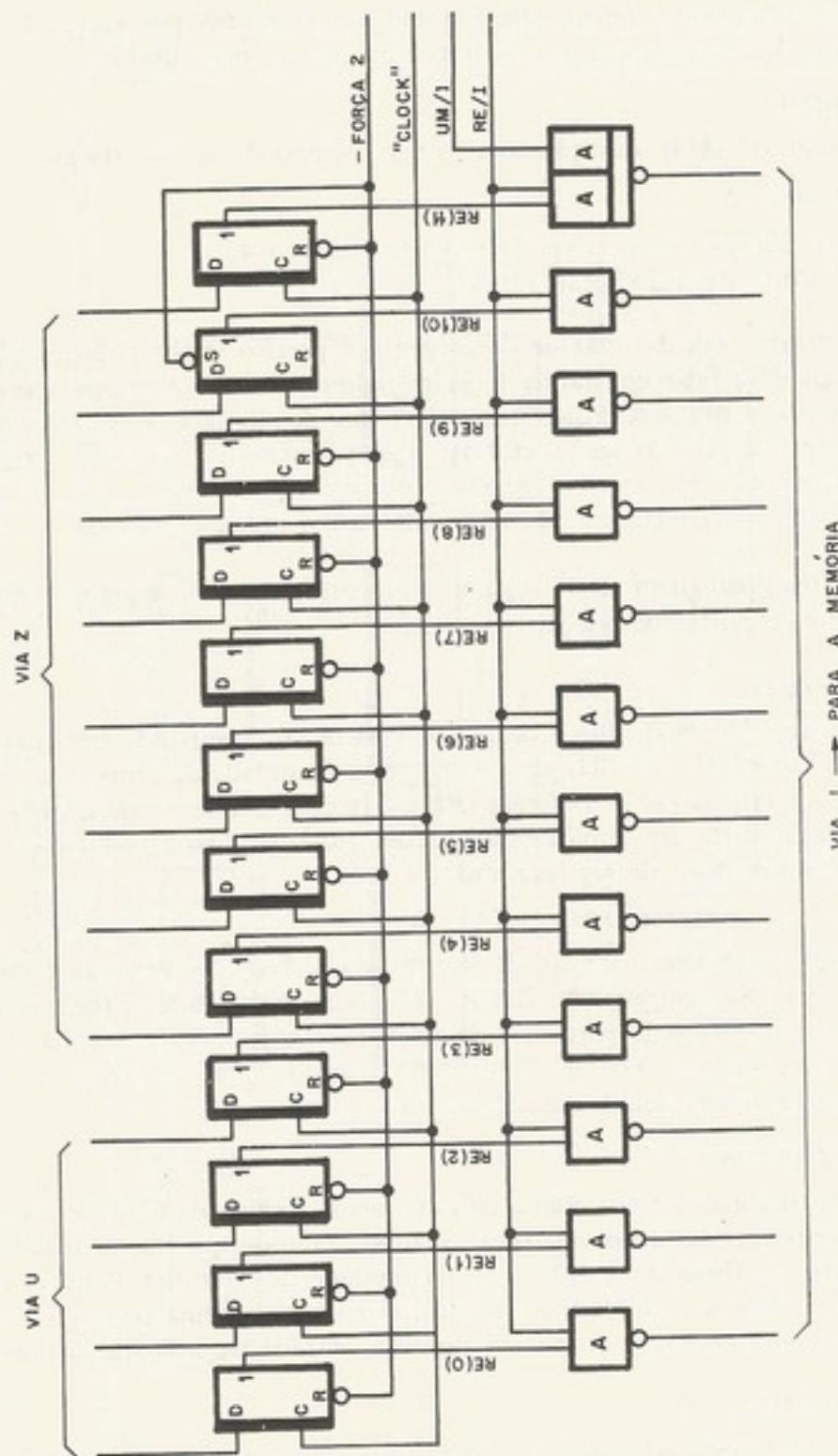


Figura 5-11. Registrador de endereço e porta de seleção 5

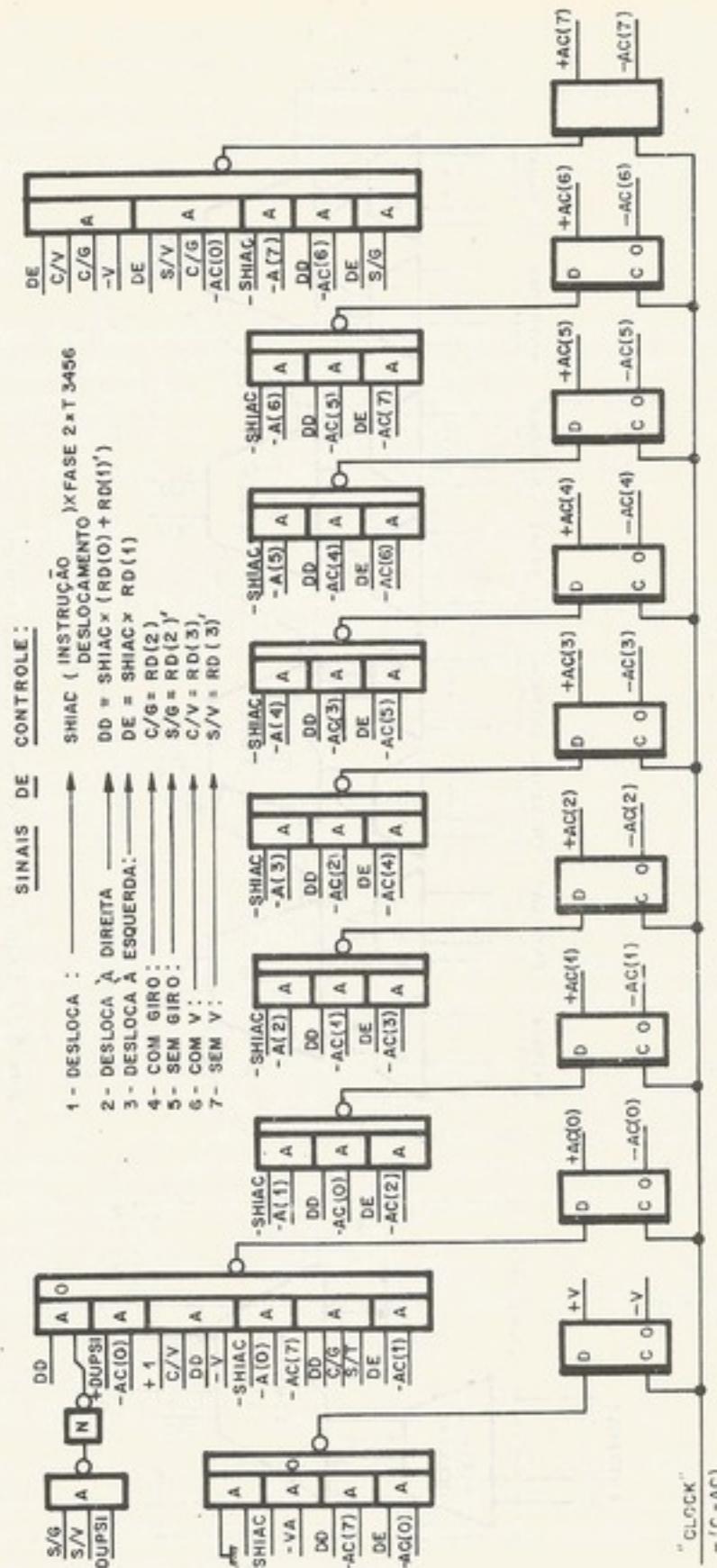


Figura 5-12. Circuito do acumulador

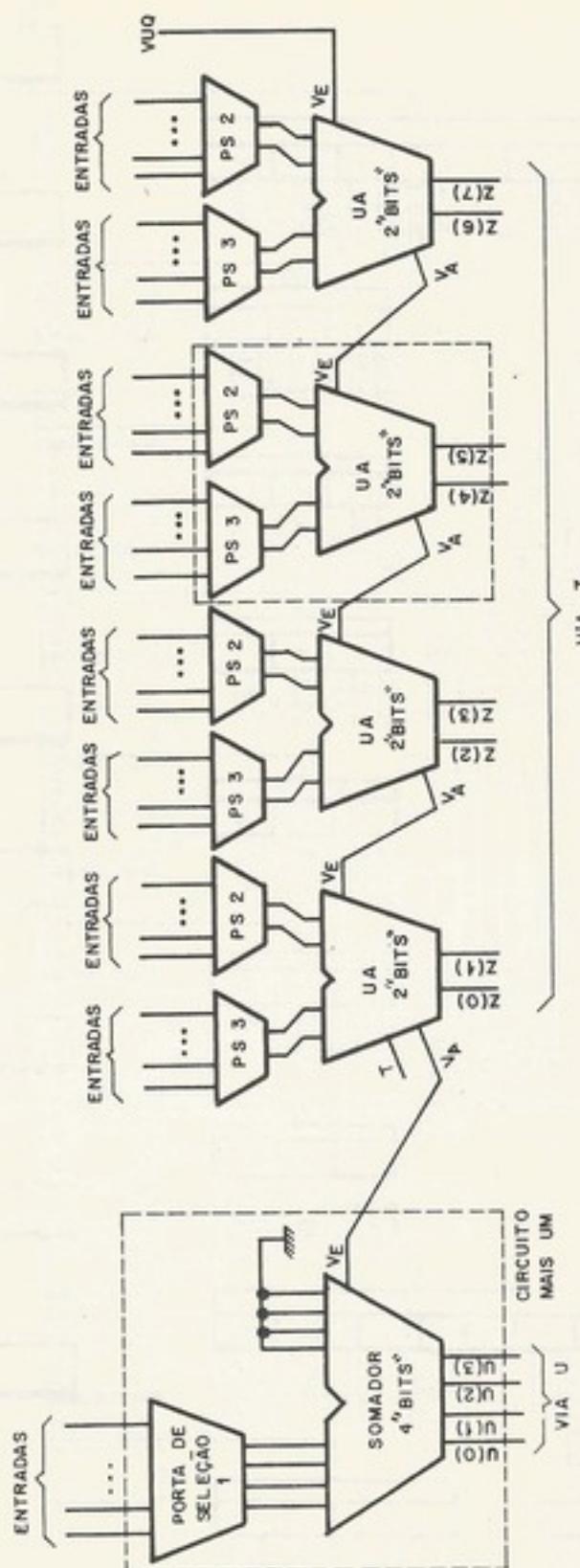


Figura 5-13. Esquema em blocos da unidade aritmética

exemplo de um m...

Registrado...

É um registrador que armazena os resultados das operações aritméticas realizadas na memória principal. Ele é usado para armazenar os resultados das operações aritméticas (adição, subtração, multiplicação, divisão, etc.) realizadas na memória principal.

Registrado...

Como o resultado da operação é armazenado no registrador, o resultado é sempre o mesmo. O resultado é obtido através da operação de adição ou subtração entre os valores armazenados no registrador e os valores fornecidos pelo usuário.

a.3. Unidade Aritmética

A função da unidade aritmética é somar ou subtrair dois números binários de 8 bits. A Fig. 5-13 mostra o esquema em blocos da unidade aritmética.

A unidade aritmética é formada por quatro unidades de soma de 2 bits, somando o resultado de cada uma delas em paralelo. Cada uma dessas unidades de soma trata dos 4 bits mais significativos de um número de 8 bits. O resultado é "desequilibrado". Esse resultado é armazenado no registrador. Foi projetado para que o resultado seja sempre significativo, ou seja, não haverá "sobras" de zeros no resultado da soma.

Para se obter o resultado da soma, é necessário somar o resultado da unidade de soma de 2 bits com o resultado da unidade de soma de 4 bits. Para isso, é necessário que o resultado da unidade de soma de 2 bits seja dividido em quatro parcelas e, em seguida, serem somadas. Assim, é necessário que o resultado da unidade de soma de 2 bits seja dividido em quatro parcelas e, em seguida, serem somadas.

Na Fig. 5-13, o resultado da soma é dividido em quatro parcelas.

a.4. Porta de Seleção

Na Fig. 5-13, a porta de seleção é usada para selecionar o resultado da soma de um circuito de soma de 2 bits ou de um circuito de soma de 4 bits cuja saída é maior que zero.

Esse circuito é controlado por uma porta de seleção (S) que é nula. Quando S é zero, a saída é a soma de 2 bits. Com S2 é a soma de 4 bits.

Registrador de dados da memória (RD)

É um registrador de 8 bits que serve com um buffer entre a UCP e a memória. Dados lidos na memória são armazenados nele. Também os dados para gravar na memória ficam nele durante o ciclo de escrita. Por esse motivo, o segundo operando das instruções aritméticas e lógicas é armazenado nele (veja, sua saída está ligada à entrada Y da unidade aritmética), já que o primeiro, conforme já foi dito, é o conteúdo do acumulador.

Registrador de instruções (RI)

Como a maioria das instruções desse computador são longas, isto é, de duas palavras, o RI é usado para armazenar a primeira palavra da instrução enquanto que a segunda palavra (endereço do operando) fica no RD até a leitura do operando. É um registrador de 8 bits provido de uma linha de set que permite forçar a instrução PUG (código 1111), necessária durante a interrupção, por um dispositivo de entrada/saída.

a.3. Unidade aritmética

A função básica da unidade aritmética é a realização das operações aritméticas e lógicas especificadas pelas instruções. São elas soma, *NAND* e *exclusive OR*, com operandos de 8 bits. A Fig. 5-10 mostra um bloco, denominado "unidade aritmética", que realiza essas funções. Esse bloco tem largura de 8 bits.

A unidade aritmética, porém, precisa executar uma outra função: operar em endereços, somando o índice ao endereço ou somando um ao contador de instruções. Por isso, inclui-se em paralelo com a unidade aritmética um circuito chamado "mais um" (*one-upper*), que trata dos 4 bits mais significativos dos endereços. Com essa providência, consegue-se somar um número de 12 bits (endereço) com um de 8 bits (índice). Poderíamos dizer que o somador é "desequilibrado", ou seja, ele soma uma parcela de 8 bits com outra de doze. Por isso, ele foi projetado de modo a fazer a soma dos 8 bits da primeira palavra com os oito menos significativos da segunda, num somador com propagação de "vai um" convencional e o "vai um" dessa soma de 8 bits é introduzido a um circuito que apenas soma esse sinal aos 4 bits mais significativos da segunda palavra (Fig. 5-13).

Para se fazer subtração, usamos as portas de seleção que estão incluídas na entrada do somador (Fig. 5-10, em blocos, e Fig. 5-14, em detalhes) para complementarmos uma das parcelas e, como o código usado pelo sistema é a codificação binária complemento de dois, forçamos um na entrada "vem um" e obtemos como resultado final essa parcela com o sinal trocado. As operações lógicas são realizadas pelos próprios circuitos do somador. São elas *NAND* e *exclusive OR*.

Na Fig. 5-13 vê-se, em blocos, a unidade aritmética e, na Fig. 5-14, encontra-se um dos blocos em detalhes. O cálculo de transbordamento é feito como explicado no Cap. 3.

a.4. Portas de seleção

Na Fig. 5-15, encontra-se o esquema de uma porta de seleção. Esse circuito é usado sempre que existem duas (ou mais) palavras, distintas, possíveis de serem colocadas na entrada de um circuito qualquer. Essas palavras são colocadas nas entradas da porta de seleção cuja saída é ligada na entrada do circuito citado.

Esse circuito tem o seguinte comportamento: com $E1/S$ e $E2/S$ no nível "zero", a saída (S) é nula. Quando $E1/S$ fica igual a "um", a via de entrada $E1$ é colocada na via de saída (S). Com $E2$ é análogo. Nessa arquitetura, existem seis portas de seleção.

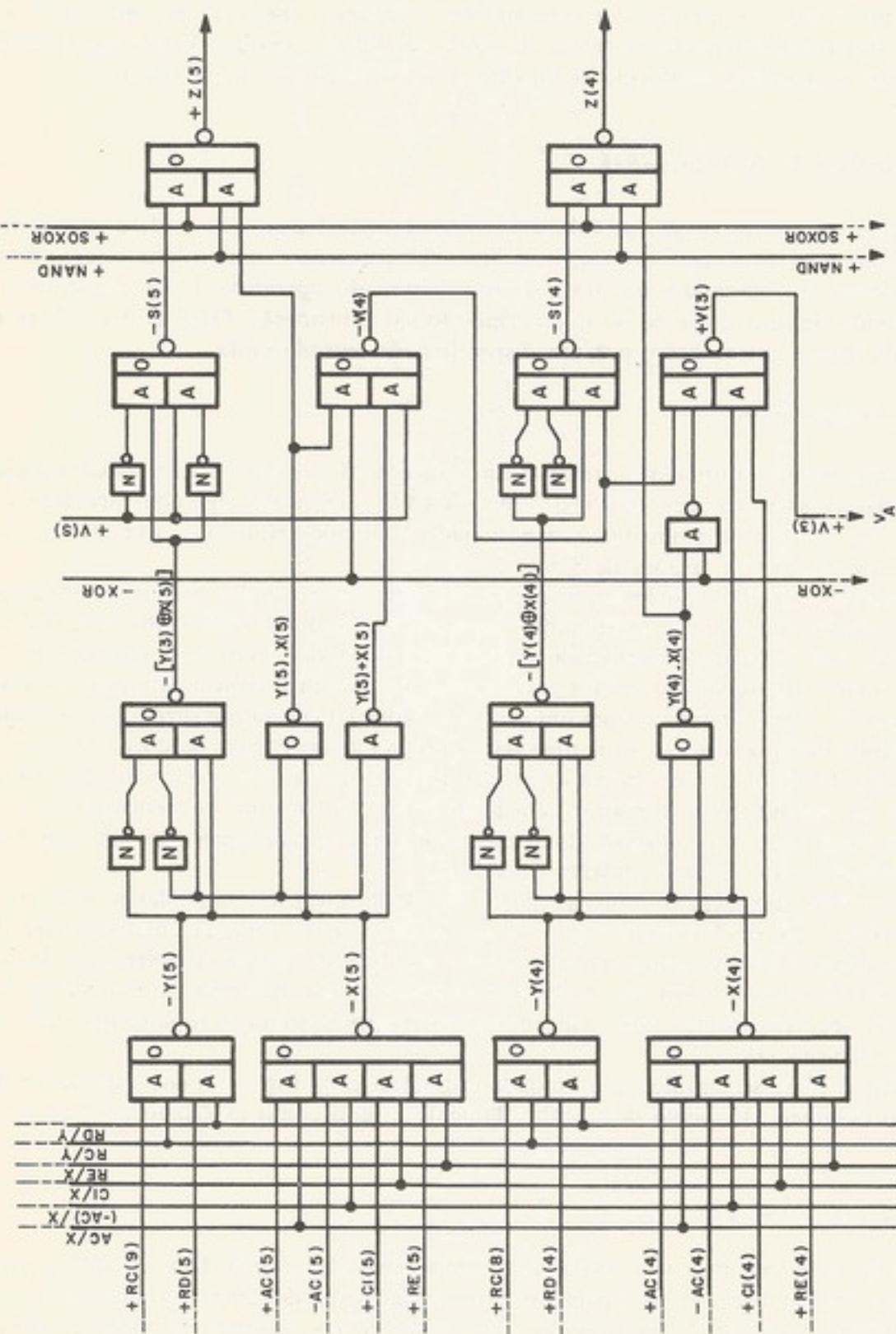


Figura 5-14. Unidade aritmética, detalhes do bloco indicado na Fig. 5-13

Figura 5-15.

Porta de Controle guentes palav
 $RJ(4-7)$
 $CI(8-3)$
 $RE(8-3)$
 $RC(8-3)$

Porta de Controle
 RD , mas
 $RC(4-3)$
de seleção II

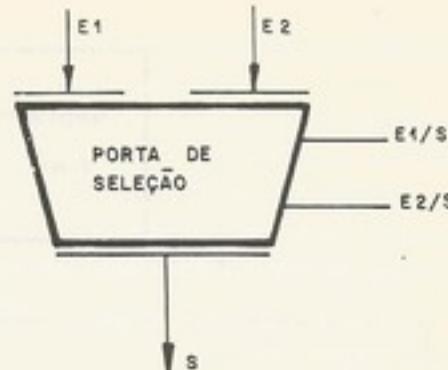
Porta de Controle de seleção II
 AC , para
 \overline{AC} , para
 $RE(4-3)$
 $CI(4-3)$

Porta de Controle
Via Z, pa
Linha, pa
Via E, pa

Porta de Controle saída zero
saída um
saída RE

Porta de Controle a via E (caso

Figura 5-15. Esquema de uma porta de seleção



Porta de seleção 1

Controla a entrada do circuito "mais um". Permite que se coloquem na via M as seguintes palavras:

- $RI(4-7)$, para endereçar o operando;
- $CI(0-3)$, para somar um ao CI ou transportar o CI para RE ;
- $RE(0-3)$, para somar o índice de registro ao endereço;
- $RC(0-3)$, para endereçar pelo painel.

Porta de seleção 2

Controla a entrada Y do somador. Seleciona entre duas palavras:

- RD , nas operações aritméticas;
- $RC(4-11)$, para introduzir dados através das chaves do painel, ou junto com a porta de seleção 1 para endereçar pelo painel.

Porta de seleção 3

Controla a entrada X do somador. Na maioria das vezes, funciona junto com a porta de seleção 1 ou 2. Seleciona uma das quatro seguintes entradas:

- AC , para operar com o acumulador;
- \bar{AC} , para complementar o acumulador;
- $RE(4-11)$, para somar o índice ao endereço;
- $CI(4-11)$, para somar um ao CI , ou transportar CI para RE .

Porta de seleção 4

Controla a entrada de RD , selecionando uma das seguintes três entradas:

- Via Z , para operandos gravados na memória;
- $Lwbo$, para palavras lidas na memória;
- Via U , utilizada pela instrução de PUG .

Porta de seleção 5

Controla o endereçamento da memória. Possibilita as seguintes três situações:

- saída zero, para endereçar o índice de registro;
- saída um, para endereçar a extensão do acumulador;
- saída RE , nos endereçamentos normais, onde a memória é endereçada pelo RE .

Porta de seleção 6

Controla a entrada do acumulador, selecionando entre via Z (funcionamento normal) a via E (caso de entrada de dados).

Figura 5-14. Unidade aritmética, detalhes do bloco indicado na Fig. 5-13

exemplo de um minicomputador

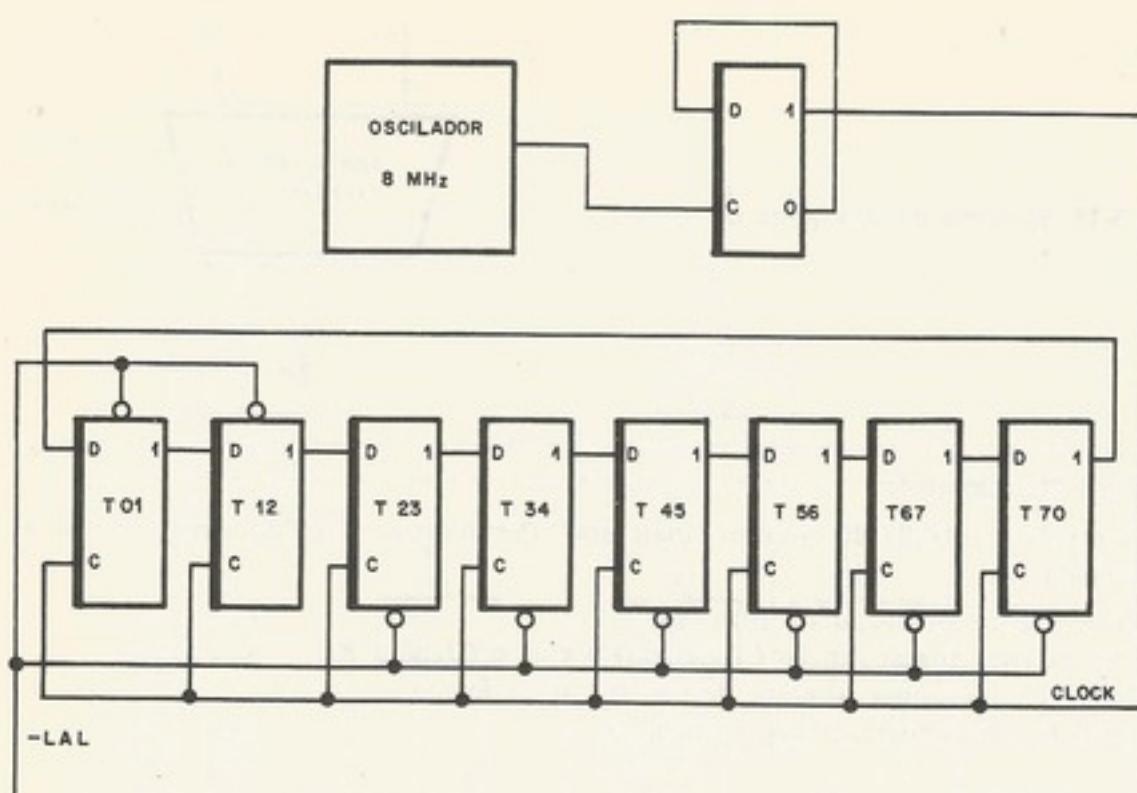


Figura 5-16. Esquema simplificado do relógio central

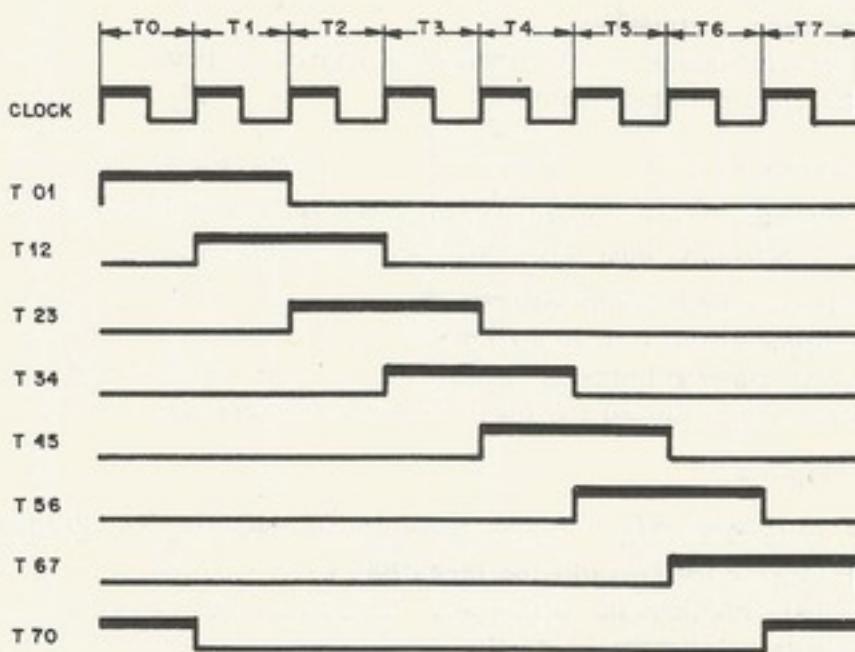


Figura 5-17. Gráfico de timing dos sinais do relógio central

a.5. Ciclo da máquina

Todos os pulsos elétricos usados pela unidade de controle para gerar sinais de *clock* ou sinais de comando de portas provêm de uma pequena unidade funcional chamada *relógio central*.

O relógio central do microcomputador do LSD está mostrado na Fig. 5-16. Basicamente, ele é constituído de um oscilador a cristal e de um *overlaped ring counter*, isto é, um deslocador em anel que funciona girando dois "1", consecutivos, conforme se vê na Fig. 5-17.

Os sinais gerados pelo relógio central são periódicos e, a esse período ($2 \mu s$), damos o nome de ciclo da memória. As oito partes do ciclo da memória (T_0 a T_7) são as unidades

de tempo memória múltiplo desses anéis.

O somador é feito partindo-se para um registrador tanto define-se como 500 ns. Como varia dependendo da impre

b. Implementação

Dividiram-se os sinais da memória. Na figura, a unidade de pastilha, da placa etc. O objetivo era atingir o máximo de simplicidade. Foi tentado, à medida que o layout e arte fizeram.

FR1-1, registrador

FR2-2, registrador

FS1-3, somador

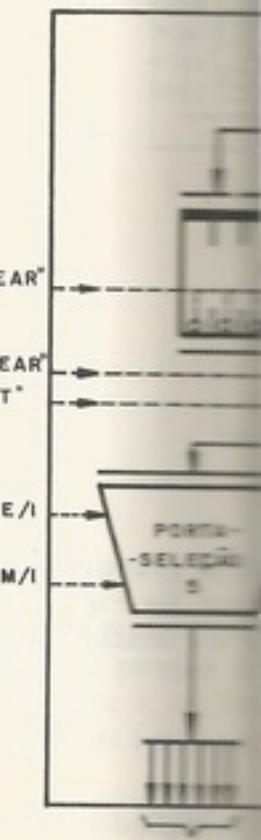
FS2-4, somador

FAC-5, acionador

FM1-6, "memória"

FCM-7, "memória"

FIN-8, síncrono



de tempo menores possíveis do sistema, ou seja, dois eventos não-simultâneos distam um múltiplo dessas unidades, com raras exceções, que mostraremos adiante.

O somador é o caminho mais importante do fluxo de dados. Um loop do fluxo de dados é feito partindo-se de um registrador, passando-se pela unidade aritmética e voltando-se para um registrador. Esse caminho tem um atraso da ordem de 400 ns, no pior caso. Portanto define-se como ciclo da máquina um período igual a duas unidades de tempo, ou seja 500 ns. Como veremos adiante, os ciclos de máquina são distribuídos convenientemente, dependendo da instrução dentro do ciclo de memória.

b. Implementação dos circuitos

Dividiram-se os circuitos do fluxo de dados em 8 placas de circuito impresso, além da memória. Na partição, levaram-se em conta inúmeros vínculos, como quantidade máxima de pastilhas de circuitos impressos por placa, número máximo de entradas e saídas da placa etc. O critério de partição foi a economia de pinos nos conectores, ou seja, tentou-se atingir o máximo de circuitos com um mínimo de entrada e saída da placa. Outro fator importante foi tentar, à medida do possível, fazer placas iguais, para se economizar na preparação do layout e arte final do circuito impresso. A partição resultou nas seguintes placas:

- FR1-1, registradores, bits pares;*
- FR2-2, registradores, bits ímpares;*
- FS1-3, somador, bits menos significativos;*
- FS2-4, somador, bits mais significativos;*
- FAC-5, acumulador;*
- FM1-6, "mais um" e porta de seleção 6;*
- FCM-7, interface com memória;*
- FIN-8, sinais para cartões de interface.*

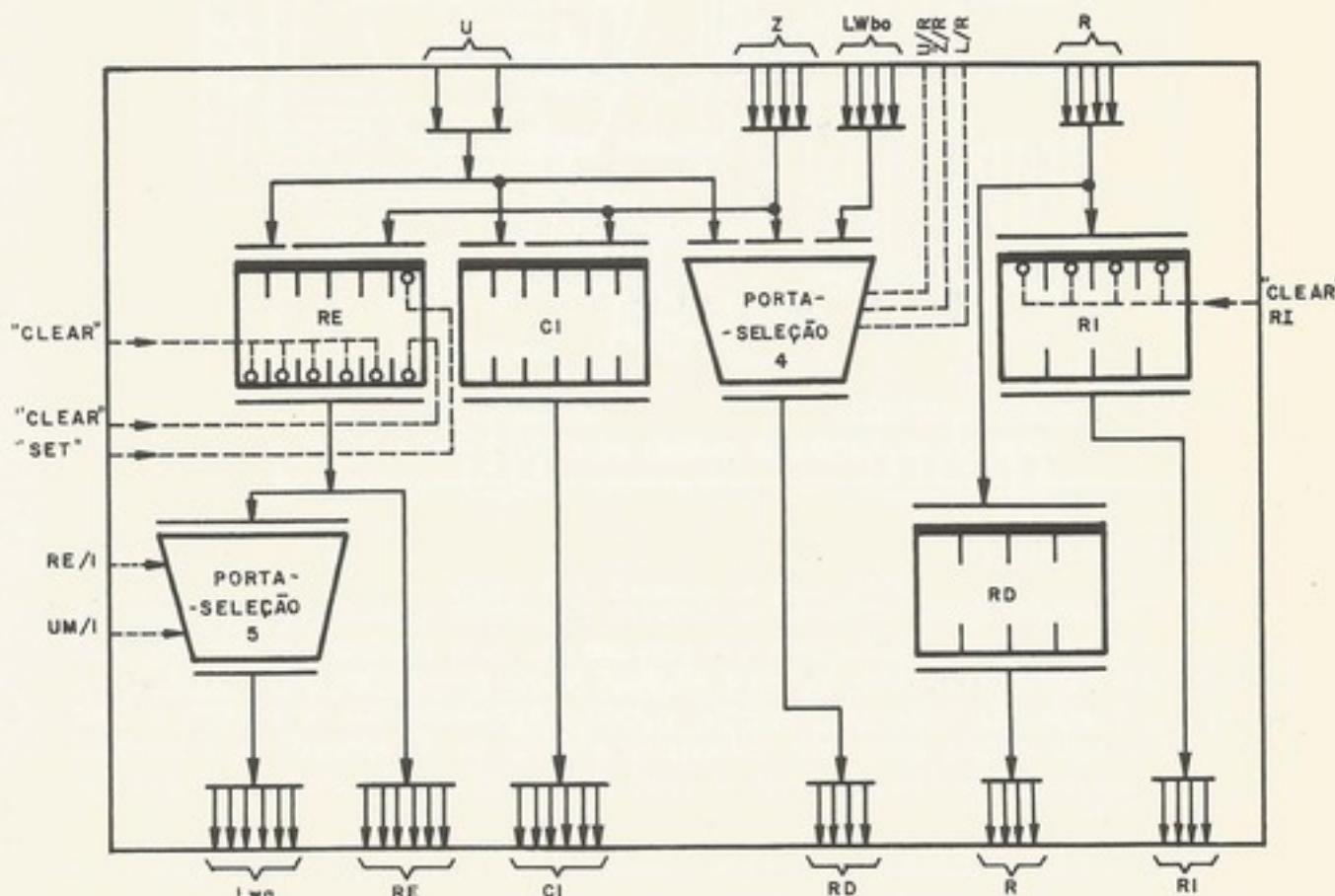


Figura 5-18. Cartão dos registradores (FR1-1, FR2-2)

gerar sinais de clock
ainda chamada relógio

Fig. 5-16. Basicamente,
isto é, um deslocador
é na Fig. 5-17.
período (2 µs), damos
a (T7) são as unidades

São esses os oito cartões que compõem o fluxo de dados. Segue-se uma breve descrição de cada um deles. Nos esquemas apresentados a seguir, as linhas são de dados e as linhas pontilhadas de controle.

b.1. FR1-1 e FR2-2

São duas placas de circuito impresso, iguais. A primeira com os bits das posições pares e a segunda das posições ímpares. Na Fig. 5-18 encontra-se um esquema do circuito dessa placa. Observar que ela inclui os registradores *CI*, *RE* — com sua linha de controle, (*set-reset*) para forçar 2 —, *RI* com outra linha de controle (*set*) para forçar o *PUG*, e *RD*. Há, ainda, nessa placa, as portas de seleção 4 e 5. Na Prancha I, encontra-se o circuito em detalhes.

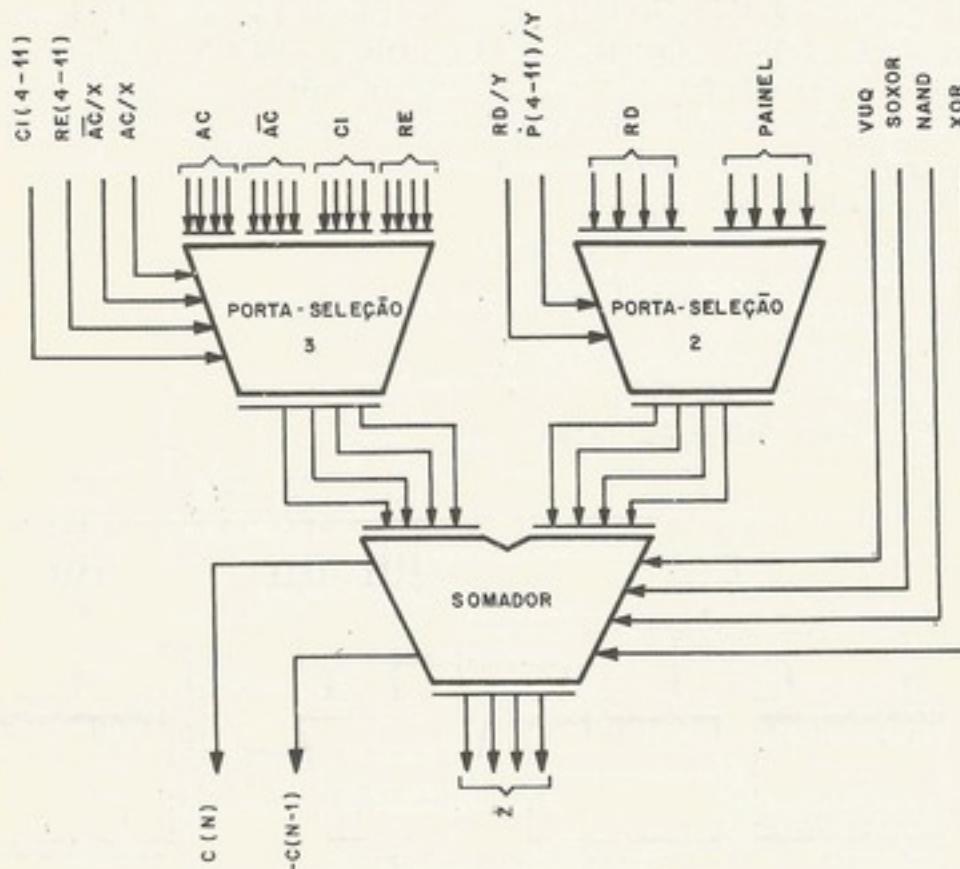


Figura 5-19. Cartões de unidade aritmética (FS1-3 e FS2-4)

b.2. FS1-3 e FS2-4

São dois cartões iguais contendo os circuitos da unidade aritmética. O primeiro corresponde aos bits menos significativos e o segundo aos mais significativos. Na Fig. 5-19, encontra-se o esquema em bloco de seu conteúdo: portas de seleção 2 e 3 e unidade aritmética. Na Prancha II, vêem-se os detalhes desse circuito.

b.3. FAC-5

O *FAC-5* é um registrador deslocador, o acumulador. Colocou-se o acumulador numa só placa, junto com os *flip-flops V* e *T*, já que sua estrutura não é uniforme a ponto de ser “quebrada em duas” iguais. A Fig. 5-20 mostra a placa em blocos e a Prancha III em detalhes.

-C-DIF
-CLEARPULSE
-SET-DIF

D-OUT
D-IN

b.4. FMD
É a placa um”, e gerenciado esquema e, na que nela fizer

b.5. FCW
É a placa corretamente

das posições pares
do circuito dessa
de controle, (*set*-
ovf, *PUG*, e *RD*. Há,
o circuito em de-

VDD
B0XOR
HAND
XOR

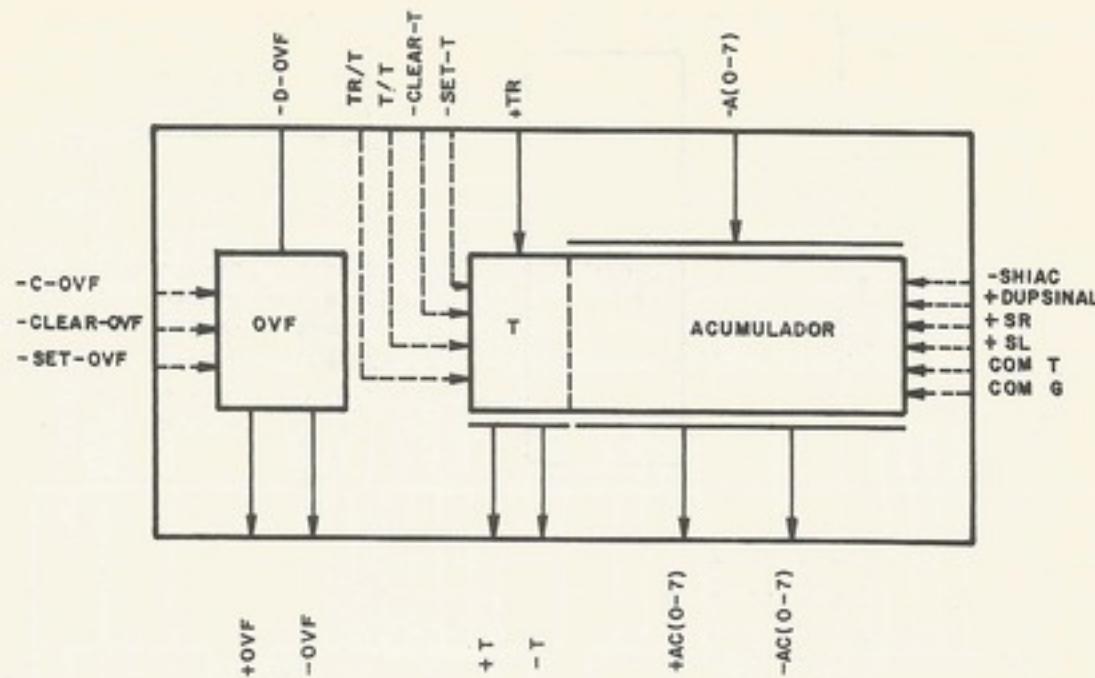


Figura 5-20. Esquema da placa do acumulador (FAC-5)

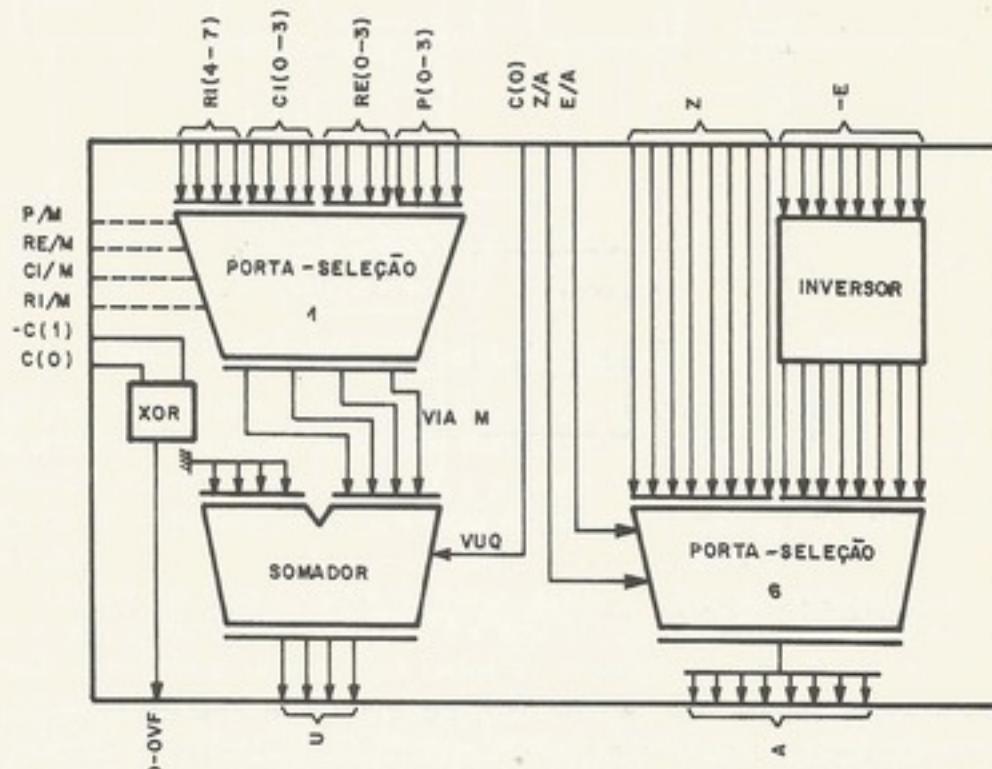


Figura 5-21. Esquema da placa FM1-6

b.4. FM1-6

É a placa que tem os circuitos da porta de seleção 6, porta de seleção 1, circuito "mais um", e geração do transbordamento para o *flip-flop T* (*overflow*). Na Fig. 5-21 vê-se um esquema e, na Prancha IV, os circuitos detalhados. Essa placa contém uma miscelânea já que nela foram colocados os pedaços do fluxo de dados que não couberam em outro lugar.

b.5. FCM-7

É a placa que gera os controles da memória. Suas funções são as seguintes: endereçar corretamente os módulos da memória (quatro); coordenar proteção das 128 últimas po-

O primeiro cor-
mos. Na Fig. 5-19,
3 e unidade arit-

o acumulador numa
ícone a ponto de ser
prancha III em detalhes.

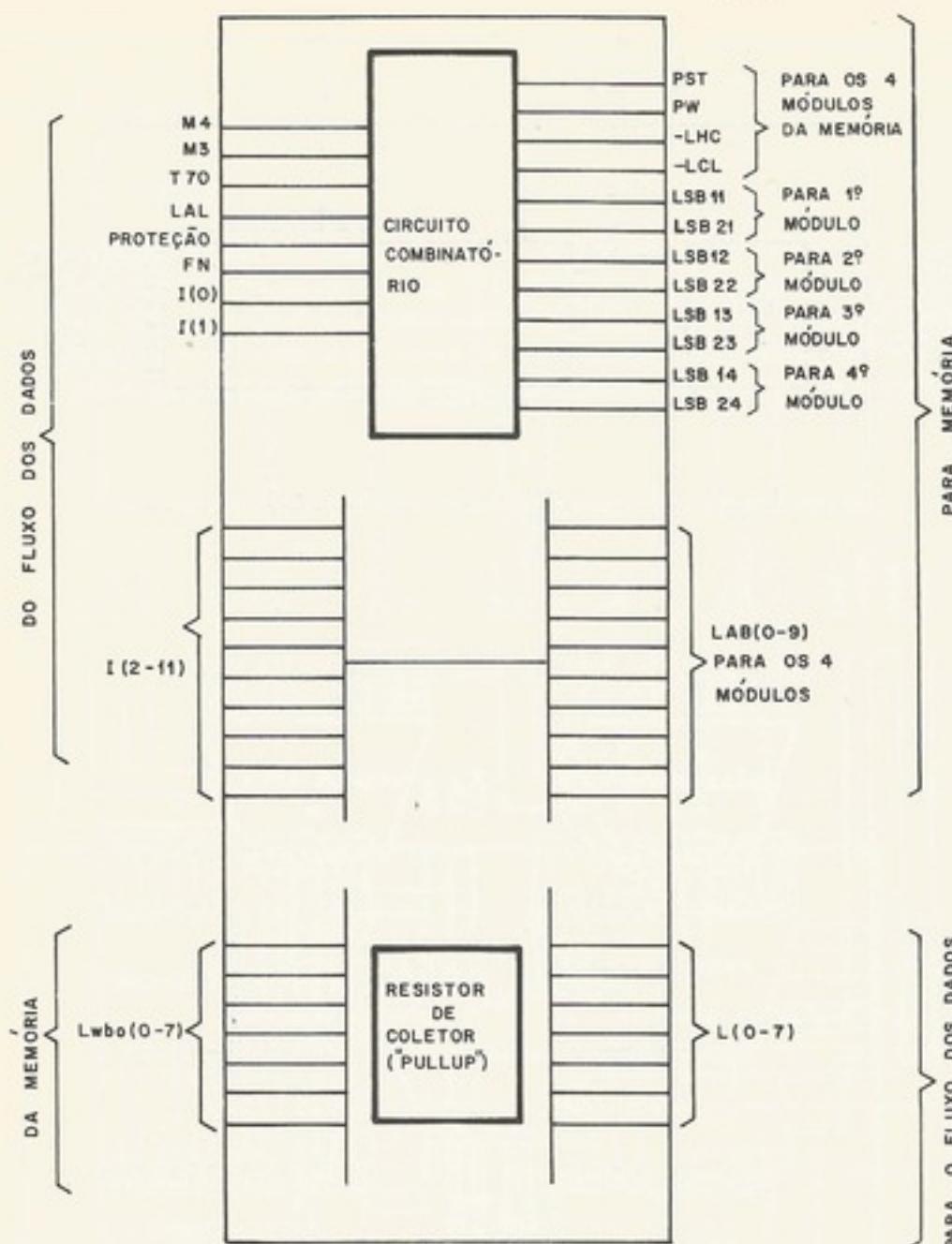


Figura 5-22. Esquema do cartão de controle de memória (FCM-7)

sições; com os sinais simples M_1 , M_3 ou M_4 , fornecidos pela unidade de controle, gerar todos os sinais necessários para o controle da memória; e, ainda, preparar os sinais de saída da memória (agrupando as saídas dos módulos num só ponto). A Fig. 5-21 mostra o esquema e a Prancha V os detalhes.

b.6. FIN-8

Para enviar os sinais para os cartões de interface, deve-se fazer com que as vias passem por portas que se abrem apenas quando é uma instrução de entrada e saída. Isso evita que existam sinais nas vias tal quando não é necessário, já que esses sinais geram ruídos. Essa é a função da placa FIN-8, além de decodificar o endereço do dispositivo, controlar a interrupção pelo canal zero (pelo painel ou por falta de força), além de aumentar a potência dos sinais de saída. Na Fig. 5-23 existe um esquema desse cartão e, na Prancha VI os circuitos em detalhes.

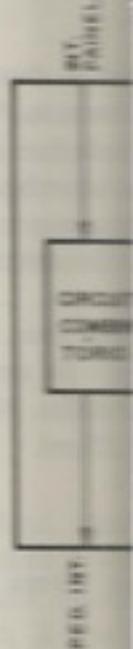
exemplo de um módulo

Os módulos só bloco: o fluxo de dados

5.5 FASES E INSTRUÇÕES

a. Fases

Uma instrução é formada por dois ciclos de memória. As fases 1 e 2 são as fases de leitura e escrita, respectivamente, por isso elas são chamadas de fases de leitura e escrita.



é, apenas a fase 1 serve para endereçar a memória e a fase 2 para escrever os dados. Em cada ciclo de memória, a fase 1 é usada para endereçar a memória e a fase 2 para escrever os dados. No caso de uma instrução de leitura, a fase 1 é usada para endereçar a memória e a fase 2 para ler os dados.

Depois de especificada a fase 1, os dois acessos são realizados.

No caso de uma instrução de leitura, a fase 1 é usada para endereçar a memória e a fase 2 para ler os dados. No caso de uma instrução de escrita, a fase 1 é usada para endereçar a memória e a fase 2 para escrever os dados.

Durante a fase 1, a unidade de controle envia o endereço para a fase 2, que é responsável por enviar o endereço para a memória.

Os oito cartões são interligados pelo painel traseiro do computador, formando um só bloco: o fluxo de dados.

5.5 FASES E MICROOPERAÇÕES

a. Fases

Uma instrução, para ser executada, passa por várias etapas. É necessário um ou dois ciclos de memória para ler a instrução da memória e colocá-la nos registradores *RI* e *RD*; são as fases 1 e 2 normalmente conhecidas como *fases de fetch* ou de busca. As microinstruções, por terem apenas uma palavra de comprimento, exigem apenas um ciclo de *fetch*, isto

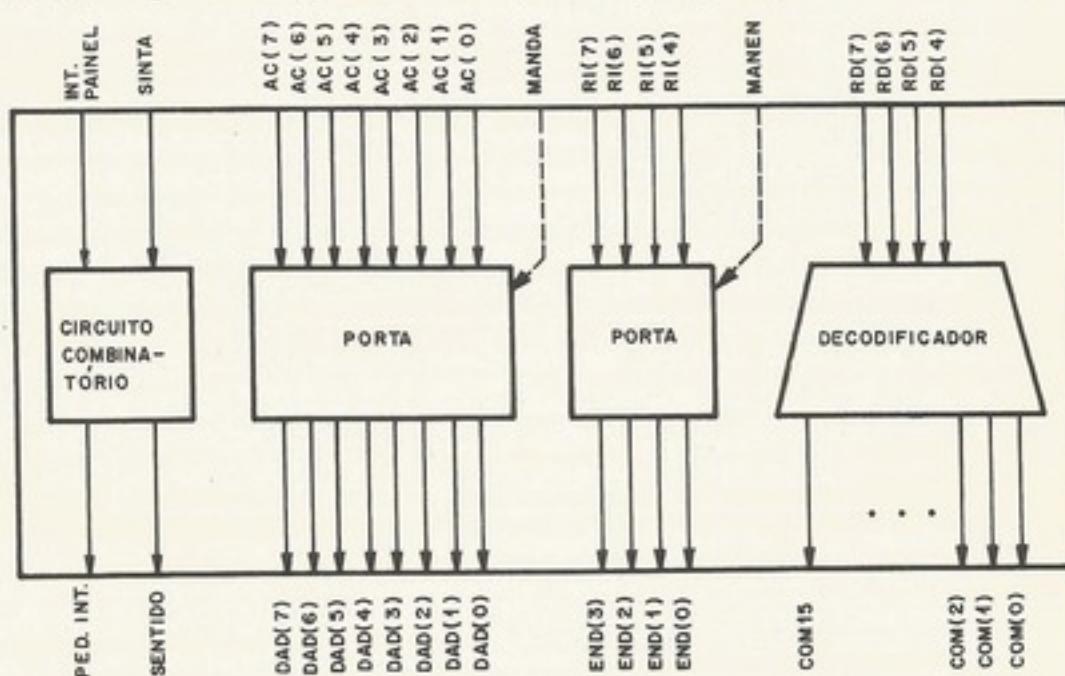


Figura 5-23. Esquema do cartão dos sinais de interface (FIN-8)

é, apenas a *fase 1*. É durante a fase de *fetch* que somamos um ao contador de instrução, para endereçar a instrução seguinte. Instruções longas têm duas fases de *fetch*; portanto somamos dois ao *CI*. Em resumo, durante a fase 1, a primeira palavra de instrução é lida da memória e colocada no *RI* e, durante a fase 2, a segunda palavra de instrução é lida da memória e colocada no *RD*.

Depois das fases de *fetch* ou busca, passa-se à fase de execução, onde a operação especificada pela ilustração é executada; é a *fase 3*. Algumas instruções, por necessitarem de dois acessos à memória, exigem dois ciclos de execução; *fase 3* e *fase 4*.

No caso de instruções indexadas, antes de entrarem nas fases de execução, existe a necessidade do cálculo do endereço efetivo, lendo o registrador de índice da memória (posição 0) e somando-o no endereço especificado. Existe uma fase especial para isso, a *fase 5*. Resumindo,

- fase 1} fases de *fetch* ou busca;
- fase 2}
- fase 3} fases de execução;
- fase 4}
- fase 5} fase de cálculo de endereço efetivo.

Durante a execução de um programa, a *UCP* vai pulando de fase em fase, sob o comando da unidade de controle. Por exemplo, da fase 1 de uma instrução de "soma indexada", evolui para a fase 2, daí para a fase 5 e volta para a fase 3. Para isso, existem cinco sinais elétricos,

mutuamente exclusivos, que são gerados durante uma fase para indicar qual a fase seguinte. São eles CF_1, CF_2, CF_3, CF_4 e CF_5 (condições para a fase 1, condições para a fase 2 etc.).

Quando existe uma instrução "pare" ou "espere" são bloqueadas todas as fases e o sistema fica sem fazer nada.

b. Microoperações

Cada instrução, em cada fase, é composta de inúmeros passos, seqüenciais no tempo, chamados de microoperações. Por exemplo, são microoperações:

incrementar um no contador de instruções, $CI \leftarrow CI + 1$;
transferir o conteúdo do contador de instruções para o registrador de endereço, $RE \leftarrow CI$;
pular para a fase 3, $F3 \leftarrow 1$.

Cada instrução, desde sua fase 1 até sua última fase, é executada, como já dissemos, pela combinação de microoperações. Por exemplo, a instrução que já analisamos, "soma indexada", tem quatro ciclos de memória, ou seja, fase 1, fase 2, fase 5 e fase 3. Cada fase dessas é composta de várias microoperações, que se sucedem ao tempo. Para a documentação da máquina, é importante que se conheçam, para cada instrução, suas fases e microoperações. Por isso, nas tabelas apresentadas a seguir (quatro), observam-se as cartas das microoperações.

Observar nas tabelas que algumas instruções estão agrupadas em algumas fases, visto que, então, elas se comportam da mesma maneira. Isso ajuda muito no projeto de unidade de controle que irá gerar sinais elétricos para a execução dessas microoperações. Nessas tabelas, o eixo horizontal é a variável tempo, nas unidades da máquina ($0,25 \mu s$).

c. Exemplo

Para ilustrar as tabelas, vejamos como o sistema executa a instrução de "soma indexada", já tão comentada. A fase 1 dessa instrução é

T_0	T_1	T_2	T_3	T_4	T_5	T_6	T_7
Memória: ler e restaurar							
	<i>Lwbo</i> <i>RI</i> ↗				<i>RE</i> ← <i>CI</i>	<i>F2</i> ← 1	
	<i>CI</i> ← <i>CI</i> + 1						

Inicia-se o ciclo lendo-se a memória. Nessa fase, a primeira palavra da instrução é lida na memória e, no final do tempo T_1 , é colocada em RI . Enquanto isso, incrementa-se um no contador de instrução. No final desse ciclo (T_5 e $T_6 = T_{56}$), colocamos o conteúdo de CI em RE para endereçar a segunda metade da instrução e, em T_7 , disparamos a fase 2, que será o ciclo de memória seguinte. É importante notar que, até T_2 , o sistema desconhecia a particular instrução que iria ser executada (no nosso exemplo, $SOMX$) e, por isso, *todas* as instruções são iguais na fase 1 e nos tempos T_0 , T_1 e T_2 .

A fase 2 de $SOMX$ será

T_0	T_1	T_2	T_3	T_4	T_5	T_6	T_7
Memória: ler e restaurar							
					<i>RE</i> ← <i>RI(4-7)</i> e <i>RD</i>		<i>F5</i> ← 1
	<i>RD</i> ↗ <i>Lwbo</i>						
	<i>CI</i> ← <i>CI</i> + 1						

Nesse ciclo, lemos a segunda palavra da memória e colocamo-la em RD , enquanto somamos um ao CI (para endereçar a próxima instrução). No final do ciclo, montamos o endereço [$RI(4-7)$ e RD] no registrador (RE), que serve para endereçar o operando, caso não seja indexado, ou para somar com o registrador de índice, caso seja indexado. Liga-se então a fase 5, já que é indexada.

A fase 5, que é a mesma para as quatro instruções indexadas é

T_0	T_1	T_2	T_3	T_4	T_5	T_6	T_7
Memória: ler e restaurar							
	$RD \leftarrow Lwbo$				$RE \leftarrow RE + RD$		$F3 \leftarrow 1$
$0 \rightarrow Lwa$							

Iniciamos lendo a memória e forçando o endereço "0" pela porta de seleção 5. No fim de T_1 , temos o índice em RD , que é somado com RE , sendo a soma guardada no próprio RE no final do ciclo. Portanto o endereço efetivo do operando já está em RE . A seguir, em T_7 , forçamos a fase 3.

A fase 3, de execução é

T_0	T_1	T_2	T_3	T_4	T_5	T_6	T_7
Memória: ler e restaurar							
	$RD \leftarrow Lwbo$			$AC \leftarrow AC + RD$	$RE \leftarrow CI$		$F1 \leftarrow 1$

Aqui, lemos o operando que fora endereçado no final do ciclo anterior. Em T_3 , esse operando é somado com o acumulador e o resultado guardado no acumulador. No final do ciclo, colocamos o conteúdo de CI em RE que corresponde a endereçar a próxima instrução, já que esta terminou. Então forçamos a fase 1.

5.6 UNIDADE DE CONTROLE

a. Considerações gerais

A unidade de controle é o centro motriz do sistema. É ela que gera todos os sinais elétricos no momento e no lugar certo para a realização das microoperações que, combinadas, resultam nas instruções.

Essa unidade tem como informação os sinais de tempo do relógio central, as fases e a instrução corrente. É, basicamente, um circuito combinatório, montado com portas da família TTL 400 de circuitos integrados, que gera os sessenta sinais que abrem portas, disparam flip-flops, dão clock em registradores, forçam um na entrada VE da unidade aritmética etc. A combinação conveniente desses sinais elétricos produz uma microoperação que é parte de uma instrução.

Por exemplo, os sinais

- $CI(0-3)/M$ (abre a porta de seleção 1 para o $CI(0-3)$),
- $CI(4-11)/X$ (abre a porta de seleção 3 para o $CI(4-11)$),
- VU (força 1 na entrada VE do somador),
- $SOXOR$ (estrutura de soma na unidade aritmética).

seguidos, depois de 500 μs do *clock*, em CI , perfazem microoperação

$$CI \leftarrow CI + 1$$

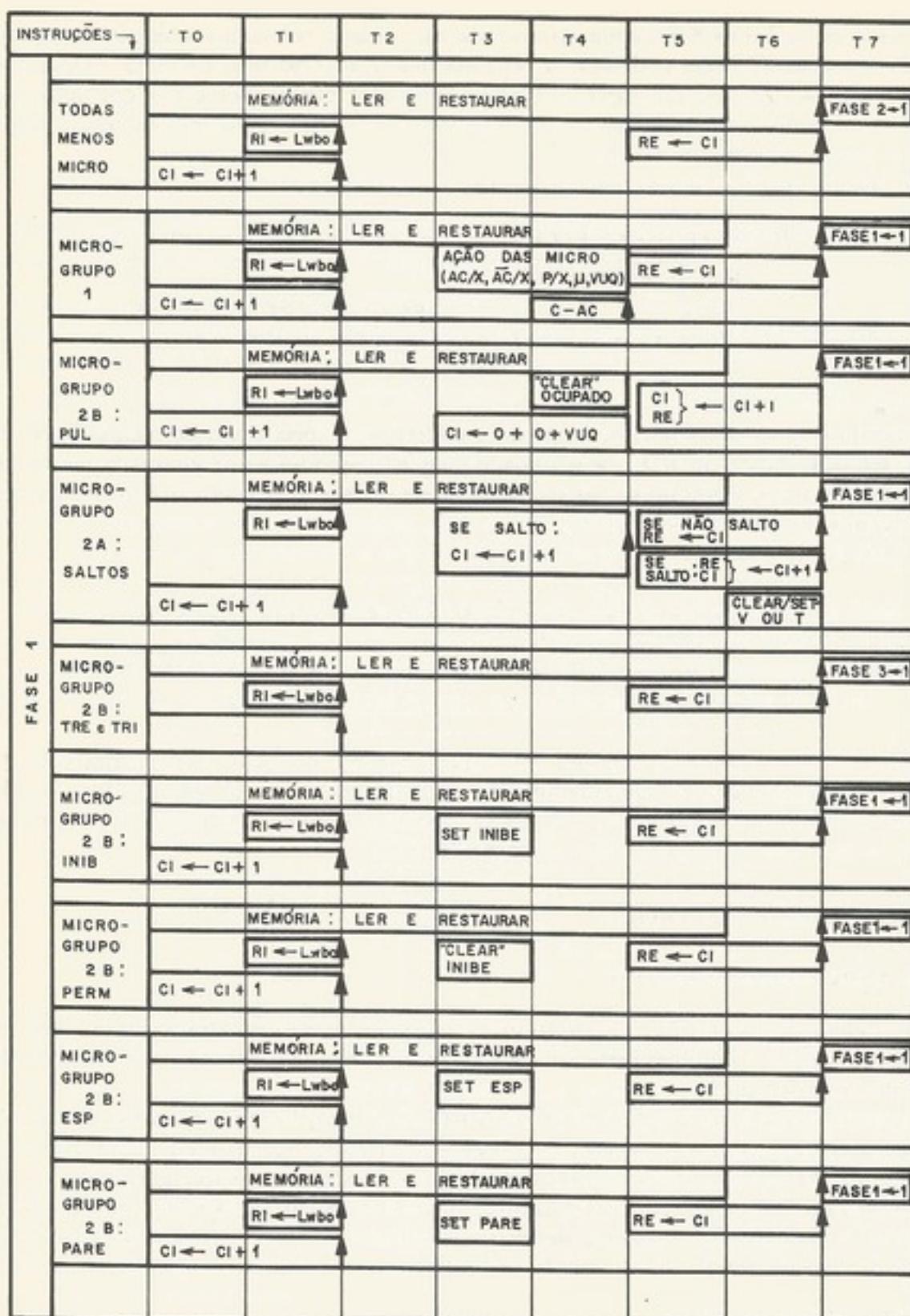


Figura 5-24. Carta de microoperações da fase 1

Portanto o projeto de unidade de controle é feito tendo-se por base as cartas de microoperações. Temos as condições e os tempos em que devem ocorrer as microoperações. Portanto tiramos das cartas as condições e tempos nos quais os sinais elétricos de controle devem ser "1" ou "0". O problema seguinte será o projeto lógico do circuito que gera esses sinais.

INSTRUÇÃO →	T0	T1	T2	T3	T4	T5	T6	T7
CAR ARM (COM OU SEM ÍNDICE)		MEMÓRIA : LER E RESTAURAR RD ← Lwbo CI ← CI + 1				RE ← RI(4-7) e RD		SEM ÍNDICE FASE 3-1 COM ÍNDICE FASE 5-1
SUS PUG		MEMÓRIA : LER E RESTAURAR RD ← Lwbo CI ← CI + 1				RE ← RI(4-7) e RD		FASE 3-1
PLA PLAX		MEMÓRIA : LER E RESTAURAR RD ← Lwbo CI ← CI + 1				RE CI ← RI(4-7) e RD		SEM ÍNDICE FASE 1-1 COM ÍNDICE FASE 5-1
PLAN		MEMÓRIA : LER E RESTAURAR RD ← Lwbo CI ← CI + 1				SE AC(0) ← RE(4-7)e RD SE AC(0) : RE ← CI		FASE 1-1
PLAZ		MEMÓRIA : LER E RESTAURAR RD ← Lwbo CI ← CI + 1				SE AC(0; RE) ← RE(4-7)e RD SE AC(0; RE) ← CI		FASE 1-1
NAND XOR		MEMÓRIA : LER E RESTAURAR RD ← Lwbo CI ← CI + 1		AC ← AC { SOMA NAND > RD XOR }		RE ← CI		FASE 1-1
DESLOCA- MENTOS		MEMÓRIA : LER E RESTAURAR RD ← Lwbo CI ← CI + 1	SE RD(4)=1 C-AC	SE RD(5)=1 C-AC	SE RD(6)=1 C-AC	SE RD(7)=1 C-AC		FASE 1-1
SAL		MEMÓRIA : LER E RESTAURAR RD ← Lwbo CI ← CI + 1	BUS Z+U ← CI + 1 SE SENTIDO: SET S SINTA = 1	SE S=1: RE ← CI + 1 SE S=1: C-AC SE S=D: RE ← CI				FASE 1-1
FNC		MEMÓRIA : LER E RESTAURAR RD ← Lwbo CI ← CI + 1				RE ← CI		FASE 1-1
SAI		MEMÓRIA : LER E RESTAURAR RD ← Lwbo CI ← CI + 1		DADOS SAÍDA				FASE 1-1
ENT		MEMÓRIA : LER E RESTAURAR RD ← Lwbo CI ← CI + 1		AC ← E		RE ← CI		FASE 1-1

Figura 5-25. Carta de microoperações da fase 2

INSTRUÇÕES		T0	T1	T2	T3	T4	T5	T6	T7
FASE 3	CAR	MEMÓRIA: LER E RESTAURAR							
	CARX	RD ← Lwbo		AC ← RD		RE ← CI		FASE1 ← 1	
	ARM	MEMÓRIA: LER SOMENTE	MEMÓRIA: ESCREVER						
	ARMX	RD ← AC				RE ← CI		FASE 1 ← 1	
	SOM	MEMÓRIA: LER E RESTAURAR							
	SOMX	RD ← Lwbo	AC ← AC + RD		RE ← CI		FASE 1 ← 1		
	PUG	MEMÓRIA: LER SOMENTE	MEMÓRIA: ESCREVER			RE ← RE + 1		FASE 4 ← 1	
FASE 4	TRE	MEMÓRIA: LER SOMENTE	MEMÓRIA: ESCREVER						
		O/Lwbo						FASE 4 ← 1	
		RD ← Lwbo	RE ← RD						
	TRI	MEMÓRIA: LER SOMENTE	MEMÓRIA: ESCREVER						
FASE 5		O/Lwbo						FASE 4 ← 1	
		RD ← Lwbo	RE ← RD						
	SUS	MEMÓRIA: LER SOMENTE	MEMÓRIA: ESCREVER						
FASE 6		SE RDZ = 0: RD ← RD - 1		SE S = 0: RE ← CI					
		SE RDZ ≠ 0: SETS	SE S = 1: CI ← CI + 1	SE S = 1: RE ← CI + 1					
	TRE	MEMÓRIA: LER SOMENTE	MEMÓRIA: ESCREVER						
		1 / Lwq						FASE 1 ← 1	
		RD ← AC	AC ← RE	RE ← CI					
	TRI	MEMÓRIA: LER SOMENTE	MEMÓRIA: ESCREVER						
		O / Lwq						FASE 1 ← 1	
FASE 7	PUG	MEMÓRIA: LER SOMENTE	MEMÓRIA: ESCREVER						
		RD ← CI (4 - 1)		(RE) ← RE + 1				FASE 1 ← 1	
FASE 8	CARX	MEMÓRIA: LER E ESCREVER							
	ARMX	O / Lwq						SE PLI: FASE 1 ← 1	
	SOMX	RD ← Lwbo			RE ← RE + RD			SE NAO PLI: FASE 4 ← 1	
	PLAX				SE PLI 1 C ← RE + RD				

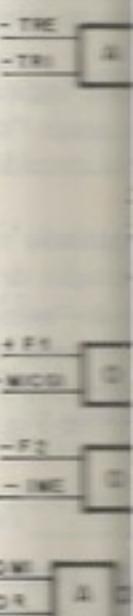
Figura 5-26. Carta de microoperações das fases 3, 4 e 5

b. Exemplo

Como exemplo bastante simples, veja que o conteúdo da inspeção na carta de condições:

instrução de LER
instrução de T
instrução de S
instruções de I
instruções de S
T34;

instrução de S
Na Fig. 5-27, a combinatória que em cinco placas de



c. Decodificação

Para a geração corrente. Para isso, os bits do RI, que são muitos em grande parte, e a decodificação em a esses grupos de RI, no caso de serem que têm a mesma

d. Controle de

Durante o funcionamento da máquina. Quando busca ou círculos de

b. Exemplo

Como exemplo do circuito da unidade de controle, vejamos um dos sinais de geração bastante simples, o sinal AC/X , ou seja, aquele que age na porta de seleção 3, fazendo com que o conteúdo do acumulador seja colocado na entrada X do somador. Por uma simples inspeção na carta de microoperações, vemos que esse sinal elétrico deve existir nas seguintes condições:

- instrução de *ARM* ou *ARMX*, na fase 3, em $T12$;
- instrução de *TRE* ou *TRI*, na fase 4, em $T12$;
- instrução de *SUS*, com $RD \neq 0$, na fase 3, em $T23$;
- instruções do grupo *MICG1*, com $RI(5) = 1$, na fase 1, em $T34$;
- instruções de *SOMI* ou *NAND* ou *XOR*, e não *SOMI* e *XOR* juntos na fase 2, no tempo $T34$;
- instrução de *SOM* ou *SOMX*, em $T34$, da fase 3.

Na Fig. 5-27, encontra-se o circuito combinatório que gera o sinal AC/X^* . Os circuitos combinatórios que geram os sinais de controle do fluxo de dados foram implementados em cinco placas de circuitos. As Pranchas de XII até XVI mostram os detalhes dos circuitos.

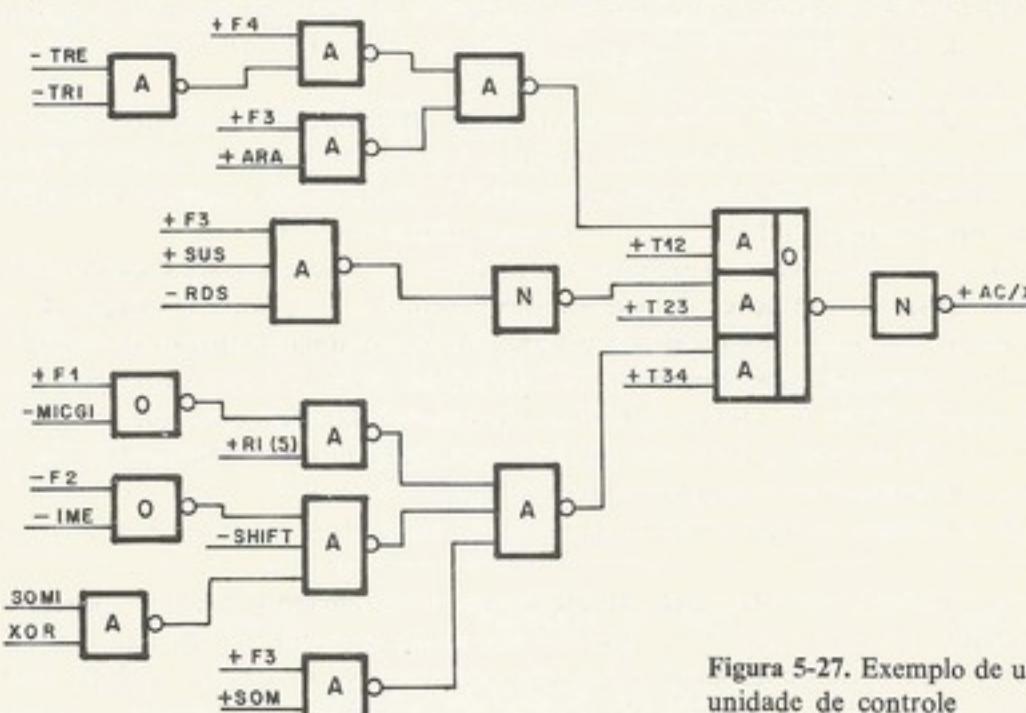


Figura 5-27. Exemplo de um sinal da unidade de controle

c. Decodificador

Para a geração correta dos sinais de controle, é importante que se saiba a instrução corrente. Para isso, existe, na unidade de controle, um decodificador, cuja entrada são os bits do RI , que armazena a primeira palavra de instrução. Como muitas instruções têm, em grande parte, as mesmas microoperações, estas agrupam-se em códigos convenientes e a decodificação em árvore (veja o Cap. 3) é eficiente, pois gera sinais de decodificação comum a esses grupos de instruções. Por exemplo, a decodificação do bit 0 (mais significativo) de RI , no caso de ser igual a 0, informa-nos que é uma das instruções *PLA*, *SOM*, *CAR* ou *ARM*, que têm a característica comum de serem indexáveis.

d. Controle de estado

Durante o funcionamento normal, podem-se distinguir cinco tipos diferentes de ciclo de máquina. Chamam-se de fase 1, fase 2, fase 3, fase 4 e fase 5. As fases 1 e 2 são ciclos de busca ou ciclos *I*. Na fase 1, a primeira palavra da instrução é lida na memória e, na fase 2,

a segunda palavra da instrução. A fase 3 é o ciclo de execução, onde é lido ou gravado o operando na memória e, como algumas instruções exigem dois acessos à memória para serem executadas, existe a fase 4. Instruções indexadas exigem um ciclo de máquina diferente, para ler o índice na posição zero da memória e calcular o endereço efetivo; é a fase 5.

Note-se que o simples controle das fases resolve o problema da seqüencialização de programa. Se, em toda a fase 1, é lida na memória a primeira palavra de uma instrução, endereçada pelo contador de instruções, e se, em toda a fase de busca, atualizar-se o contador de instruções, bastará, após se executar cada instrução, encaminhar-se automaticamente, para a fase 1, para que a seqüencialização do programa seja garantida.

Para o controle das fases, o "gerador de sinais de controle" fornece cinco sinais, mutuamente exclusivos,

- CF1*, condição para fase 1;
- CF2*, condição para fase 2;
- CF3*, condição para fase 3;
- CF4*, condição para fase 4;
- CF5*, condição para fase 5;

Durante uma fase, esses sinais indicam qual será a fase seguinte. Por exemplo, se se está na fase 2 de uma instrução de *CAR*, *CF3* está no nível "1" e os outros no nível "zero", indicando que a fase seguinte é a fase 3.

Na placa de controle de estado, existem cinco *flip-flops* cujos estados indicam fases, um *flip-flop* para cada fase. O controle das fases é feito colocando-se os sinais "condições de fase" na entrada *D* dos *flip-flops* e no inicio de *T7*, dando-se um pulso na entrada de controle (*clock*) do *flip-flop*. A Fig. 5-28 ilustra essa idéia.

Durante o funcionamento, o sistema vai pulando de fase em fase, seguindo o que lhe dita os cinco sinais de condições de fase. Se, por exemplo, após uma instrução de *PUG* (a fase 4 é a última fase dessa instrução), o sistema encontrar uma instrução de "soma", ocorrerá o seguinte: durante a fase 1, o sinal *CF2* estará no nível 1, o que provocará, em *T7*, o disparo do *flip-flop* "fase 2"; na fase 2, *CF3* é que será "1", fazendo com que o *flip-flop* "fase 3" se ligue em *T7*; na fase 3, *CF1* será ativado para ler a nova instrução. Na Fig. 5-8 tem um diagrama que ilustra, no tempo, essa evolução.

Ao ligar o sistema, deve-se iniciá-lo na fase 1. O sinal *-LAL*, visto na Fig. 5-28, tem essa função.

Com essa filosofia de fases, fica simples parar o sistema no meio do processamento (instrução "pare", por exemplo). Se o "gerador dos sinais de controle" utilizar sempre o sinal de fase para produzir qualquer sinal, a inibição dos cinco sinais de fase, inibirá todo o sistema. O sinal "roda" (Fig. 5-28), quando igual a "0", inibe as fases parando o computador.

É controlando o sinal "roda" que se operam os modos especiais de funcionamento, como o ciclo único e a instrução única. Esse sinal é composto por três outros (Fig. 5-29), "roda 1", "roda 2" e "roda 3".

"Roda 1"

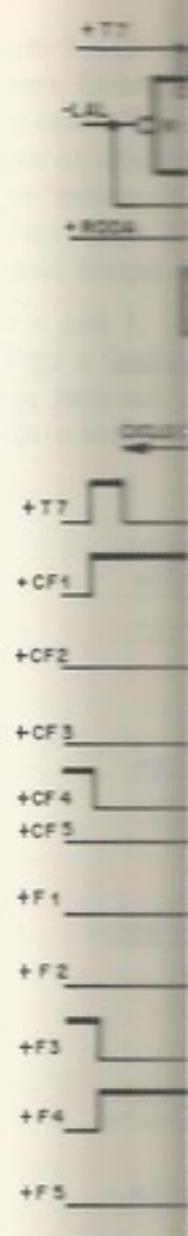
Usado durante o modo normal de funcionamento. Fica sempre no nível "1" até que se pare o sistema. As condições para ligá-lo e desligá-lo são as que seguem.

Ligar. Quando se aciona o botão de partida ou quando se está no estado de espera e chega um pedido de interrupção.

Desligar. Sempre antes de uma fase 1, nas seguintes circunstâncias: instrução de "pare", instrução ou acionamento do "espere" ou tirando-se do modo normal de funcionamento.

"Roda 2"

Usado no modo especial "instrução única" (IS). É ligado quando se está nesse modo de funcionamento e se pressiona o interruptor de partida. É desligado sempre que se vai entrar numa fase 1.



"Roda 3"

Utilizado no modo de funcionamento de um ciclo. Utilizado para iniciar o processo de rodagem.

Uma vez que a rodagem é iniciada, o operador e o sistema permanecem nela até que a rodagem seja finalizada. O sinal de rodagem é gerado a partir dos outros sinais.

Um pouco mais detalhadamente, deve existir um sinal de rodagem que é usado para controlar os flip-flops, sendo o sinal de rodagem gerado a partir do sinal de partida.

lido ou gravado o operando na memória para serem usados em uma máquina diferente, para isso é a fase 5.

A sequencialização de uma instrução, endereçando-se o contador automaticamente,

emite cinco sinal

Por exemplo, se se

estados indicam fases,

segundo o que lhe

Fig. 5-28, tem essa

modo do processamento

utilizar sempre o sinal

“roda”

“modo 3”

“modo 1” até que se

no estado de espera e

instrução de “pare”,

modo de funcionamento.

se está nesse modo

sempre que se vai

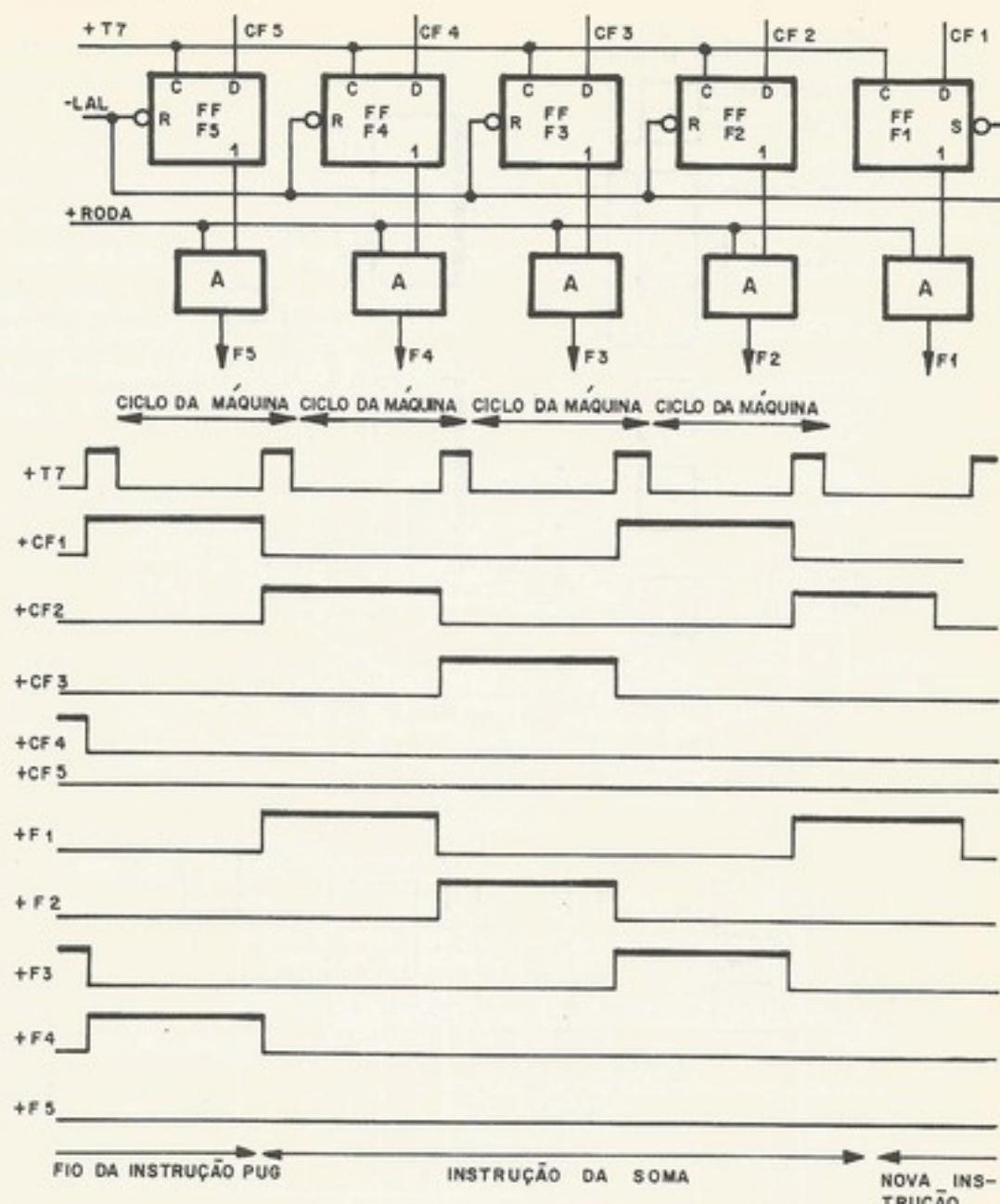


Figura 5-28. Controle das fases

“Roda 3”

Utilizado no modo especial “ciclo único” (CS). É um sinal que fica no nível 1 por apenas um ciclo. Utiliza-se o próprio sinal de partida, que dura um ciclo, para gerá-lo.

Uma tarefa importante do circuito “controle de estado” é a provisão da interface entre o operador e o sistema. Já se comentou que o operador, ao selecionar o modo de operação, acaba, no final, controlando o sinal “roda”. O acionamento do interruptor “espere” irá ligar o flip-flop correspondente, que, por sua vez, desligará o sinal “roda 1”. O interruptor “preparação” (preset) age diretamente no circuito de geração do LAL (limpa ao ligar), parando o relógio central, forçando a fase 1, desligando interrupção etc.

Um pouco mais delicada é a geração do sinal “partida”, já que, a cada pressão da chave ele deve existir por apenas um ciclo, começando em T0 e acabando em T7. Isso porque esse sinal é usado em inúmeros pontos onde o sincronismo é vital. Um circuito com três flip-flops, sendo o primeiro eliminador de ruído (bounce) da chave, gera esse sinal, chamado “partida”.

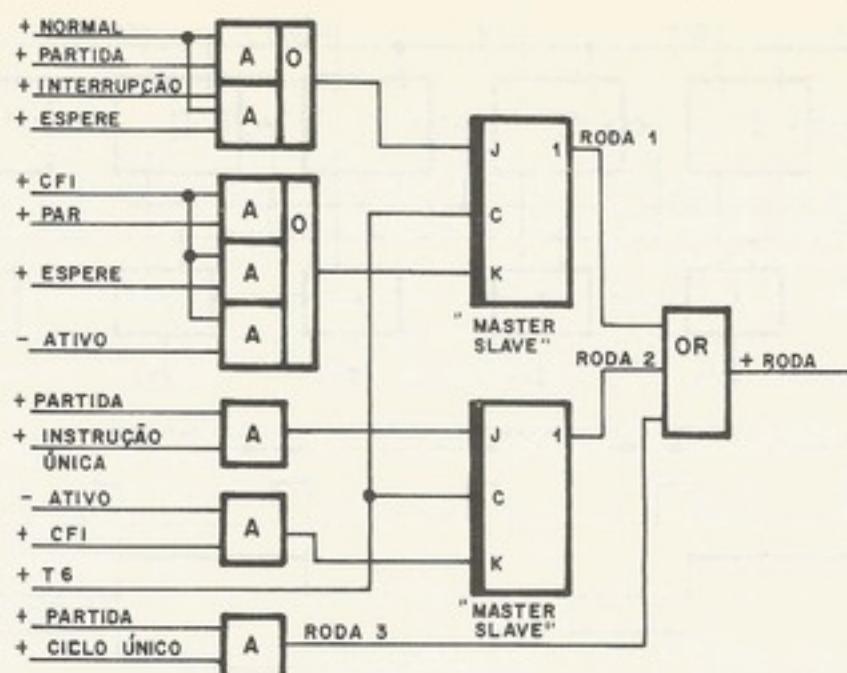


Figura 5-29. Geração do sinal "roda"

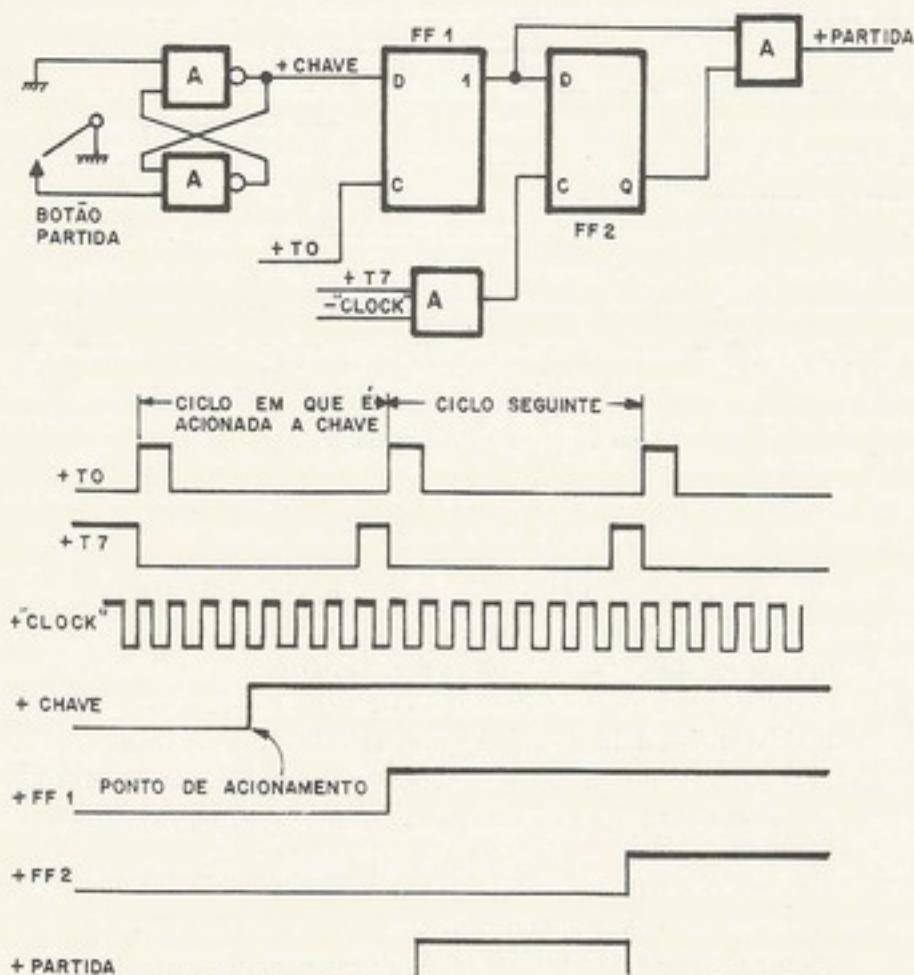


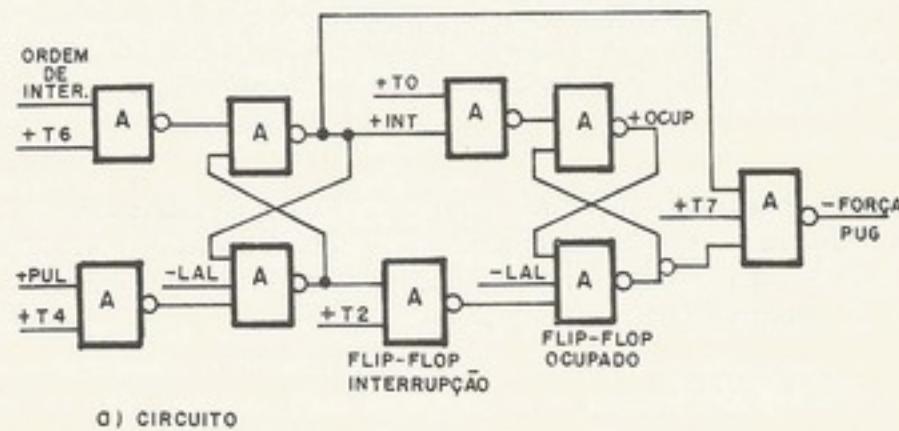
Figura 5-30. Geração do sinal "partida"

Outra função é a geração do sinal "partida". Quando o sinal de partida é gerado, a fase 1 se inicia. O sinal de partida é gerado dependendo das condições (ver se não é o caso) e é chamado "ordem de partida" ou "sinal de interrupção" no tempo T. Nesse meio tempo, o sinal de partida liga a fase 2, desliga a fase 1, força a instrução de partida, força o endereço de memória e

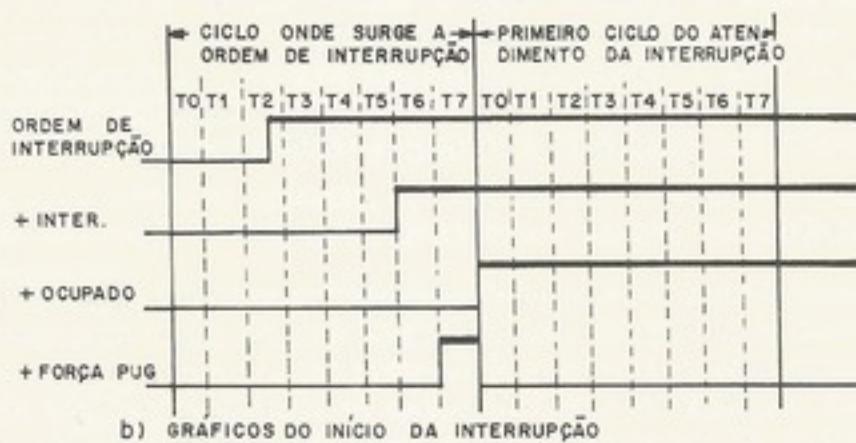
liga a fase 3, desliga a fase 2, força a instrução de partida, força o endereço de memória e

Outra função do circuito "controle de estado" é controlar a interrupção (veja o próximo item). Quando chega um pedido de interrupção, ele é atendido antes que a próxima fase 1 se inicie. O pedido de interrupção passa por algumas portas que testam certas condições (ver se não está ocupado, se não está inibido, se é fase 1 etc.) e se transforma num sinal chamado "ordem de interrupção". Esse sinal (Fig. 5-31) dispara um flip-flop chamado "interrupção" no tempo T_6 . No segmento de tempo T_0 seguinte, é disparado o flip-flop "ocupado". Nesse meio tempo (em T_7), é gerado o sinal "força PUG", que realiza o seguinte:

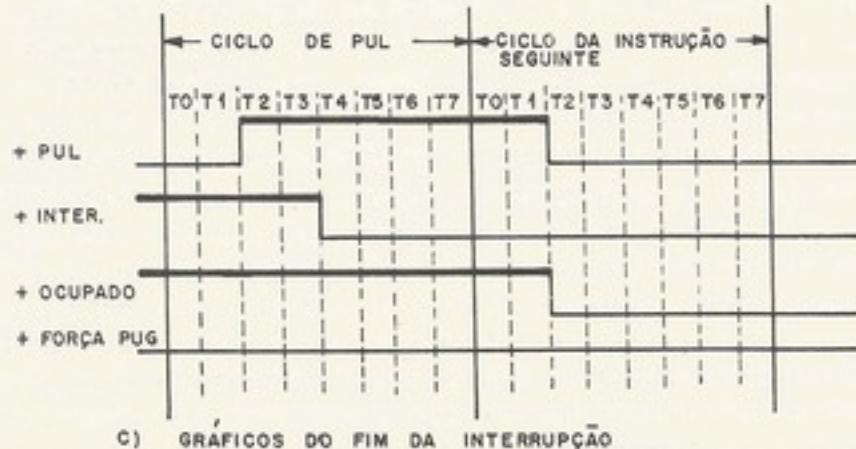
- liga a fase 3,
- desliga a fase 1,
- força instrução de PUG no RI (set RI),
- força o endereço 2 no RE.



a) CIRCUITO



b) GRAFICOS DO INÍCIO DA INTERRUPÇÃO



c) GRAFICOS DO FIM DA INTERRUPÇÃO

Figura 5-31. Mecanismo de interrupção

Isso tudo equivale a iniciar uma instrução de *PUG* para a posição 2 da memória.

Os *flip-flops* *INT* e "ocupado" ficam ligados durante todo o tempo de atendimento da interrupção, e apenas uma instrução de *PUL* (ou acionamento da chave "preparação") os desliga. Para que não se perca o endereço de volta ao programa principal, que, durante o atendimento da interrupção, ficou armazenado na posição 2 da memória, o *flip-flop* "ocupado" é desligado no ciclo seguinte ao ciclo de *PUL*, aproveitando-se o fato de que, enquanto o *flip-flop* ocupado se mantiver no nível "1", nenhum outro pedido de interrupção será atendido.

Na Prancha XI, existem os circuitos da placa de controle de estado. Nessa prancha encontram-se alguns outros detalhes que merecem algumas palavras.

Gerador do LAL

É um circuito de componentes discretos. Basicamente é um *SCR* que, ao ligar o sistema, encontra-se cortado, fazendo com que *LAL* = "1". Ao se acionar, pela primeira vez, o interruptor de partida, o *SCR* conduz, levando o *LAL* para "0".

Sinal FN

É um sinal, chamado *fase normal*, controlado por uma chave que se encontra dentro da máquina. Quando *FN* = "1", o sistema funciona normalmente. Ao se acionar a chave fazendo *FN* = "0", deixa de existir a evolução das fases, ficando o sistema rodando preso a uma fase. É utilizado na depuração do sistema.

Sinal "partida atrasada"

É um sinal que inicia em *T6* de um sinal "partida" e acaba em *T5* do ciclo seguinte. Ele é usado para limpar os *flip-flops* "pare" e "espere". Quando se aciona o interruptor de partida para sair de um desses estados, o sinal "roda 1" é energizado no final do ciclo de partida. Com o "roda 1" ligado, nova instrução será lida e o decodificador que estava dando o sinal da instrução anterior de "pare" ou "espere" passa a indicar a nova instrução. Então, pode-se limpar (no ciclo seguinte ao sinal de partida) o estado de "pare" ou "espere".

5.7 INTERRUPÇÃO

Suponha que queiramos gravar num disco magnético, que esteja ligado ao nosso computador, vários blocos de informação. Um bloco, uma unidade que o disco grava, é formado de inúmeras palavras. Existe um cartão de interface fazendo a comunicação entre o disco e a *UCP*, que tem um *buffer* de tamanho igual ao do bloco. O que a *UCP* faz é preencher o *buffer* da interface. (O programador indica isso com uma instrução do tipo *SAI, n, c*.) Quando termina o preenchimento, a *UCP* envia ao cartão de interface um sinal de "pode gravar no disco" a partir de uma instrução tipo *FNC, n, c* e fica esperando o término da gravação para preencher o *buffer* com o novo bloco. Esse tempo de espera da *UCP* é ocioso e poderia ser aproveitado para fazer outras coisas. É aí que entra o conceito de *interrupção*.

A idéia geral de interrupção é a seguinte: a *UCP* está executando um programa normal, quando chega um pedido de interrupção do dispositivo doze (por exemplo). Então ela pára o que estava fazendo, executa um programinha especial para esse dispositivo, ao término do qual volta ao ponto onde parara e continua o processamento normal.

Vejamos agora o nosso problema do disco. A *UCP* carrega o *buffer* com um bloco de palavras e volta, no fim, ao processamento normal, enquanto o bloco é gravado no disco. Quando o disco termina de gravar o bloco, o cartão de interface interrompe a *UCP*, que irá parar o que estava fazendo para encher novamente o *buffer* e voltar ao seu trabalho anterior.

Evidentemente a importância desse conceito é enorme, e apenas demos uma idéia intuitiva e bastante vaga do que seja. Podemos imaginar que o sistema seja ligado a máquinas

e aparelhos de menor porte, deles, é interrompido o processamento normal.

No minicomputador, segue.

Como já sabemos, o índice e extensão da rotina tratada deve terminar a execução da rotina tratada pela unidade de set (1111 = *PUL*) no endereço 2.

Todas essas informações assim que chegam ao minicomputador são colocadas em "processamento é desligado" da tradutora da instrução de *PUL* que equivale ao endereço 2.

Quando o sistema de uma instrução é interrompido, provoca uma interrupção, apenas na rotina geral. Por isso, é necessário que o sinal "ocupado" no endereço 2, só ocorre

5.8 MODOS DE OPERAÇÃO

Além do modo de operação normal, uma instrução de operação que são os seguintes:

grupo 1

grupo 2

Existem seis chaves mutuamente exclusivas.

a. Grupo 1

"Ciclo único"

É um modo de operação que evolui apenas sob comando do operador do sistema. Cipalmente, é dividido

2 da memória.
de atendimento
“preparação”)
principal, que, durante
memória, o flip-flop
o fato de que,
fim de interrupção
Nessa prancha

ao ligar o sis-
peia primeira vez,

encontra dentro
acionar a chave
rodando preso

do ciclo seguinte.
o interruptor de
no final do ciclo
lizador que estava
a nova instrução.
“pare” ou “espere”.

nosso compu-
grava, é formado
ção entre o disco
CP faz é preencher
do tipo SAI, n, c.)
um sinal de “pode
m o término da
da UCP é ocioso
de interrupção.

programa normal,
Então ela pára
ativo, ao término
m um bloco de
gravado no disco.
a UCP, que irá
trabalho anterior.
uma idéia intui-
ligendo a máquinas

e aparelhos de medições e que, quando houver alguma irregularidade com qualquer um deles, é interrompido para resolver o “galho”. Enquanto isso não acontecer, ele atende ao processamento normal.

No minicomputador do LSD, a interrupção funciona da maneira que expomos a seguir.

Como já sabemos, as posições 0 e 1 da memória são reservadas para o registrador de índice e extensão do acumulador. Nas posições 4 e 5, existe uma instrução de desvio para a rotina tratadora da interrupção. Quando chega um pedido de interrupção, a UCP espera terminar a execução da instrução corrente e, em vez de ir para a fase 1 da seguinte, ela é forçada pela unidade de controle a ir para a fase 3 (execução) enquanto o RI recebe um pulso de set (1111 = PUG) e o RE recebe também um pulso na sua linha de set-reset, sendo forçado o endereço 2.

Todas essas coisas equivalem à introdução de uma instrução de PUG para a posição 2 assim que chega um pedido de interrupção. Com essa instrução de PUG nas posições 2 e 3, é colocado um “pulo incondicional” (código 0000) para o CI do programa normal e o processamento é desviado para as posições 4 e 5, que têm um “pulo incondicional” para a rotina tratadora da interrupção. No final do atendimento da interrupção, existe uma instrução de PUL que equivale a um “pulo incondicional” para a posição 2.

Quando o sistema atende a um pedido de interrupção ele só atenderá ao seguinte depois de uma instrução após a de PUL, para não perder o endereço do programa normal. Isso provoca uma situação incomum, pois a instrução de PUL deverá agir, limpando a interrupção, apenas na instrução seguinte, que, no caso, é um PLA, que é uma instrução de uso geral. Por isso, é necessário o circuito visto na Fig. 5-31, que mostra como o PUL limpa o sinal “ocupado” no ciclo seguinte. Para atendê-lo, basta lembrarmos que o tempo T2, após um T4, só ocorre no ciclo seguinte.

5.8 MODOS DE OPERAÇÃO ESPECIAIS

Além do modo normal de operação, aquele onde o sistema fica operando até encontrar uma instrução de “pare” ou “espere”, existem outros modos, que chamamos de especiais, que são os seguintes:

grupo 1 “ciclo único”,
“instrução única”;

grupo 2 “carrega endereço”,
“carrega posição”,
“mostra posição”.

Existem seis chaves no painel para selecionarmos um dos modos de operação, os quais são mutuamente exclusivos.

a. Grupo 1

“Ciclo único”

É um modo de funcionamento onde o sistema, a cada acionamento do botão de partida, evolui apenas um ciclo. Isso permite que, de ciclo em ciclo, isto é, de fase em fase, o operador do sistema acompanhe a evolução de seu programa. Seus objetivos são, principalmente, didáticos e de diagnose de falhas do sistema.

"Instrução única"

É um modo de funcionamento onde o sistema, a cada acionamento do botão de partida, executa uma instrução e pára na fase 1 da próxima. Serve para o operador acompanhar o seu programa de instrução a instrução. Seu objetivo é o *debug* de programas.

A implementação física desses modos de funcionamento foi feita controlando os sinais de fase. Existe um sinal elétrico chamado "roda", que controla as fases. No modo normal de funcionamento, esse sinal fica permanentemente no nível 1, permitindo a franca evolução do programa. No modo especial "ciclo único", a cada pressionamento do interruptor de partida, o sinal "roda" eleva-se ao nível 1 até o fim de T_7 , quando volta a zero e, assim, a cada partida que se dá, teremos a liberação do sinal de fase por um ciclo, fazendo com que exista o processamento normal por uma fase apenas. No outro modo especial de funcionamento ("instrução única"), o comportamento do sistema é análogo com a única diferença de que o sistema pára de operar a cada vez que vai entrar na fase 1. Portanto o sistema, a cada acionamento do botão de partida, executa *uma* instrução completa e pára. O leitor encontra na Fig. 5-36 um esquema lógico do circuito de controle do modo de operação.

b. Grupo 2

Os modos especiais de funcionamento desse grupo têm a característica particular de fazer o sistema funcionar *sem fases*. O sinal do interruptor de partida tem a forma vista na Fig. 5-20.



Figura 5-32. Sinal de partida

O sistema, com o modo especial de funcionamento desse grupo opera usando o sinal de partida ao invés do de fase, tendo uma seqüência de microoperações específicas.

Carrega endereço

Com o acionamento do interruptor de partida, o conteúdo do registrador de chaves do painel é transferido para o registrador de endereço da memória e o contador de instrução (Fig. 5-33).

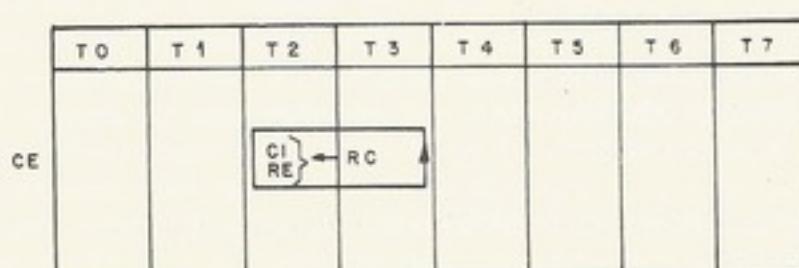
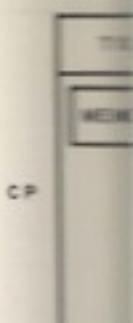


Figura 5-33. Distribuição das microoperações no modo "carrega endereço"

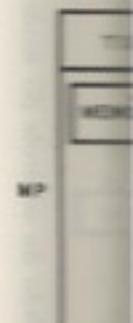
Carrega posição

Ao se acionar o interruptor de partida, o conteúdo dos 8 bits menos significativos do registrador de chaves do painel é armazenado na posição de memória cujo endereço está no RE (Fig. 5-34).



Mostra posic

Com o accu
cujo endereço
do conteúdo d



Como voce
está na iniciativa
programa (carreg
remos endereço
grama na memó
que permite a i

5.9 SUMÁRIO

Após termos
um sistema digi
onde procuramos
Neste ponto
mentos e uma re
sistema digital. E
dissemos e, salvo

A partir de
de "arquitetura"

do botão de parada para o operador acompanhar os programas.

Visualizando os sinais de saída. No modo normal de funcionamento a franca evolução do interruptor de partida é para zero e, assim, a unidade fazendo com que especial de funcionamento a única diferença é que o sistema, a unidade para. O leitor muda de operação.

única particular de forma vista na

memória usando o sinal específico.

operador de chaves
operador de instrução

significativos do endereço está

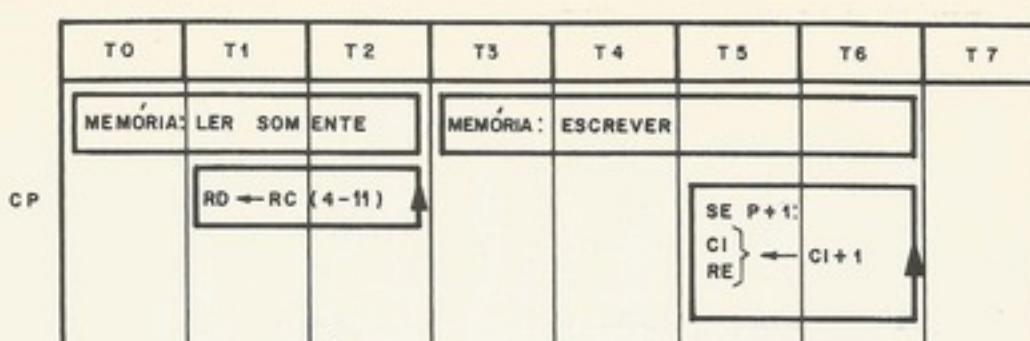


Figura 5-34. Seqüência das microoperações no modo "carrega posição"

Mostra posição

Com o acionamento do interruptor de partida, o conteúdo da posição de memória cujo endereço está no RE é transferido para o RD, permitindo ao operador a visualização do conteúdo dessa posição da memória (Fig. 5-35).

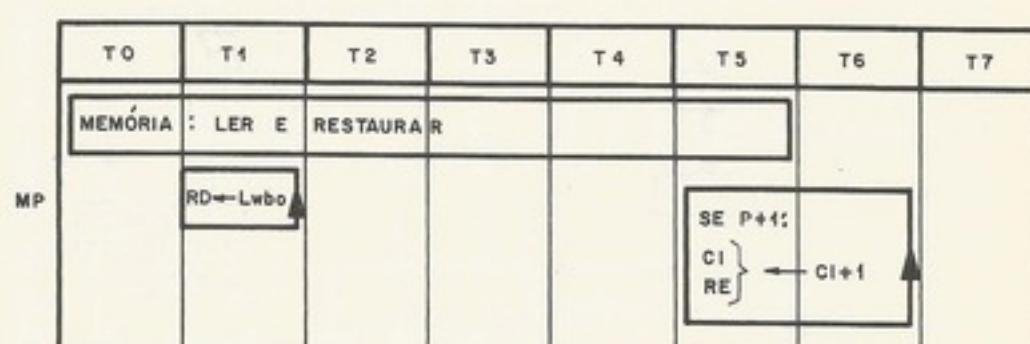


Figura 5-35. Seqüência das microoperações no modo "mostra posição"

Como você já deve ter notado, a importância dos modos de funcionamento do grupo 2 está na inicialização do sistema e *debug* de programas. Se quisermos carregar o primeiro programa (carga fria), deveremos fazê-lo pelas chaves do painel. Ao ligar o sistema, deveremos endereçar o "programa carregador" pelas chaves. É possível a análise de um programa na memória, ou mesmo sua modificação, pelo próprio painel. Enfim, é esse grupo que permite a interação do operador no sistema, através das chaves.

5.9 SUMÁRIO

Após termos estudado, nos capítulos anteriores, os elementos e conceitos que formam um sistema digital, vimos, neste capítulo, uma breve descrição de um minicomputador onde procuramos mostrar, em síntese, um exemplo da reunião daqueles conceitos.

Neste ponto, acreditamos que você já tenha adquirido uma visão detalhada dos elementos e uma visão geral do conjunto, entendendo como funciona e como se projeta um sistema digital. Evidentemente, muitas dúvidas existem, pois, sobre muitos detalhes, nada dissemos e, sobre muitos conceitos, apenas demos uma idéia vaga.

A partir de agora, nos próximos capítulos, veremos, com mais detalhes, os conceitos de "arquitetura" e "entrada/saída".

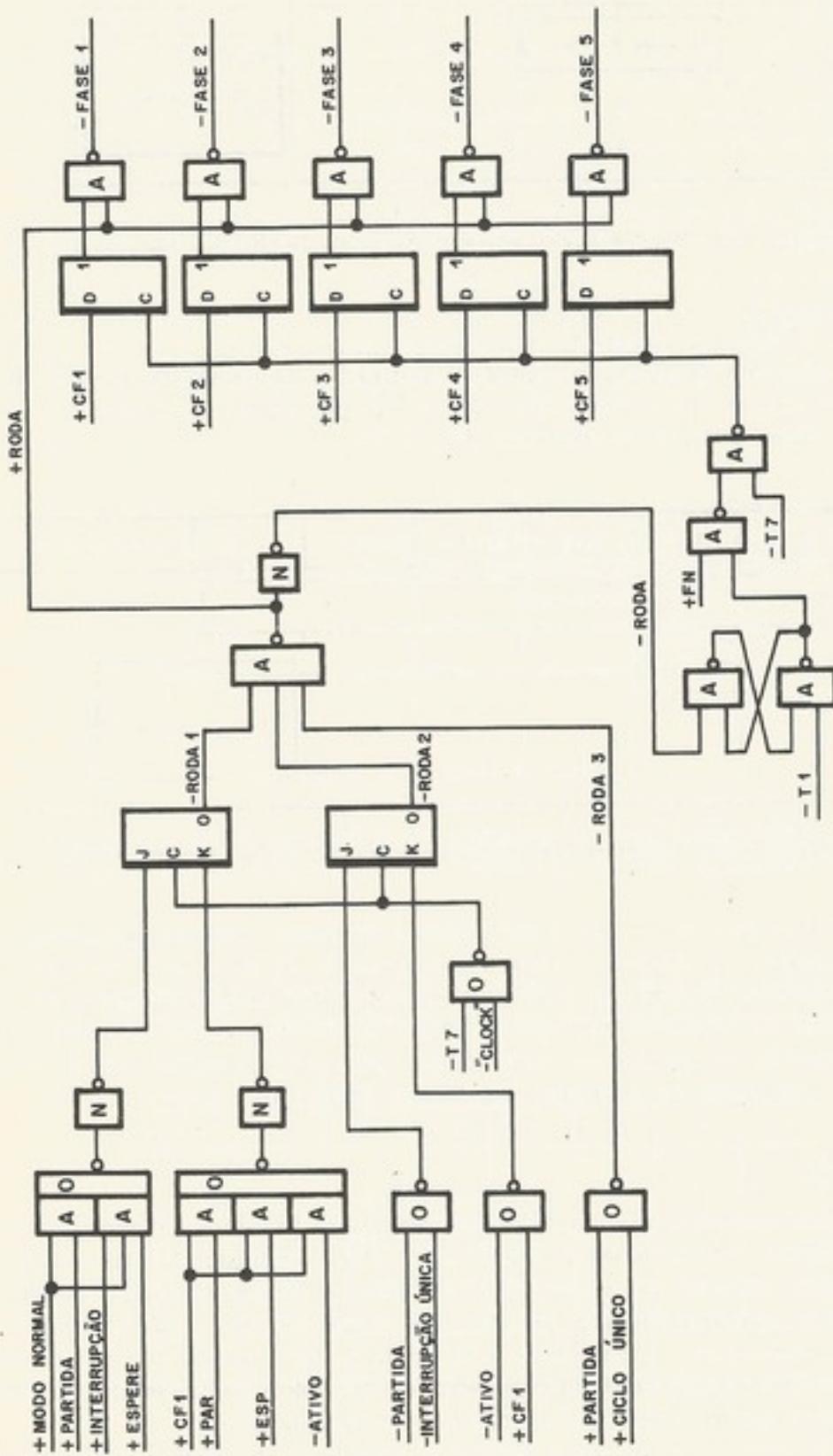


Figura 5-36. Esquema simplificado do controle do modo de operação

5.10 PRANCHAS

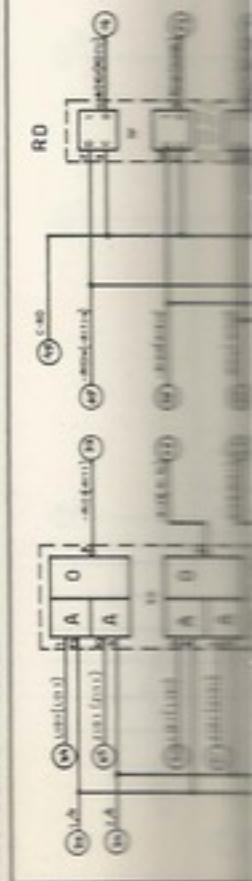
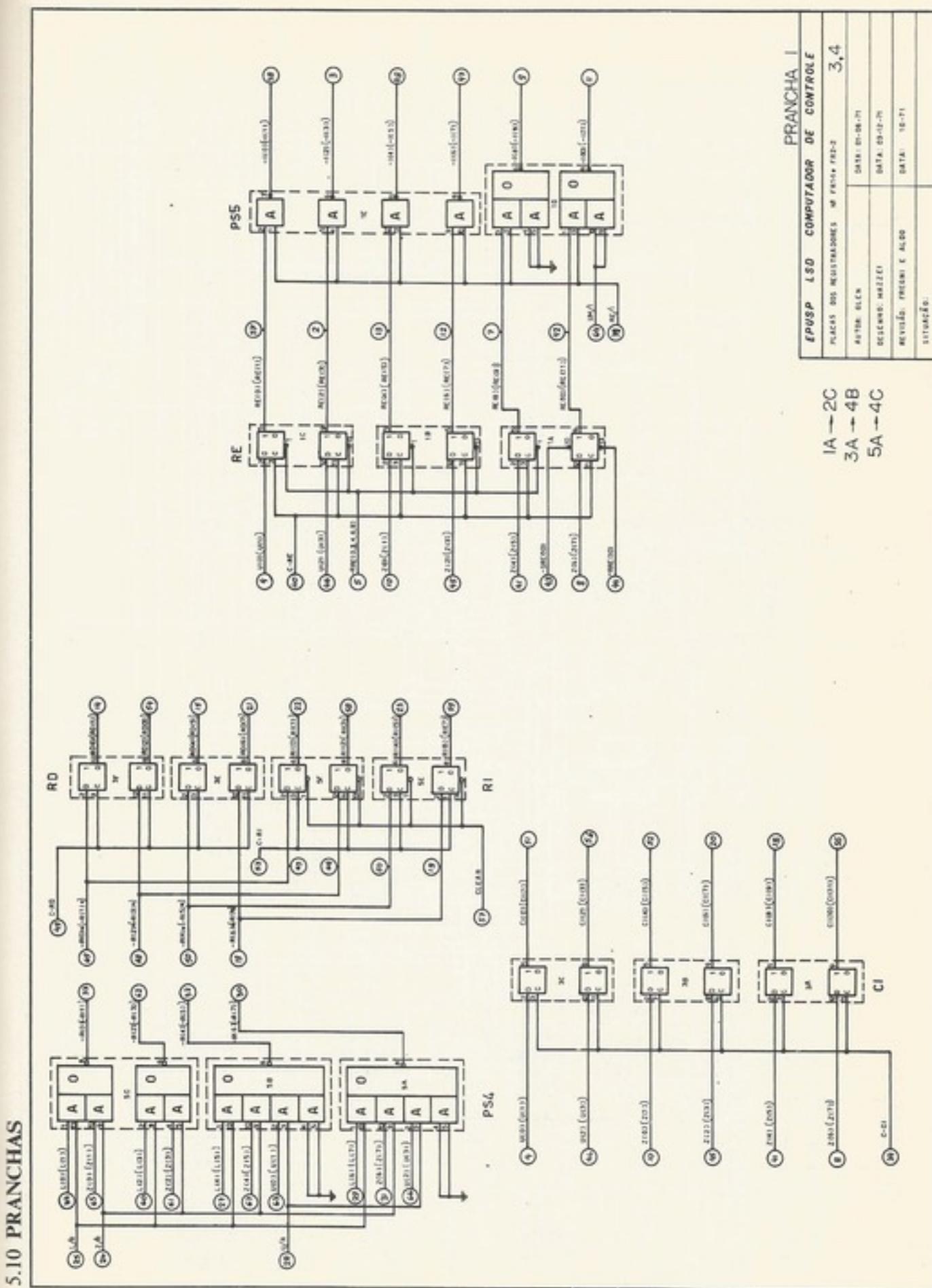
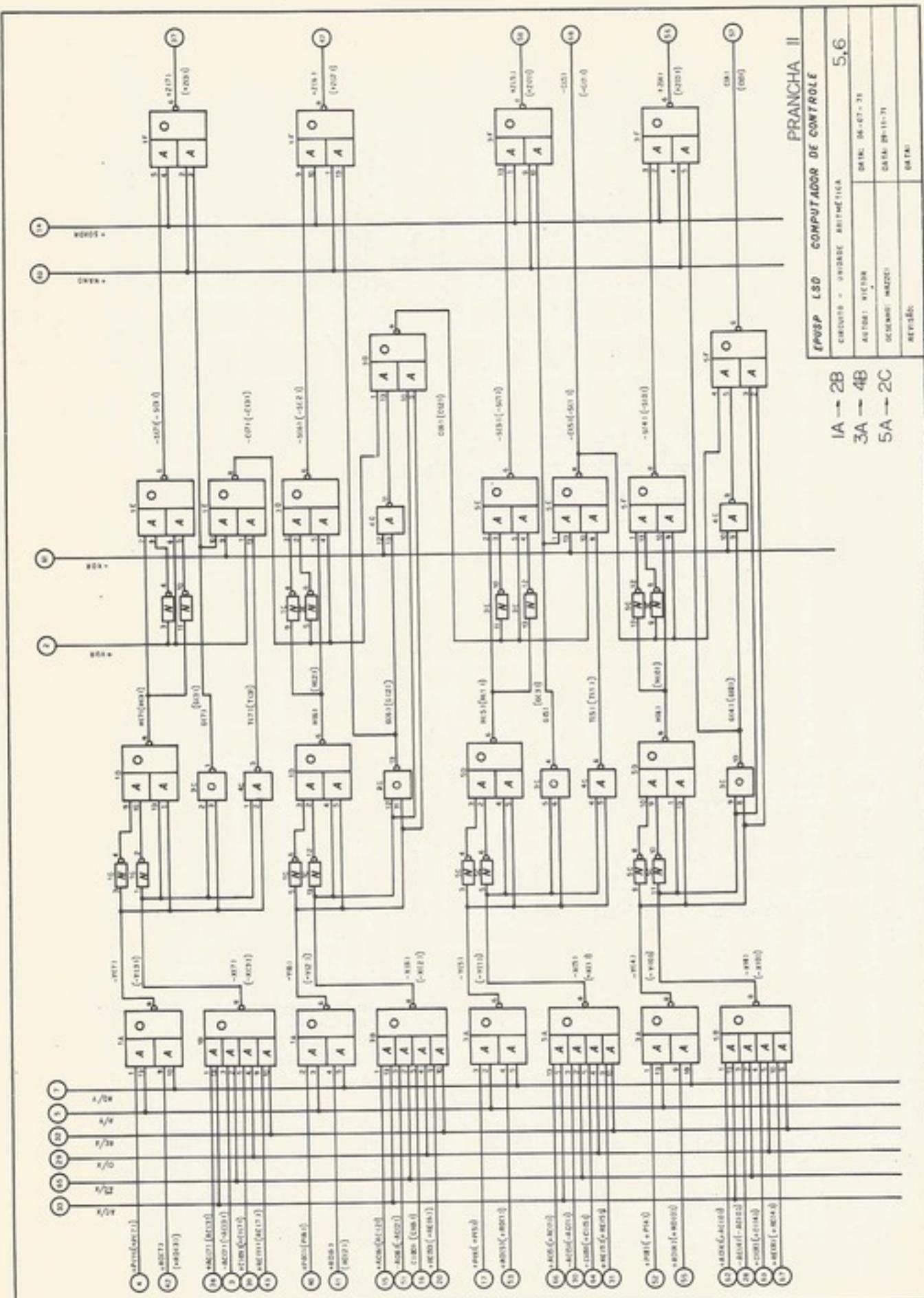


Figura 5-36. Esquema simplificado do controle do modo de operação

exemplo de um minicomputador

209



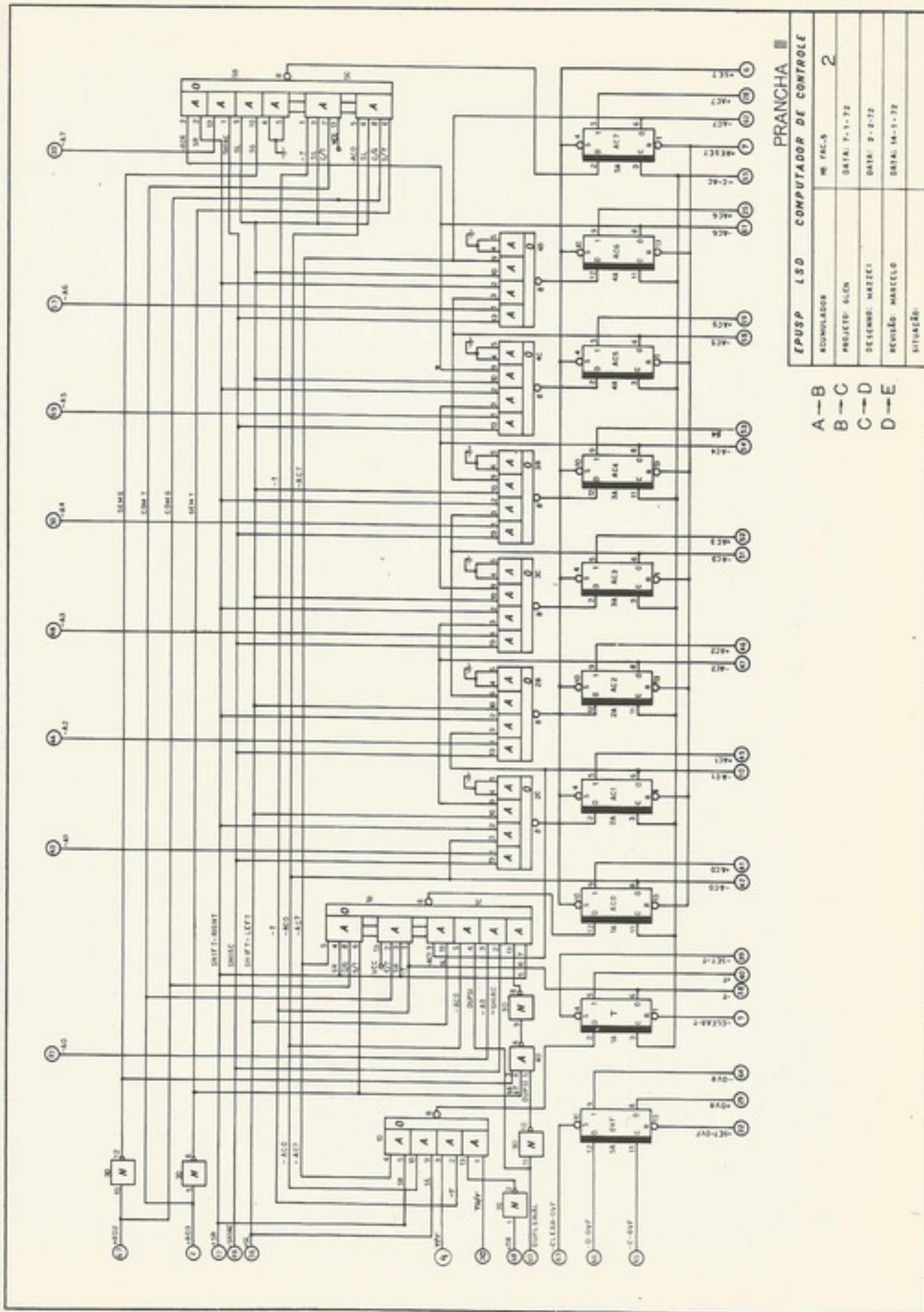


PRANCHAS

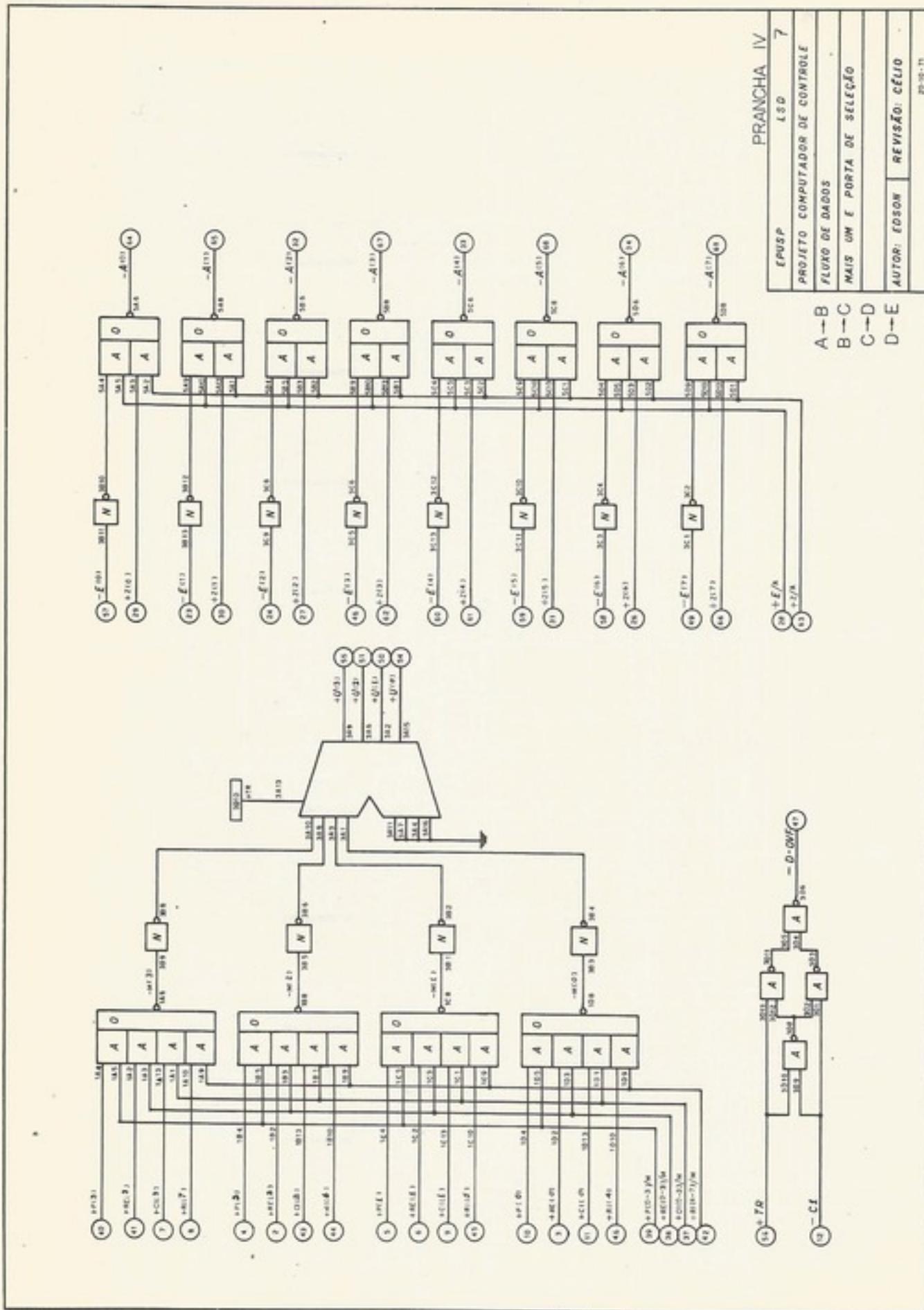
PRANCHA II	
EPUSP	LSD COMPUTADOR DE CONTROLE
1A → 2B	CIRCUITO = UNIDADE ARQUITETICA
3A → 4B	PROJETO: VICTOR
5A → 2C	DATA: 06/07/71
ACERVO:	DESENHO: MARCELO
SITUAÇÃO:	DATA: 27/11/71

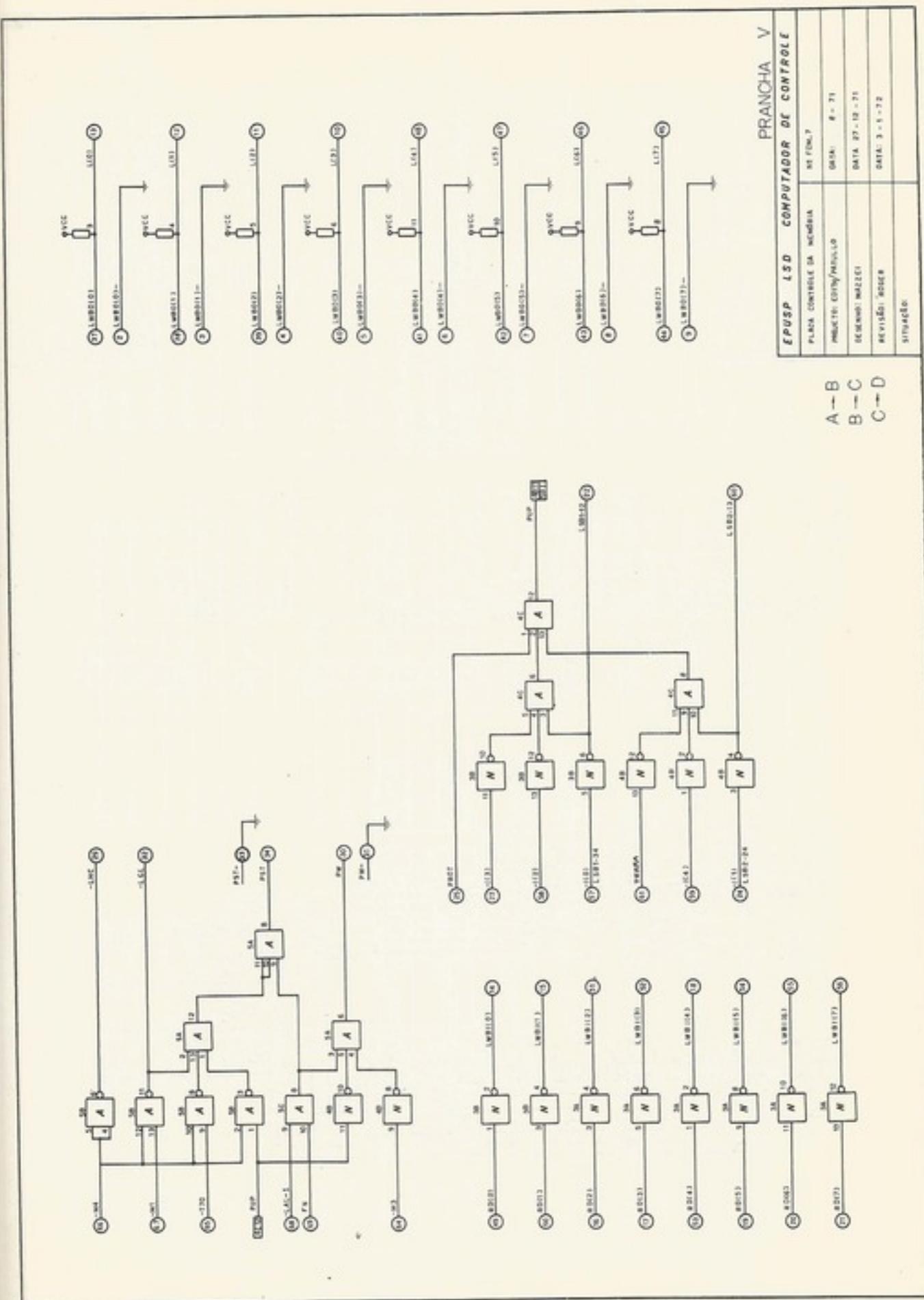
exemplo de um minicomputador

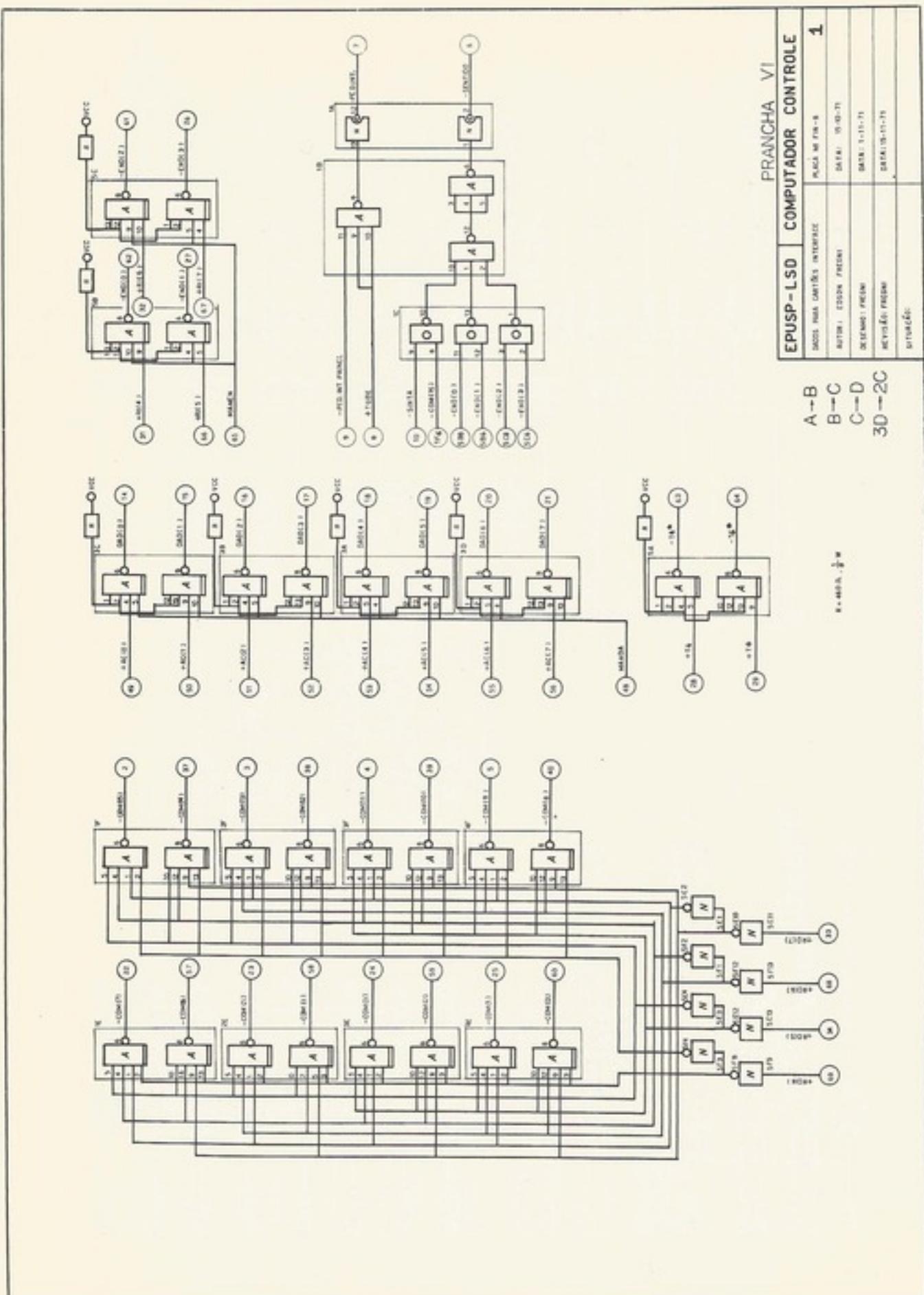
211

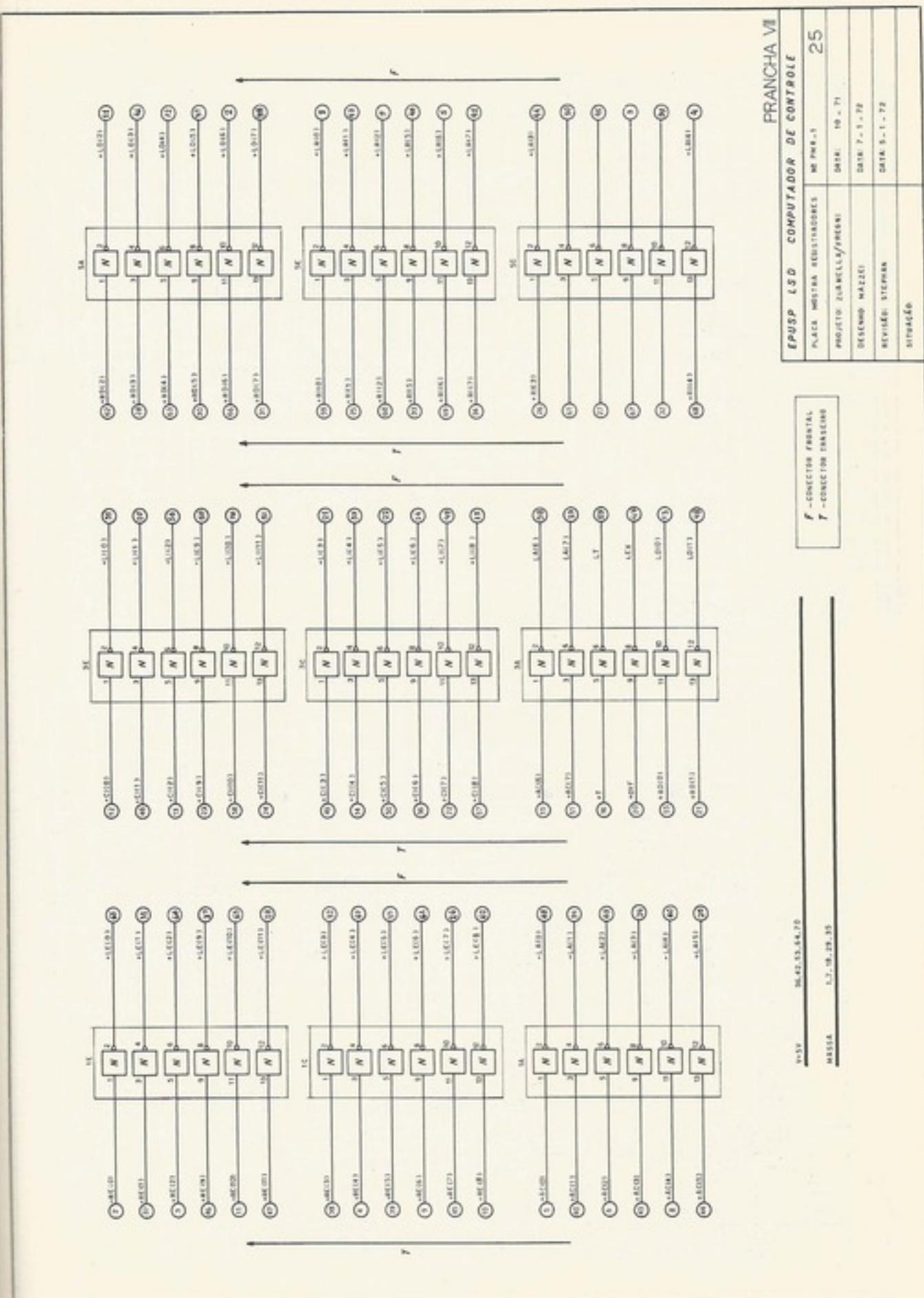
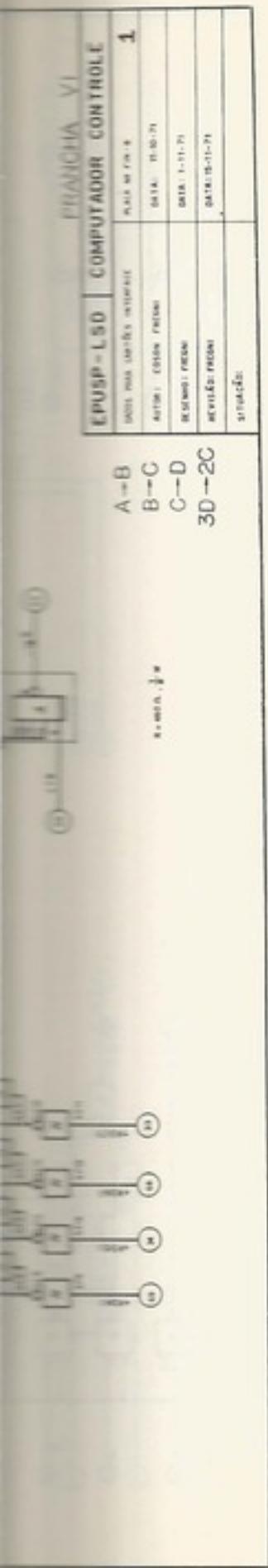


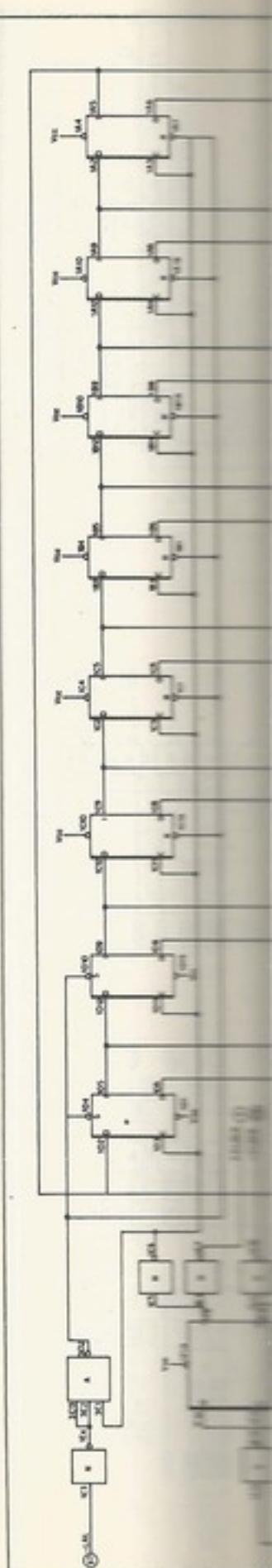
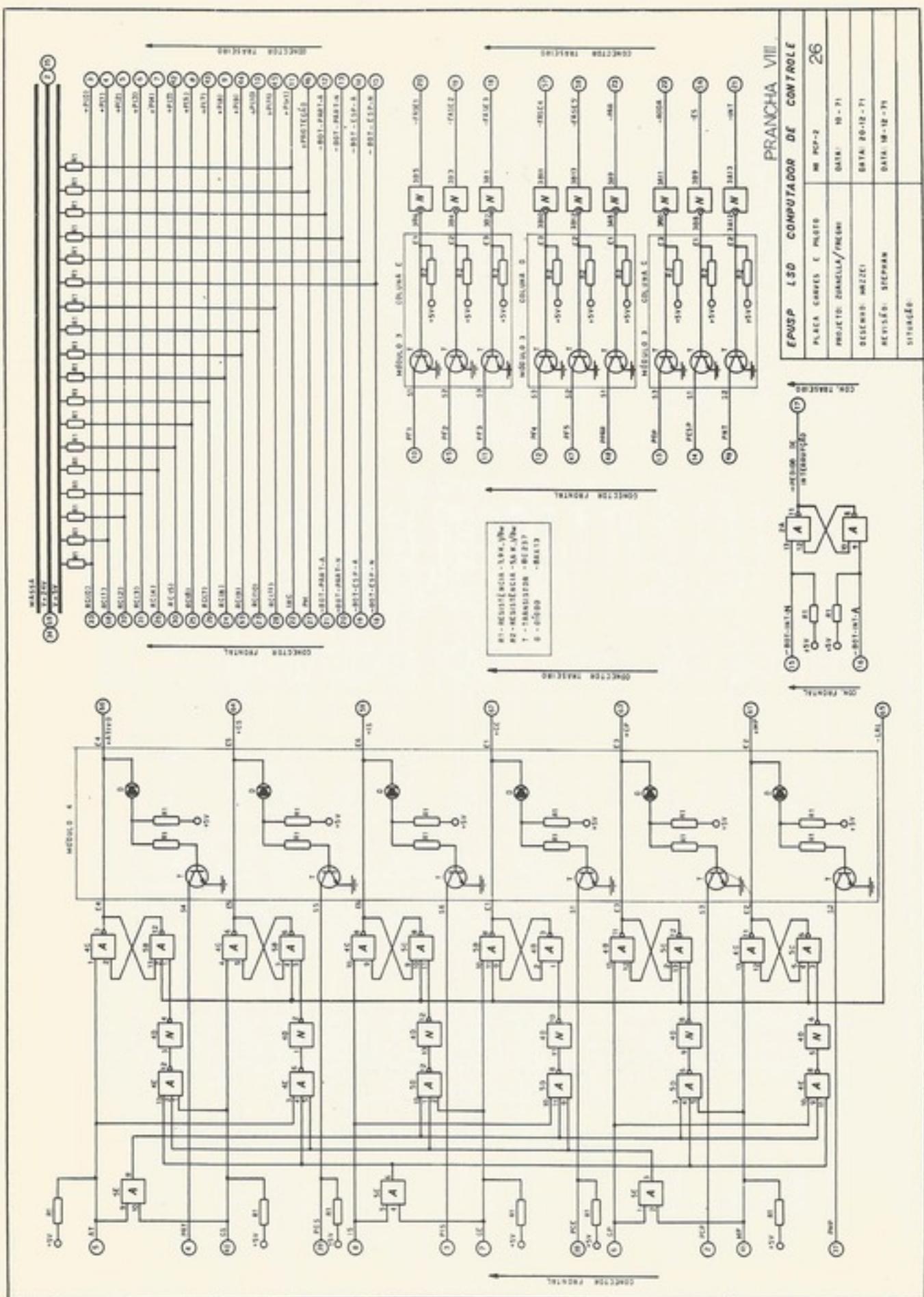
exemplo de um m-

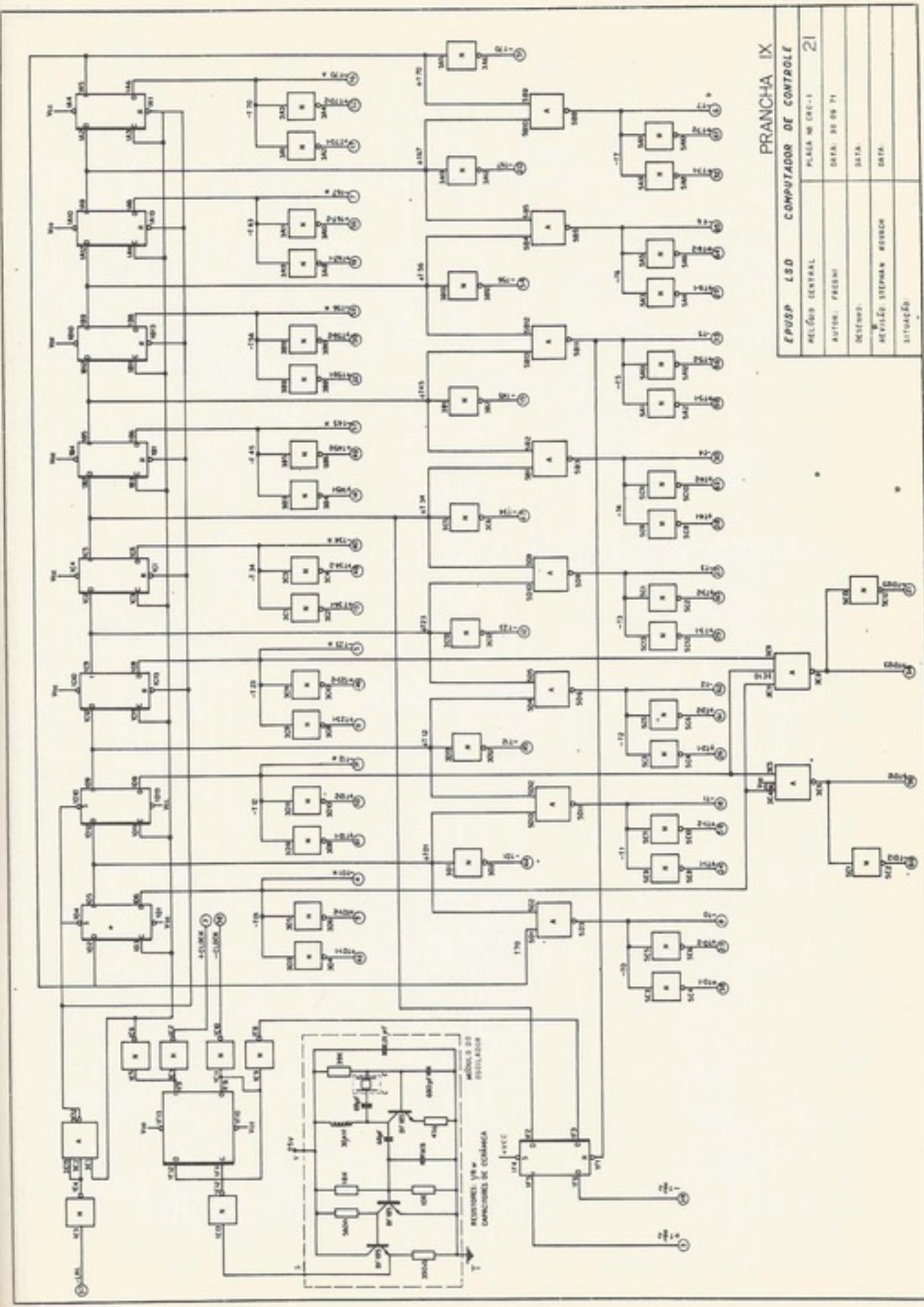
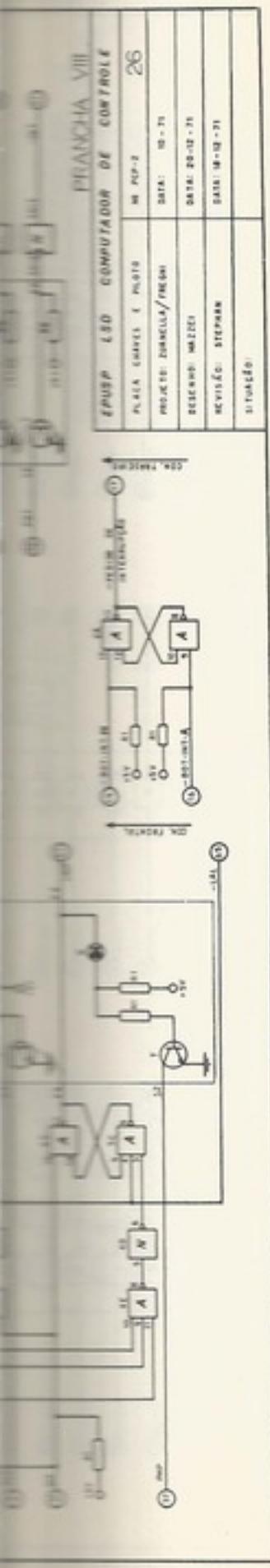


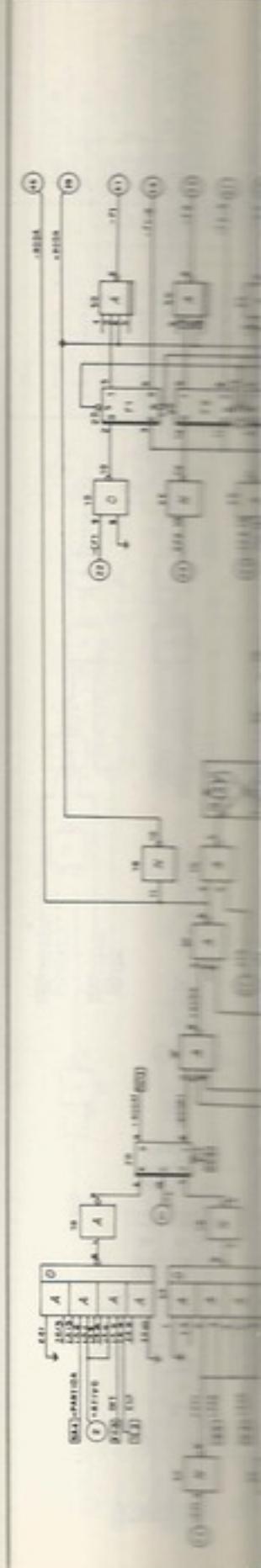
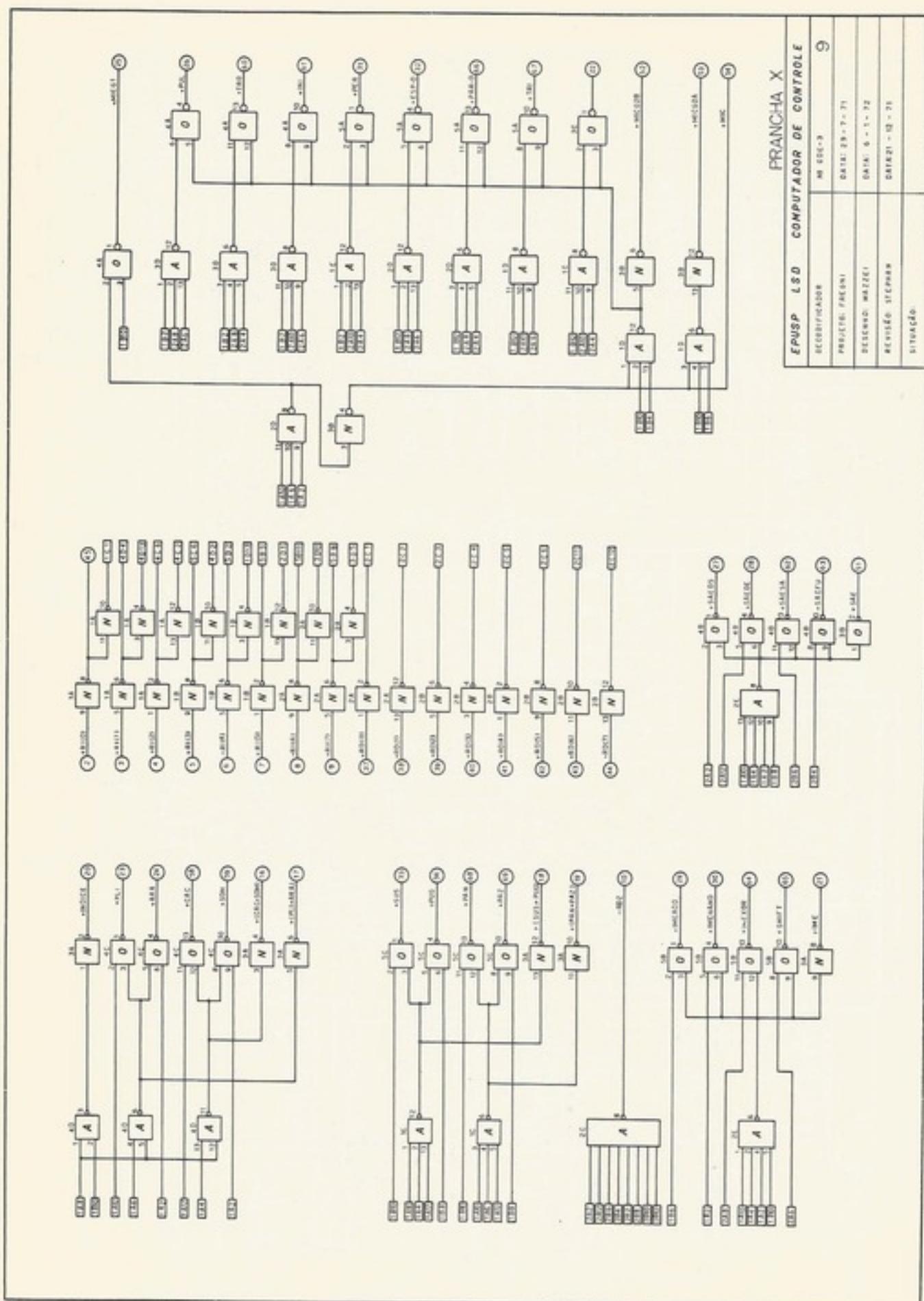


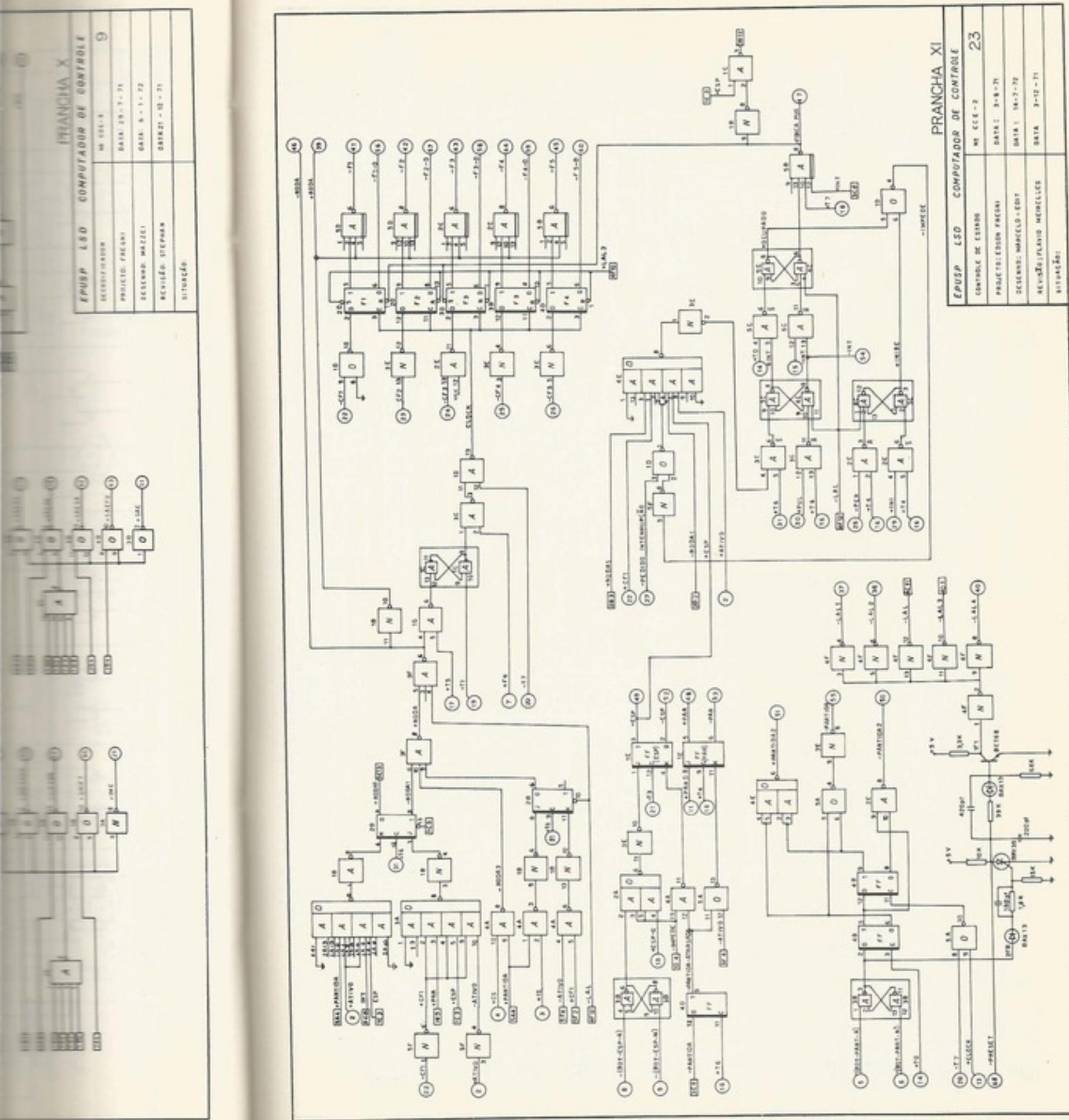


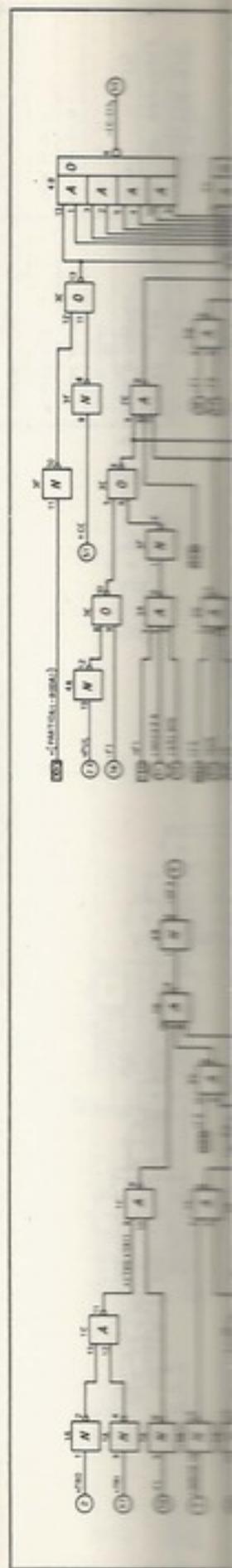
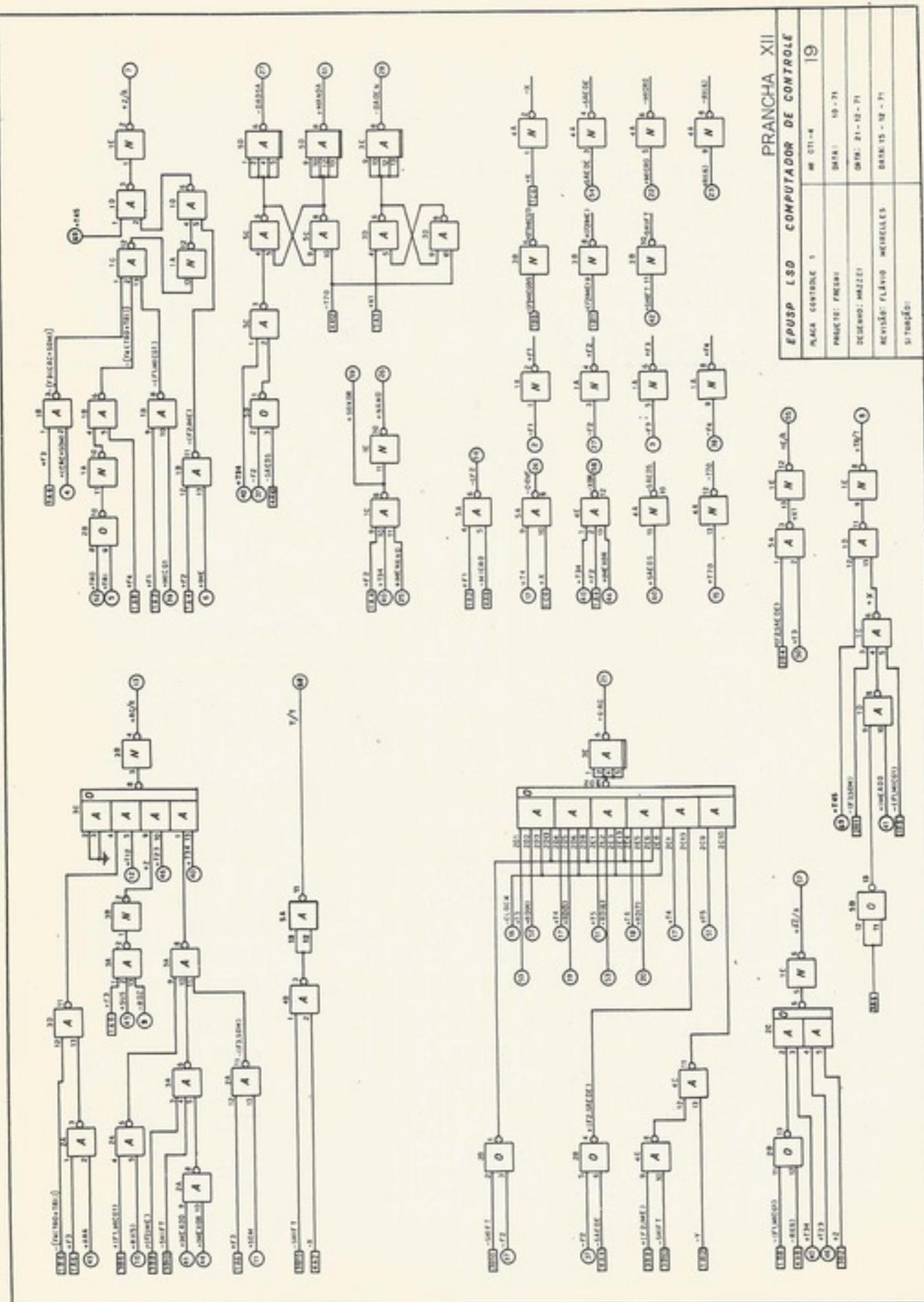


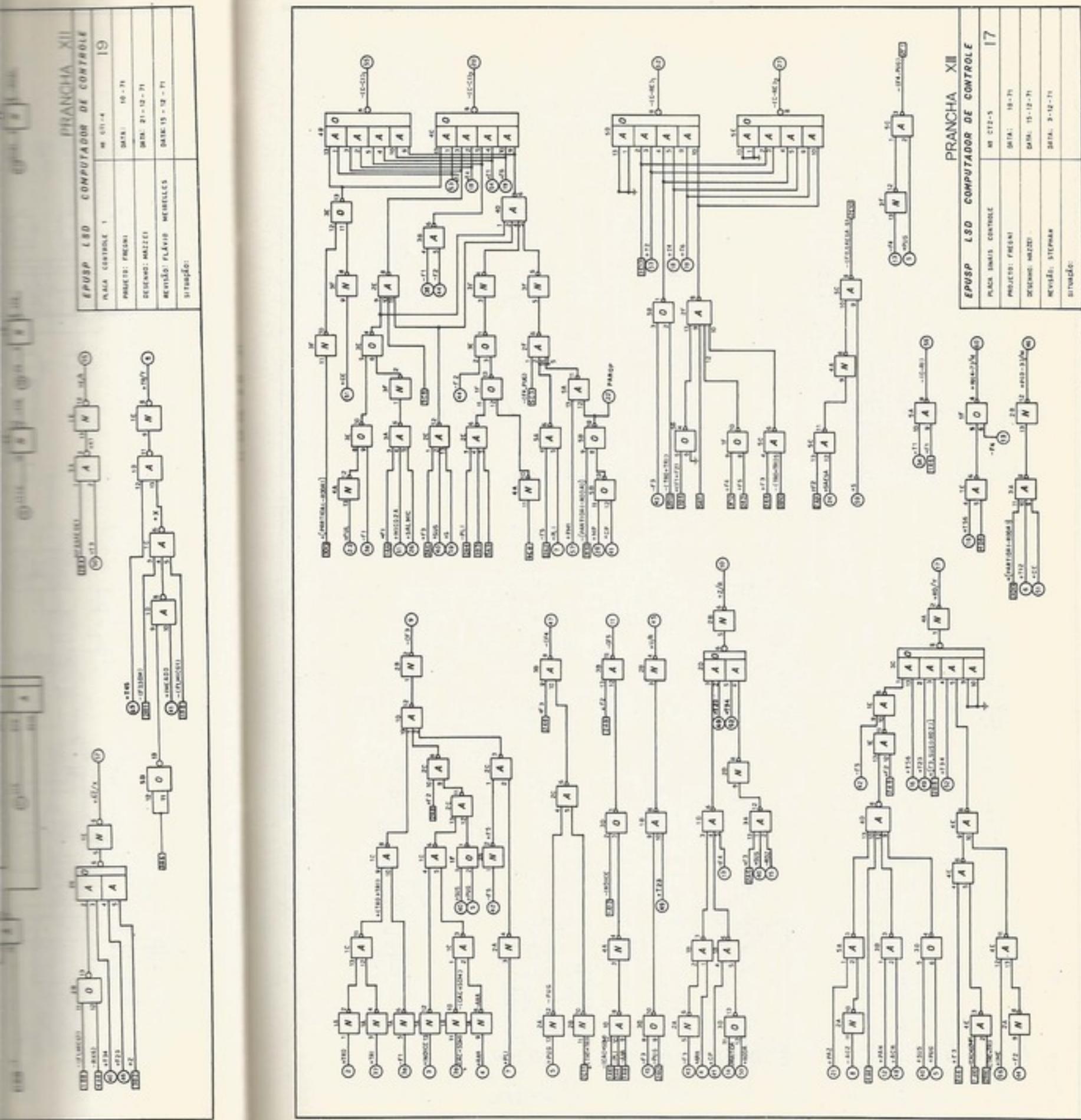


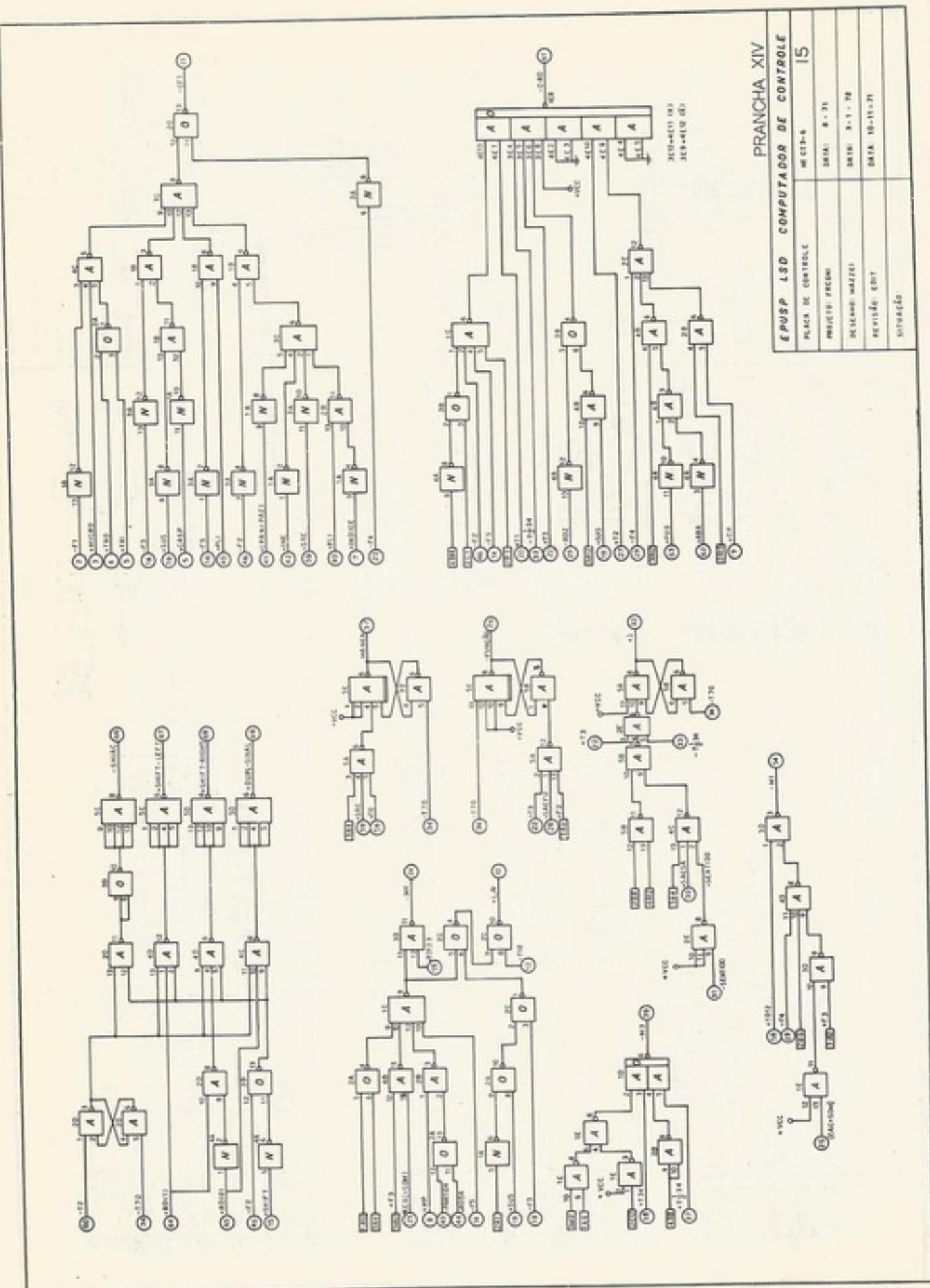




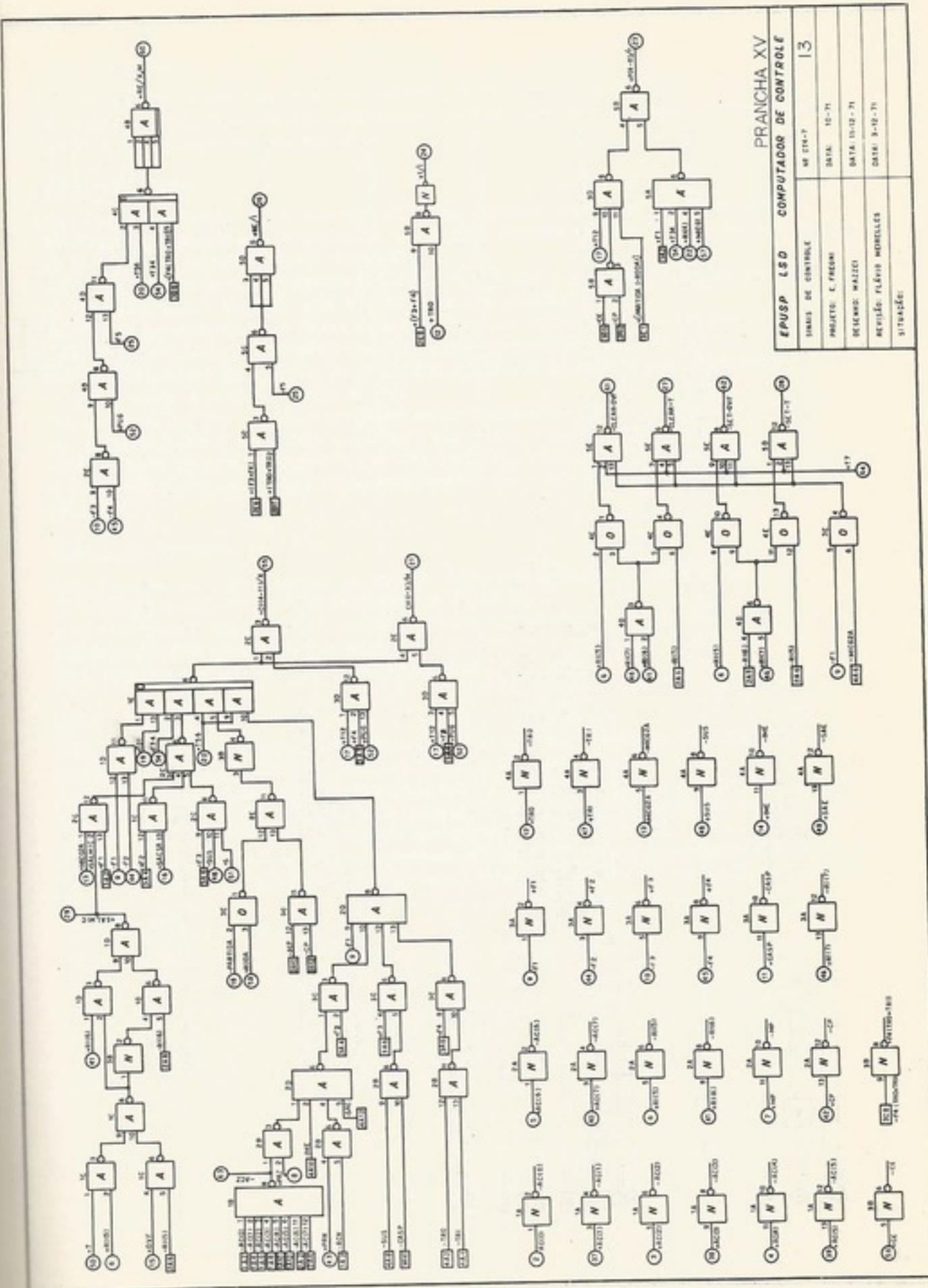
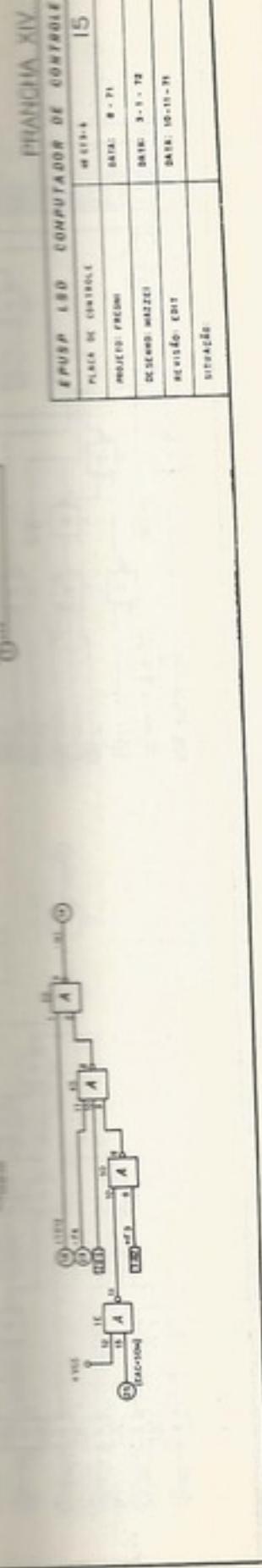


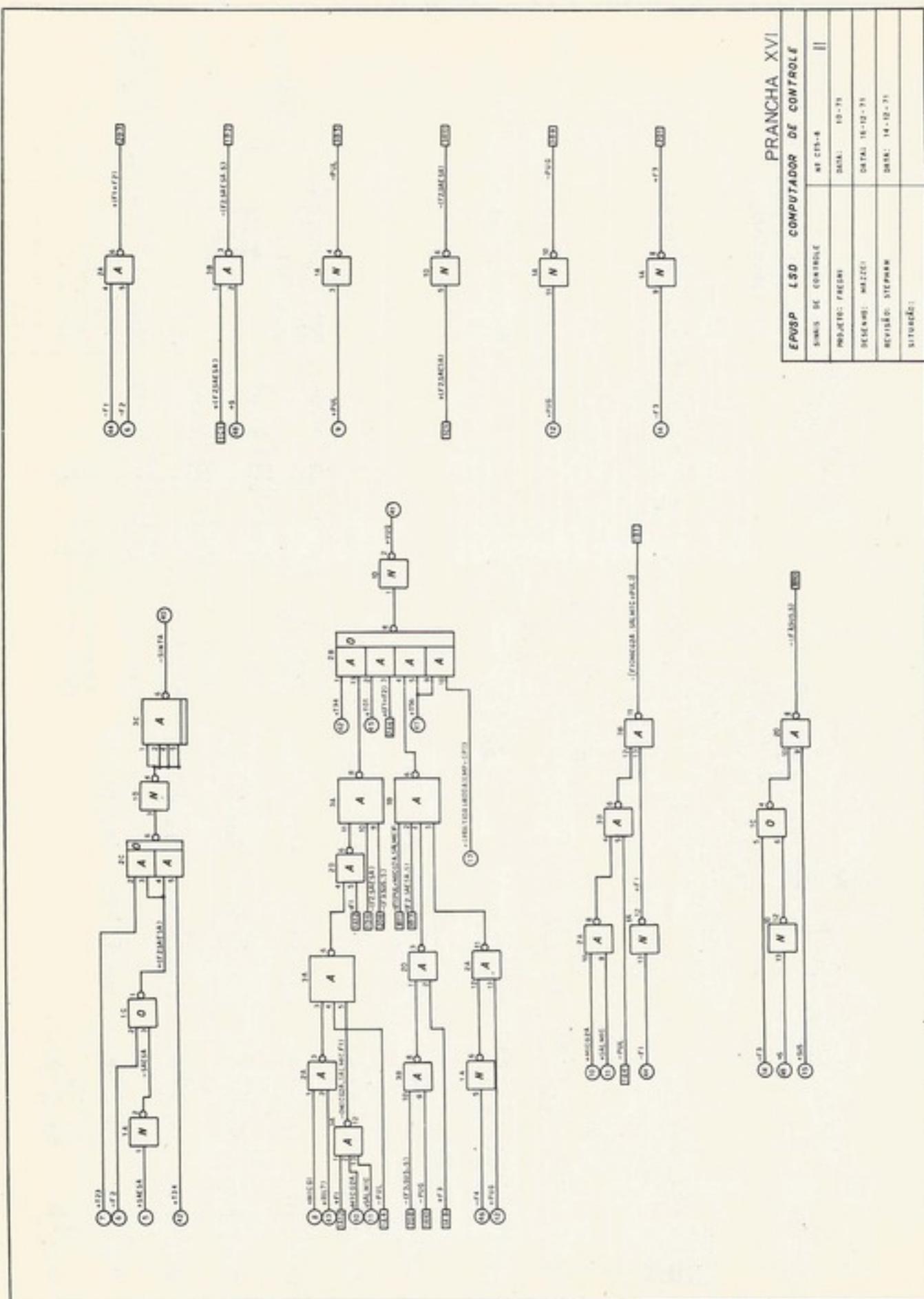






exemplo de um minicomputador





PRANCHA XVI

COMPUTADOR DE CONTROLE

AT CTR-8

DATA: 10-15

MEM: 10-15

REG: 10-15

REV: STEPHAN

DATA: 14-19-21

LITERARIO:

exemplos de com

EXERCÍCIOS

Sobre a arqu

5-1. Projete o

5-2. O que se

Quais as mo

Sobre as instr

5-3. O que a

5-4. O que a

de PLUG pa

5-5. Se o com

da instruçõe

5-6. Qual nõ

seria 1001 000

5-7. Qual a s

de que o co

5-8. Como se

5-9. E como

5-10. Faga os

A e B, send

105 e 106.

Sobre o fluxo

5-11. Faga os

5-12. Projete

5-13. Modifica

5-14. Faga os

Sobre microc

5-15. Faga os

(a) PLA, (b)

5-16. Consid

"endereç", a

5-17. Quais s

5-18. Adminis

as formas de

EXERCÍCIOS

Sobre a arquitetura

- 5-1. Projete uma arquitetura para uma memória de 1K palavras de 8 bits.
 5-2. O que se perderia se usássemos um somador de 4 bits com velocidade igual ao dobro da de 8 bits? Quais as modificações na arquitetura?

Sobre as instruções

- 5-3. O que aconteceria com a instrução *SUS* com endereço zero? Qual sua utilidade?
 5-4. O que aconteceria se o programador, por engano, incluisse no meio de um programa a instrução de *PUG* para o próprio endereço da *PUG*?
 5-5. Se o conteúdo do acumulador for 11001011 e $V = 1$, qual o conteúdo do acumulador e de V depois da instrução "giro à esquerda com V " (*GEV*) de três posições?
 5-6. Qual o conteúdo do acumulador, depois de executar a instrução *NAND* com 00001111 (a instrução seria $1101\ 0100\ 0000\ 1111$), sabendo que, inicialmente, seu conteúdo era $1101\ 1001$?
 5-7. Qual a maneira mais rápida de multiplicar por *menos um* conteúdo do acumulador? Lembre-se de que o código é o complemento de dois.
 5-8. Como se faz $T = 0$ ("limpa transbordamento"), sem alterar o acumulador?
 5-9. E como se faz $V = 1$ ("dispara vai um") sem alterar o acumulador?
 5-10. Faça um programa que guarde, nas posições $(101)_{10}$ e $(102)_{10}$, o resultado da soma dos números A e B , sendo ambos em dupla precisão armazenados em: A , nas posições 103 e 104, e B , nas posições 105 e 106.

Sobre o fluxo de dado

- 5-11. Faça um esquema completo da unidade aritmética, incluindo as saídas V_A e T .
 5-12. Projete o circuito de "mais um" visto na Fig. 5-14.
 5-13. Modifique o relógio central, mudando o *overlap ring counter* para um contador binário com um decodificador para gerar os sinais corretos.
 5-14. Faça um esquema do registrador de instrução.

Sobre microoperações e controle

- 5-15. Faça um esquema das microoperações, fase por fase, explicando cada uma delas, das instruções: (a) *PLA*, (b) *ARM*, (c) *SOMI*, (d) *SUS*, (e) *PUG*, (f) *TRE*, (g) *TRI*.
 5-16. Como seria a seqüência das microoperações de uma instrução que tivesse o formato *SUB* e "endereço", que equivaleria a subtrair do acumulador o operando endereçado no campo "endereço"?
 5-17. Quais os sinais elétricos de controle que produzem a micro-instrução $RD \leftarrow AC + RD$?
 5-18. Admita que um programador se enganasse e fizesse um *PLA* para seu próprio endereço. Desenhe as formas de onda dos seguintes sinais: (a) fase 1, (b) memória "modo 4", (c) *VUQ*, (d) *RI(4-7)/M*.

capítulo 6

ARQUITETURA DA UCP

6.1 INTRODUÇÃO

Nas primeiras gerações de computadores eletrônicos, a arquitetura da unidade central de processamento era muito ligada ao *hardware*, e esse tipo de projeto continua até hoje em sistemas menos sofisticados. Como vimos, no microcomputador do capítulo anterior, a escolha do tamanho da palavra foi baseada no *hardware*, nesse caso, a memória disponível. Na terceira geração, com o advento de sistemas de grande porte, a arquitetura tornou-se mais afastada do *hardware*. Além do mais, considerando que sistemas operacionais e programas monitores aumentam em potência e sofisticação, os projetistas da arquitetura começaram a se preocupar em facilitar a tarefa dos programadores. Assim, podemos dizer que a arquitetura de computadores é um assunto que, hoje, requer conhecimentos de projeto lógico e programação, isto é, de *software* e *hardware*.

Uma única arquitetura pode ter muitas implementações; notamos que a arquitetura do sistema IBM S/360 foi implementada em vários modelos (os modelos 25, 30, 40, 50, 65, 75, 85 e 95). No entanto é possível que um sistema digital possa “emular” outras arquiteturas, ou seja, possa se comportar como mais de uma arquitetura. Por exemplo, com uma combinação de *hardware* e microprogramação, o S/360 modelo 25 (da arquitetura S/360) tem a capacidade de executar programas do IBM 1401, um computador da segunda geração.

Neste capítulo, estudaremos, através de alguns exemplos, a representação de informação, o formato de instruções, endereçamento, desvios e demais assuntos da arquitetura.

Reforçaremos o que foi dito sobre a arquitetura do microcomputador do Cap. 5, e estudaremos o impacto da multiprogramação em sistemas da terceira geração.

6.2 FORMATOS DE INFORMAÇÃO

a. Tamanho de palavra

Um vínculo importante na estrutura de um computador é o tamanho (número de *bits*) da palavra. Em máquinas de pequeno porte, como os minicomputadores, tendo-se em vista que uma grande parte do custo do sistema é a memória principal, faz-se economia usando-se uma palavra curta com 8 ou 12 *bits*. As máquinas de grande porte geralmente dispõem de palavras de tamanho maior, por exemplo, a palavra *CDC 6600* é de 60 *bits*.

Relacionada à Tab. 6-1, encontra-se uma lista de alguns sistemas, ao lado do respectivo tamanho básico da palavra.

b. Relacionamento operando/instrução

A maioria dos computadores usa a mesma memória para armazenar as instruções e os dados (operandos). Então deve existir um relacionamento entre o tamanho das instruções, dos operandos e da palavra básica. No caso do IBM 7090, os números (operandos) e as instruções são de 36 *bits*. Essa situação é muito comum, acontecendo também no *HP 2116B*,

Tabela 6-1. Relacionamento de sistemas e tamanho de palavra

Sistema	Tamanho (bits)
IBM 1401	8
CDC-160	12
PDP-8	12
HP-2116B	16
W Prodac 580	18
IBM S/360	32
IBM 7090	36
IAS (primeira geração)	40
Burroughs 5000	48
CDC-6600	60

PDP-8 e Prodac 580. Vimos que, no microcomputador do Cap. 5, o operando básico é uma palavra de 8 bits, enquanto que uma instrução tem duas palavras.

No computador Varian 520/i, existe um registrador especial para indicar o tamanho dos operandos de 8, 16, 24 ou 32 bits.

O Burroughs 5000 usa uma palavra de 48 bits como operando básico, mas, quanto à instrução, a palavra é dividida em quatro "silabas" de 12 bits⁽¹⁾. Cada silaba é uma instrução. Alguns sistemas dispõem de tamanhos variáveis de instruções. Por exemplo, o IBM 1401 usa uma palavra (carácter de 8 bits) para o código da instrução.

Instruções do 1401 com um operando explícito são de quatro palavras (uma para o código de operação e três para endereço). Além do mais, o 1401 tem instruções de sete palavras (uma para códigos de operação, três para o endereço do primeiro operando e três para o endereço do segundo operando).

No 1401, o dígito é o dado básico, mas os operandos são de tamanhos variáveis. O operando é uma cadeia de dígitos. O endereço do operando indica o dígito menos significativo e o dígito mais significativo é indicado por um bit de sinalização (chamado *word mark*) no próprio carácter. Esse tipo de esquema que dispõe de operandos de tamanho variável é chamado *variable field length (VFL)*.

O S/360 tem três tamanhos permitidos para a instrução: 16 bits (2 bytes ou "meia-palavra") para a instrução curta em que os operandos já estão nos registradores centrais, 32 bits para a maioria das instruções, e 48 bits para as instruções *VFL*.

Quando o tamanho da palavra é pequeno, uma técnica bastante usada para aumentar a precisão dos números é juntar duas palavras vizinhas e tratá-las como um único operando, chamado "dupla palavra". Em sistemas mais simples, o tratamento da dupla palavra é efetuado através de sub-rotinas (*software*) em vez de *hardware*.

c. Formato dos operandos

O formato de números binários de n bits foi discutido no Cap. 2: ponto fixo e ponto flutuante. Ressaltamos aqui que o projetista deve levar em conta as vantagens e desvantagens das três representações de números negativos. Um outro tipo de palavra é o carácter ou dígito decimal (como no 1401) e os operandos são cadeias de caracteres. No IBM S/360, uma cadeia de caracteres pode ser em código EBCDIC (ou ASCII) de 8 bits ou pode ser em decimal "empacotado" (*packed decimal*) em que dois dígitos de 4 bits cada compartilham de um byte de 8 bits (veja a Fig. 6-1). O sinal é o "meio-byte" menos significativo. Indicado na Fig. 6-1 está o peso da posição. Existem instruções decimais que operam em decimal com operandos nesse formato. O tamanho do operando é indicado na própria instrução que opera na cadeia de caracteres.

A vantagem de *VFL* é que todas as posições na memória são aproveitadas. Em sistemas simples de *VFL* como o IBM 1401 e IBM 1620, o fluxo de dados é econômico, tendo largura

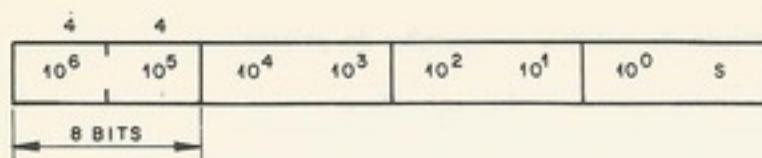


Figura 6-1. Decimal "empacotado"; dois dígitos por byte

de um carácter. Nas primeiras gerações, muitos computadores projetados para fins de processamento comerciais (em vez de cálculos científicos) usavam *VFL* com caracteres ou dígitos decimais. Assim, não é necessária a conversão digital a binário e vice-versa. Além do mais, no caso do *S/360*, o compilador *COBOL* (linguagem de alto nível para fins comerciais) usa os formatos e instruções decimais.

Um outro tipo de operando é usado no Burroughs 5500. Esse operando é chamado de autodescrito porque os bits da própria palavra indicam o tipo de dado. Nesta altura, o leitor deve ser avisado que a arquitetura do *B-5000* e seus derivados, como os *B-5500* e *B-6700*, não são da "escola tradicional", mas, no entanto, são bastante originais e merecem estudo. Portanto pretendemos utilizar o *B-5500* bem como as arquiteturas mais "tradicionais" como exemplos, mas aqui não pretendemos explicar os formatos de dados do *B-5500* em grande detalhe. Queremos apenas abordar o assunto, deixando o leitor com o conhecimento da existência dessa filosofia importante. Os formatos das palavras comuns do *B-5500* da classe autodescrita aparecem na Fig. 6-2. (Os formatos dos control words não são mostrados.) Os operandos já estão em ponto flutuante. O bit 0 no estado "0" indica que a palavra é um operando em vez de uma palavra de controle, ou descrição de dados ou programa.

O descritor de dados (*data descriptor*) é usado no esquema de segmentação, indicando um bloco de palavras, o tamanho e o endereço inicial. O bit 2 (o bit de presença, *P*, ou *presence bit*) indica se o segmento está na memória ou não. Se o segmento não está na memória (o bit *P* está em "0"), o sistema busca o segmento, colocando-o na memória, acertando o endereço do descritor e, depois, "ligando" o bit *P* no estado "1".

A palavra descritor de programa [Fig. 6-2(c)] é usada para sub-rotinas ou subprogramas. Para os que conhecem *ALGOL*, isso é um *procedure* e, para os que conhecem *COBOL*, um *paragraph*.

O bit 4 indica o modo da sub-rotina. O modo "carácter" não usa palavras autodescritas, mas simplesmente trata os operandos de 48 bits com oito caracteres de 6 bits cada.

O bit 5, "ligado" (no estado "1"), indica se a "pilha" (veja a Sec. 6.3) está sendo usada para passar parâmetros à sub-rotina. O campo de bits 18 a 32 indica a localização de uma palavra de controle (não discutido aqui), usada para o retorno ao programa que chamou a sub-rotina. O campo de bits 33 a 47 indica a localização da primeira instrução da sub-rotina. Para demais informações relativas às palavras autodescritas do *B-5500*, recomendamos ao leitor, o manual da Burroughs⁽¹⁾.

O propósito dessa estrutura de informação é facilitar a segmentação (veja o Cap. 4) e o tratamento de matrizes. Um esquema semelhante ao *B-5000* está sendo usado no computador da Rice University (Houston, Texas).

Um conjunto de localizações consecutivas na memória primária é chamado de *bloco* e, para cada bloco, existe uma palavra-código (*codeword*). Encontra-se na Fig. 6-3 o formato dessa palavra-código⁽⁴⁾.

A palavra-código é útil na descrição de matrizes e também vale para um programa de instruções consecutivas.

d. O formato das instruções

No computador *IAS* da primeira geração, a instrução era organizada em dois campos: código de instrução de 8 bits e endereço do operando de 12 bits. Esse tipo de instrução chama-se "endereço simples" (*single address*).

Quando um operando já está no acumulador e o resultado da operação está sendo colocado no acumulador, é necessário apenas um endereço na instrução para indicar um

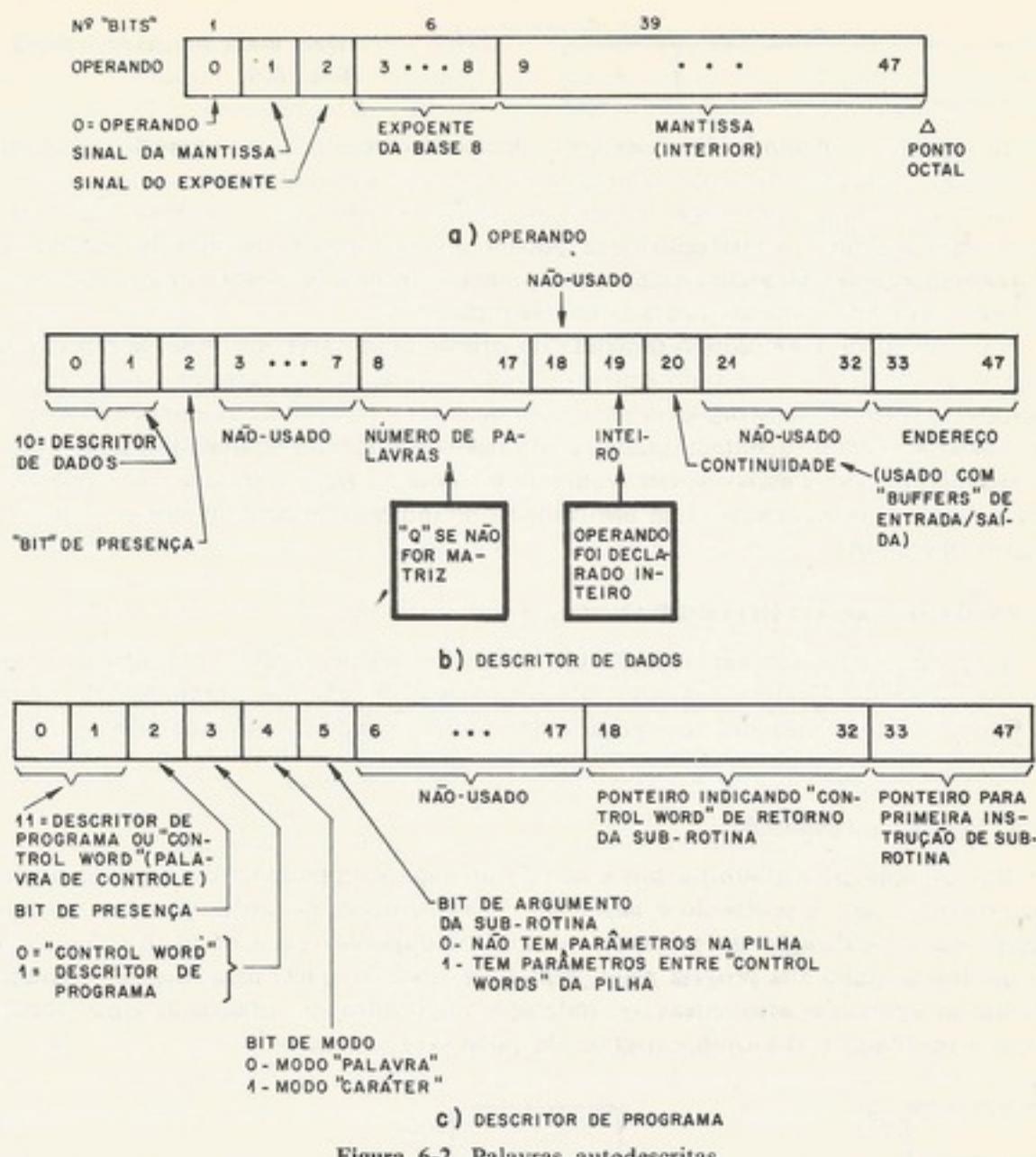
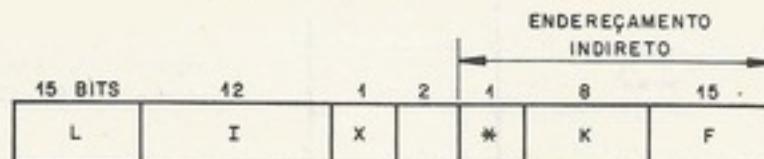


Figura 6-2. Palavras autodescritas



[Observação. *L*: tamanho do bloco; *I*: um deslocamento relativo à primeira palavra do bloco; *X*: "ligado" se o bloco referido consiste de demais palavras-códigos; *: "ligado" se o endereçamento indireto está sendo usado; *K*: se for indexado, o campo *K* indica o registrador de índice; *F*: campo *F* indica uma palavra dentro do bloco. (A primeira palavra do bloco é sempre *F-I*)]

Figura 6-3. Palavra-código do computador da Rice University

segundo operando. Na instrução de armazenar acumulador na memória, o endereço especifica a posição de memória onde seria gravado o acumulador.

No computador IBM 1620, por exemplo, sem acumulador, o tipo de aritmética feita é o "somador à memória" (*add-to-memory*). O sistema, semelhante ao IBM 1401 tem instruções com o formato visto na Fig. 6-4.

O código de operação é campo "0", formado por dois dígitos. Os campos *P* e *Q* são os endereços com cinco dígitos decimais cada. Uma operação típica é a soma do operando do



Figura 6-4. Formato da instrução do IBM 1620

endereço Q com o operando do endereço P , deixando o resultado no lugar do operando P . Esse tipo de instrução é de dois endereços, ou endereço duplo (*two address instruction*).

Além dos campos principais (campo do código de operação, e campos para fins de endereçamento direto), a instrução pode possuir outros campos que indicam modificações e demais informações. Quando estudarmos os endereçamentos indiretos e indexados, veremos que ambos são diferenciados por bits na instrução.

Em sistemas de VFL, com o tamanho do operando variável, um meio de especificar-se o número de dígitos no operando é através de um campo de comprimento (*length field*) na instrução. Também, nas instruções de desvio condicional, é possível usar-se um bit na instrução para indicar se o desvio acontece quando a condição é verdadeira ou quando é falsa. Quando uma arquitetura possui mais que um acumulador (como no HP 2116B, que tem registradores A e B) a instrução geralmente tem um campo que indica qual acumulador é usado como primeiro operando.

6.3 ESPECIFICAÇÃO DO OPERANDO

O operando, quando não especificado pelas técnicas de *imediato* ou *implicado*, é normalmente especificado através do seu endereço. A especificação do operando pode ser pelos endereçamentos: (a) imediato ou implicado; (b) direto; (c) indireto; (d) indexado; (e) relativo; (f) abreviado.

a. Imediato ou implicado

Na instrução *SUS* ("subtrai um e salta") do microcomputador do capítulo anterior, o operando "1" que é subtraído é *implicado*. Nas instruções de endereçamento simples, o endereço ou a localização do primeiro operando é implicado: o acumulador. No B-5500, cada programa tem a sua própria *pilha* (*push-down store* ou *stack*) para guardar operandos.

Para as operações aritméticas, os endereços implicados são tirados de cima (topo) da pilha e o resultado é colocado em cima da pilha (veja a Fig. 6-5).

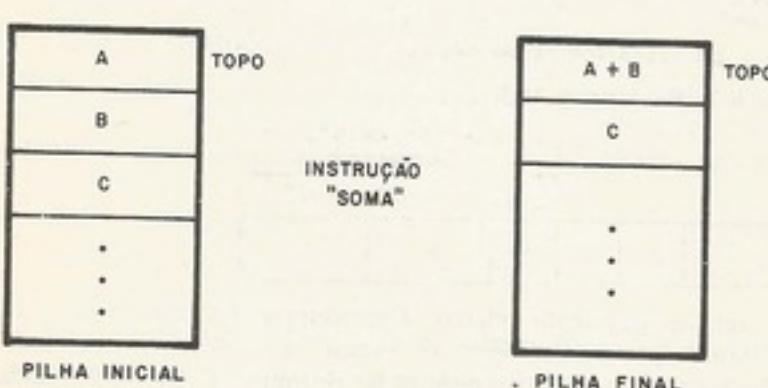


Figura 6-5. Localização dos operandos no B5500

Nesse sistema, a silaba (instrução) para somar tem apenas dois campos, um campo indicando que a classe da silaba é uma operação e outro campo indicando qual operação. A instrução é sem endereço (*zero-address*). O B-5500 também tem silabas da classe "chamar operando" (*operand call*), que tem um campo de endereço. Instruções da classe "chamar operando" são usadas para retirar um operando da memória primária e colocá-lo em cima da pilha ou para tirar o operando de cima da pilha e guardá-lo na memória primária.

Uma outra maneira de especificar o operando é através da própria instrução. Esse tipo de endereçamento é chamado de *imediato*. O operando imediato reside no outrora campo de endereço. O endereçamento imediato é usado no microcomputador do Cap. 5.

Uma variação de endereçamento imediato é aquele que usa como operando a palavra seguinte de instrução. Assim, se a instrução N usa a técnica de endereçamento imediato com essa variação, a seqüência de palavras no programa aparece como "instrução ($N - 1$), instrução N , operando do N , instrução ($N + 1$), instrução ($N + 2$) etc."

b. Direto

Endereçamento *direto* é aquele onde o campo de endereço da instrução corresponde ao endereço do operando. Esse tipo de endereçamento era muito comum na primeira geração. Os esquemas de endereçamento indireto, indexado ou relativo utilizam endereçamento direto como uma base para a formação do *endereço efetivo* do operando.

c. Indireto

No esquema de endereçamento *indireto*, o conteúdo do campo de endereço da instrução é o endereço do endereço do operando. Em outros termos, a palavra retirada da memória através de endereçamento direto não é o operando mas sim o endereço do operando. Acontece que, em alguns sistemas, esse "operando" é um endereço indireto do segundo nível, e que o endereçamento indireto assim pode reiterar até que seja terminado por um bit no último endereço indireto. No *HP 2116B*, a palavra é formada por 16 bits e, para endereçamento indireto, o bit mais significativo é usado como um sinal para indicar a continuação do modo indireto. Encontrando um endereço cujo bit mais significativo está no estado "0", os 15 bits menos significativos da palavra tornam-se o endereço efetivo, indicando o operando. Caso contrário, esses 15 bits indicam um outro endereço, aumentando o nível de endereçamento indireto. Um problema de endereçamento indireto de multinível é que, se, por um erro de programação, ocorrer uma malha de endereçamento fechada, a máquina nunca sairá do modo indireto. Uma outra desvantagem de endereçamento indireto de multinível é que um bit de endereço é gasto. Por exemplo, a capacidade máxima da memória do *HP 2116B*, sem se preocupar com instruções especiais, é de 32K palavras, porque só 15 bits são disponíveis para endereçamento indireto. Também, com esse tipo de endereçamento, o programador deve tomar cuidado para não endereçar indiretamente números negativos (cujos bits mais significativos são "1"). Com endereçamento indireto de um nível só, o *HP 2116B* podia acomodar uma memória de capacidade 64K palavras. Richards⁽⁵⁾ duvida, uma vez que exista endereçamento indireto, que endereçamento indireto de multinível compense os gastos. Uma descrição gráfica de endereçamento indireto aparece na Fig. 6.6.

d. Indexado

O endereçamento indexado foi desenvolvido num computador da University of Manchester, Inglaterra, onde foi chamado de B-Box. O propósito do B-Box foi facilitar a construção de malhas (*looping*) pelo programador. Essa foi uma das primeiras vezes em que o hardware foi desenvolvido para facilitar a programação. Quando o endereçamento é feito por indexação, o endereço efetivo é a soma do campo de endereço da instrução com o registrador de índice.

Em sistemas com um único registrador de índice, basta indicarmos se o endereçamento é indexado ou não. Como alguns sistemas possuem três ou mais registradores de índice, então é necessário um campo na instrução para indicar qual o indexador a ser usado. Sistemas com indexação necessitam de providências para carregar, incrementar (ou decrementar) e armazenar os registradores de índice. Também é importante a existência de instruções para testar o registrador de índice.

Em muitos sistemas, o registrador de índice é implementado em hardware como registrador central. Em alguns sistemas, para economizar, como visto no microcomputador do Cap. 5, o registrador de índice é uma palavra na memória principal. Essa técnica também tem a vantagem de que as mesmas instruções para carregar e armazenar, referentes à localizações na memória principal, também servem para o registrador de índice. É também

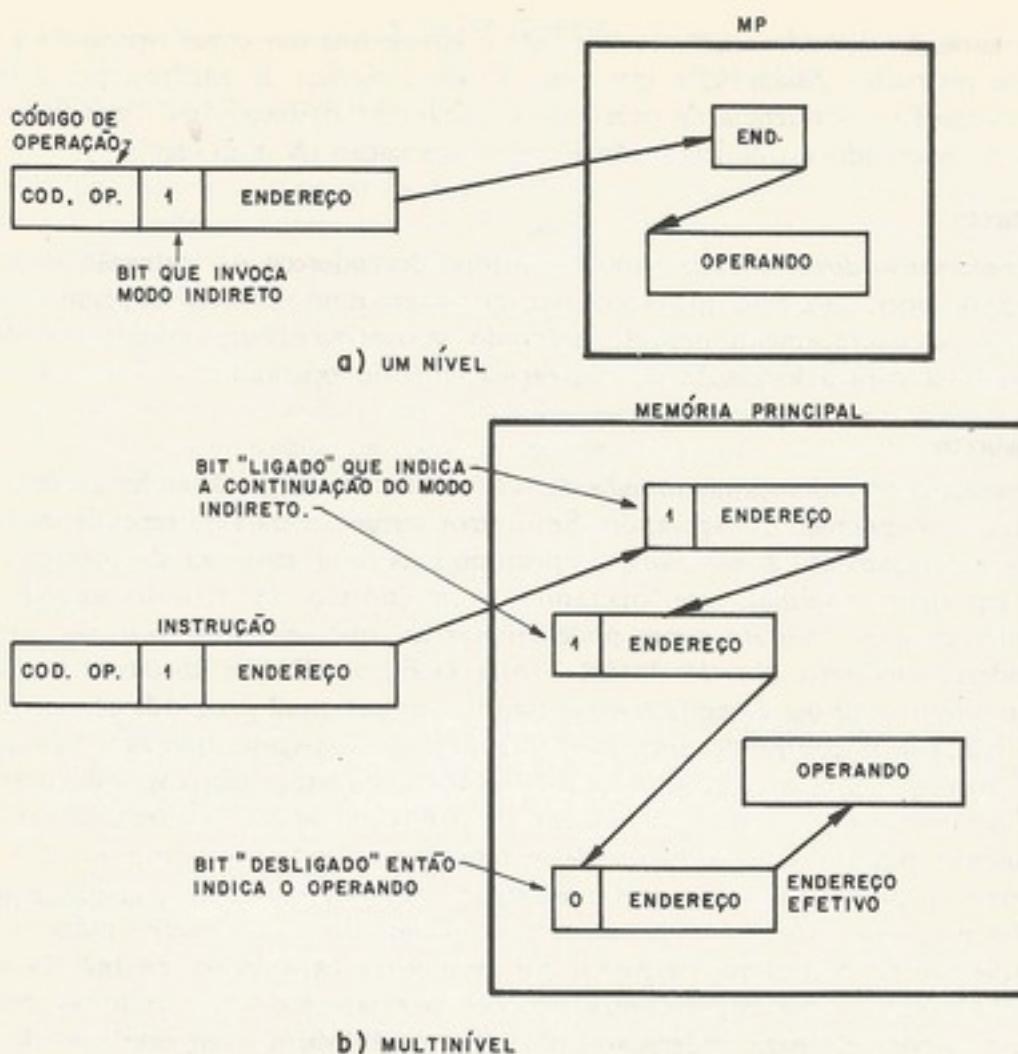


Figura 6-6. Endereçamento indireto

comum decrementar, testar e desviar no conteúdo do registrador de índice. Assim, quando o índice ficar com conteúdo nulo, haverá meios de sair da malha. A instrução *SUS* do microcomputador pode ser utilizada dessa maneira. A função de indexação é mostrada na Fig. 6-7.

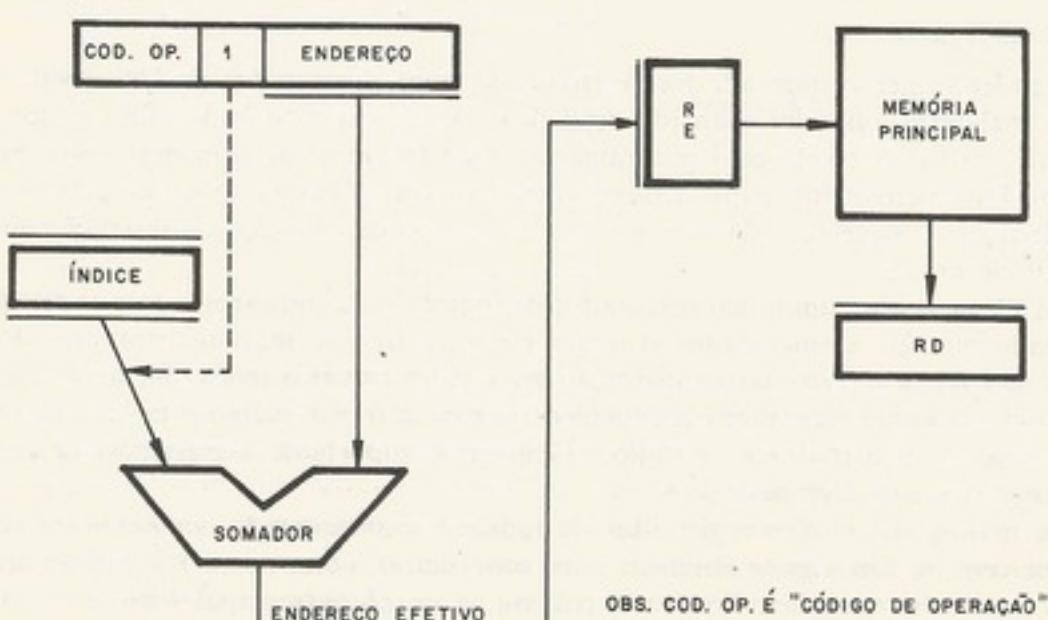


Figura 6-7. Endereçamento indexado

e. Relativo

Endereçamento *relativo* é semelhante ao endereçamento indexado e, quando usado num sistema, permite o encurtamento do campo de endereço da instrução, economizando, assim, *bits*. Com endereçamento relativo, o endereço efetivo é a soma do conteúdo do *registrator de base* e o campo de endereço da instrução:

$$\text{endereço efetivo} = (\text{campo de endereço}) + (\text{registrator de base}).$$

Muitas vezes o endereço efetivo é relativo ao contador da instrução. Assim, o programa pode desviar facilmente para instruções próximas e pode ter acesso às variáveis e constantes locais. Uma outra vantagem de endereçamento relativo é a relocação de programas, bastando apenas mudar o registrador de base e carregar o programa no outro lugar da memória principal. No caso do *CDC 160*, o endereçamento relativo foi usado para economizar *bits*. Com apenas 12 *bits* na instrução, o campo de endereço tinha 7 *bits* permitindo endereçamento de até ± 63 posições relativo ao contador da instrução.

f. Abreviado

Um outro tipo de endereçamento é chamado *abreviado*, ou *truncado*. Nesse tipo, o campo de endereço da instrução fornece os *bits* menos significativo do endereço e os *bits* mais significativos são provenientes de algum registrador. Esse último registrador (dos *bits* mais significativos) pode ser o contador da instrução, mas também pode ser um registrador especialmente projetado para isso. Esse tipo de endereçamento é utilizado em sistemas de pequeno porte para economizar *bits* na instrução, possibilitando o uso de uma memória maior que a permitida pelos *bits* do campo de endereço.

No caso do *PDP-8* (máquina com palavra de 12 *bits*) com endereçamento indireto, podem-se alcançar apenas 4K palavras ($2^{12} = 4K$). Para estender o endereçamento até 32K, que necessita de 15 *bits* de endereço, o sistema usa dois registradores com extensão de 3 *bits* cada, o *I-EXT*, que indica os 3 *bits* mais significativos das instruções e operandos diretos e o *D-EXT*, registrador com extensão de 3 *bits* para operandos indiretos. Como no caso do registrador indexador, o programador precisa ter meios (instruções) para carregar e armazenar *I-EXT* e *D-EXT*.

g. Combinações

Em alguns sistemas, depois de formar-se o endereço relativo ou abreviado, a palavra endereçada pode ser interpretada como sendo indireta. A combinação de endereçamento relativo ou abreviado com indexação é viável também. Uma outra combinação interessante é de endereçamento indireto com indexação. O endereçamento indireto com índice funciona assim: (1) usa-se o endereço do campo de endereço da instrução para (2) se retirar o endereço indireto; (3) soma-se esse novo endereço com o registrador de índice; (4) o resultado é admitido como endereço efetivo do operando. Veja a Fig. 6-8, onde nesse caso, é chamado de "indireto com pós-indexação". Também existem arquiteturas com endereçamento indireto com pré-indexação.

6.4 A POTÊNCIA DO CONJUNTO DE INSTRUÇÕES

O que deve constar em um conjunto completo de instruções, que permita ao programador fazer qualquer tipo de computação? Não pretendemos abordar o assunto teórico de computabilidade, mas é interessante saber que, para se fazer qualquer cálculo, o problema não é tanto a potência das instruções, mas principalmente o tamanho do armazenamento disponível. Então, na máquina Turing, esse problema foi resolvido através de uma fita de comprimento infinito. A unidade de controle pode deslocar a fita uma palavra à direita, uma palavra à esquerda e pode imprimir um carácter na fita. O esquema é visto na Fig. 6-9.

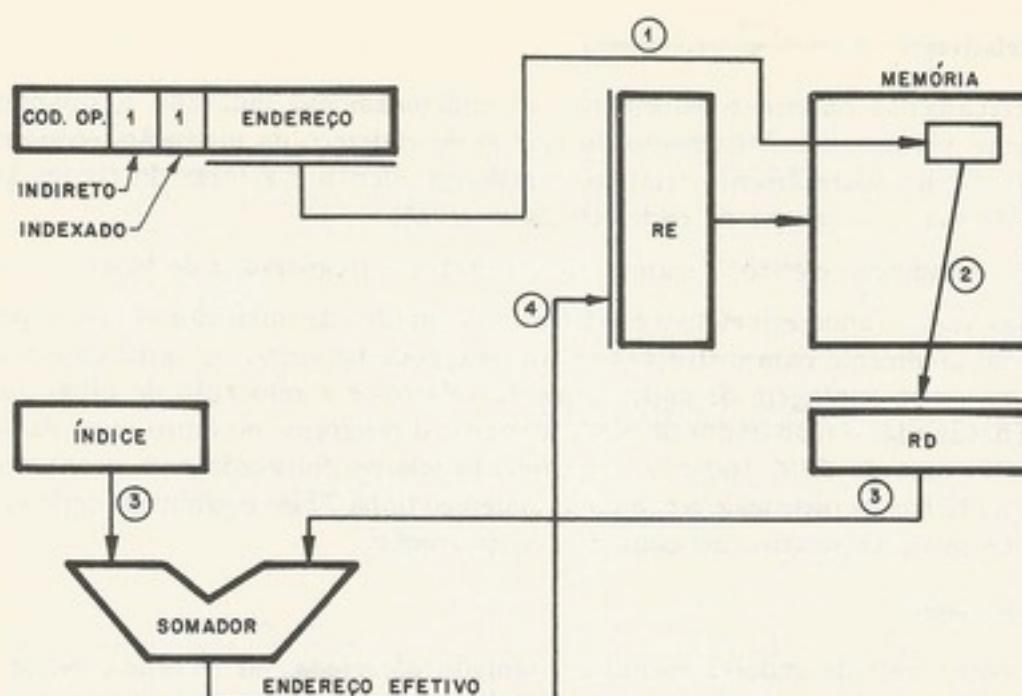


Figura 6-8. Endereçamento indireto com pós-indexação

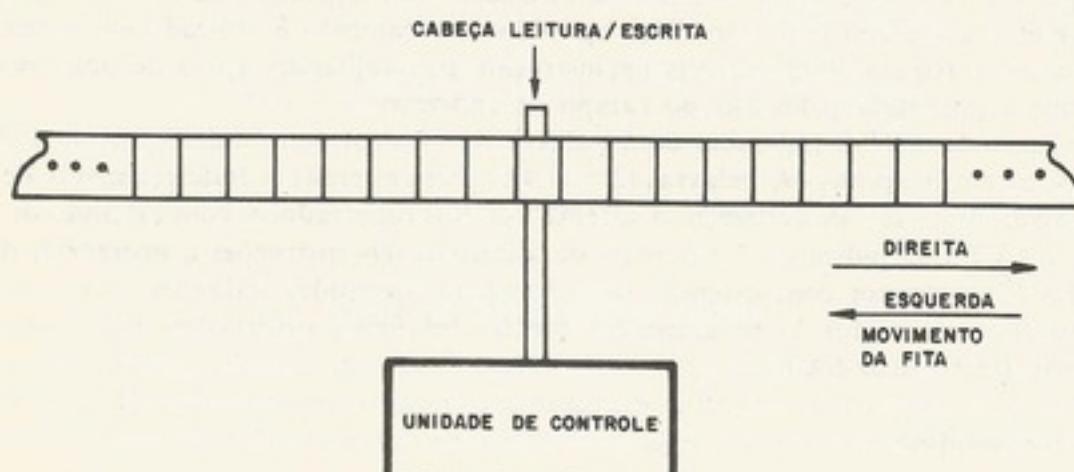


Figura 6-9. Máquina Turing

O “alfabeto” é (a_0, a_1, \dots, a_n) , onde o símbolo a_0 é “branco”, isto é, quando o quadro da fita está em branco, o quadro contém o símbolo a_0 . O conjunto de instruções necessário para se fazerem computações com esse esquema é surpreendentemente simples.

O pesquisador Wang⁽⁶⁾ descobriu que o conjunto de instruções da Tab. 6-2 é suficiente para fins de computabilidade. Esse conjunto de instruções, que opera uma unidade de controle de uma fita como na Fig. 6-9, é chamado *W-machine*.

Em geral, as arquiteturas de computadores digitais dispõem das seguintes classes de instruções:

- (a) aritmética; (b) referências à memória; (c) desvios; (d) deslocamento e operação booleana (manipulação de bits); (e) supervisão e entrada/saída; (f) outras instruções (Edit, Pack, Unpack, Convert).

a. Instruções aritméticas

A máquina PDP-8 tem o mínimo de instruções aritméticas necessárias. Só pode somar com o acumulador. Para subtrair, o programador tem de usar a técnica de complementação

Tabela 6-2. Conjunto de instruções do *W-machine*

Mnemônico	Função
m_i	Imprimir símbolo a_i na fita ($i = 1, 2, \dots, n$)
e	Apagar o quadro na fita (ou seja, imprimir a_0)
$+$	Deslocar a fita um quadro à esquerda
$-$	Deslocar a fita um quadro à direita
$T(N)$	Transferir o controle do programa para linha N se o símbolo embaixo da beira leitura/escrita estiver em a_i , senão o controle pas- sarará para a seguinte instru- ção

do subtraendo. Sistemas mais sofisticados podem ter instruções para somar e subtrair em dupla precisão, para multiplicar e dividir, e para fazer essas operações diretamente em ponto flutuante.

b. Instruções referentes à memória

As instruções de referência à memória têm a finalidade de carregar ou armazenar os registradores centrais (os acumuladores, indexadores, registradores de base etc.). Também nessa categoria, podemos ter instruções para transportar blocos ou operandos *VFL* de um lugar da memória a outro.

c. Desvios

A capacidade de desviar é muito importante. Com uma ampla classe de desvios, a tarefa do programador é bastante facilitada. O "pulo incondicional" transfere o controle a um certo lugar no programa. Às vezes, esse lugar é chamado de *endereço-alvo* (*target*) do desvio. O "pulo condicional" provê meios ao programador de testar uma situação. Se a condição está satisfeita, o controle passa para a instrução armazenada no endereço-alvo; se não, o controle passa para a instrução vizinha a seguir. Um tipo de desvio interessante é chamado de *salto* (*skip*); o controle passa condicionalmente para a segunda instrução a seguir, pulando (saltando) uma instrução. Em muitos casos, a instrução pulada é um "pulo incondicional". A vantagem do salto é que o campo de endereço não é necessário, e esses bits podem ser aproveitados para as especificações das condições dos saltos.

Um tipo de desvio muito importante na aplicação da técnica de sub-rotinas é um desvio que guarda o atual conteúdo do contador da instrução. Assim, através dessa quantidade, o controle pode voltar ao programa principal que chamou a sub-rotina. Um lugar comum para se guardar o contador da instrução (que, nesta altura, deveria ser incrementado para indicar a instrução seguinte) é a primeira palavra da sub-rotina (como no microcomputador do Cap. 5) e o controle passa para a instrução que mora na segunda palavra da sub-rotina.

Essa técnica provavelmente foi aplicada pela primeira vez no computador *LPG-30* da primeira geração do fabricante Royal McBee.

Depois da execução da sub-rotina, temos de devolver o controle ao programa principal que a chamou. Com máquinas com endereçamento indireto, isso é normalmente feito através de um pulo incondicional indireto à primeira palavra da sub-rotina. Essa é a palavra onde

foi guardado o contador da instrução do programa principal. Assim, o controle passa para a instrução que segue aquela do programa principal que pulou à sub-rotina.

Uma outra maneira de se fazer ligação à sub-rotina é guardar o contador da instrução num registrador de índice. Esse tipo de instrução foi usada no IBM 704 (computador de válvulas a vácuo) e também no IBM 7090, onde foi chamada de *branch and link* (desviar e ligar). Essa técnica tem a vantagem de permitir que a sub-rotina busque parâmetros usando o registrador de índice. Na Fig. 6-10, vemos o método usado pela instrução do tipo "desviar e ligar" para juntar à sub-rotina um bloco de parâmetros armazenados na cadeia de instruções do programa principal.

Antes de desviar e ligar, o programa executa um pulo incondicional para a "desviar e ligar".

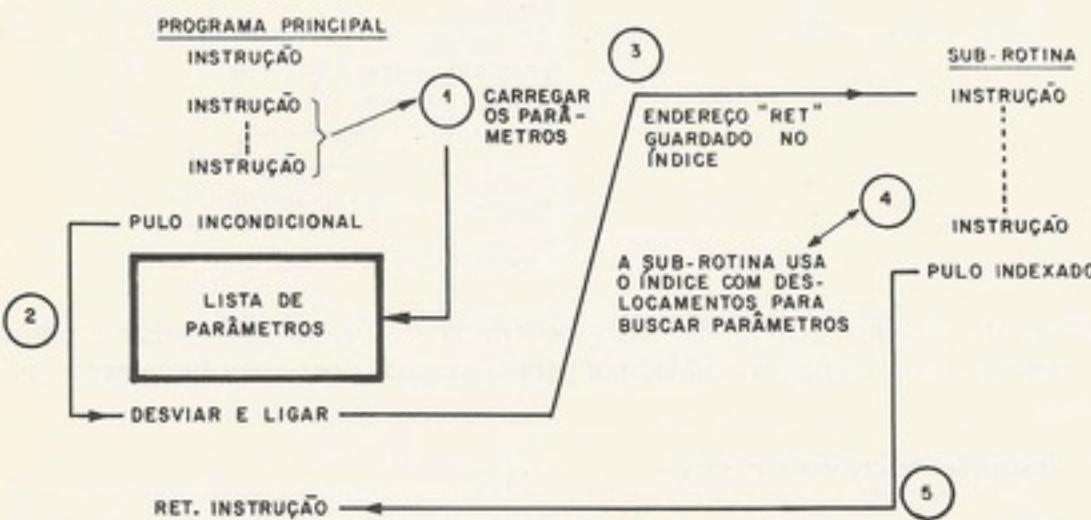


Figura 6-10. Obtendo parâmetro através do registrador de índice

Assim, subtraindo dois do conteúdo do registrador de índice usado para a ligação (o conteúdo -1 daria a instrução "desviar e ligar"), obtém-se o primeiro parâmetro. Para voltar o controle ao programa principal, um pulo para zero, indexado com o valor no registrador de índice usado na ligação, já indica a instrução após a "desviar e ligar" do programa principal.

d. Deslocamento e operações booleanas

Embora a capacidade de deslocar seja conveniente na implementação de rotinas para multiplicar e dividir, essa capacidade é aproveitada nas operações chamadas "manipulação de bits", (*bit manipulations*). Juntamente com as operações booleanas, dão ao programador a capacidade de transcodificar, fazer conversão decimal-binária e binária-decimal, de "empacotar" e "desempacotar" dígitos decimais, examinar individualmente os bits, de uma palavra de *status* ou uma palavra cujos bits indicam a causa de uma interrupção.

As instruções booleanas, usadas com palavras chamadas "máscaras" (*masks*) podem servir para extrair ou inserir bits do acumulador. Por exemplo, vamos supor que queremos "extrair" os 4 bits menos significativos do acumulador de 8 bits, ou seja, limpar os 4 bits mais significativos. Basta usarmos a operação *AND* com máscara 00001111. Assim,

$$\begin{array}{r}
 \text{AAAA AAAA} \\
 \text{AND } 0000 1111 \\
 \hline
 0000 \text{ AAAA}
 \end{array}$$

Suponhamos agora que queremos "ligar" o bit 0 do acumulador. Isso pode ser efetuado através do *OR* com 1000 0000:

$$\begin{array}{r}
 \text{AAAA AAAA} \\
 \text{OR } 1000 0000 \\
 \hline
 1AAA AAAA
 \end{array}$$

e. Supervisão e entrada/saída

Nas primeiras gerações de máquinas pequenas, a entrada/saída era feita com instruções específicas de entrada/saída. Em sistemas bem simples, o programador carrega o acumulador com uma palavra de saída, endereçando depois o dispositivo através de uma instrução cujo campo de endereço indica o dispositivo. (No IBM 704, o sistema usava o registrador *MQ* para entrada/saída, para que o programador pudesse continuar usando o acumulador.) Em sistemas com um programa supervisor, monitor, ou sistema operacional, a entrada/saída é feita pelo supervisor. Isso obrigava comunicação com o supervisor. A implementação dessa necessidade, em alguns sistemas da segunda geração (por exemplo, o Prodac 580), foi feita com o aproveitamento dos códigos de operação ilegais. Nesses sistemas, a unidade de controle, ao decodificador um código de operação ilegal, provocava uma interrupção. Assim, o supervisor era chamado para tratar da interrupção e executava a chamada "pseudo-operação". Na terceira geração, esse conceito foi formalizado e, no *S/360*, foi criado um código de operação especial, chamado *SVC (supervisor call)*. Em alguns sistemas, para que o supervisor possa ter controle completo das atividades do sistema, existem certas instruções, chamadas privilegiadas (como as de entrada/saída), que só podem ser executadas quando o sistema está no modo "supervisor".

Essas instruções, além da entrada/saída, geralmente, tratam do sistema de interrupção e dos demais assuntos como a proteção da memória. Veja a Sec. 6.7.

f. Outras instruções

Uma instrução muito útil em sistemas comerciais é a *edit* (redação). Essa instrução facilita a preparação de uma linha de escrita. Por exemplo, vamos supor que queremos imprimir o número de seis dígitos 003750, e que esse número represente uma quantia de cruzeiros. A operação de redação fez uma conversão para a seguinte linha:

Cr\$ 37,50.

Uma outra espécie de instrução é a de *conversão*. A conversão é uma operação comum, e essas conversões podem ser de alfa-numérico a decimal, decimal a binário, e vice-versa.

6.5 O PDP-8

a. Introdução

Nas seções anteriores, foram explicados assuntos de tamanho e formato de dados, endereçamento e tipos de instruções. Esses assuntos não são independentes. Por exemplo, resolvendo-se o número de bits da palavra, já existem implicações quanto ao formato da instrução. Relativamente à arquitetura, as "peças" devem se encaixar bem, de maneira que não haja desperdícios ou ineficiências. Por exemplo, em geral não é eficiente fornecer um campo de endereço de 20 bits se o sistema nunca vai ter mais de 64K palavras de memória principal. (Talvez só em casos onde o sistema disponha de meios para implementar uma memória virtual.) Por outro lado, se desejamos endereçar até 32K palavras na memória principal e só existem 7 bits no campo de endereço da instrução, a arquitetura tem de compensar essa falta de bits de um modo ou de outro. Para fins de estudo dessa "arte" de arquitetura, e para a visualização de como as "peças" se encaixam, sem nos preocuparmos com os detalhes dos circuitos lógicos, apresentamos a seguir a arquitetura da UCP do *PDP-8*.

A máquina *PDP-8* foi um êxito como um dos primeiros e mais populares minicomputadores da terceira geração. O sistema não é complicado do ponto de vista do *hardware*; o objetivo dos arquitetos era o baixo custo, com bom aproveitamento e utilização dos circuitos do sistema. Sendo a memória um dos componentes mais caros do sistema, os engenheiros conseguiram (como no caso do *CDC 160*) um projeto usando uma palavra de apenas 12 bits. Evidentemente, esse tamanho basta para certas aplicações de controle, aquisição de dados e instrumentação.

As características principais do *PDP-8* são apresentadas na Tab. 6-3. Para que a apresentação seja completa, estão incluídas informações sobre entrada/saída, um assunto que será abordado em mais detalhes no capítulo seguinte.

Tabela 6-3. Características principais do *PDP-8*

1. Palavra	Binário, 12 bits, complemento de dois e inteiros sem sinal
2. Instrução	Uma palavra por instrução, código de operação de 3 bits, tipo sem endereço e endereço simples, usando acumulador como operando
3. Endereçamento	Direto truncado, indireto, "auto-indexado"
4. Aritmética	Soma em complemento de dois. (Instrução de subtração não existe.) Opção de multiplicação e divisão. <i>Flip-flop link</i> para facilitar precisão dupla
5. Memória	Básica de 4 kΩ palavras de 12 bits. Com opção, até 32 kΩ palavras
6. Ciclo da máquina	Baseada no ciclo da memória
7. Entrada/saída	Uma via de entrada/saída, transferência ao acumulador sob controle do programador. Interrupção de um nível só. Opção de <i>data break</i> , uma espécie de <i>cycle-steal</i> .

b. As instruções

O formato da instrução básica está mostrado na Fig. 6-11. Esse formato vale para os códigos de operação vistos na Tab. 6-4.

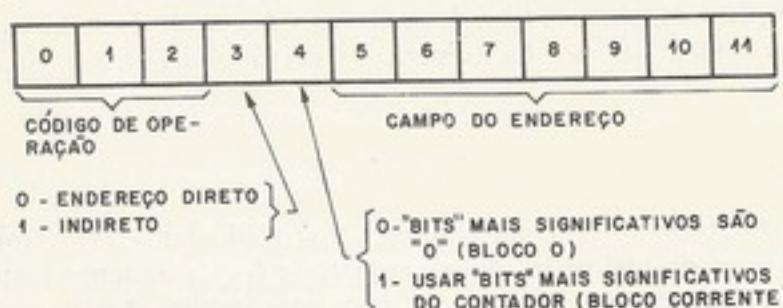


Figura 6-11. Formato das instruções básicas do *PDP-8*

Os bits 5 a 11 são os 7 bits menos significativos do endereço. O próprio campo do endereço da instrução apenas indica uma palavra de um bloco de 128 palavras. Uma memória de 4K palavras possui 32 desses blocos, e o bit 4 da instrução indica em que bloco estará a palavra endereçada. Só podem ser dois blocos: bloco "0" ou o bloco em que "mora" a atual instrução.

Com o bit 3, a instrução especifica se a palavra retirada do bloco "0" ou do bloco corrente é o próprio operando (endereçamento direto) ou, caso contrário, o endereço do operando (endereçamento indireto de um nível só). Com endereçamento indireto, o programador pode endereçar qualquer palavra dentro de 4K palavras. O fato de, além do bloco 0, só o bloco corrente de 128 palavras poder guardar operandos endereçados diretamente ou guardar ponteiros para endereçamento indireto, torna o problema de programação mais difícil. O programador deverá usar o bloco "0" para ponteiros e operandos comuns aos blocos do programa e deverá organizar seu programa em blocos de 128 palavras. É um problema o fato de, quando o *CI* é incrementado de um bloco com 128 palavras ao próximo, todos os

Para que a apresentação do assunto que

Tabela 6-4. Instruções básicas do PDP-8

Código	Mnemônico	Função
000	AND	Operação booleana AND entre o conteúdo do endereço efetivo e o acumulador
001	TAD	Somar o conteúdo do endereço efetivo com o acumulador, em complemento de dois
010	ISZ	Incrementar o conteúdo do endereço efetivo e saltar a próxima instrução se o resultado for zero
011	DCA	Armazenar o acumulador no endereço efetivo, deixando o acumulador limpo
100	JMS	Pular para sub-rotina, deixando o CI no endereço efetivo e passando o controle ao endereço efetivo mais um.
101	JMP	Pulo incondicional; o controle passa ao endereço efetivo

antigos operandos locais não poderem ser endereçados; então a fronteira entre blocos não pode ser cruzada sem se pensar nas consequências. Nesse sentido, endereçamento relativo ao CI (usando-se a soma do CI e o deslocamento) eliminaria a barreira artificial a fronteira dos blocos. Mas, de qualquer maneira, a tarefa do programador é dificultada porque a instrução não dispõe de mais bits de endereço.

O sistema tem uma técnica chamada *auto-index*: se uma palavra entre os endereços $(10)_8$ e $(17)_8$ está sendo usada como ponteiro indireto (bit 3 = "1", bit 4 = "0", e os bits 5 a 11 entre $(10)_8$ e $(17)_8$, então o ponteiro será incrementado de "1". Por exemplo, se a palavra 10 indica uma tabela e o ponteiro for 1000, a execução da instrução TAD, endereço 10, indireto, vai deixar a palavra 10 com o ponteiro 1001.

O código de operação "110", com mnemônico IOT é usado para funções de entrada/saída. O formato aparece na Fig. 6-12.

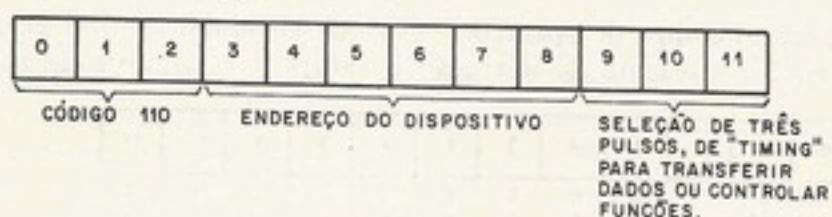


Figura 6-12. Formato da instrução IOT do PDP-8

O código de operação que resta é "111", que realmente indica uma classe de instruções que os projetistas chamaram de *operate* (operar). Certos bits dessa instrução controlam diretamente certas funções no fluxo de dados, mas o leitor não deve levar a impressão de que o PDP-8 é microprogramado. Antes de explicar as instruções de código "111", obviamente devemos explicar o fluxo de dados. Veja, na Fig. 6-13, o gráfico do fluxo de dados.

A memória é endereçada através do registrador de endereço (REM), que recebe o endereço da via Z, à saída da unidade aritmética. O registrador de dados da memória (RDM) recebe a palavra lida da memória e segura os dados durante duas escritas. O RDM também recebe da via Z. Realmente, a unidade aritmética é o ponto focal do fluxo de dados e sua

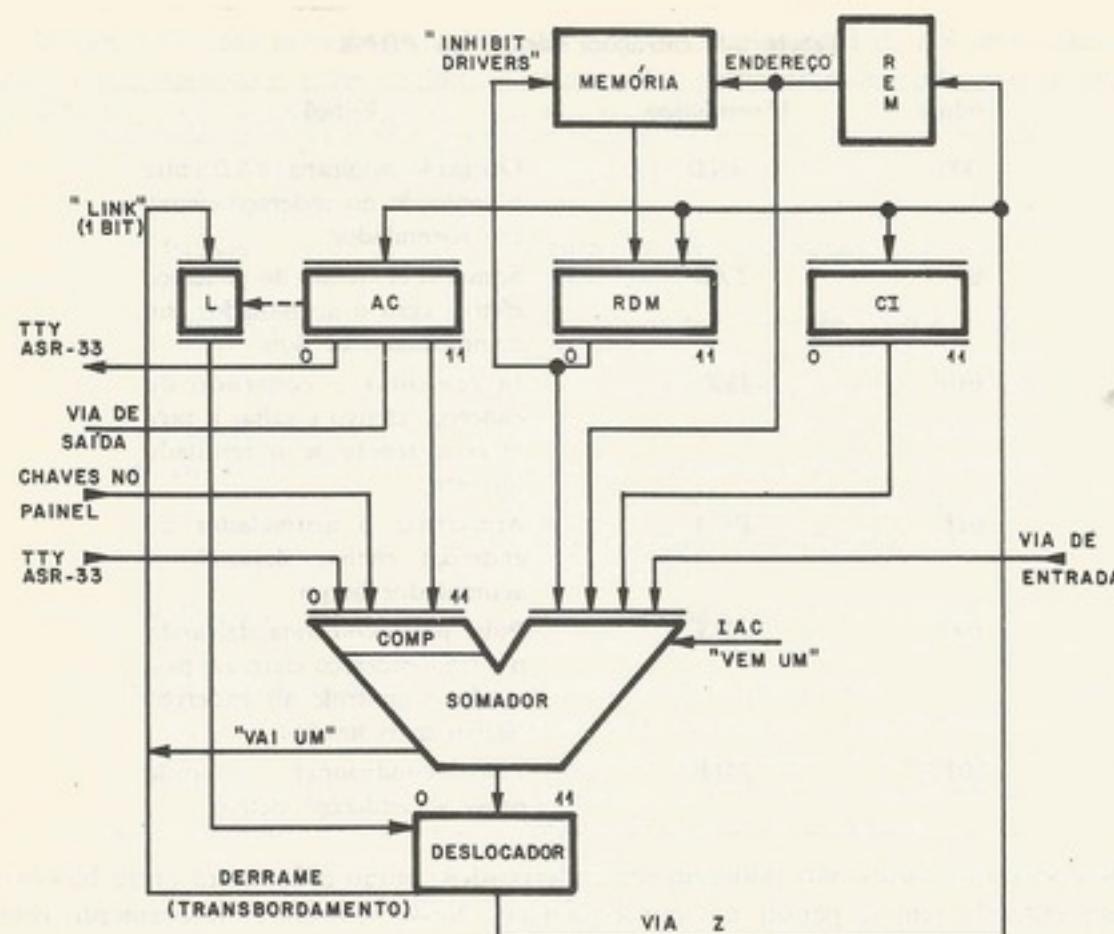


Figura 6-13. O fluxo de dados do PDP-8.

saída, via Z , é o único caminho de distribuição de informação para os registradores centrais REM , acumulador (AC) e o contador da instrução (CI).

Existe um *flip-flop* chamado *link* que é mais ou menos uma parte do acumulador. Durante a operação de somar (*TAD*), o “vai um transbordo” do somador muda o estado de *link* e, durante as operações de deslocamento, o *link* recebe o bit de transbordamento. Também durante deslocamentos o *link* preenche a vaga. Com essa filosofia, os deslocamentos do *PDP-8* realmente são giros de uma palavra de 13 bits (os doze da palavra normal mais o *link*).

As instruções do código de operações “111” são divididas em dois grupos, dependendo do estado do bit 3. O formato da instrução do grupo 1 está mostrado na Fig. 6-14.

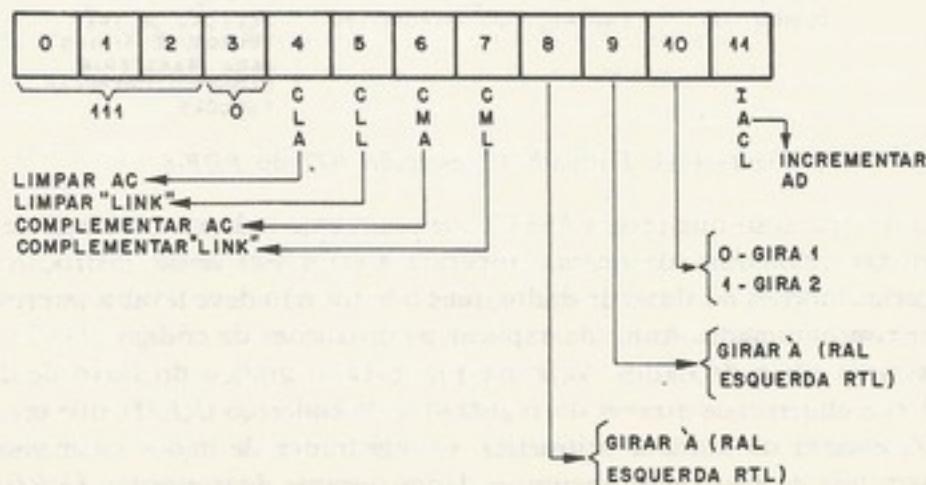


Figura 6-14. Formato das instruções do grupo 1

Essa instrução é executada basicamente em um ciclo do fluxo de dados onde o conteúdo do acumulador passa através do somador. Em seguida, o circuito de deslocamento é devolvido ao acumulador. Os bits de instrução do grupo 1 controlam certas portas e sinais de controle durante essa passagem do conteúdo do acumulador.

A atuação das instruções do grupo 1 pode ser esclarecida da seguinte maneira:

- 1) o acumulador, através de uma porta de seleção, é enviado ao somador; fechando-se essa porta, surge o efeito de "limpar AC" (bit 4). O mesmo acontece com "link" (bit 5);
- 2) a entrada esquerda do somador passa por uma porta "verdadeiro/falso" que, ou passa o bit como entrou ou passa o complemento (bit 6). O mesmo acontece com o link (bit 7);
- 3) passando pelo somador, o bit 11 (IAC) controla o "vem um quente", se é para incrementar;
- 4) a saída do somador passa pelo circuito de deslocamento, onde ela pode ser girada ou não, conforme os bits 8 e 9, ou de uma posição, ou duas (bit 10).

Os bits da instrução, grupo 1 podem ser usados em várias combinações para fazer várias coisas. Por exemplo, controlando os bits 6 e 11, podemos formar o complemento de dois do acumulador; usando os bits 4, 5, 7, 9 e 11 podemos colocar o inteiro "3" (0000 0000 0011) no acumulador (limpar o acumulador, colocar o "link" no estado "1", incrementar e depois girar uma casa à esquerda).

O formato das instruções do grupo 2 está mostrado na Fig. 6-15.

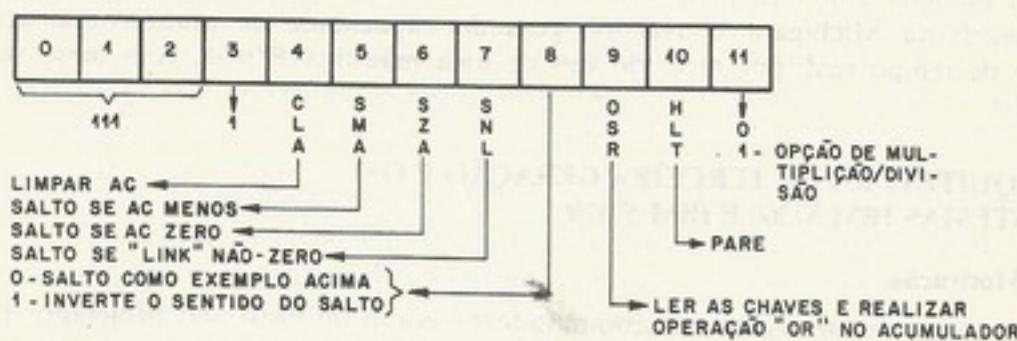


Figura 6-15. Instruções do grupo 2 do PDP-8

O grupo 2 permite que se testem condições no fluxo de dados. (Também, usando os bits 4 e 9, o programa pode ler o estado das chaves no painel.) O bit 8, no estado "1" ("ligado"), inverte o sentido do salto. Supondo-se que o bit 8 = 0, se uma das condições selecionadas pelos bits 5, 6 ou 7 estiver satisfeita, o controle do programa vai saltar. Com o bit 8 no estado "1", a situação estará invertida, e o salto será feito só se não fosse saltar antes.

Esse grupo se comporta ainda como um salto incondicional quando os bits 5, 6 e 7 estão todos em "0" (nenhuma condição selecionada) e com o bit 8 no estado "1".

A capacidade de testar o link é muito útil para se examinarem os bits de uma palavra. Basta colocar-se a palavra no acumulador, girar a palavra através de instruções do grupo 1 e, depois, saltar com o link usando a instrução do grupo 2.

c. Estados principais e interrupção

Para a execução de instruções, a unidade de controle usa três estados principais: *ciclo I* (que usa *CI* para retirar a instrução); *indireto* (zero, um, ou vários ciclos da memória, dependendo da instrução e dos níveis de endereçamento indireto); e *execução* (um ciclo que completa a execução da instrução). Uma interrupção (ressaltamos que é uma instrução executada fora da ordem normal, provocada por uma fonte no sistema) só é permitida no PDP-8 (como no microcomputador do Cap. 5), depois da execução de uma instrução. No PDP-8, acontecendo interrupção, o *CI* é armazenado no endereço 0, e a instrução seguinte

executada é aquela do endereço 1. A interrupção do sistema não admite níveis de prioridade, nos quais uma interrupção proveniente da uma fonte com mais prioridade interrompe os níveis com menos prioridade.

d. Comentário

Um ponto de partida para o projeto do *PDP-8* foi a economia na memória principal e no *hardware*. Com isso, o conjunto de instruções não ficou muito rico e o endereçamento tornou-se mais difícil. A falta de potência nas instruções existentes no sistema obriga o programador a usar uma grande quantidade de instruções no programa. Também com as restrições no endereçamento, o programador tem de gastar palavras na memória para guardar ponteiros para os dados que ele não possa endereçar diretamente.

Observamos aqui a influência do fluxo de dados na arquitetura. O endereçamento foi "truncado", criando barreiras entre blocos, possivelmente porque endereçamento relativo não era uma solução possível. Não podiam somar o campo de endereço da instrução [posições (5-11) do *RDM*] com o *CI* porque ambos registradores alimentam a mesma entrada do somador. Para outras provas do fato de a arquitetura ser influenciada pelo fluxo de dados, basta indicarmos a instrução do código "111", especialmente as opções do grupo 1.

Um fator muito interessante com o *PDP-8* é que, com um conjunto de instruções bastante modesto, o programador ainda pode implementar aplicações interessantes.

Uma aplicação de programação feita no *PDP-8* foi a de "concentrador de dados". Essa aplicação foi feita com um programa supervisor chamado *RAMP* (*random access multi-programmed*) na Michigan University, gerando capacidade de multiprogramação num ambiente de tempo real, por meio do uso de uma máquina *PDP-8*, com memória de 4K palavras⁽⁷⁾.

6.6 ARQUITETURA DA TERCEIRA GERAÇÃO E OS SISTEMAS IBM S/360 E IBM S/370

a. Motivação

Os sistemas pequenos, os minicomputadores, como foi visto, são projetados para economizar *hardware*. Muitos desses sistemas são aplicados para algum trabalho específico, não havendo necessidade de grandes investimentos em *software*. Por outro lado, muitos usuários necessitam de um sistema de uso geral, pois a empresa usa seu computador em várias aplicações. Um dos acontecimentos revolucionários da segunda geração foi o crescimento de *software*. Provavelmente, a mudança mais profunda na área de computação nos últimos dez anos foi no *software*.

Hoje em dia, o programador de sistemas com *software* fica totalmente isolado do *hardware*. Em sistemas de grande porte, o sistema operacional tem muito a fazer: contabilidade, gerência das informações (arquivos e acesso a eles), escalação dos serviços, etc. Além do sistema operacional, um segundo aspecto do crescimento de *software* foi a adoção de linguagens de alto nível, necessitando o projeto de compiladores para cada arquitetura diferente. Uma terceira inovação foi a multiprogramação. Com a preocupação com o tempo ocioso da *UCP*, enquanto esperava pela entrada/saída, surgiu a seguinte idéia: se houvesse, na memória principal, um programa e dados prontos para rodar, este poderia aproveitar a *UCP* no tempo ocioso do outro programa. A generalização dessa idéia, chamada *multiprogramação*, exige certas características desejáveis da arquitetura e requer um bem projetado programa supervisor.

b. As direções da arquitetura

Acabamos de descrever a situação como estava no fim da segunda geração. Com o *software* desempenhando um papel de importância sempre maior e com esse *software* custando sempre mais, é interessante estudar esse impacto na arquitetura do *hardware*. Mais

uma vez, podemos ver a realimentação do efeito do *software* no *hardware*. Desta vez, o impacto é um pouco mais profundo do que o da providência de endereçamento indexado ou de um pulo de sub-rotina, ou até interrupção de programa.

A primeira reação a essa situação parece ser o Honeywell *H-800*, projetado com "multiprogramação em hardware" (cerca de 1958). A máquina tinha oito conjuntos idênticos de acumulador, *CI* e demais registradores guardados numa memória pequena de "rascunho" (*scratchpad*). O sistema podia ter, no máximo, oito programas ativos simultaneamente. Também tinha *status* para cada conjunto, indicando se estava "pronto", "esperando E/S" ou "não sendo usado". Em *hardware*, a unidade de controle alternará a execução de instruções entre os programas ativos executando uma instrução de cada programa ativo em maneira *round-robin*. Essa técnica facilita a escalação de programas.

A reação mais dramática à situação no fim da segunda geração foi a da Burroughs, exemplificada na arquitetura do *B-5000* (descrito na Fall Joint Computer Conference de 1963) e seus sucessores, como o *B-5500* e *B-6700*. Nessa arquitetura, foi previsto o monitor (tinha dois estados internos, "mestre" e "normal"), segmentação de memória (descritor de dados) e a utilização de linguagens de alto nível (a pilha interagindo com instruções numa cadeia de *polish notation*). O aspecto do conjunto de instruções parece ser influenciado pela linguagem *ALGOL*, também como o tratamento de sub-rotinas. A técnica de compilação aproveitando *polish notation*⁽⁸⁾ e a pilha também influencia muito a arquitetura do *B-5000*.

A "Polish notation" (mas não descritores) foi usada também no computador inglês *KDF-9*, descrito por Davis⁽⁹⁾ em 1960. A máquina da Rice University⁽⁴⁾, que foi desenvolvida por volta de 1962, usava a técnica de descritores de dados e programas, mas não se serviu da "*polish notation*", e sim um formato de instrução mais convencional.

R. S. Barton⁽¹⁰⁾ cita as vantagens da arquitetura da Burroughs: a cadeia de instruções consta de operações, "nomes" (descritores) e operandos imediatos, em vez de instruções do tipo convencional (por exemplo, endereço único). A sintaxe dessa linguagem é bastante simples. A cadeia de instruções necessita de menos espaço que outras formas, pois existem muitas referências implicadas à memória que não aparecem explicitamente. Além do mais, o mecanismo da pilha representa uma medida de alocação automática da memória; a pilha serve para resultados intermediários, parâmetros de sub-rotinas, variáveis, e permite procedimentos recursivos e interrupção em qualquer profundidade.

Uma reação ao *software* nas linhas mais clássicas foi a da IBM. O projeto da arquitetura do sistema IBM *S/360* começou no fim de 1961, e o *S/360* foi anunciado em 1964⁽¹¹⁾. Muitas idéias no *S/360* relativas à multiprogramação têm suas raízes no projeto stretch, a máquina IBM 7030 de grande porte que foi entregue ao Los Alamos Scientific Laboratory, em 1961⁽¹²⁾. A IBM introduziu uma "família" de modelos, todos (com exceção dos modelos 20 e 44 e depois o 67) com a mesma arquitetura. Na segunda geração, a IBM havia fornecido compiladores e supervisores para vários sistemas, por exemplo, IBM 1401, IBM 1410, IBM 1710, IBM 7040, IBM 7070, IBM 7080 e IBM 7090. A idéia da "família" foi, em vez de se escrever e manter uma dúzia de montadores ou compiladores Fortran, por exemplo, por que não escrever só um ou dois na linguagem *S/360*, que serve para todos os modelos? Isso obrigou que a arquitetura fosse uma reunião da filosofia "série *VFL decimal*" da série 1401 para aplicações comerciais e a filosofia "paralelo, palavra ponto fixo, binário" da série 7090 para aplicações científicas. Essa integração beneficia o usuário também quando ele muda de *UCP* para aumentar a potência do sistema, ele não tem de reprogramar tudo novamente.

6.7 A ARQUITETURA DO IBM *S/360*

Como no caso do *PDP-8*, queremos ver uma arquitetura "completa e bem integrada". Infelizmente, o *S/360* é muito complexo e, portanto, não é nossa intenção entrar em muitos pormenores. Apresentamos aqui, então, o que julgamos importante para melhor atender à direção tradicional da arquitetura da terceira geração, e a influência dos seguintes pontos:

(1) sistemas operacionais; (2) utilização de linguagens de alto nível; (3) multiprogramação; (4) conceito de "família".

a. Ponto de vista do programador

O S/360 consta de uma memória principal, endereçável no nível de byte, uma palavra de 8 bits. Os registradores centrais, chamados *general purpose registers* (GPR) são em número de 16, de GPR 0 a GPR 15. Esses registradores são acumuladores de 32 bits. Uma pesquisa no IBM 7090, que tem um único acumulador, mostrou que, em trinta por cento dos casos, o programador guardava um resultado intermediário na memória só para recarregá-lo no acumulador, algumas instruções depois.

Por simulação, determinou-se que, com quatro acumuladores, noventa por cento dessa redundância (armazenar só para depois se carregar) podia ser eliminada. Por isso, o S/360 tem acumuladores múltiplos. A razão de ter dezesseis é que os acumuladores, como veremos, servem como indexadores e registradores de base. (O sistema inglês Pegasus, em 1957, utilizou registradores centrais como acumuladores ou indexadores.) O sistema também dispõe de quatro registradores de 64 bits cada, usados para guardar números em ponto flutuante. O esquema de entrada/saída é feito através de um sistema digital autônomo chamado *canal*. O aspecto lógico do sistema é mostrado na Fig. 6-16.

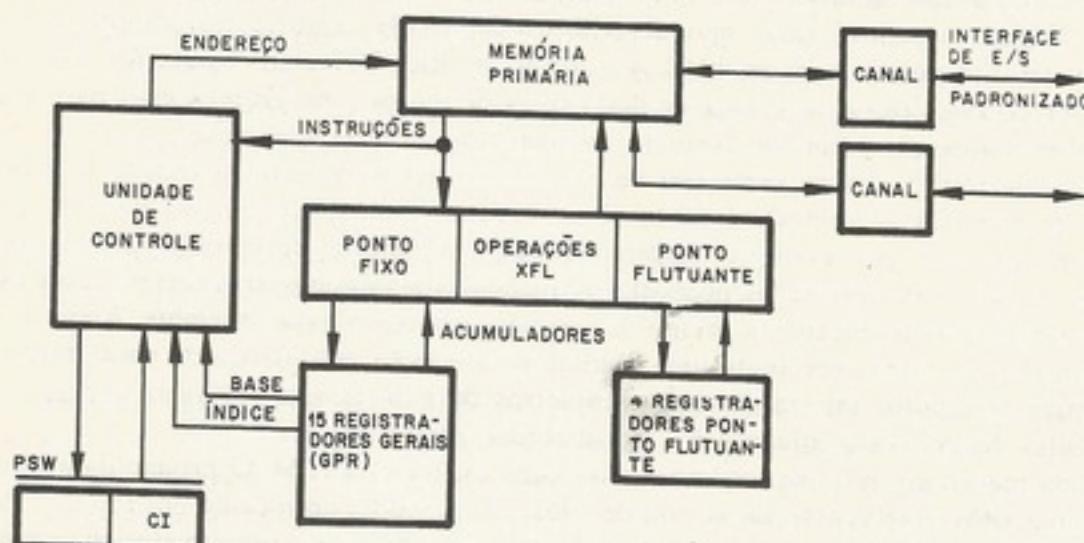


Figura 6-16. Aspecto lógico do S/360 e S/370

Conceitualmente, a figura é o desenho do "fluxo de dados" da arquitetura. A arquitetura do S/370 tem poucas diferenças com a do S/360, e a maioria das modificações ocorre na área de entrada/saída.

b. Formato dos dados

A unidade básica é o *byte*, quantidade de 8 bits. Essa quantidade serve para um carácter no código alfa-numérico *ASCII* ou *EBCDIC* na comunicação Homem-máquina.

Quatro bytes formam uma palavra de 32 bits, a quantidade básica para operações em ponto fixo. Em ponto flutuante, existem duas representações: curta e longa. A única diferença é que a longa dispõe de mais 32 bits de mantissa (nesse caso, a fracionária) aumentando a precisão. Para aritmética *VFL* em decimal, existe a representação "decimal empacotado" (*packed decimal*), onde 4 bits formam um dígito decimal do código *BCD*. O problema de reconciliar o "decimal *VFL*" do IBM 1401 e o "binário, comprimento fixo da palavra" do IBM 7090 foi resolvido permitindo as duas representações. Correspondente a cada formato de dado, existem seus próprios códigos de operação, os operandos não são

“auto-descritor” do campo de nome
tentar operar
é útil em um
Os tipos
-palavra de 1
dois. Isso per
de dados de u
de um, tem

Os númer
existem mui
conversão d
ponto flutuan

Existem in
existem insu

c. Formatos

Os format
em comprim

O tipo de
indica qual s
não necessari
(GPR). Basic
rando, denunci

(c) multiprogra-

uma palavra
só em número
Uma pesquisa
cento dos casos,
recarregá-lo no

por cento dessa
Por isso, o S/360
como veremos,
em 1957, uti-
também dispõe
ponto flutuante.
chamado canal.

INTERFACE
DE E/S
PADRONIZADO

"autodescritos". Uma instrução de ponto fixo, por erro do programador, pode operar num campo decimal VFL sem que o hardware do sistema perceba. (Mas se uma instrução decimal tentar operar com um dígito ilegal em BCD, provoca uma interrupção.) O ponto flutuante é útil em aplicações científicas, isto é, com programas compilados da língua-fonte Fortran.

Os tipos de dados são mostrados na Fig. 6-17. Não está mostrado o tipo *halfword* (meia-palavra de 2 bytes). Os números em ponto fixo, são representados em complemento de dois. Isso porque o sistema dispõe de muitos modelos. No modelo 30, com largura de fluxo de dados de um byte, não convém tratar do "retorno do transporte" exigido pelo complemento de um, nem da recomplementação do sinal e amplitude.

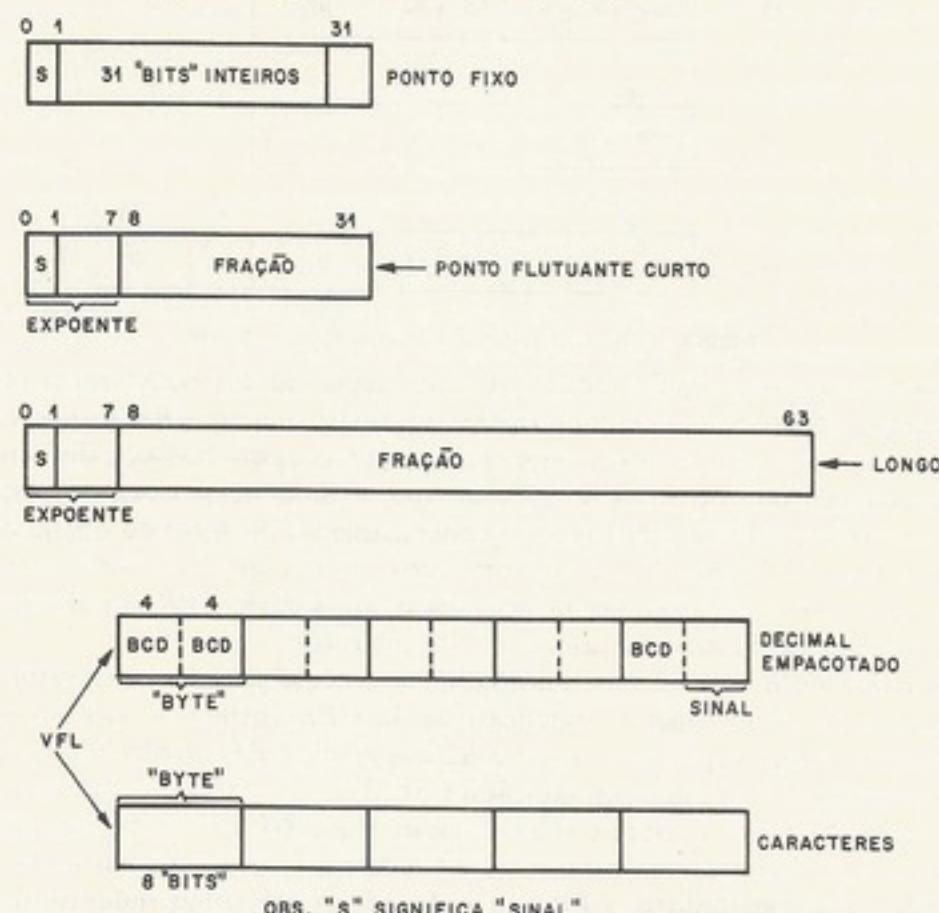


Figura 6-17. Formato de dados, S/360

Os números decimais usam sinal e amplitude. Em muitas aplicações comerciais, não existem muitos resultados negativos e o esquema sinal e amplitude é mais conveniente para conversão (desempacotamento) na entrada/saída. (Para uma explicação do formato do ponto flutuante, veja o Cap. 2).

Existem instruções de conversão binária para decimal empacotado e vice-versa. Também existem instruções de "empacotar" e "desempacotar" caracteres EBCDIC.

c. Formatos da instrução e do endereçamento

Os formatos da instrução são como os da Fig. 6-18. Os tamanhos são de 2, 4 ou 6 bytes em comprimento.

O tipo do formato é distinguido pelo próprio código de operação; a primeira coluna indica quais são os bits mais significativos do código de operação. Instruções do tipo RR não necessitam referência à memória principal, os operandos estão nos registradores centrais (GPR). Basicamente, a instrução realiza uma operação entre o primeiro e o segundo operando, deixando o resultado no lugar do primeiro operando.

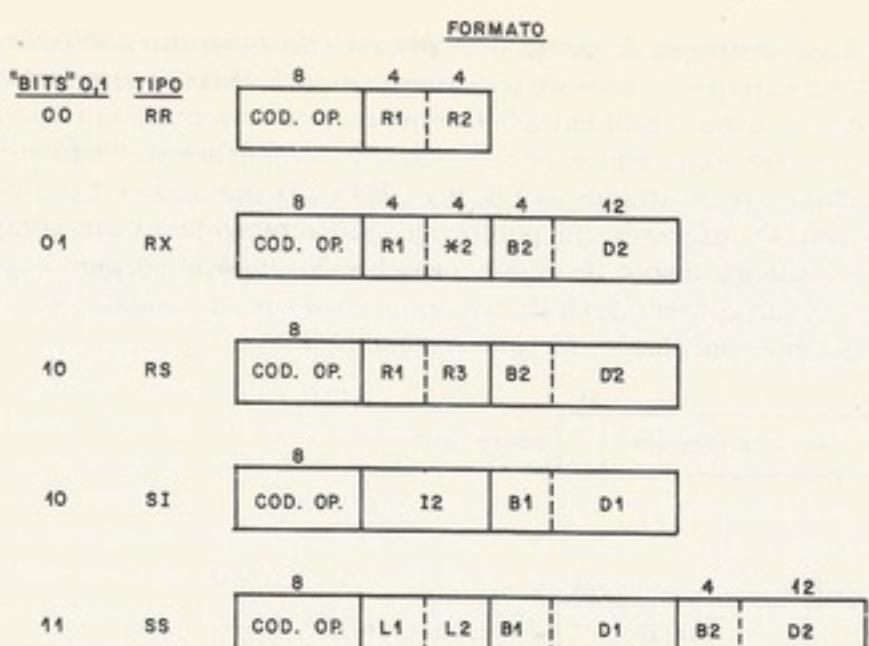


Figura 6-18. Formato da instrução do S/360

O campo *R1* tem 4 bits para indicar qual dos registradores *GPR* será o primeiro operando; igualmente *R2* indica o segundo operando. Instruções tipo *RR* dispõe de operações aritméticas (ponto fixo ou flutuante) comparações, operações booleanas, e desvios. Instruções do tipo *RX* dispõem de operações do mesmo tipo, só que, nesse caso, o operando é uma palavra na memória em vez de um *GPR*. O endereçamento do segundo operando usa a técnica *relativa à base*. O campo *B2*, se não for “zero”, indica qual dos *GPR* entre 1 e 15 vai servir como base. Daí o endereço ser formado pela soma desse *GPR* e o *deslocamento*, uma quantidade positiva de 12 bits do campo *D2* da instrução.

O registrador *GPR* tem 32 bits, mas, como o sistema permite no máximo 24 bits de endereço, apenas os 24 bits menos significativos do *GPR* entram no cálculo do endereço.

Por exemplo, vamos supor que o campo $B2$ seja 4, que $GPR\ 4$ tenha 5 673 e que o campo $D2$ tenha a quantidade 7. O endereço indicado é $5\ 673 + 7 = 5\ 680$. No caso em que $B2 = 0$, o endereço seria $\cdot D2$; nesse exemplo, seria 7, mesmo que $GPR\ 0$ tenha algo diferente. Nas instruções tipo RX , encontramos indexação indicada pelo campo $X2$ não-nulo. Nesse caso, $X2$ indica qual dos registradores $GPR\ 1$ a $GPR\ 15$ servirá como indexador.

Algumas instruções do tipo *RS* têm três operandos (veja a Fig. 6-18). Aqui, uma das aplicações é a de desvio baseado num índice, onde o segundo operando é o alvo do desvio, o *GPR R1* é uma conta sendo incrementado pela quantidade guardada no *GPR R3* e comparado com o *GPR R3 + 1*.

Instruções de deslocamento *RS* não usam *R3*. O número de deslocamentos é indicado pelos 6 bits menos significativos do endereço do segundo operando. Nas instruções tipo *SI*, o campo *I* é um byte de informação imediato.

As instruções do tipo SS utilizam os dois operandos na memória, em campos de comprimento variável (*VFL*). Os operandos podem ser caracteres *EBCDIC*, *ASCII*, ou decimal empacotado. Os operandos são indicados por (*GPR B1 + D1*) e (*GPR B2 + D2*) respectivamente. No caso de decimal, os comprimentos dos campos são indicados por *L1* (4 bits) e *L2* (4 bits) respectivamente e, portanto, os campos decimais empacotados são, no máximo de 16 bytes ou seja, de trinta e um dígitos e o sinal. (O *COBOL* exige, pelo menos dezoito dígitos mais o sinal.) No caso de caracteres, o campo *L1, L2* (8 bits) indica um comprimento de 1 até 256 bytes. Nessas instruções, a arquitetura foi influenciada pelo êxito de sistemas comerciais como o IBM 1401.

Os arquitetos do S/360 tinham experiências variadas. Mas, segundo Brooks, "a experiência mais importante foi a inevitabilidade de maior memória"⁽¹³⁾. Assim, foi desenvolvido

o endereçamento com o registrador de base, para economizar bits de endereço na instrução e, ao mesmo tempo, dar a capacidade de endereçar 16M bytes de memória primária. Usando-se um registrador de base também se facilita a tarefa de *relocação*.

O problema de carregar programas na memória é complexo em sistemas multiprogramados. Não se pode prever onde o programa será colocado. O programa carregador tem, na hora de carregar um programa na memória, de acertar o endereçamento das instruções. Na maioria dos sistemas da segunda geração, isso necessita de mudança de campos de endereços nas instruções, por uma quantidade chamada *relocation constant*. A técnica de endereçamento relativo à base já simplifica essa tarefa: o programa carregador carrega a *relocation constant* no GPR, sendo usado como base o "deslocamento" que já está na instrução e não precisa de modificação alguma. Na realidade, a relocação não é tão simples assim, mas é evidente que o endereçamento com registrador de base é um exemplo onde o *hardware* que calcula o endereço efetivo ajuda o *software* fazer relocação.

A arquitetura do S/360 não tem endereçamento indireto. No entanto, é muito comum, em *software*, guardarem-se ponteiros para várias quantidades em tabelas, sendo conveniente utilizarem-se esses endereços no modo indireto. Contudo, o S/360 não tem esse tipo de endereçamento e, portanto, o programador carrega um dos GPR com o ponteiro através da instrução *load register* (mnemônico *LR*) e, depois, usa o GPR como registrador de base com deslocamento "0". Por isso, a ocorrência da instrução *LR* é surpreendentemente alta.

No S/360, a instrução "desvio condicional" (mnemônico *BC*) opera junto com um código de dois bits chamado o *código da condição*, armazenado em dois *flip-flops*. As instruções aritméticas e as comparações deixam informações sobre o resultado no código de condição (*CC*): se zero, positivo, negativo, ou estouro, caso adição ou subtração. No caso de operações booleanas, o *CC* indica se o resultado foi zero (veja a Tab. 6-5).

Tabela 6-5. Código de condição (*CC*)

Adição ou subtração	<i>CC</i>
Resultado zero	00
Resultado negativo	01
Resultado maior que zero	10
"Estouro" (<i>overflow</i>)	11

Junto com a instrução *BC* do desvio condicional vem um campo de 4 bits (*tag*), um bit para cada estado do *CC*. Durante a execução, se o estado do *CC* corresponde à condição indicada por "1" no *tag*, o controle passa para o endereço efetivo da instrução *BC*. Assim, o *tag* de "1111" é um desvio incondicional. Esse tipo de desvio representa uma generalização e uma economia sobre os vários desvios condicionais, especialmente quando existem dezenas de acumuladores, em vez de apenas um. O desvio para sub-rotina é o *branch and link* do IBM 704 e, mais tarde, o do IBM 7090, isto é, o *CI* fica armazenado num registrador de índice, ou seja, num GPR.

As instruções aritméticas são bastante poderosas, pois têm ponto flutuante, multiplicação e divisão em *hardware*. Existem sete instruções de multiplicação e de divisão. Isso porque existem sete tipos de dados e formatos. Ponto fixo, de 32 bits, ponto flutuante longo e ponto flutuante curto, ambos com instruções *RR* ou *RX*, e ainda existem as operações em decimal.

O sistema dispõe de instruções poderosas do tipo *VFL* (formato *SS*). A instrução *translate* faz uma procura de uma seqüência de bytes, cada byte do argumento sendo substituído por um byte de função, segundo uma tabela de funções. Uma outra instrução, mnemônico *TRT*, procura, numa seqüência de bytes, certos bytes de interesse. No caso de um compilador, os bytes de interesse normalmente são os delimitadores (carácter em branco, vírgula, ou *carriage return*) de uma linguagem de alto nível. A arquitetura também dispõe de duas instruções, *EDIT* e *EDIT AND MARK*, para redigir um campo, preparando-o para ser escrito.

6.8 FEIÇÕES DO S/360 E S/370 PARA MULTIPROGRAMAÇÃO

a. Introdução e o sistema Stretch

O começo da multiprogramação aconteceu com um computador Univac Scientific, usado para os cálculos de um grupo de engenheiros que fazia experiências com um túnel de vento⁽¹⁴⁾. As experiências não eram freqüentes, mas, quando havia dados do túnel de vento para processar, os engenheiros atrapalhavam qualquer outro que estivesse sendo processado no computador, mesmo que se estivesse no meio de um programa longo. A solução desse problema foi a *interrupção de programa*, que envolveu uma modificação em *hardware*. Depois da implantação da interrupção, quando os engenheiros queriam utilizar o computador, interrompiam o programa em andamento, transferindo o controle a um programa especial (tratador da interrupção). O programa tratador guardava o estado dos registradores centrais e demais informações (como o estado do *flip-flop* de transbordamento), do programa interrompido. Daí, o programa passava o controle ao programa dos engenheiros. Quando os engenheiros não precisavam mais do computador, o controle passava para um programa que restaurava os registradores centrais e demais informações do programa interrompido e devolvia o controle à instrução interrompida. Em um sentido, o sistema foi multiprogramado: possuía dois programas, normalmente um ativo e o outro esperando ativação.

Além do Honeywell H-800, o computador Stretch⁽¹²⁾ foi um dos primeiros em que se implantou a multiprogramação. No projeto Stretch, desenvolveram-se umas quatro características em *hardware* para ajudar: relógios especiais; interrupção flexível; proteção da memória; entrada/saída assíncrona com processamento.

Com dois relógios, um *timer* e o outro "hora do dia", o sistema poderia ser interrompido periodicamente para se verificar se algum programa estava num *loop* e também poderia indicar ao usuário quando o seu programa tivesse sido rodado. O esquema de interrupção permitia níveis de prioridade, e permitia ao supervisor inhibir certas fontes de interrupção enquanto estivesse tratando com uma outra interrupção. Além do mais, certas condições no programa, como "dividir por zero" e "estouro", foram tratadas através de uma interrupção "exceção de programa".

Para proteger um usuário dos outros e para proteger o programa supervisor dos programas usuários, projetou-se um esquema de proteção de memória principal no qual o usuário localiza-se em uma região da memória entre dois limites. O programa não podia fazer referências fora dos limites.

O esquema da entrada/saída, também usado nos IBM 709 e 7090, utilizou um canal operando assincronamente com o processamento. Ao terminar uma tarefa de *E/S*, o canal interrompia a *UCP*, sendo tratado pelo supervisor. Isso permitia uma superposição de *E/S* de um programa, com processamento de um outro, vantagem da técnica de multiprogramação. Esse assunto será abordado no Cap. 7.

b. Proteção da memória

Como o *S/360* foi influenciado (em muitos aspectos) pelo projeto do Stretch, esse sistema também tem todas as características relacionadas anteriormente para ajudar multiprogramação. Mas o esquema de proteção de memória é um pouco diferente.

A memória está dividida em blocos de 2K bytes. Cada bloco tem uma "chave de proteção" de 4 bits. Com isso, cada programa tem uma "chave de armazenamento" de quatro bits. Quando o usuário tenta escrever na memória, a "chave de proteção" e a "chave de armazenamento" são comparadas. O programa pode escrever, se houver, um dos seguintes três casos: (1) as chaves compararam; (2) a "chave de proteção" é "0000" (bloco "não-trancado"); ou (3) a "chave de armazenamento" é "0000" (chave-mestra usada pelo supervisor). Cada canal de *E/S* também tem sua "chave de armazenamento". No *S/360* só existe proteção de armazenamento, mas, no *S/370*, foi também implementada uma opção para proteção contra a leitura de blocos sem autorização (*fetch protection*).

c. Relógios

A utilização da arquitetura de programação. Aumentado em sua primária (ver)

d. O PSW

No micro só guarda o C fisticados, gue

No *S/360*, o sistema tem para depois mmação. No *S/370* igual, com em

O PSW a

Para interrupções PSW. No *S/360* de máquina é sinal externo

Nome

Máscara do m

Chave

CMWP

Código da inter

ILC

CC

Máscara do p

Contador da in

c. Relocação e transcodificação dinâmica do endereço

A utilização dos registradores de base facilita a relocação e, portanto, esse aspecto da arquitetura também é considerado como uma parte de *hardware* que facilita a multiprogramação. Aqui, também deve ser citada a transcodificação dinâmica de endereço implementado em *hardware* no S/360, modelo 67, justamente para facilitar a gerência da memória primária (veja o Cap. 4).

d. O PSW e o mecanismo de interrupção

No microcomputador e no PDP-8, quando o programa está interrompido, o *hardware* só guarda o *CI* para depois poder voltar ao programa interrompido. Em sistemas mais sofisticados, guarda-se o "estado" do programa que é mais informação que o *CI*.

No S/360, foi definido um *program status word* ou *PSW*. Sempre que se tem interrupção, o sistema tem de guardar o *status* e o contador da instrução do programa interrompido, para depois recomeçar o programa exatamente onde foi interrompido sem a perda da informação. No S/360 o *PSW* consiste nessa informação (veja a Fig. 6-19). O *PSW* do S/370 é igual, com exceção do bit 12, que não é mais usado; não existe mais a opção *ASCII*.

O *PSW* ativo é, segundo a arquitetura, registrado na unidade de controle da *UCP*. Para interromper, uma substituição é efetuada, e a unidade de controle recebe um novo *PSW*. No S/360 há cinco níveis de interrupção, na seguinte ordem de prioridade: (1) erro de máquina (*machine check*); (2) erro no programa; (3) chamada do supervisor (*SVC*); (4) sinal externo; (5) entrada/saída.

Nome	Função								63
	0	11 12	15 16	31	32	33 34	35 36	39 40	
Máscara do sistema	MÁSCARA DO SISTEMA	CHAVE	CMWP	CÓDIGO DA INTERRUPÇÃO	ILC	CC	MÁSCARA DO PROGRAMA	CONTADOR DA INSTRUÇÃO	
Chave									
CMWP									
Código da interrupção									
ILC									
CC									
Máscara do programa									
Contador da instrução									

Figura 6-19. O *program status word* do S/360

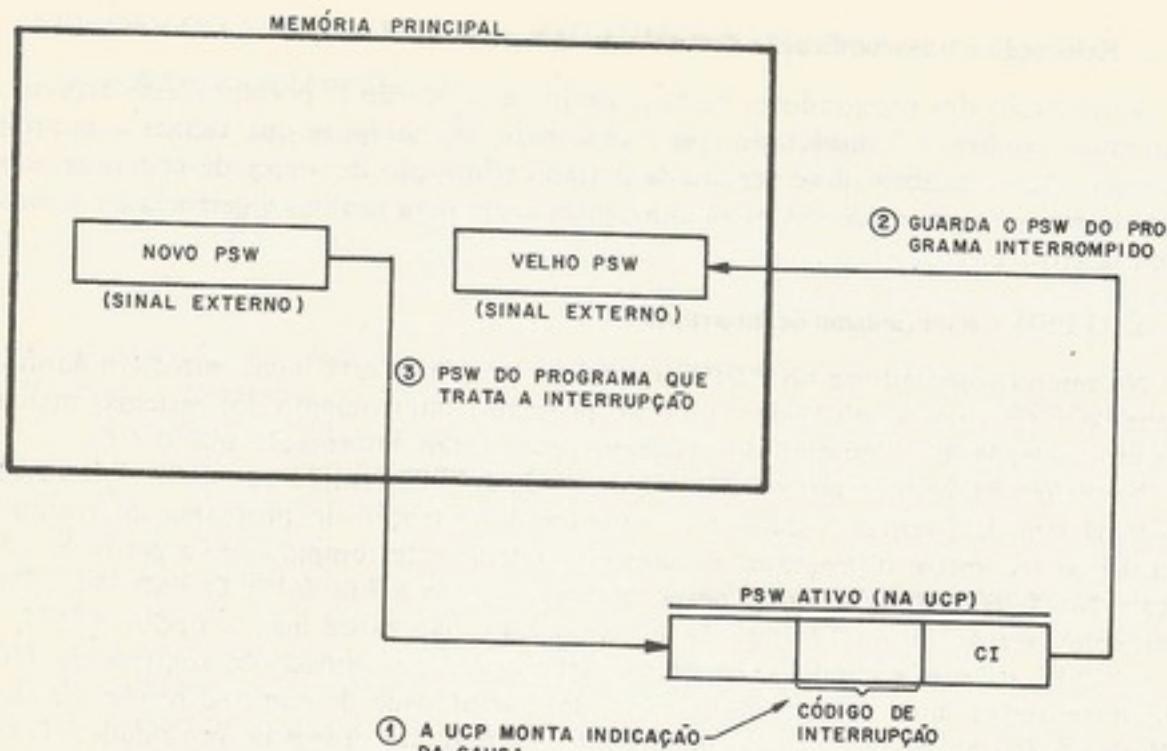


Figura 6-20. A troca de *PSW* para tratar da interrupção do sinal externo

Para cada fonte, existe uma palavra dupla de 8 bytes chamada "novo *PSW*" e uma palavra dupla chamada "velho *PSW*" (veja a Fig. 6-20). Quando houver interrupção proveniente do sinal externo⁽¹⁾, o *PSW* do programa interrompido recebe 16 bits de código da interrupção no campo dos bits 16 a 31 e, então⁽²⁾, fica armazenada no lugar "velho *PSW*" da fonte sinal externo. (O código da interrupção indica de onde veio a interrupção: o timer, a chave no painel, o sinal externo número um etc.). Para tratar da interrupção⁽³⁾, a unidade de controle carrega o registrador *PSW* ativo com o "novo *PSW*", que pertence ao sinal externo. O campo "contador da instrução" desse *PSW* indica o começo do programa tratador da interrupção. Esse programa tratador tem o cargo de guardar os *GPR* do programa interrompido. Quando o programa tratador da interrupção acaba com o tratamento da interrupção, o supervisor restaura os *GPR* do programa interrompido e carrega o registrador *PSW* ativo com o conteúdo da dupla palavra "velho *PSW*", que pertence à fonte "sinal externo", que armazenou o *PSW* quando o sinal externo foi interrompido.

e. Comentário sobre o *PSW*

É interessante fazer algumas observações sobre a interrupção do S/360, como visto no *PSW*.

Observe que o bit 14 do *PSW* indica o estado de espera (*wait state*). Aqui, a filosofia é a de que os programas, e não o computador, devem ser parados. Num sistema multiprogramado, não é desejável permitir ao programador executar uma instrução "pare" que rompa os outros programas ativos no sistema.

Observe também o campo "máscara do programa", bits 36 a 39 do *PSW*. É aqui que tratamos do problema "o que fazer com o estouro de capacidade". Para evitar o desperdício de sempre colocar instruções na linha do programa para testar o código de condição depois de operações aritméticas em que estouro pode acontecer, o programador pode optar por deixar uma situação excepcional interromper. Daí, o controle pode passar para uma rotina escrita pelo programador. Como alternativa, se o programador não está interessado nos estouros, ele pode "desligar" (*mask*) esse tipo de interrupção através da máscara.

Além das interrupções de "estouro", existem outras causas de interrupção de programa que o programador não pode desligar: endereçamento a um lugar na memória não existente

Esse ponto foi plenamente provado pelo Mills⁽⁷⁾. No entanto a arquitetura do sistema pressionava que não é necessário grande auxílio de hardware para se fazer multiprogramação.

Projetados para um sistema operacional foi custado através do PDP-8 e do S/360. Porém o contraste entre as arquiteturas de mini ou microcomputadores e de computadores da palavraria, modo de instrução, especificação do operando e variões tipos de instrução.

Neste capítulo, estudamos os assuntos comuns aos vários tipos de arquitetura: tambores como volante.

A arquitetura de sistemas exige conhecimentos sobre hardware, no sentido de saber de software, para se entenderem os circuitos lógicos, e exigir, também, conhecimentos o que pode ser feito economicamente em circuitos lógicos, e exigir, também, conhecimentos para se compreenderem os sistemas náuticos e aeroespaciais.

6.9 SUMÁRIO

Em um sistema em que vários processos estão simultaneamente utilizando os vários recursos do sistema, deve haver algum mecanismo para coordenar os recursos do tipo que responde ao "byte de fechadura" para "0000 0000".

g. A instrução "test and set" (TS)

Para facilitar a "integridade" e melhor controlar o sistema, existem dois estados (segundo supervisor) estariam carregada no registrador PSW ativo.

f. O estado "supervisor"

Facilita a manutenção da "integridade" do sistema.

Essa fiscalização do hardware sobre o programa ajuda na depuração dos programas com dígitos ilegíveis; e outros tipos de erros no programa.

No sistema; tentar escrever num lugar protegido sem a propria chave; tentar executar uma

instrução privilegiada sem estar no estado supervisor; tentar operar num campo decimal

ou num campo decimal que não é decimal.

Facilita a manutenção da "integridade" do sistema.

</div

LEITUR

Ivan Flores, C.
C. C. Foster, C.
C. G. Bell e A. N.
Small Computer

queno, ou seja, o minicomputador, é bem parecida com os sistemas da primeira geração, enquanto que a arquitetura da terceira geração foi muito influenciada pela interface entre hardware e software. Nesse sentido, apresentamos a arquitetura do S/360 como a evolução "tradicional" da arquitetura, ao contrário da filosofia da arquitetura do B-5000, que, segundo Barton⁽¹⁰⁾, representa uma simplicidade e elegância tal que merece maiores notícias e estudos. Para esse fim, também indicamos, nos lugares apropriados, as características dessa filosofia.

Usando o S/360 como a base, estudou-se a interface entre hardware e software relativamente aos controles do sistema, especialmente do programa supervisor. Os assuntos abordados incluem o PSW, o estado "supervisor" e as instruções privilegiadas, interrupção e proteção da memória.

Segundo Brooks, "em breve a arquitetura do S/360 terá sido essencialmente a da consolidação (*clean-up operation*), uma integração sistemática dos desenvolvimentos arquiteturais e experiências de duas décadas"⁽¹³⁾.

Brooks observou que provavelmente o esforço de arquitetura do futuro seria na área de entrada/saída, uma área não tão bem entendida. Esse assunto será abordado no Cap. 7.

EXERCÍCIOS

- 6-1. Por que o PDP-8 usou uma palavra tão pequena (12 bits)?
- 6-2. Discuta a possibilidade de implementar-se endereçamento indireto no microcomputador do Cap. 5.
- 6-3. No sistema IBM 1130, os três registradores de índice foram implementados como palavras 1, 2 e 3 da memória. Quais são as vantagens e as desvantagens?
- 6-4. Existem duas filosofias de implementação do desvio de sub-rotina. Explique. ("Dica": o LGP-30 e o IBM 704.) Quais são as vantagens da filosofia do 704?
- 6-5. No PDP-8, como o programador pode tratar das interrupções múltiplas?
- 6-6. Por que a função de "test and set" do S/360 não pode ser implementada pelo programador através de instruções de carregar o acumulador, desviar e armazenar o acumulador?

BIBLIOGRAFIA

- (1) "Burroughs B-5500 Information Processing Systems", Reference Manual, 1964
- (2) C. B. Carlson, "The Mechanization of a Push-Down Stack", Proc. AFIPS, Vol. 23, pp. 243-250, (FJCC, 1963)
- (3) R. V. Bock, "An Interrupt Control for the B-5000 Data Processor System", Proc. AFIPS, Vol. 23 (FJCC 1963), pp. 229-241
- (4) Jane G. Jodeit, "Storage Organization in Programming Systems", Comm. ACM, Vol. 11, pp. 741-746, novembro de 1968
- (5) R. K. Richards, *Electronic Digital Systems*, John Wiley, New York, 1966
- (6) Hao Wang, "A Variant to Turing's Theory of Computing Machines", Journal of the ACM, Vol. 4, pp. 63-92, 1957
- (7) David L. Mills, "Multiprogramming in a Small Systems Environment", Second Symposium on Operating Systems Principles (20 a 22 de outubro de 1969, Princeton University), Publ. ACM, pp. 112-119
- (8) C. L. Hamblin, "Translation to and from Polish Notation", The Computer Journal, outubro de 1962
- (9) G. M. Davis, "The English Electric KDF-9 Computer System", Computer Bulletin, Vol. 4, p. 119, 1960
- (10) R. S. Barton, "Ideas for Computer Systems Organization: A Personal Survey", Software Engineering (Julius Tou, editor), pp. 7-66
- (11) IBM Systems Journal, Vol. 3, exemplares números 2 e 3, 1964
- (12) W. Buchholz (editor), *Planning a Computer System, Project Stretch*, McGraw-Hill, New York, 1962
- (13) F. P. Brooks, Jr., "The Future of Computer Architecture", IFIPS Congress, pp. 87-91, New York, 1962
- (14) Jules Mersel, "Program Interruption on the Univac Scientific Computer", Proc. WJCC, p. 52, fevereiro de 1956

LEITURAS COMPLEMENTARES

- Ivan Flores, *Computer Organization*, Prentice-Hall, 1969
C. C. Foster, *Computer Architecture*, Van Nostrand Reinhold, 1970
C. G. Bell e A. Newell, *Computer Structures: Readings and Examples*, McGraw-Hill, 1971
Small Computer Handbook, Digital Equipment Corp. (sobre o PDP-8), edição 1970

capítulo 7

ENTRADA/SAÍDA

7.1 INTRODUÇÃO

Vimos, no Cap. 4, que existem dois tipos de dispositivos para entrada/saída (E/S) ao sistema: a E/S "verdadeira" (para comunicação entre o Homem e a máquina) e a E/S de armazenamento secundário (arquivos a longo prazo). Esses dois tipos de E/S, com poucas exceções, utilizam o mesmo esquema conforme a arquitetura de E/S do sistema.

Assuntos importantes relativos à entrada/saída de computadores são a interface com os dispositivos e o controle das operações de entrada/saída. Quanto ao controle, destaca-se a técnica de interrupção do programa ativo na unidade central de processamento (UCP). A memória primária e o armazenamento secundário foram explicados no Cap. 4 e aqui pretendemos mostrar a ligação entre eles, ou seja, a arquitetura dos esquemas de E/S.

Nos anos recentes, os circuitos lógicos, a memória principal e a unidade central de processamento têm sido aumentadas em velocidade muito mais do que a velocidade de transferências de dados e o tempo de acesso dos dispositivos periféricos de E/S. Então, orienta-se a arquitetura de E/S de sistemas de médio e grande porte com técnicas de minimização do tempo ocioso da UCP durante a espera de dados dos periféricos (veja a Fig. 7-1). Por outro lado, na arquitetura de minicomputadores, o arquiteto explora técnicas de economia de hardware.

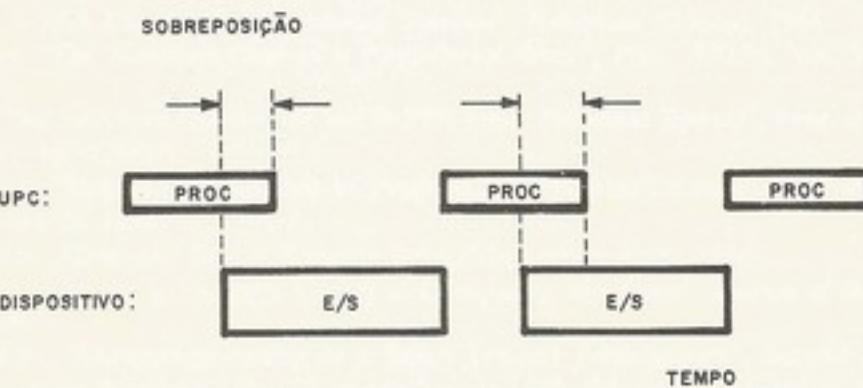


Figura 7-1. Sobreposição de processamento com E/S

O subprocesso de *entrada* é a transferência de dados de um veículo de informação à memória e o subprocesso de *saída* é a transferência de informações da memória primária para um dispositivo que as aceita. O processo de E/S pode ser subdividido (no tempo) em três fases importantes; o *início*, a *transferência de dados* e a *finalização*.

Na fase de *início*, a UCP e o dispositivo se reconhecem, ou seja, o dispositivo é selecionado. Assim, a UCP notifica o dispositivo que vai transferir dados. Nesse ponto, a UCP pode verificar o estado (*status*) do dispositivo porque, se o dispositivo não estiver ligado, ou se estiver ocupado, a segunda fase não poderá prosseguir. Na segunda fase, o dispositivo, geralmente, transfere dados ou, em alguns casos, executa um comando como posicionamento

do veículo. Na terceira fase, a finalização, trata-se da verificação de que tudo foi executado sem falhas. Alguns sistemas, os menos sofisticados, não têm essa fase. E outros sistemas, orientados à confiabilidade, utilizam técnicas de *recovery* e *retry*. No processo de *recovery*, o sistema tenta recuperar as informações possivelmente perdidas e, no mínimo, tenta isolar o programa afetado pela falha para poder continuar com o processamento dos outros programas. A técnica de *retry* força o reinício da situação antes da falha, e uma nova tentativa de execução do comando de *E/S* que sofreu a falha.

7.2 DISPOSITIVOS E SUAS INTERFACES

a. Funções dos periféricos

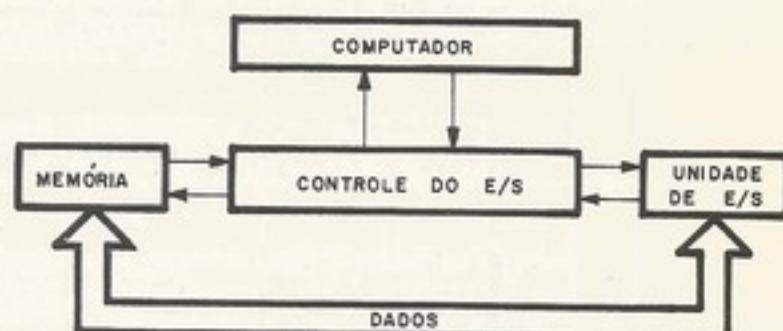
A maioria dos periféricos (dispositivos de *E/S*) tem duas ou três funções principais. A primeira é controlar o veículo de transporte da informação (fita, papel, disco etc.), a segunda, ler a informação e montar palavras, e a terceira, desmontar palavras e escrever informação. Além disso, os periféricos têm o problema "o que fazer quando há falha?". Esse assunto pertence à filosofia da confiabilidade do sistema.

No projeto e controle dos periféricos e suas interfaces, temos de levar em conta os seguintes detalhes: as operações mecânicas do dispositivo; as funções do dispositivo; o controle do equipamento; a integração do periférico no sistema; o tratamento das falhas.

b. A essência do controle dos periféricos

A essência do controle de dispositivos é mostrada na Fig. 7-2, onde fica evidente que a parte importante, a razão de ser, de *E/S* é a transferência de dados entre a memória primária e as unidades periféricas. Em mais detalhes, observar Fig. 7-3.

Figura 7-2. O controle de *E/S*



A operação começa com um comando ("ler, escrever") para transferir uma certa quantidade de palavras à *MP* (bloco 1), começando com um certo endereço (bloco 2). Um programa de microoperações (bloco 3) executa os comandos, controlando o periférico (blocos 4 e 5).

No fluxo, acontece a montagem e desmontagem de palavras (bloco 6), pois a palavra da *MP* não é necessariamente do mesmo tamanho que o dispositivo usa. Para cada palavra transferida, o endereço (bloco 7) e o contador (bloco 1) têm de ser atualizados. No caso mais simples, como veremos, a *E/S* trata blocos do tamanho de uma só palavra.

c. A ligação entre a UCP e o dispositivo

Surge o problema de como fazer a comutação do dispositivo com a *MP*. Em sistemas da primeira geração, a *E/S* estava ligada com os registradores centrais da *UCP*, e os "comandos" de *E/S* eram instruções do programa. O esquema era "radial" no sentido de que cada dispositivo tinha sua própria interface (veja a Fig. 7-4). Nesse esquema, o processamento pára enquanto o sistema espera pelo término da operação de *E/S*, e não tem sobreposição de *E/S* com a *UCP*.

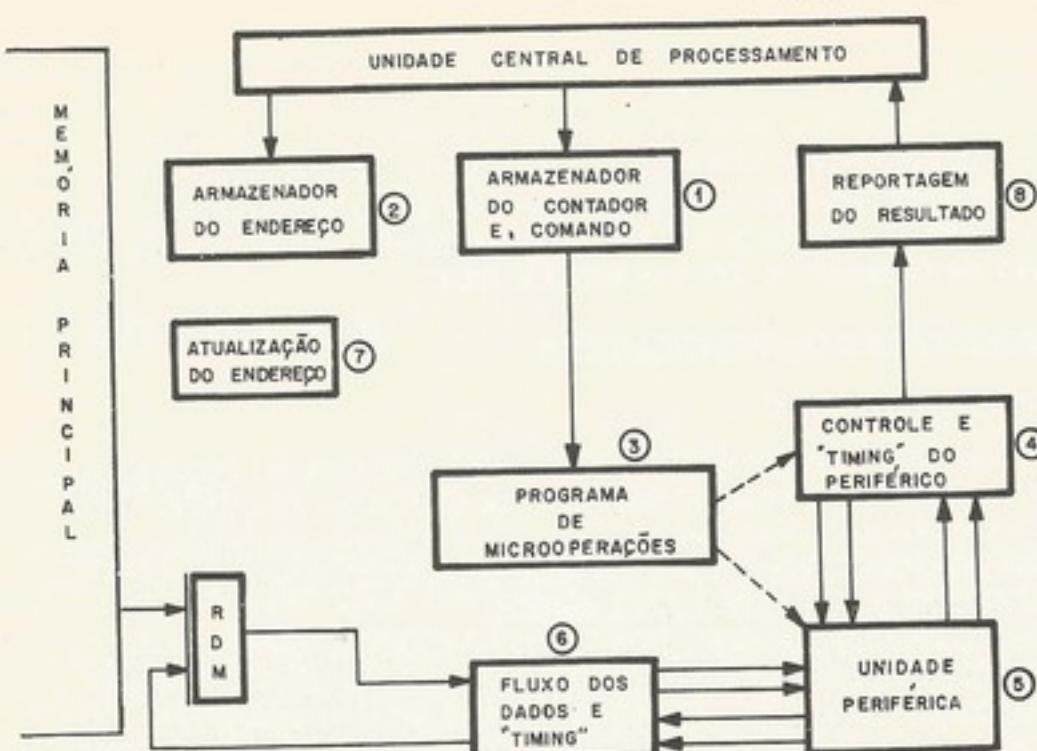


Figura 7-3. O funcionamento de E/S em blocos

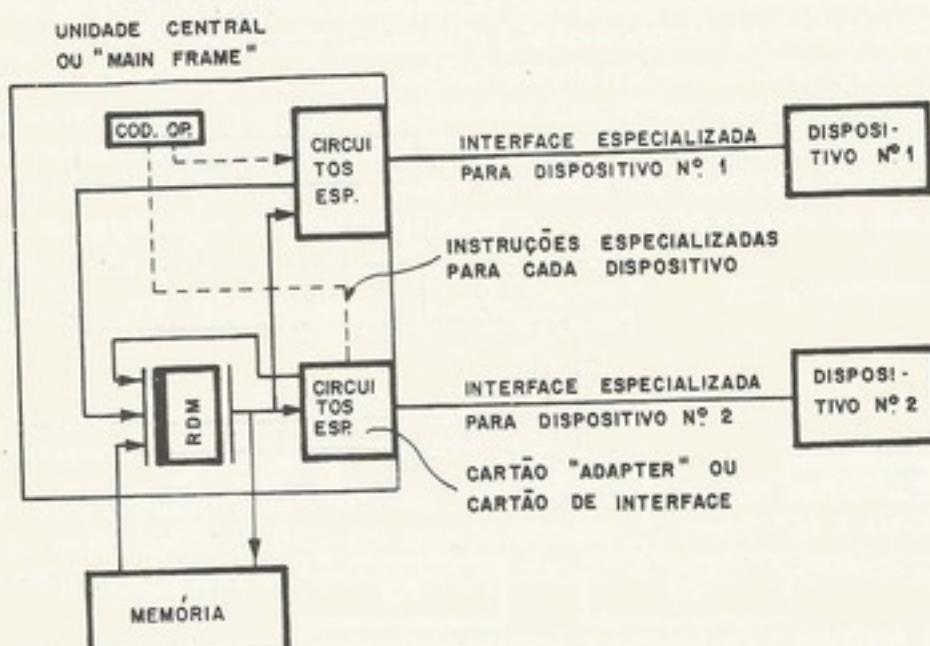


Figura 7-4. Sistema "radial" com interfaces especializadas

As três fases de E/S são todas em software, com poucas sub-rotinas em comum entre o dispositivo 1 e o dispositivo 2. Os sinais de interfaces são particulares ao próprio dispositivo, isto é, a interface é especializada.

Em alguns sistemas desse tipo, dentro da UCP, é projetada uma via (barramento) de E/S padronizada, mas com circuitos especiais para o controle dos dispositivos ainda colados no *main frame* da UCP, isto é, os circuitos especiais são "empacotados" junto com os circuitos da UCP. Um sistema desse tipo foi descrito no Cap. 5.

Outro tipo de interface é chamada de padronizada (*standard interface*), essa técnica permite a colocação de periféricos em série, evitando a necessidade de cartões especializados de interface de E/S na unidade central (veja a Fig. 7-5).

entraida/saida

Observar lado da UCP pinos no lado através de sua interface para e que, quando padronizada, é face padronizada. Assim, o design outras exigências.

Para fins de periféricos não exemplo, pode (tape drives, perfuradora de c.

Quando duas ou mais "multiplex".

Em instalações transitoriamente usar a interface nesse caso, a

d. Os sistemas

Os cartões
(2) extratos de
(sinais de sincronismo como real ou
"ocupado", etc.)

A descrição de função (junto do endereço) é o endereço da memória. Em outros sistemas forma o dispositivo trapezoidal (pode ser



Figura 7-5. Dispositivos ligados à UCP com interface padronizada

Observar que, com um cabo de interface padronizado assim, uma única interface no lado da UCP pode servir a todos os dispositivos que estão ligados ao cabo (o que economiza pinos no lado da UCP). Quando isso é feito, a parte na UCP distingue entre dispositivos através de um endereço, isto é, cada dispositivo tem seu próprio endereço. A vantagem da interface padronizada é que os dispositivos podem ser projetados independentes da UCP, e que, arquiteturalmente, os dispositivos podem servir a qualquer UCP que utiliza a interface padronizada, dando uma certa flexibilidade ao usuário do sistema. A desvantagem da interface padronizada é que cada dispositivo necessita de circuitos especializados juntos a ele. Assim, o dispositivo precisa hospedar circuitos lógicos e suas fontes de tensão e, possivelmente, outras exigências como, por exemplo, tomar providências contra ruidos.

Para fins de economia, uma possibilidade é a de juntar os circuitos lógicos de vários periféricos numa única unidade, chamada *IO control unit* (unidade de controle de E/S). Por exemplo, podemos ter uma única unidade de controle para dez unidades de fita magnética (*tape drives*). Também, foi feita uma única unidade de controle para uma leitora, uma perfuradora de cartões e para impressora.

Quando vários dispositivos compartilham de uma interface assim, é possível entrelaçar duas ou mais mensagens em tempo, como mostra a Fig. 7-6. Essa técnica chama-se “multiplex”.

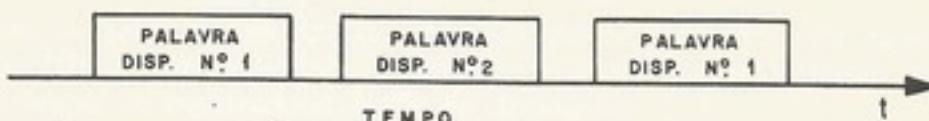


Figura 7-6. Aparecimento de palavras multiplexadas na interface

Em interfaces não-multiplexadas, é necessário o término das três fases de E/S (início, transferência de um bloco de palavras e finalização) para que um outro dispositivo possa usar a interface. Em sistemas pequenos, o “bloco” é freqüentemente de uma palavra só e, nesse caso, a “multiplexação” não é uma solução adequada.

d. Os sinais da interface

Os cabos de interface podem ter até sete tipos de sinais (Fig. 7-7): (1) saída de dados; (2) entrada de dados; (3) endereço do dispositivo (caso interface padronizada); (4) timing (sinais de sincronização ou coordenação, síncrono ou assíncrono); (5) controle (comandos como *read backwards*, *write etc.*); (6) status (indicadores da situação ou *flags*, como “pronto”, “ocupado”, “erro”, etc.); e (7) interrupção.

A decodificação do endereço no caso da interface padronizada pode ser feita através de fiação (*jumpers*) no dispositivo. Assim, uma mudança dos fios efetua uma modificação do endereço reconhecido. Em alguns sistemas, com a via de E/S junto à UCP (sistema radial), o endereço do dispositivo é dado pela localização do cartão de interface (o HP 2000 é assim). Em outros sistemas, os endereços dos dispositivos são designados (pré-determinados) conforme o dispositivo, por exemplo, no PDP-8 os endereços $(50)_8$, $(51)_8$ e $(52)_8$ já indicam um traçador (*plotter*).

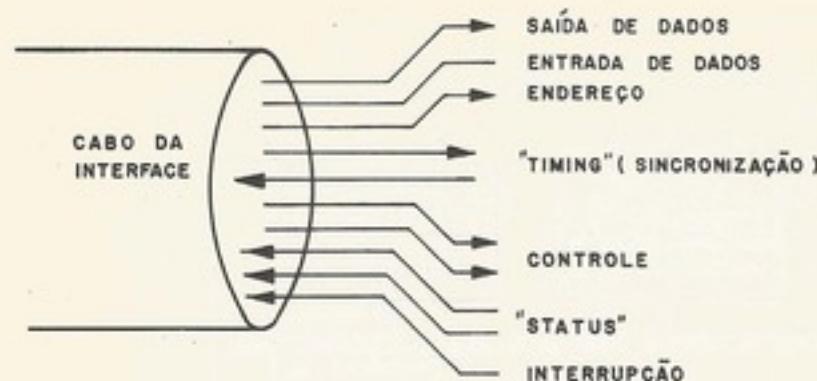
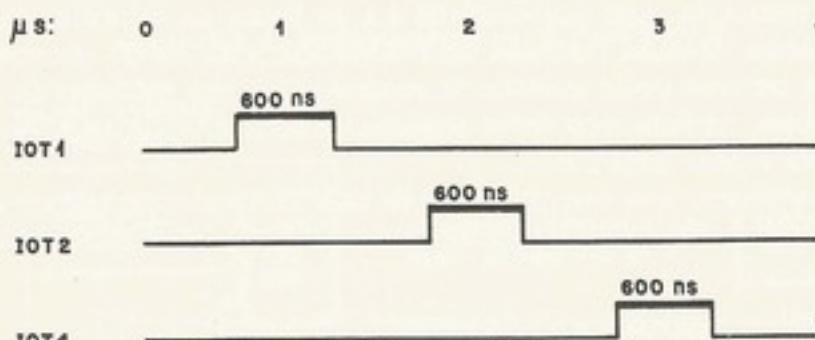


Figura 7-7. Sinais de interface

e. O "timing" da via de E/S

e.1. Caso sincrono

Quando a via de E/S é sincronizada, os sinais de *timing* normalmente são provenientes do relógio central da UCP e, portanto, o ciclo da via é vinculado ao ciclo da máquina. No caso do PDP-8, para a instrução E/S, a UCP gera três pulsos de *timing* chamados *IOT1*, *IOT2* e *IOT4*. A ocorrência relativa no tempo desses pulsos está mostrada na Fig. 7-8. Esses sinais *IOT* passam através de um *gate* de controle para, depois, aparecerem como sinais *IOP*.

Figura 7-8. O *timing* da via de E/S do PDP-8

A instrução de E/S do PDP-8, que reside no RI, tem um campo de bits (bits 9, 10 e 11) que seleciona quais pulsos *IOP* vão aparecer na via E/S (Fig. 7-9). Esses pulsos *IOP* são usados no dispositivo para controlar as operações. Por exemplo, no caso do traçador do PDP-8, com endereço $(52)_8$ para o dispositivo, os pulsos têm os sentidos mostrados na Tab. 7-1.

Tabela 7-1. Os pulsos da via de E/S do PDP-8

Pulso	Comando
<i>IOP1</i>	<i>Pen left</i>
<i>IOP2</i>	<i>Drum up</i>
<i>IOP4</i>	<i>Pen down</i>

Daremos um outro exemplo mostrando como o programador pode interrogar o dispositivo. O teclado tem endereço $(03)_8$. Com pulso *IOP1*, a operação é "salto no flag do teclado" (*skip on keyboard flag*). A interface tem um fio chamado *skip* que alimenta a UCP. O funcionamento dessa operação está na Fig. 7-10.

A importância desse tipo de salto no PDP-8 é que o pulso de 600 ns *IOP-1* é usado como sinal de controle para colocar (ou comutar) o estado do *flip-flop flap* na via *skip*, por 600 ns. Mais ou menos 300 ns depois da geração do sinal *IOP-1*, a UCP do PDP-8 "tira uma amostra" da via *skip*. Dependendo do resultado da amostra, a UCP salta ou não.

entrada/saída

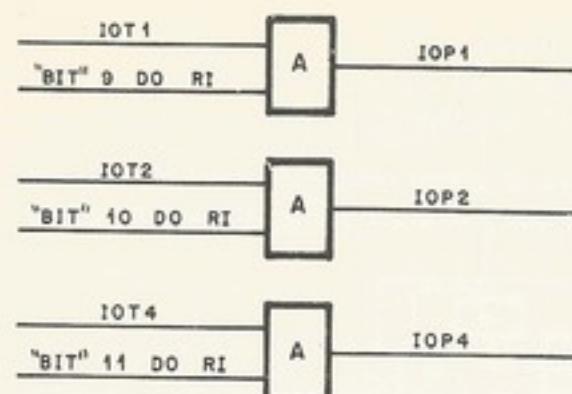


Figura 7-9. Gates de controle que selecionam os pulsos IOP

são provenientes da máquina. No chamados *IOT1*, *IOT2* e *IOT4*. Esses são sinalados *IOP*.

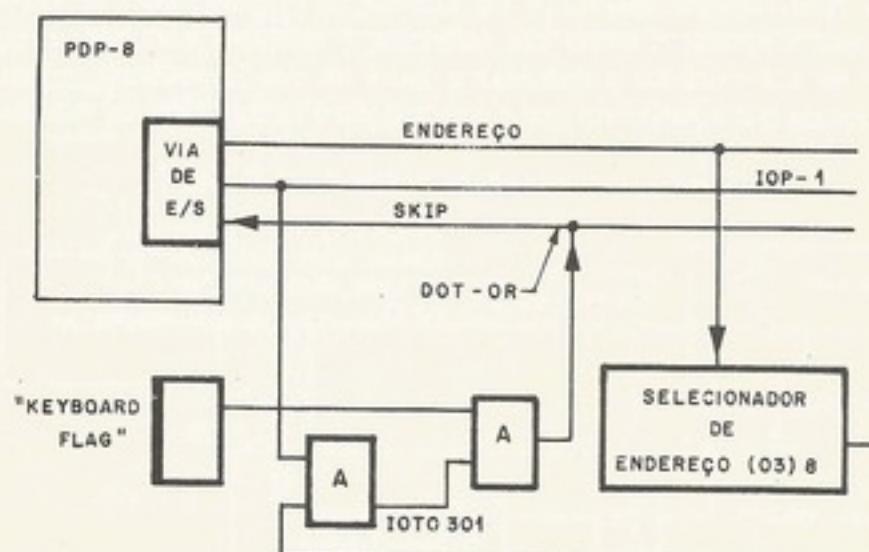


Figura 7-10. O skip on flag do PDP-8

Em geral, os três pulsos de sincronização no *PDP-8* são usados para os seguintes fins: *IOP-1*, saltos; *IOP-2*, limpar os *flip-flops* de *status* ou *flags*, ou o acumulador; *IOP-4*, copiar, carregar ou limpar os registradores *buffer* do dispositivo.

e.2. Caso assíncrono

Sistemas de via de *E/S* com regulagem assíncrona não usam o ciclo de máquina, nem largura pré-determinada dos pulsos. Normalmente a única especificação é que os sinais de controle mantêm-se estáveis por um prazo mínimo, ou que eles respondem dentro de um certo prazo. A vantagem desse tipo de interface é a sua flexibilidade. O esquema, se padronizado, permite a colocação de dispositivos de várias velocidades na via. Também um dispositivo projetado para a via pode trabalhar com várias *UCP* (mas não no mesmo tempo), desde que eles usem a mesma interface padronizada.

Os sinais de controle da via têm significados como "pedir entrada", "entrada reconhecida" e "função feita". Em geral, cada sinal do "controlador" (*master*) da via provoca uma resposta de reconhecimento do controlado. O *PDP-11* usa uma técnica de *timing* assíncrono chamada "univia", que explicaremos em seguida. Para os usuários da via, um é o "mestre" e o outro é o "escravo". As transferências de dados são efetuadas das seguintes maneiras:

- o "mestre" coloca o endereço do dispositivo na subvia de endereço, e coloca a subvia de comando no estado *dati* (*data-in*, ou seja, "entrada");
- o "mestre" ativa (coloca no estado "1") o sinal *MSYN* (*master synchronizer*);
- o "escravo" (dispositivo), reconhecendo o seu endereço e o comando *dati*, na borda de ataque de *MSYN*, coloca a palavra na subvia *data*.
- o "escravo" ativa o sinal *SSYN* (*slave synchronizer*);

- o "mestre", sentindo o *SSYN*, copia a subvia *data*;
- o "mestre" desativa (coloca no estado 0) o sinal *MSYN*;
- o "escravo", sentindo essa resposta do "mestre", desativa *SSYN*.

A desvantagem desse tipo de via é que pode gastar muito tempo em *hand-shaking*, ou seja, reconhecendo respostas. Assim, esse tipo de via não parece ser vantajoso para a ligação entre a *UCP* e a *MP*.

f. Características tecnológicas da via

Suporemos aqui estar a via de *E/S* fora da *UCP*. Com distâncias compridas e bordas de ataque e fuga relativamente velozes, a via adquire as características de linhas de transmissão. Assim, para diminuir o efeito das reflexões, é costume controlar o comprimento dos tocos (*stubs*) ou até evitá-los e terminar a linha na sua impedância característica. As ligações unidirecionais são aquelas em que a informação é transmitida em uma única direção (Fig. 7-11).

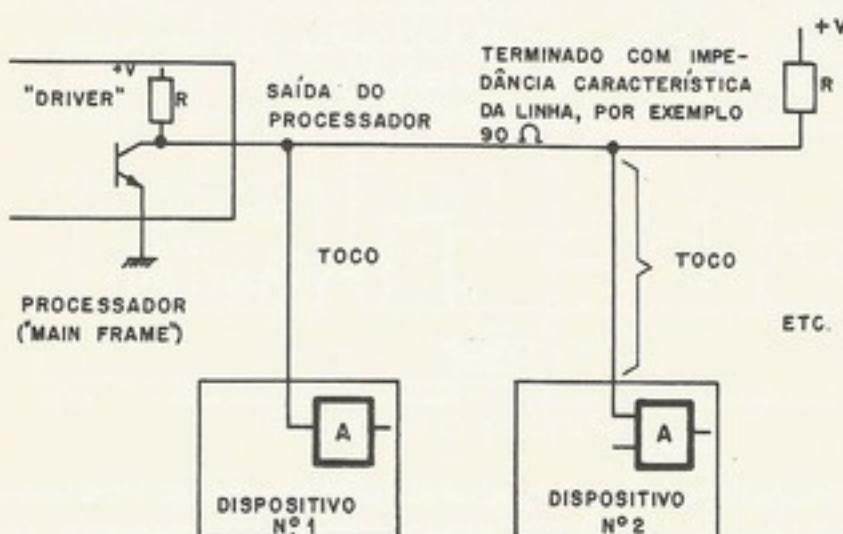


Figura 7-11. Via unidirecional com tocos

Na maioria dos cabos de interface fora do "main frame", em vez de se usarem tocos, usam-se terminações em cada dispositivo, como mostra a Fig. 7-12.

Algumas interfaces podem aproveitar a técnica de *DOT-OR* para enviar sinais à *UCP*, como no caso da via. Quando isso não é prático, é sempre possível formar uma cadeia de blocos lógicos tipo *OR*, como mostra a Fig. 7-13.

É também viável usarem-se ligações bidirecionais como na univia do *PDP-11*. Os sinais de controle e prioridade determinam a direção em que as informações fluem (Fig. 7-14).

7.3 A INTERFACE DO PROGRAMA E O DISPOSITIVO

Ressaltamos que o programa em execução na memória principal é o volante do sistema. Então, de que maneira o programa consegue se comunicar com os dispositivos? Em alguns sistemas, o programa usa instruções para o controle direto do dispositivo e, em outros sistemas, existe um canal de entrada/saída que age como intermediário. Alguns minicomputadores têm uma opção para dispositivos velozes (como o disco) chamado *direct memory access* (acesso direto à memória), que "rouba" ciclos da memória (*cycle-steal*). Um sistema de grande porte, o *CDC 6600*, usa um processador periférico para controlar a entrada/saída.

As técnicas de interface entre o programa e o dispositivo são mostradas na Tab. 7-2 e serão explicadas a seguir.

o "junk-shaking", ou
uso para a ligação

empilhadas e bordas
de linhas de trans-
istor o comprimento
característica. As
uma única direção

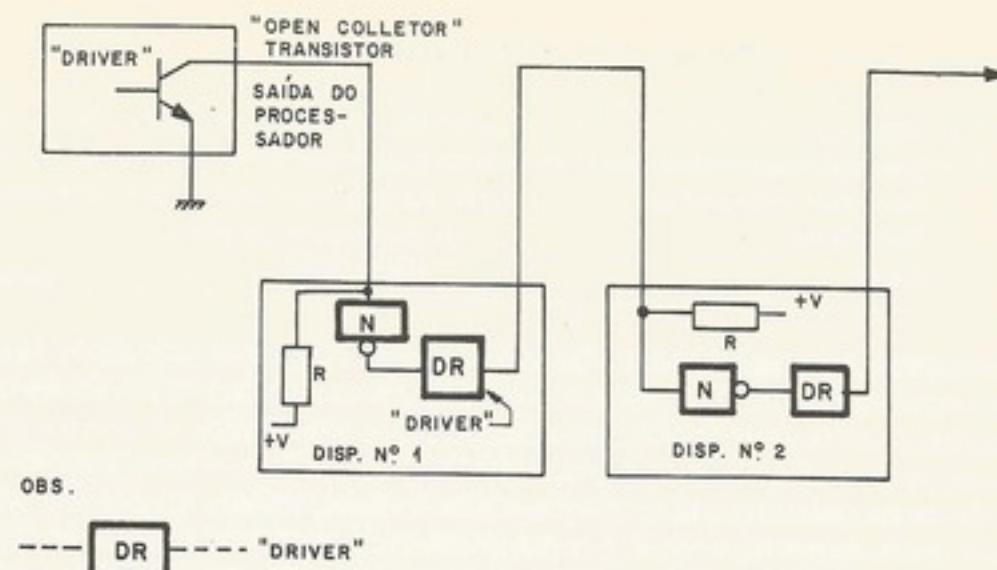


Figura 7-12. Interface unidirecional sem tocos

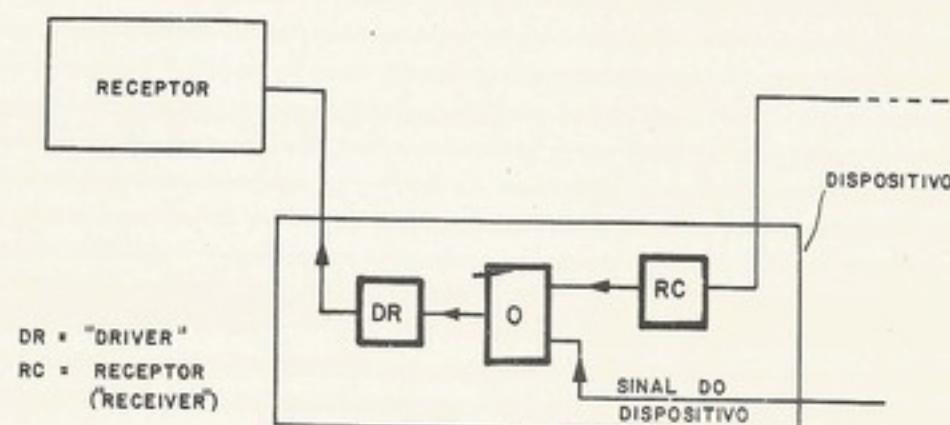


Figura 7-13. Sinais de entrada da UCP numa interface serial que não usa a técnica de DOT-OR

a. Programado

Na primeira geração, quando os processadores eram lentos, havia instruções para o controle direto dos dispositivos. Essa técnica pode ser chamada E/S "imediata". Assim, a UCP esperava enquanto o dispositivo funcionava. Por exemplo, no sistema IBM 1401, existe a instrução "ler cartão". As três fases (início, transferência de dados e término) acontecem antes da execução da instrução seguinte. No caso do 1401, existe uma região definida na memória principal de oitenta caracteres, para receber o conteúdo do cartão. Uma van-

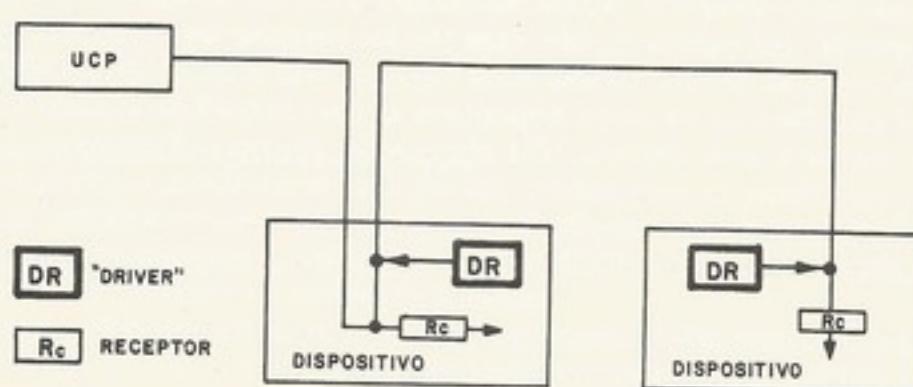


Figura 7-14. Interface bidirecional

Tabela 7-2. Controle de entrada/saída

a) Programado	b) Direct memory access (DMA)
Instrução simples (imediata)	(Cycle-steal)
Sense-loop (skip-on-flag)	
Interrupção	
c) Canal	d) Processador periférico (CDC 6600)

tagem desse sistema, embora não muito flexível, é que o programador não tinha de se preocupar com a transferência de cada byte; o hardware se encarregava da atualização do endereço para a próxima palavra e da transferência das palavras à memória.

O método do *sense loop* é um *loop* no programa que esperava o término de E/S. O *loop* é feito através de um salto que interroga, via o *skip bus* da interface, um *flip-flop* no dispositivo. Para o início da operação de E/S, o programa pode interrogar o *status* do dispositivo para verificar se o dispositivo está pronto e não ocupado. Daí, pode-se usar o *sense loop* para transferir cada palavra. Para esse fim, o programa tem um contador do número de palavras a ser transferido, que ele decrementa a cada transferência. Quando o *status* indica que o dispositivo está pronto, para transferir a próxima palavra, o programa entra numa sub-rotina de uns vinte ciclos para transferir dados, atualizar o endereço e a conta, e verificar se é para terminar (isto é, se a conta está zero). Para a fase de finalização, o programa pode verificar que o *status* não indica erro algum.

Quando a UCP está num *loop* da E/S, alegamos que isso representa um tempo ocioso no que, talvez, um programa poderia aproveitar a UCP. Se o programa esperando a E/S pudesse entrar num estado de "espera" enquanto outro programa (ou até o mesmo programa) utilizasse a UCP, então poderíamos diminuir o tempo gasto pela UCP esperando num *sense loop* na E/S.

A técnica de interrupção evita o tempo ocioso na UCP, se há outra coisa a fazer. O programa que iniciou a E/S pode fazer outra coisa enquanto espera a interrupção. Assim, recebendo e verificando a causa da interrupção, o programa desvia para uma sub-rotina que desempenha a fase 2; isto é, transferir dados e atualizar o endereço, o contador, e desviar, se for a última palavra a ser transferida.

b. Acesso direto à memória (DMA)

Em sistemas que dispõem de acesso direto à memória, inicio e a finalização da transferência de um bloco entre a memória e o dispositivo é normalmente efetuada através da via de E/S, mas a transferência das palavras, uma por uma é feita pelo DMA. Essa transferência usa a técnica de *cycle-steal* (ciclo furtado) da memória; isto é, o hardware suspende a execução da atual instrução enquanto a interface DMA transfere a palavra. A técnica de *cycle-steal* normalmente implica a parada do relógio do fluxo de dados da UCP. Assim, a UCP nem "percebe" que perdeu o ciclo. Essa técnica necessita mais circuitos lógicos que o *sense loop*, por causa da criação de uma nova interface com a memória principal (o DMA normalmente só trabalha com um único dispositivo) e, pela necessidade de parar e recomeçar a UCP. Mas a técnica é mais eficiente, pois só necessita de um a cinco ciclos de máquina para efetuar a transferência de uma palavra, em vez das vinte e tantas necessitadas pela técnica de interrupção ao desviar o processamento para uma rotina tratadora. A vantagem é que um dispositivo como um disco pode transferir um bloco de palavras sem interrupção do programa.

Depois que a interface DMA transferir a última palavra, ela interromperá o programa da UCP para notificar o programa do término de sua tarefa.

A interface DMA, no entanto, tem de guardar a informação "endereço" e "conta" para saber onde colocar (ou retirar) a próxima palavra e para saber quando parar a operação.

Algumas interfaces têm seus próprios registradores para isso e alguns casos, como o *PDP-8*, utilizam lugares na memória para isso. Então, no caso do *PDP-8*, o *cycle-steal* rouba pelo menos três ciclos: um para guardar a informação, e dois para atualizar o endereço e a conta. Em qualquer caso, a interface *DMA* representa um compromisso em que a interferência da *E/S* é diminuída, mas o *hardware* é mais caro.

c. Canal

O canal é uma generalização do *DMA* (embora o canal tenha vindo primeiro), porque ele pode trabalhar com mais de um dispositivo. O canal geralmente se comunica com seus dispositivos através de uma interface padronizada (como no *S/360*) ou com *crossbar switch* (como no *B-5500*) (veja o Cap. 4). Enquanto o *DMA* só faz a segunda fase (transferência) das fases de *E/S*, o canal executa as três fases. O próprio canal guarda e atualiza o "endereço" e a "conta" dos blocos a serem transferidos, também dispõe de registradores *buffer* para montar e desmontar palavras do tamanho da memória ao tamanho da interface padronizada.

No caso do sistema *IBM S/360*, o canal pode ser visualizado como um "computadorzinho" que executa um programa de canal que consta de palavras de controle. Desde que a *UCP* tenha seu contador de instrução (*CI*) indicando a instrução a ser executada, o canal tem um contador que indica a próxima palavra de controle do canal (*CCW*).

Para efetuar uma programação de *E/S*, o programa da *UCP* normalmente monta um programa de *CCW*, na memória, que descreve as etapas da operação desejada. A *UCP* dá partida ao canal, indicando o endereço do programa de canal. O canal e a *UCP* continuam trabalhando, operando em paralelo. Quando o canal acaba de executar esse programa, ele interrompe a *UCP*. Estudaremos logo em seguida esses assuntos em mais detalhes.

d. Processador periférico (*CDC 6600*)

Do conceito de canal, foi observado que o seu comportamento é o de um computadorzinho retirando seus comandos da mesma memória que a *UCP*. No caso do sistema *CDC 6600*, cada "canal", ou seja, *processador periférico*, tem 4K palavras de sua própria memória para armazenar o "programa de canal" e para agir como um *buffer* entre a *MP* da *UCP* e o dispositivo. O processador periférico é, então, uma generalização da idéia de canal. Mas, também, o processador periférico no *CDC 6600* pode fazer mais de um canal. Notamos especialmente o fato de o programa de controle do sistema, ou seja, sistema operacional, rodar num processador periférico.

Como um último comentário sobre o processador periférico, diremos que ele não resolve em si o problema da interface e controle dos dispositivos, mas sim que ele mesmo tem de usar uma combinação das técnicas anteriormente explicadas.

7.4 INTERRUPÇÃO

a. Classes e tratamento geral de interrupção

No Cap. 5, explicamos o esquema de *interrupção* de um minicomputador. É uma situação em que a próxima instrução a ser executada pela *UCP* é determinada pelo *hardware* do sistema e não pelo programa atualmente rodando.

Também no Cap. 6, apresentamos a interrupção no *S/360*, que faz a troca de *PSW* e admite um código de interrupção para ajudar na determinação da causa. Uma vez descoberta a técnica de interrupção, surgiram muitas aplicações para aproveitá-la. A Fig. 7-15 mostra as classes de interrupções existentes nos sistemas atuais.

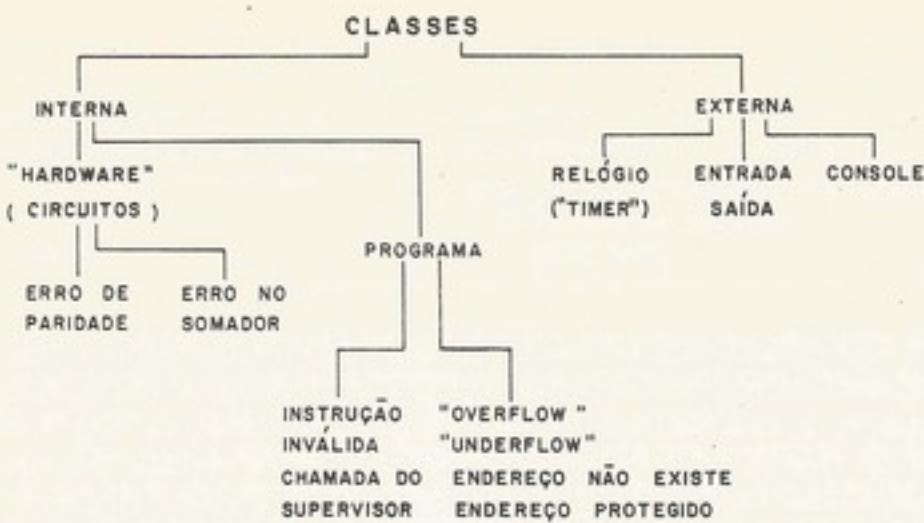


Figura 7-15. Árvore de classificação das interrupções, com exemplos

Havendo interrupção, cabe ao sistema tratá-la. Em geral, o sistema obedece a seguinte seqüência de atividades:

- suspende o programa corrente;
- guarda o *status* ou estado (*CI*, registradores centrais);
- trata com a causa da interrupção;
- atualiza o estado do sistema e determina o programa seguinte a rodar;
- restaura o *status* do programa a ser rodado.

b. Interrupção de nível simples

O tipo de interrupção do minicomputador do Cap. 5 é de nível simples (*single-level*), onde não se permite interrupção sobre interrupção. O PDP-8 também tem interrupção de nível simples. Assim, o computador tem de tratar a interrupção atual antes que uma outra interrupção ocorra, mesmo que essa última seja de prioridade mais alta. Nesse caso, o sistema tem geralmente, apenas uma fonte de interrupção: um fio da via de *E/S* (veja a Fig. 7-16). Qualquer dispositivo querendo interromper provoca a interrupção através do mesmo fio.

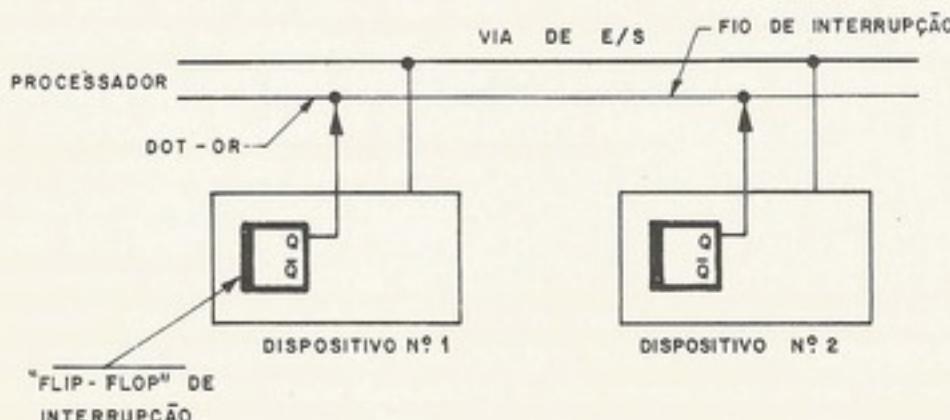


Figura 7-16. Exemplo de interrupção de nível simples

Essa interrupção "desvia" o processador para um único programa, que tem de descobrir a causa da interrupção. Isso é feito por programação, interrogando cada dispositivo usando a técnica de saltar no *status*. O programa endereça cada dispositivo à via de *E/S*, e desvia se o "flip-flop de interrupção" está no estado "1". No caso em que dois dispositivos estão interrompendo ao mesmo tempo, primeiro o dispositivo tratado é aquele que é interrogado primeiro.

entrada/saida

A "prioridade" há muitos dias de overman.

No PDP-8 de sub-sistema "0" e a UCP programa ativa cartão perfumado significa que a *on data resultante* existe o flip-flop.

c. Interrupção

Um método em grupos ou cada fonte tem um programa tratado pré-determinado, e para a interrupção as fontes de interrupção basta uma resolução.

Em sistemas de níveis organizados

O sistema malmente o mais "baixo" é a fonte de nível.

No caso medidas. Com usuário juntas (1) Os pedidos de interrupção de cutado só se partilhar de

d. Hardware

Pretendendo temas de nível.

Na Fig. 7-17 uma função genérica

A "prioridade" é implementada por *software*. Essa técnica é bastante lenta e, quando há muitos dispositivos, alguns de alta velocidade (como fita magnética), tem a possibilidade de *overrun*.

No PDP-8, quando um dispositivo interrompe, o processamento executa um desvio de sub-rotina (*subroutine jump*) (*JMS*) para localização "0". Isso guarda o *CI* na localização "0" e a *UCP* executa a primeira instrução do programa atendedor na localização "1". O programa atendedor é responsável pela determinação da causa. Por exemplo, a leitora de cartão perfurado interrompe sempre que o *flip-flop data ready* estiver disparado. Isso significa que a informação da coluna do cartão está pronta. A rotina usa o comando 6631 (*skip on data ready*) para determinar esse fato. Para indicar que o cartão saiu da estação de leitura, existe o *flip-flop card done flag*.

c. Interrupção de níveis múltiplos

Um método um pouco mais sofisticado para determinar a causa é a partição das causas em grupos ou conjuntos parecidos e o fornecimento de uma "fonte" para cada grupo. Agora cada fonte tem o seu endereço especial, que indica a primeira instrução do seu próprio programador. Por exemplo, no S/360, existem cinco fontes. Para cada fonte há um lugar pré-determinado na memória principal para guardar o *status* (*PSW*) do programa interrompido, e para fornecer o *PSW* do programa que trata da interrupção. Quando o sistema de interrupção admite níveis múltiplos, sempre existe a possibilidade de que duas (ou mais) fontes de interrupção queiram interromper no mesmo tempo. Para resolver esse problema, basta uma *rede de prioridade*, que é de fácil implementação em *hardware* (veja o Cap. 3).

Em sistemas de controle de processo, é desejável um sistema de interrupção de vários níveis organizado, como na Fig. 7-17.

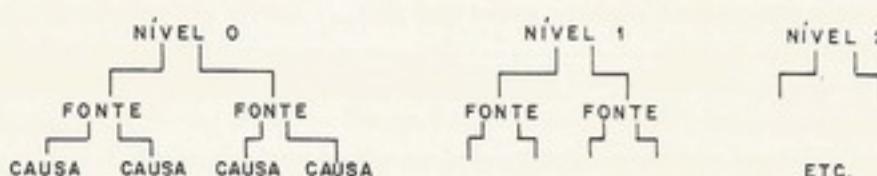


Figura 7-17. Hierarquia nos sistemas *multilevel*

O sistema de prioridade age entre os níveis e não entre as fontes do mesmo nível. Normalmente o nível 0 é o nível mais alto. Se houver pedido de interrupção pelo nível 0, o nível mais "baixo" será interrompido. Mas uma fonte de nível 3 nunca pode interromper outra fonte de nível 3.

No campo de controle de processos, existem vários sinais do processo de alarme e de medidas. Como não é econômico prover um endereço para cada fonte de interrupção, o usuário junta algumas fontes no mesmo nível. O sistema *multi-level* tem essas vantagens. (1) Os pedidos em comum de um nível são agrupados e podem interromper se não existe interrupção de nível mais alto (por exemplo, um programa de atualização pode ser executado só se não existe alarme algum). (2) As fontes de tratamento semelhantes podem compartilhar de uma mesma sub-rotina.

d. Hardware generalizado para cada nível

Pretendemos aqui generalizar o *hardware* e funções encontrados na maioria dos sistemas de níveis múltiplos.

Na Fig. 7-18, os sinais (1), (2), (3), (4) e (5) são controlados por programação, sendo isso uma função geralmente do programa monitor (executivo, supervisor ou sistema operacional).

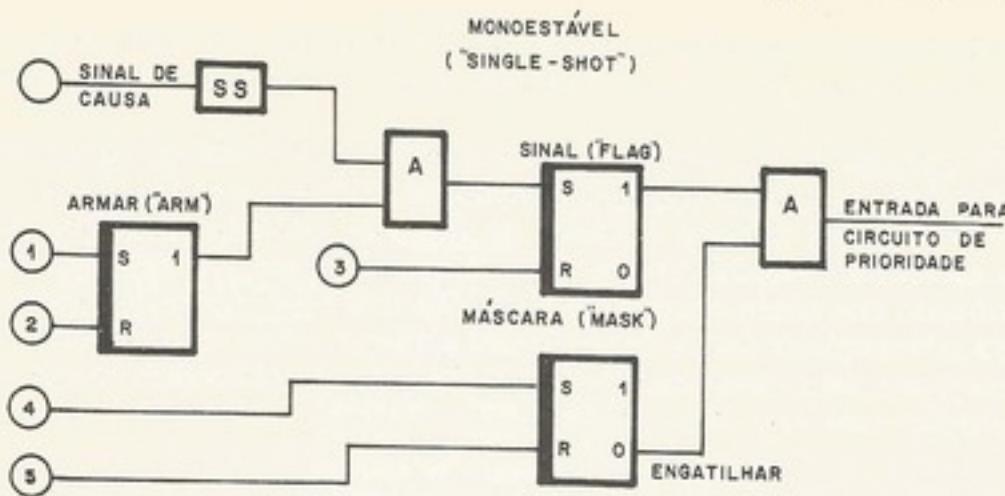


Figura 7-18. Elementos de um sistema multilevel

Num sistema de interrupção, para cada acontecimento que requer uma interrupção, o sistema terá uma das três possíveis respostas:

- ignorá-lo, nem lembrando que ocorreu (por exemplo, durante o inicio);
- guardar o fato, para tratá-lo depois;
- permitir a interrupção na hora.

O primeiro ponto não parece útil, mas existem situações em que desejamos desprezar uma interrupção, tais como quando o dispositivo está ligado fora de linha.

O segundo ponto, que fornece armazenamento para tratar da interrupção mais tarde, necessita do *flip-flop* chamado "sinal" (*flag*, Fig. 7-18).

O terceiro ponto, que trata da interrupção na hora, necessita de um sistema de níveis múltiplos e determinação de prioridade. Com o *hardware* da Fig. 7-18, podemos implementar, por *software*, qualquer uma das três respostas, desde que tenhamos a rede de determinação de prioridade.

Na Fig. 7-18, o monoestável fornece um pulso para disparar o *flip-flop* "sinal". Se a causa da interrupção for proveniente de um contato mecânico, teremos que tratar do *contact bounce* (Cap. 3). A finalidade do monoestável é disparar o *flip-flop* "sinal" uma vez apenas por acontecimento.

É o *flip-flop* "armar", que controla o "sinal"; com "armar" limpo, o nível ignora o pulso do monoestável. O *flip-flop* "máscara" é usado para impedir que o *flip-flop* "sinal" faça o pedido de interrupção. Isso é útil quando um programa de baixa prioridade está rodando, mas, por um pequeno trecho, não convém ser interrompido. Então somente nesse trecho crítico ele dispara os *flip-flops* "máscara" dos níveis de prioridade mais alta. Por exemplo, se o nível 3 estiver ativo e não for conveniente interromper pelo nível 2, ele pode disparar

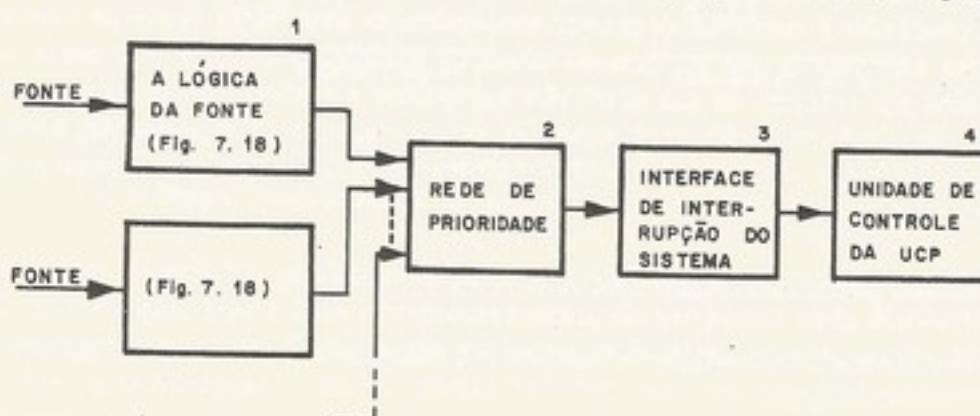


Figura 7-19. Diagrama em blocos de um esquema de interrupção

a "máscara" do nível 2. Agora, se o nível 2 quiser interromper, o sistema não liga para ele e o programa do nível 3 continua ativo até que a "máscara" do nível 2 seja limpa.

A idéia da "máscara" é chamada *arm/disarm* e, em alguns sistemas (como o Xerox Data Systems, Sigma 7⁽³⁾), de *enable*, e (no HP 2116 B) de *control*.

Vimos, na Fig. 7-18, o *hardware* da parte do nível de interrupção. A parte do *hardware* de interrupção que faz a interface com a unidade de controle da UCP do sistema é um pouco mais complicado (veja o bloco 3 da Fig. 7-19).

O sistema tem de acusar o recebimento da interrupção no inicio do seu tratamento e não deve ficar confuso se a mesma fonte interromper uma segunda vez. Além do mais, o sistema deve distinguir entre fontes de interrupção do mesmo nível, e não limpar o *flip-flop* "sinal" sem verificar que todas as fontes desse nível foram tratadas.

e. Técnicas para efetuar a interrupção

Em geral, existem duas técnicas de efetuar a interrupção, (a) forçando os *bits* do próprio endereço no *REM* e (b) forçando os *bits* de um desvio no *RD* ou *RI*. As duas técnicas ocorrem no começo de ciclo I (Fig. 7-20).

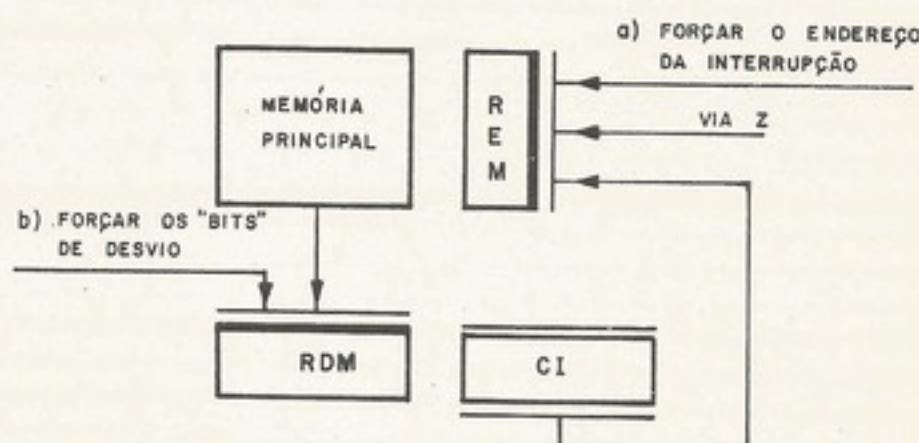


Figura 7-20. Técnicas para realização da interrupção

A técnica (a) é a mais comum. No HP 2116 B, existem endereços fixos para cada nível ("fonte" equivale a "nível", no HP), em octal: 04, *power fail*; 05, *memory protect*; 06, *DMA chan 1 completion*; 07, *DMA chan 2 completion*; e 10 a 77, dispositivos de *E/S*.

Existe uma diferença no *timing* da técnica (a) e técnica (b). A técnica (a) precisa aprontar o endereço de interrupção antes que a técnica (b).

Veja, na Fig. 7-21, que, com a técnica de modificar o registrador *RD*, o sistema pode aproveitar o tempo de acesso da memória.

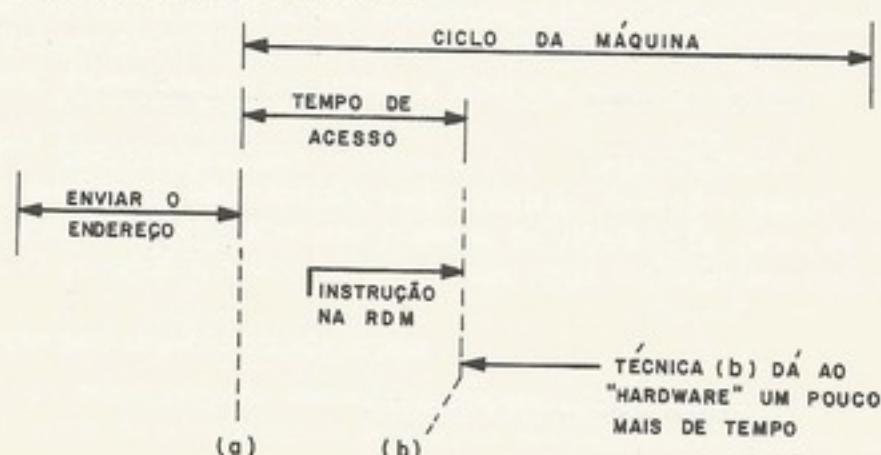


Figura 7-21. Tempo de "forçar" os *bits* para técnicas (a) e (b)

Existe uma outra técnica. No *HP 2116 B*, há uma fase (estado principal) de controle chamada *interrupt*. Com essa técnica (que pode ser utilizada tanto com o método de forçar o endereço quanto com o de forçar a instrução no *RDM*), um ciclo da máquina é usado para facilitar a transição para o novo programa. No *HP 2116 B*, é a fase 4 (veja a Fig. 7-22).

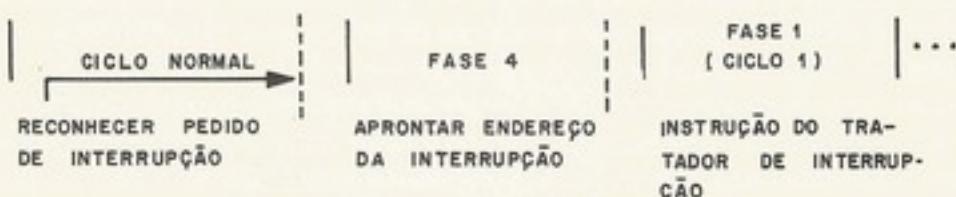


Figura 7-22. Ciclo especial de interrupção (fase 4)

No sistema *PDP-8*, a interrupção está implementada com a técnica (b), onde o código de operação da instrução “pulo de sub-rotina” é forçado no *RD*, com o campo de endereço nulo.

Assim, o *CI* é armazenado no endereço “0” e a primeira instrução do programa atendedor fica no endereço “1”. Observar que o *PDP-8* é realmente de um nível só, e tem apenas um lugar para colocar o endereço do programa interrompido.

Nos sistemas *IBM 1130* e *1800*, usa-se também a técnica de forçar uma instrução no *RD*, porém com mais sofisticação. Os bits do campo do código de operação forçado são os bits de um desvio de sub-rotina, mas no campo de endereço fica o endereço que pertence à fonte de interrupção. Durante o ciclo I, a atualização do *CI* é inibida para poder armazenar o endereço correto do programa interrompido.

O *hardware* que coloca a identificação do nível interrompido no campo de endereço (no caso do *IBM 1800*) pode aproveitar a rede de prioridade que já existe (veja a Fig. 7-23).

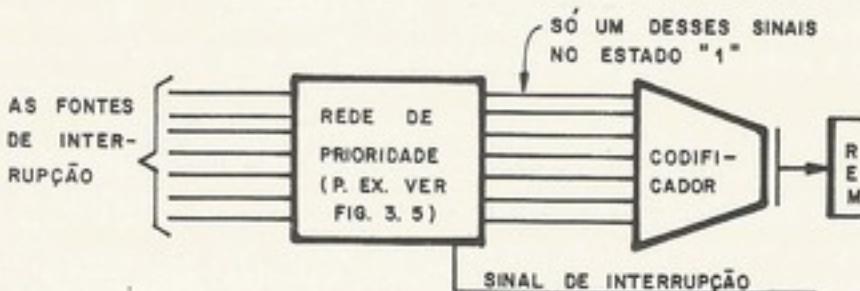


Figura 7-23. Identificação da fonte pela técnica “matriz”

Com esta técnica, é possível que cada fonte precise de um fio de entrada pelo processador. No *HP 2116 B*, a idéia da Fig. 7-23 é usada, mas a “rede de prioridade” é espalhada, um pouco em cada dispositivo; são as entradas ao “codificador” que vêm de cada dispositivo. (Isso será visto no item 7-5, que explica o sistema de interrupção do *HP*.)

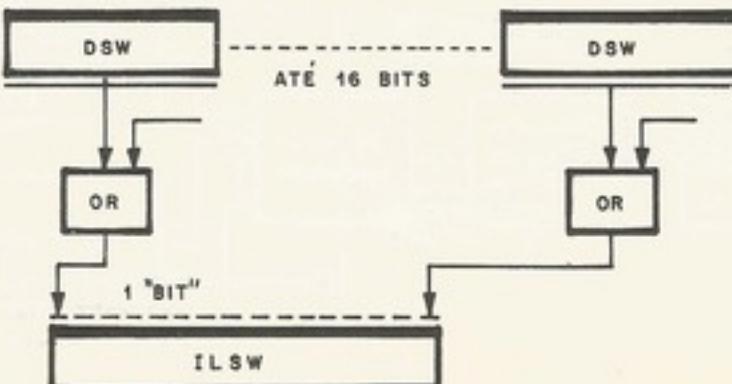


Figura 7-24. Formação da *ILSW* no *IBM 1800*

f. Técnicas para descobrir a causa da interrupção

Muitas vezes, um programa tratador de interrupção tem de descobrir a causa específica da interrupção, embora sendo conhecido o nível e/ou a fonte da interrupção. A rede de prioridade só funciona entre os níveis e não entre as fontes do mesmo nível.

Quando o nível tem apenas uma fonte, o problema de identificar a causa é facilitado, mas, caso contrário, existem as seguintes técnicas: (1) registrador de *status*, (2) comando especial e (3) interrogação.

Registrador de *status*

Como foi explicado relacionado ao *S/360*, o velho *PSW* dispõe de um campo *interrupt code*. Esse campo do *PSW*, carregado por *hardware*, identifica a fonte de interrupção através do programador, examinando os seus bits.

Comando especial

Uma segunda maneira de se descobrir a causa está exemplificada no *IBM 1800*⁽⁷⁾. Nessa máquina, cada dispositivo dispõe de uma palavra de 16 bits, chamada *DSW* (*device status word*). O *DSW* tem um bit para identificar a causa de uma possível condição ou interrupção. As fontes de interrupção identificam-se com os dispositivos e os níveis de interrupção (*interrupt level*) são conjuntos de dispositivos (veja a hierarquia na Fig. 7-17).

O *hardware* é organizado como na Fig. 7-24, os *DSW* dos dispositivos do mesmo nível alimentam uma palavra de *status* para o nível; isso se chama a *ILSW* (*interrupt level status word*).

Um bit da *ILSW* corresponde a um dispositivo. Então esse bit estará no estado “1” se houver qualquer bit do *DSW* no estado “1”. Vamos supor, então, que um dispositivo qualquer, por exemplo, a leitora de cartões, queira interromper. O pedido passa através do *hardware* do nível próprio e, em seguida, o programa tratador do nível é acionado. Esse programa executa um comando especial, *sense interrupt level*, que carrega o acumulador com o *ILSW* do nível. No *ILSW* o bit que pertence à leitora de cartões está no estado “1”. Descobrindo isso, o programa executa um outro comando *sense device* para a leitora, que carrega o acumulador com o *DSW*.

Uma técnica semelhante à do *IBM 1800* é mostrada na Fig. 7-25(a). Todos os dispositivos são ligados em paralelo na via de *E/S*, mas há um fio que percorre, serialmente, os dispositivos. Esse fio, quando no estado “1”, indica ao dispositivo, se o *flip-flop* de interrupção está no estado “1”, para colocar seu endereço no *bus de E/S* e passar “0” ao vizinho. Assim, o dispositivo mais próximo à *UCP* tem a prioridade mais alta.

No exemplo da Fig. 7-25(b), a *UCP* está executando o comando especial e, então, o fio em série está no estado “1”. O dispositivo 1 não está interrompendo; então ele passa o sinal para dispositivo 2. O dispositivo 2 está interrompendo; então ele coloca seu endereço na via e passa o sinal no estado “0” para o dispositivo 3. O dispositivo 3, mesmo querendo interromper, não tem chance porque ele recebe o sinal em série no estado “0”.

Interrogação

O terceiro método de identificação, aqui chamado de “interrogação”, foi explicado relacionado ao *PDP-8*, onde o programa endereça os *flip-flops* de *status* de várias causas de interrupção. O processo é muito lento, mas não exige *hardware* especializado.

g. Interrupção para casos de falta de energia

Existe um tipo de interrupção que muitos sistemas têm chamado *power fail*, cujo propósito é evitar a perda de informação de programas parcialmente executados, devido a uma falta de energia. A detecção da queda de energia provoca uma interrupção que desvia a *UCP* ao programa tratador, que guarda o *status* do sistema na memória principal (que deve ter

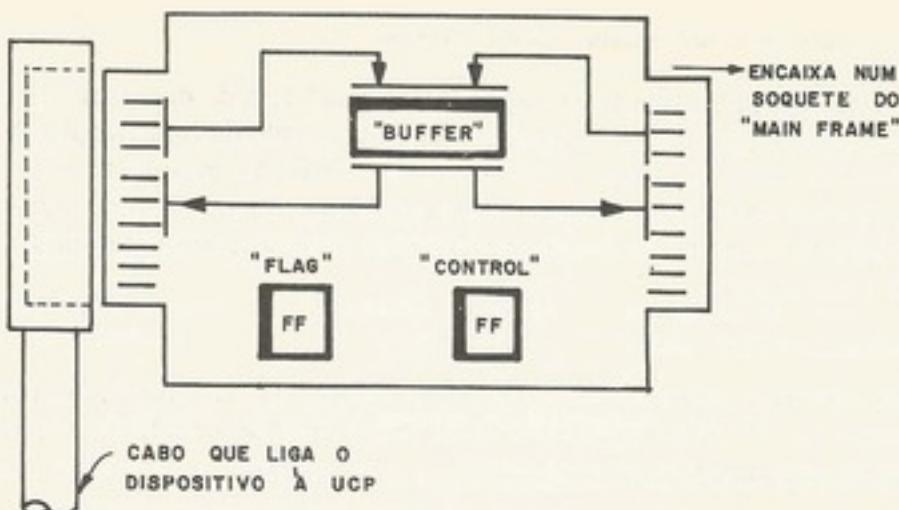


Figura 7-25. (a) Identificação da fonte pelo comando de colocar o endereço na via

núcleos de ferrite) e liga alguns sinais. Na volta da energia, ao recomeçar o sistema, um programa restaura o *status* guardado, para poder recomeçar o programa parcialmente executado. Esse tipo de interrupção, infelizmente, só representa uma resolução parcial do problema de falta de energia. Se o sistema está fazendo uma operação de entrada/saída quando ocorre essa interrupção, provavelmente que não haverá tempo para o término da operação. Também, a interrupção *power fail* não adianta muito em sistemas com memórias voláteis (monolíticas). Para centros de processamentos de dados onde a confiabilidade, e a "integridade" da informação é importante, existe uma técnica para evitar problemas de falta de energia, que é acrescentar um conjunto motor-gerador com bastante inércia para alimentar as fontes do sistema. Ao receber a interrupção de *power fail*, a inércia do motor-gerador sustenta o sistema até o término das operações de E/S e permite até a gravação do conteúdo da memória principal num veículo magnético.

7.5 INTERRUPÇÃO NO HP 2116B

a. Assuntos gerais

Já ressaltamos a nossa filosofia de que conhecer as "peças" não implica conhecimento do sistema. Nessa seção, então, pretendemos explicar um esquema completo de interrupção, usando um minicomputador para que o leitor possa entender um sistema integral. Para esse fim, mudaremos do nível de generalidades e estudaremos detalhadamente o esquema do *HP 2116 B*⁽⁵⁾.

Esse sistema se encaixa, relativamente às interfaces, como um sistema radial (Fig. 7-4). Os circuitos especializados são colocados num cartão de interface. Todos esses cartões têm flip-flops *flag* e *control*, e esses flip-flops são sincronizados com sinais *T2* e *T5* do relógio

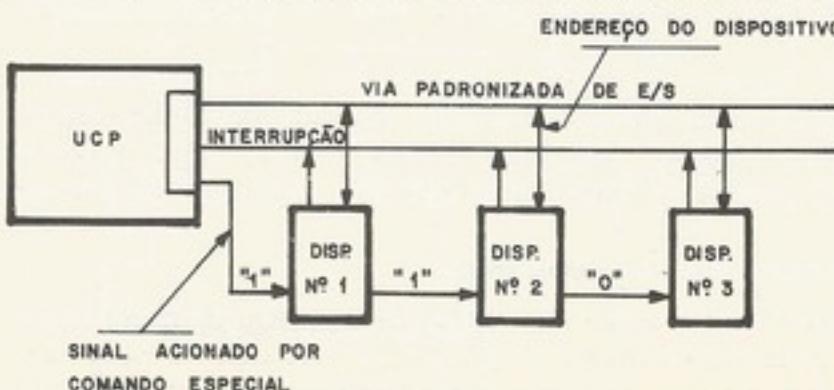


Figura 7-25. (b) O cartão de interface, *HP 2116B*

central. Os cartões recebem sinais de dados do registrador de interface da UCP e recebem sinais de controle do decodificador do código de operação do RI. Os cartões geralmente têm um registrador *buffer* para transferir dados entre o dispositivo e os registradores A e B. A função do cartão de entrada/saída pode ser vista na Fig. 7-25(a).

b. Os níveis e a seqüência da interrupção

O sistema tem até 64 níveis de interrupções, ordenados através de prioridades. As prioridades fixas são as seguintes: 4, *power fail*; 5, proteção da memória; 6, *DMA 1*; e 7, *DMA 2*. Os níveis 0, 1, 2 e 3 não existem.

Os níveis de interrupção de 8 a 63 pertencem aos dispositivos de E/S; o nível é determinado pelo soquete em que o cartão da interface está colocado. Cada dispositivo ou nível *i* tem um fio *IRQ_i* (*interrupt request*) que serve como entrada para um circuito codificador de endereço. A interrupção é realizada através da técnica de forçar esse endereço no registrador de endereço (RE) da memória (Fig. 7-20). Assim, a UCP retira uma instrução “desvio de sub-rotina” (código *JSB*) para o programa tratador do nível. O programa pode ser interrompido depois de qualquer ciclo, e não somente depois da execução completa de uma instrução. Para facilitar a transição, a máquina tem um estado principal, fase 4 (Fig. 7-22), que controla um ciclo de máquina que decrementa o contador *CI* e, depois, carrega o RE com o endereço codificado, referente à Fig. 7-26. A transição ocorre nas seguintes etapas:

- 1) na fase 4, o *CI* é decrementado de 1 e o *IRQ* é codificado para formar o *service request address* e colocada no *RE*;
- 2) o *RE* indica uma instrução de desvio de sub-rotina (geralmente); essa instrução é guardada na localização 4 para o nível 4, localização 5 para o nível 5, etc.;
- 3) o desvio de sub-rotina, *JSB*, é retirado da *MP* na fase 1 (ciclo *I*) e o campo “código da operação” vai para o *RI*, e o endereço passa para o *RE*;
- 4) durante o começo da fase de execução, o *CI* é incrementado de 1 e colocado no *RD* (por isso foi decrementado na fase 4);
- 5) o endereço efetivo do *JSB* passa do *RE* para o *CI* [o desvio de sub-rotina é do tipo *LPG-30* (veja o Cap. 6); o *CI* é guardado no alvo do desvio e a execução começa com a instrução seguinte];
- 6) no final da fase de execução, o *CI* + 1 passa para *RE*, pronto para começar a fase 1 da próxima instrução.

c. A rede de prioridade

Vamos então descobrir como está implementada a rede de prioridade. Essa rede está espalhada nos cartões de interface. É importante entender que existe um fio que liga em série os níveis de interrupção, na ordem de prioridade. Esse fio, quando entra num cartão de interface, é chamado *PRH* (*priority high*) e sai como *PRL* (*priority low*). O dispositivo da prioridade mais alta recebe o sinal *PRH* no estado “1”. Se ele não quiser interromper, passará o sinal *PRL* para o próximo nível de prioridade também no estado “1”; caso contrário, passará o *PRL* ao estado “0”.

O *PRL* do nível mais alto passa a ser o *PRH* do próximo nível. Para poder interromper, precisa receber *PRH* no estado “1”. Veja o circuito da determinação de prioridade na Fig. 7-27.

d. Os “flip-flops” “control” e “flag”

Agora estudaremos de que maneira o cartão de interface, o subsistema de interrupção e o programa interagem durante uma interrupção. Para cada nível, existe um *flip-flop* chamado *control* no cartão de interface que, quando no estado “1”,arma ou engatilha a interrupção para aquele nível. Quando o *control* está limpo, o nível “lembra” da interrupção, mas não tenta interromper. Para controlar o *control* de cada nível, o programador tem duas

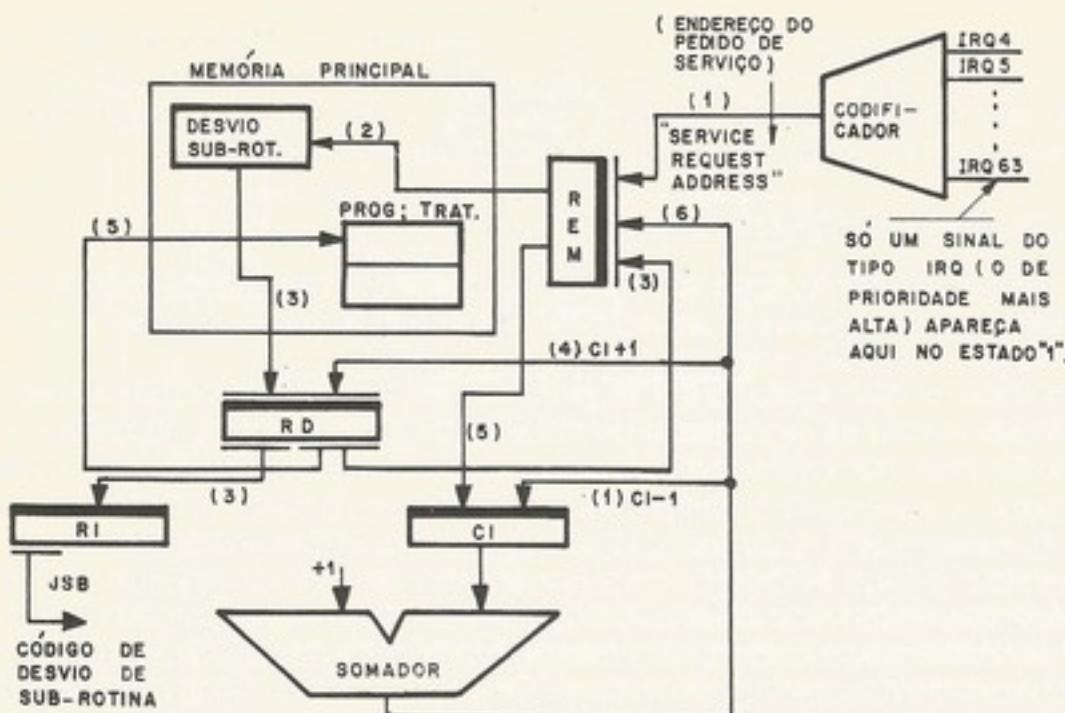


Figura 7-26. Realização da interrupção

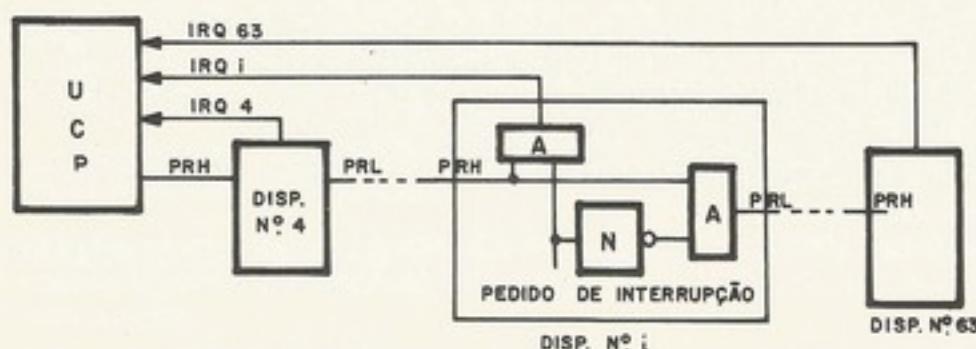


Figura 7-27. A rede (espalhada) de prioridade

instruções disponíveis: *STC* (*set control*) para disparar o *control*, e *CLC* (*clear control*) para limpar o *control*. Para lembrar da interrupção ou para efetuar uma interrupção, o cartão de interface tem um *flip-flop flag*. Esse *flip-flop* pode ser disparado pelo dispositivo, conforme uma situação que exija a interrupção. Para fins de controle, o programador também tem disponível as instruções necessárias para controlar o *flag*: *STF* (*set flag*) para disparar o *flag* e *CLF* (*clear flag*) para limpá-lo. Deve-se entender que o campo de endereço das instruções *STC*, *CLC*, *STF* e *CLF* indica qual dispositivo está sendo afetado, (então, *STCi* indica “disparar o *control* do dispositivo *i*”). Há mais um vínculo que afeta o circuito:

o sistema dispõe de um *flip-flop system enable* quearma (*enables*) o subsistema de interrupção; sem *system enable*, no estado “1”, a interrupção está desarmada, e nenhuma interrupção pode acontecer.

e. O projeto da parte da interrupção do cartão de interface

A explicação do circuito no cartão de interface seguirá um caminho “indireto”, no sentido de que pretendemos mostrar também como os engenheiros projetistas de *hardware* costumam raciocinar: começaremos projetando um circuito simples, que funciona “quase” corretamente, e o melhoraremos em seguida. A primeira tentativa aparece na Fig. 7-28.

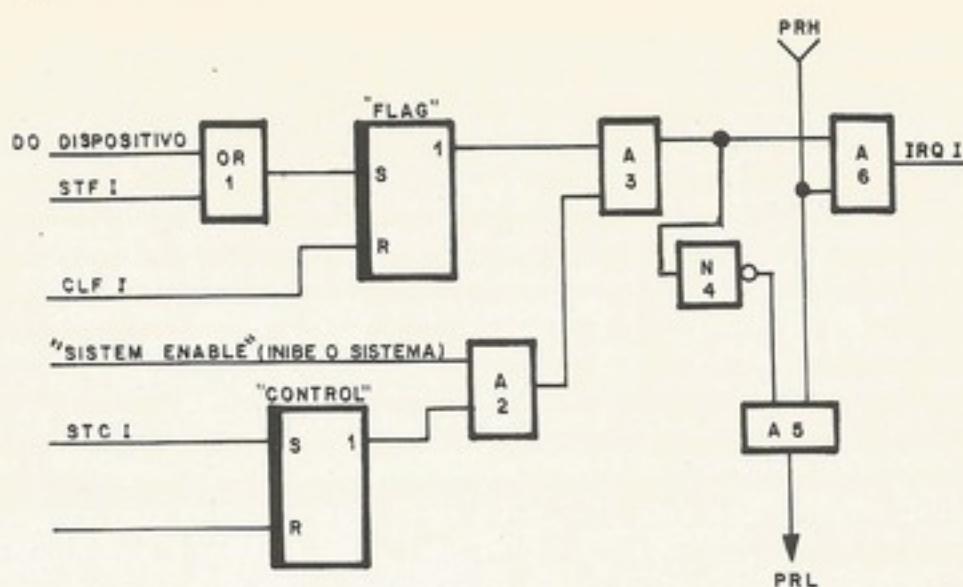


Figura 7-28. Circuito de interrupção no cartão de interface, primeira tentativa

O problema com esse circuito é que ele continuará tentando interromper, isto é, não há meio de “desligar” o IRQ_i , mesmo quando o programa tratador de interrupção i está sendo rodado. O que necessitamos é um sinal de *reconhecimento* que acuse ter sido o pedido atendido e que “desligue” o IRQ_i mas que continue a bloquear o PRL . Esse sinal vem do subsistema de interrupção da UCP e chama-se *IAK* (*interrupt acknowledge*). Mas não podemos usar o *IAK* para limpar o *flag*, pois, assim o cartão deixaria o sinal *PRH* passar, com *AND 3* no estado “0” e *N 4* no estado “1”, através de *AND 5* para *PRL*, permitindo um dispositivo de mais baixa prioridade interromper. Para prevenir isso, foi criado mais um *flip-flop*, o *flag buffer*, que guarda a interrupção do dispositivo antes de passar para o *flag* (Fig. 7-29).

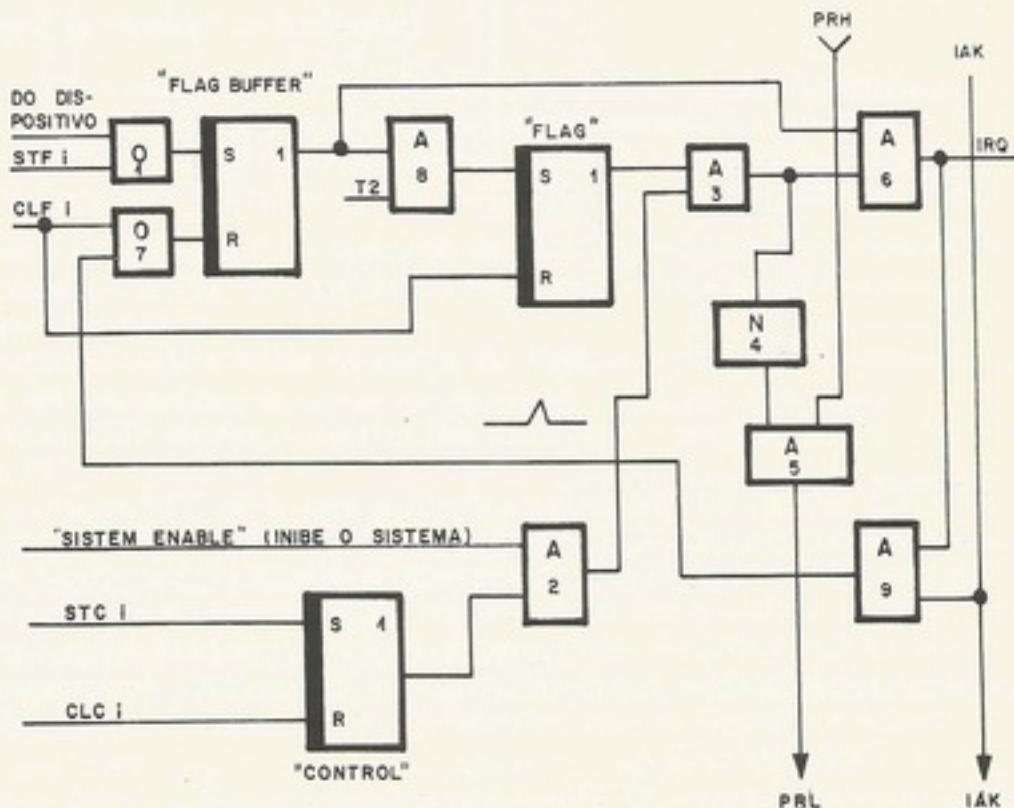


Figura 7-29. Circuito de interrupção no cartão de interface, segunda alternativa

Observar que através do *AND* 6, para gerar o *IRQ*, precisamos do *flag buffer* e do *flag*. Então, para acusar a interrupção, basta limpar o *flag buffer*. O *flag* continua disparado; então o *PRH* está bloqueado e não passará ao próximo nível no estado "1". Notamos também, que o mesmo sinal *IAK* passa para todos os dispositivos. No *AND* 9, aproveita-se o fato que o *IRQ_i*, na realidade, sai de uma rede de prioridade, e que só existe um sinal *IRQ_i*, no estado "1", no sistema. Com o *IAK* e o *IRQ_i*, o *flag buffer* desse único dispositivo é limpo.

Na Fig. 7-29 há um possível problema com uma malha (*loop*) com o *flag buffer*, *AND* 6, *AND* 9, e *OR* 7. Na subida de *IAK*, a saída do *AND* 9 sobe, limpa o *flag buffer*, o *AND* 6 cai para "0" e o *AND* 9 cai para "0". Isso implica que a saída do *AND* 9 estará no estado "1" por muito pouco tempo. Para evitar isso, podemos usar um *flip-flop* no lugar do sinal *IRQ*, como mostra a Fig. 7-30. Assim, o *IRQ* será limpo a cada *T₂*, mas, no próximo *T₅*, ele será disparado novamente, dado que o *flag buffer* continua disparado e que o nível *i* continua recebendo o *PRH* no estado "1".

Observamos aqui que, durante esse "projeto", as especificações do problema foram modificadas. Assim, seria difícil resolver esse problema por métodos clássicos com a descrição da tabela de fluxo de Huffman. O circuito da Fig. 7-30, com dez sinais de entrada, requer uma tabela de fluxo de Huffman de 1024 colunas.

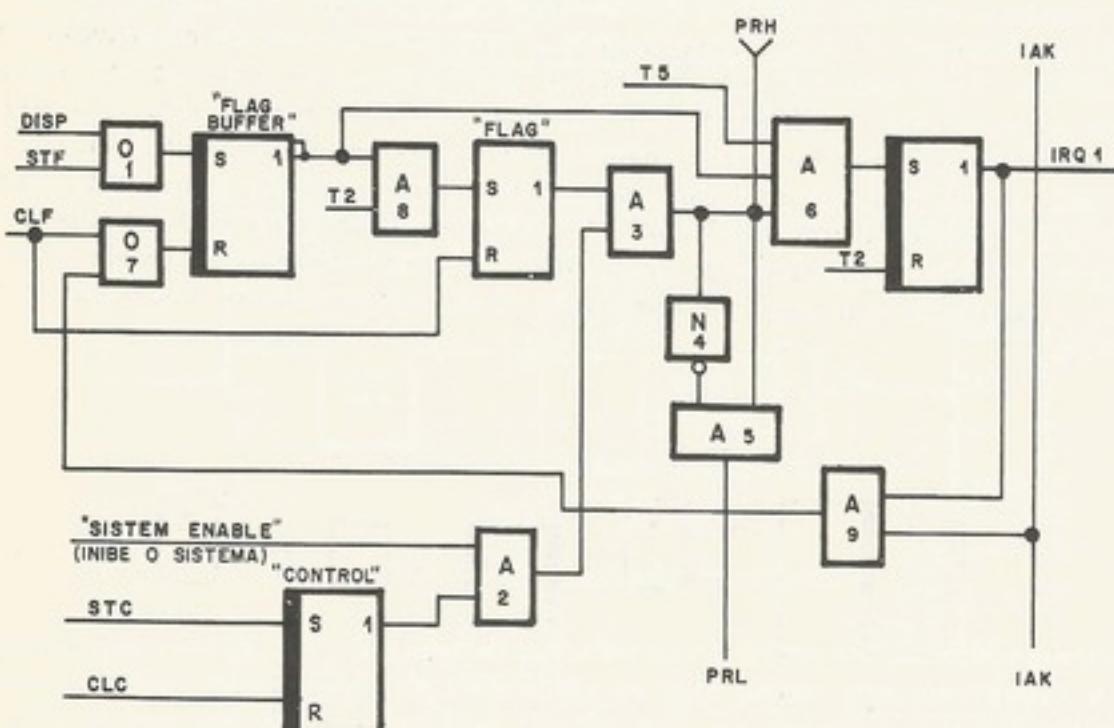


Figura 7-30. Sistema de interrupção no cartão de interface

f. O subsistema de interrupção da UCP

Vamos estudar como se pode efetuar a transferência para a fase 4. Em primeiro lugar, há certas situações em que não seria conveniente ter-se o próximo ciclo de fase 4. Faltando essas condições, existe o sinal *enable service request* ou *ESR* (arma pedido de serviço). O que inibe o *ESR* é a fase 5 (ciclo de endereçamento indireto) ou mesmo a fase 4 (não queremos uma fase 4 seguindo outra fase 4, pois assim perderemos o *CI* do programa interrompido) ou durante a execução de uma instrução que afeta a saída da rede de prioridade do sistema: *STC*, *CLC*, *STF* e *CLF*.

Um outra situação em que o *HP 2116 B* não aceita interrupção é quando a máquina está no estado *halt* (parado). Então exige o sinal *run* 1, que indica estar a máquina rodando, para entrar na fase 4. Quando se determina uma interrupção do programa atual, a máquina

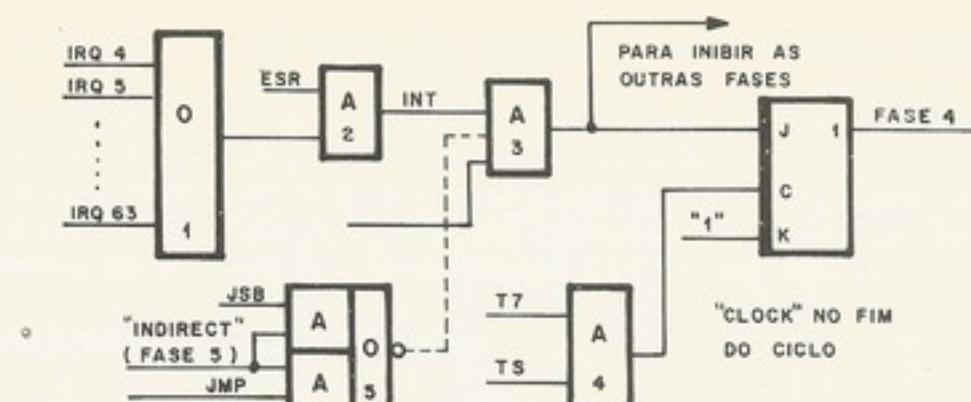


Figura 7-31. Circuito para efetuar fase 4, ciclo de interrupção

inibe a mudança para as outras fases. (Veja a Fig. 7-31, para uma primeira tentativa do circuito.) Explicaremos a seguir a porta com linha tracejada.

Infelizmente o circuito sem a modificação mostrada com o bloco lógico não funciona bem com o software da interrupção. No fim do programa tratador da interrupção, a volta ao programa interrompido é efetuada através da instrução *clear flag (CLF)* e um desvio indireto (*JMP I*) à primeira palavra da rotina. Não convém ser interrompido entre a execução da instrução *CLF* e a *JMP I*. Vamos supor o contrário e que nível 8 tenha interrompido nível 11 e guardado o *CI* na primeira palavra do nível 8. Agora, durante o tratamento do nível 8, o nível 10 quer interromper. Em nossa hipótese, quando o nível 8 termina, e entre a execução do *CLF* de nível 8 (que limpa o *flag* deixando o *PRH* passar) e o *JMP I* do mesmo nível 8; o nível 10 interrompe, já que recebeu o sinal *PRH*. Vamos supor agora que o nível 8 gere uma nova interrupção. Ele interrompeu o nível 10 (o nível 8 tem prioridade sobre o nível 10) e guarda o endereço do lugar interrompido do nível 10 na primeira palavra do programa tratador do nível 8, perdendo assim o *CI* do nível 11. Para evitar isso, o circuito que efetua a fase 4 fica com bloco lógico da Fig. 7-31.

g. Carta de “timing” da interrupção

Para melhor entender como a interrupção foi implementada, estudaremos o exemplo do nível 8 da Fig. 7-32. Começaremos com o sistema desarmado, o que inibe qualquer interrupção, e o *flip-flop control* do nível 8 limpo, que inibe interrupção do nível. Em relação aos tempos um a oito indicados na Fig. 7-32, indicamos a seqüência dos eventos.

1. Com a instrução *STF 00*, o sistema de interrupção é armado, o sinal *system enable* da Fig. 7-30 está no estado “1”. Também, sendo instrução de código *STF*, o sinal *ESR (enable service request)* é inibido.
2. Mostramos aqui a borda de ataque do sinal de interrupção do dispositivo, talvez através de um *single-shot*.
3. O sinal *DISP* dispara o *flip-flop flag buffer*, mas o próprio *flip-flop flag* só dispara com sinal *T2*.
4. A instrução *STC 08* dispara o *flip-flop control* durante *T4*, que, agora, permite ao nível 08 interromper; o sinal *PRL*, que passa como *PRH* para o nível 09, é inibido; então os níveis de prioridade mais baixos que 08 não podem interromper. Também, sendo instrução de código *STC*, o sinal *ESR* é inibido.
5. Com tempo *T5*, o *flip-flop IRQ 08* dispara.
6. Ao tempo *T7*, o circuito da Fig. 7-31 não efetua fase 4 porque o sinal *ESR* está sendo inibido [veja (4)].
7. O sinal *INT* (Fig. 7-31) entra no estado “1”, sendo válido *ESR*.
8. Dessa vez a unidade de controle passa para a fase 4, efetuando a interrupção.
9. Sendo a fase 4, o sinal *ESR* é inibido, proibindo outra interrupção. Na fase 4, o *CI* é decrementado e o endereço 08 é encaminhado ao *RE* da memória.

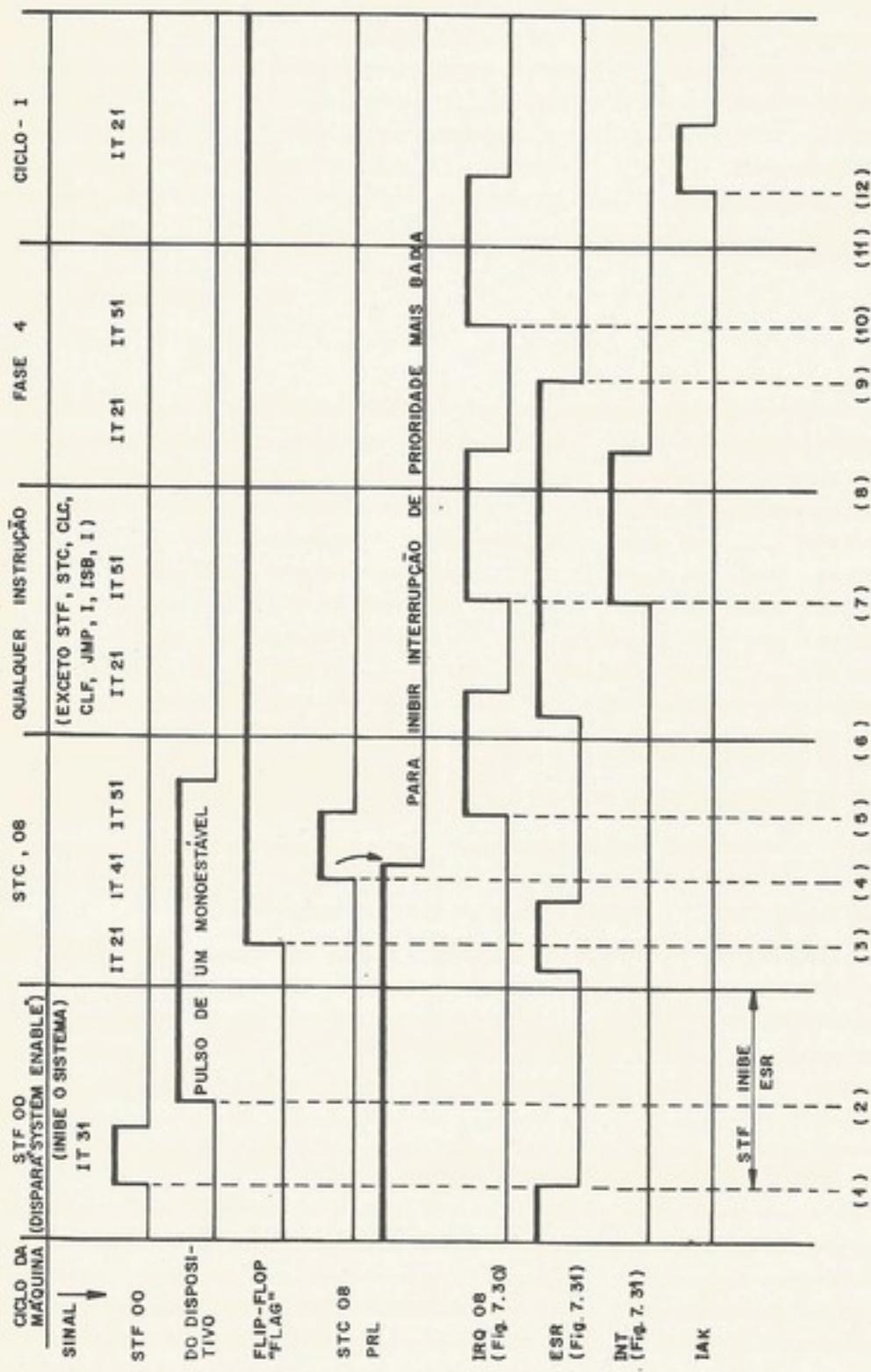


Figura 7-32. Carta de *timing* da interrupção do *HP 2116B*

10. O *flip-flop IRQ* 08 dispara, pois o cartão de interrupção não recebeu reconhecimento.

11. A unidade de controle entra na fase 1 (ciclo *I*) para retirar a instrução do endereço 08, provavelmente um *JSB* à rotina tratadora do nível 08.

12. A unidade de controle acusa o recebimento da interrupção através do sinal *IAK*. Esse sinal limpa o *flag buffer* do nível 08, que inibe os futuros disparos do *flip-flop IRQ* 08 até o recebimento de outro sinal de interrupção *DISP* do dispositivo.

7.6 A ARQUITETURA DE E/S NO S/360

Na seção anterior, estudamos um exemplo da interrupção de um minicomputador. Agora vamos estudar a arquitetura de *E/S* do *S/360*. Essa arquitetura foi projetada para um sistema multiprogramado que seria controlado por um sistema operacional (programa de controle). Esse esquema é muito flexível, mas essa flexibilidade vem acompanhada de um compromisso. Por exemplo, para “emular” a instrução “ler cartão” do *IBM 1401*, o programa de instruções do *S/360* é relativamente longo.

a. Assuntos gerais

No *S/360*, a *UCP* começa uma operação de entrada/saída pela indicação do endereço de um programa de canal (*channel program*) e o endereço do dispositivo relevante. Daí, a *UCP* e o canal operam independentemente até que o canal interrompe a *UCP*, avisando-a de que o programa de canal foi executado, e providencia informações de *status*. O canal é um processador de programa de canal que retira os comandos e executa-os. O motivo dessa estrutura foi sobrepor processamentos com entrada/saída e, ainda mais, permitir operações múltiplas de entrada/saída. Assim, as fitas magnéticas podem operar simultaneamente com a leitora de cartões.

O canal se comunica com os dispositivos através de uma interface padronizada. Então, do ponto de vista do canal, ele não pode, nem deve, distinguir entre os dispositivos.

Então por que não se ter a própria *UCP* colocando dados na interface e aceitando-os da mesma? A entrada/saída nos minicomputadores funciona assim, mas, com essa estratégia, a autonomia do canal está perdida. Em blocos, o esquema do canal está mostrada na Fig. 7-33.

b. Os compromissos da implementação

Em linhas gerais, as interfaces do programador com o canal são o programa do canal e a área da *MP* chamada *buffer* de entrada/saída. A interface do canal com os dispositivos é a interface padronizada. A interface do canal com a *MP*, como visto na Fig. 7-33 não faz parte da arquitetura. Ao invés disso, cada modelo da arquitetura tem seus próprios compromissos; por exemplo, no *S/360* modelo 25, um sistema de baixo custo, o canal está implementado por microprograma; e no *S/370* modelo 165, um sistema de grande porte, os canais são sistemas digitais estanques implementados em *hardware*.

Admitindo-se que cada modelo da arquitetura possua seu próprio sistema de memória primária, fica evidente que a implementação do canal, tal como a da *UCP*, é um assunto de engenharia e não de arquitetura. Embora a largura da transferência de dados através da interface padronizada seja de um byte (8 bits mais paridade), a largura da palavra de *MP* pode variar entre os modelos. Então o canal faz uma espécie de *buffering*. Por exemplo, no *S/360* modelo 65, o canal recebe 8 bytes da *MP* de uma vez e os envia um por vez ao dispositivo; em sentido contrário, o canal monta uma palavra de 8 bytes para guardá-la na *MP*, roubando um ciclo só da *UCP*.

Figura 7-32. Curva de timing da interrupção do *HP 2116B*

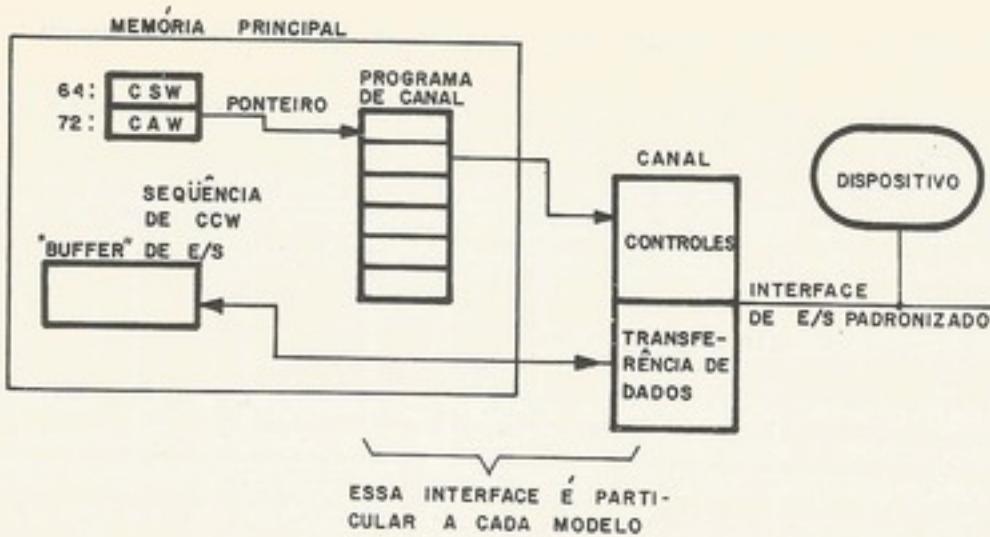


Figura 7-33. O esquema do canal do S/360 e S/370

c. O formato e os tipos de comando

O canal retira e executa os comandos da memória principal. Os comandos são chamados *channel control words* (*CCW*). Essas palavras são de 8 bytes, com o formato mostrado na Fig. 7-34.

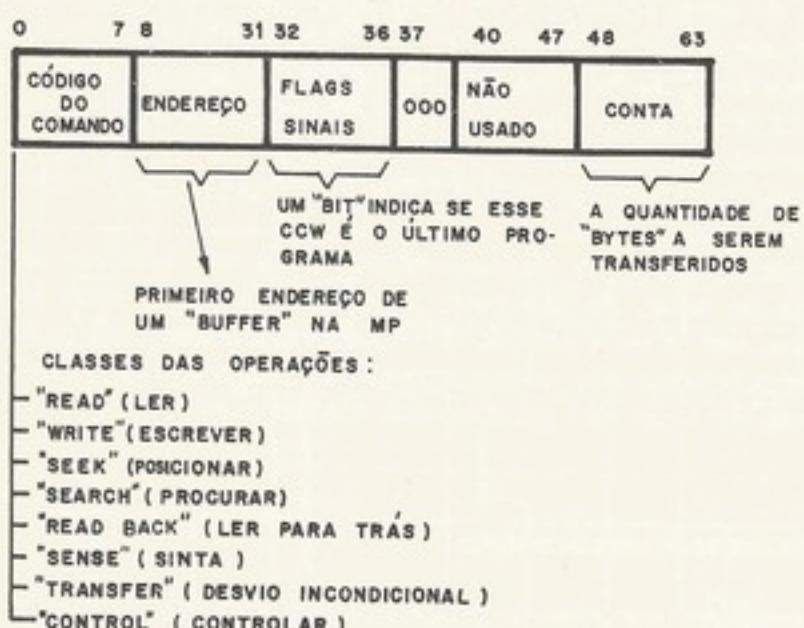


Figura 7-34. O formato da CCW.

As classes de comandos, conforme o byte de comando (bits 0 a 7), são vistos na Tab. 3.3.

Tabela 7-3 Definição do tipo de comando

Na Tab. 7-3, os bits *M* são chamados de "modificadores". A interpretação dos bits *M* depende do tipo do dispositivo endereçado. Com os bits *M*, então, dentro das classes de comando, permite-se mais de um código. Em geral, o canal se interessa mais pela classe do comando, para saber o tipo de operação (e a direção da transferência), mas ele passa o código do comando diretamente ao dispositivo. É o dispositivo que realmente interpreta o comando. Por exemplo, um certo código pode significar uma coisa para a impressora e outra para a fita magnética. Observe que o canal nem sabe o tipo de dispositivo com quem ele está se comunicando. O comportamento do canal depende somente da classe do comando, e os pormenores da execução do comando são de responsabilidade do dispositivo.

Algumas classes de comando são de interesse; o *write* indica a direção de transferência, da *MP* ao dispositivo, começando com o endereço indicado (campo 8 a 31). A cada byte transferido, o canal decrementa a "conta" (campo 48 a 63) até que esta fique zero, indicando o fim da transferência. O endereço do último byte é o endereço *mais* a "conta" original. No caso do *read*, a transferência é na direção dispositivo-*MP*. O caso do *read backward* é para quando, por exemplo, a fita magnética está se deslocando na direção oposta à da escrita. Então os bytes são colocados primeiro no endereço *mais alto* do *buffer*, sendo o último byte colocado na localização do campo de endereço (8 a 31) *menos* a "conta". (No *read backward*, o campo de endereço do *CCW* é o endereço *mais alto* do *buffer*.) O comando de controle *seek*, interpretado por um disco, é usado para posicionar o mecanismo "leitura/escrita", e o comando de controle *search* é usado para o disco comparar um campo-chave do próximo registro do disco com o campo na localização indicada pelo *CCW*. O comando *search* também dispõe de um tipo de desvio condicional; o canal salta o próximo *CCW* se a comparação está correta. A operação de *sense* efetua a transferência de um byte de *status* do dispositivo ao canal para montar um *channel status word* (*CSW*), que é uma palavra de 8 bytes indicando o *status* do canal e o dispositivo. No caso da classe de *control*, a interpretação realmente está feita no dispositivo: pode ser "avançar papel" no caso da impressora, ou de *rewind*, no de fita magnética, ou, como já vimos, *seek* ou *search* no caso de disco.

d. A seqüência típica das operações

1. O supervisor precisa achar um *buffer* do tamanho necessário para receber dados ou já deve ter um *buffer* pronto para descarga.
2. O supervisor prepara um programa de canal.
3. O supervisor carrega o *CAW* da localização 72 com o endereço do primeiro *CCW* do programa de canal.
4. Tudo começa em relação ao canal com a instrução *SIO* (*start input output*) executada pela *UCP*. O bit menos significativo do endereço efetivo dessa instrução indicam o canal e o dispositivo. A *UCP* notifica o canal do dispositivo afetado.
5. O canal retira o *CAW* da memória para poder retirar o primeiro *CCW* do programa de canal.
6. O canal manda o código do comando do *CCW* ao dispositivo através da interface padronizada, recebendo de volta um byte de *status*.
7. O canal monta um código de condição *CC* (*condition code*) para o *PSW* da *UCP*, que indica se o canal pode ou não executar o programa de canal. (Talvez não possa porque já está ocupado ou o canal ou o dispositivo.) No caso do *S/370*, existe a instrução *SIO fast release* em que a *UCP* não espera para o canal gerar o código de condição, se o canal ou dispositivo estiver ocupado, o canal interromperá a *UCP* depois. Depois de receber o *CC* do canal, a *UCP* e o canal continuam, independentes um do outro.
8. O canal transfere dados, envia comandos etc., executando o programa até encontrar um *CCW* sem *flag* indicando a última palavra do programa de canal.
9. Após terminar o último *CCW*, o canal monta seu *status* e o *status* do dispositivo e no *CSW*, e pede uma interrupção da *UCP*. Concedida a interrupção, o *CSW* está colocado na palavra 64 da *MP* (Fig. 7-33).

10. O programa supervisor examina o CSW para verificar se tudo foi bem, e o programa usuário que provavelmente estava esperando na E/S está notificado.

Na seqüência apresentada, supõe-se que haver erro ou outro problema com o andamento do programa de canal. A maioria das operações de E/S é, felizmente, desse tipo. No entanto é verdadeiro que uma boa parte dos esforços, tanto dos programadores das rotinas de E/S como dos engenheiros que implementam o canal, envolve-se com o que fazer quando há caso anormal. É nessa área também que se concentram muitos esforços na fase de depuração.

e. Unidades de controle

Em algumas situações, o canal se comunica através da interface padronizada com uma "unidade de controle", em vez de se comunicar diretamente com o dispositivo (Fig. 7-35).

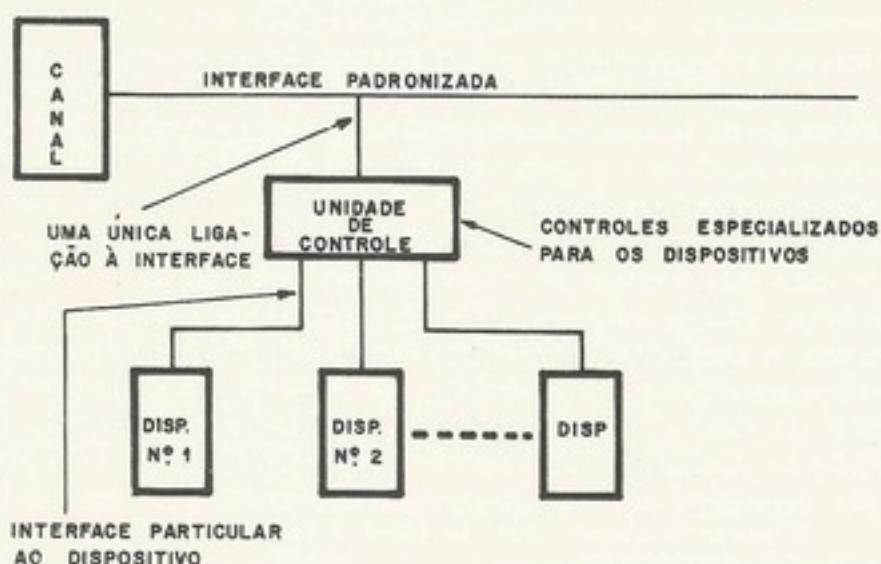


Figura 7-35. Configuração da unidade de controle

Há duas situações em que vale a pena fazer uma unidade de controle de dispositivos. (1) Quando se trata do conjunto de dispositivos que quase todo sistema tem (leitura de cartão, perfuração de cartão e impressora); e (2) quando os dispositivos são iguais como nos *drivers* de fita e de disco.

No primeiro caso, a unidade tem circuitos especiais para efetuar ligações com três dispositivos, e o usuário tem de usar os dispositivos para os quais as interfaces particulares foram projetadas.

No segundo caso, a unidade de controle junta os controles que são comuns aos dispositivos (pois os dispositivos são idênticos).

f. As características dos canais

Os dispositivos de leitura e perfuração de cartões e o de impressão são lentos; então aproveitam o canal "multiplex" (sempre canal 0, no S/360 e S/370). Assim, os três dispositivos podem ser ativos ao mesmo tempo, e os bytes transferidos compartilham em tempo a interface. O canal 0 tem armazenamento para todos os CCW ativos (até 256). Além disso, o canal tem de lembrar o endereço do dispositivo, o endereço na memória do programa de canal e a "conta" (count). Essa informação está situada em uma palavra chamada UCW (*unit control word*).

Os outros canais no S/360 são chamados de *selectors*. Esses canais normalmente são para dispositivos de maior velocidade, como fita e disco magnéticos. Para esse tipo de canal, ele é ocupado (*busy*) quando está executando um programa de canal. Assim, o dispositivo

o programa
com o anda-
desse tipo. No
madores das rotinas
o que fazer quando
na fase de de-
padronizada com uma
dispositivo (Fig. 7-35).

de dispositivos.
leitura de cartão,
como nos drivers
ligações com três dis-
interfaces particulares
comuns aos dispo-

pode mandar um bloco de bytes através da interface sem se preocupar com os outros dispositivos na interface. O canal *selector* só guarda uma *UCW* por vez. No *S/370*, foi projetado um tipo de canal que reúne características do multiplex com o *selector*, chamado *block multiplexer*. A desvantagem do *selector* é que o canal só fica disponível depois de cumprir seus deveres relacionados ao último *CCW* de um programa de canal. Basicamente, o canal foi ampliado para poder executar mais de um programa de canal de uma vez, mas retendo a capacidade de mandar um bloco de bytes de uma vez. No canal *block multiplexer* do *S/370*, guarda-se mais de um *UCW* para executar vários programas de canal. O chaveamento entre programas de canais ocorre só depois de um dispositivo desligar-se da interface, livrando assim a interface. Isso ocorre tipicamente na execução de comandos em que não há transferência de dados. Quando a interface está livre, um dispositivo com programa de canal em andamento pode pedir novamente uma ligação com o canal. O canal *block multiplexer* também pode operar, em modo *selector*, conforme o estado de um *flip-flop* controlado pelo sistema operacional.

g. A interface padronizada

A interface padronizada do *S/360* tem 34 sinais. Esses sinais são agrupados conforme a utilização (Fig. 7-36) em quatro grupos: (1) as vias, que transferem os dados; (2) as linhas

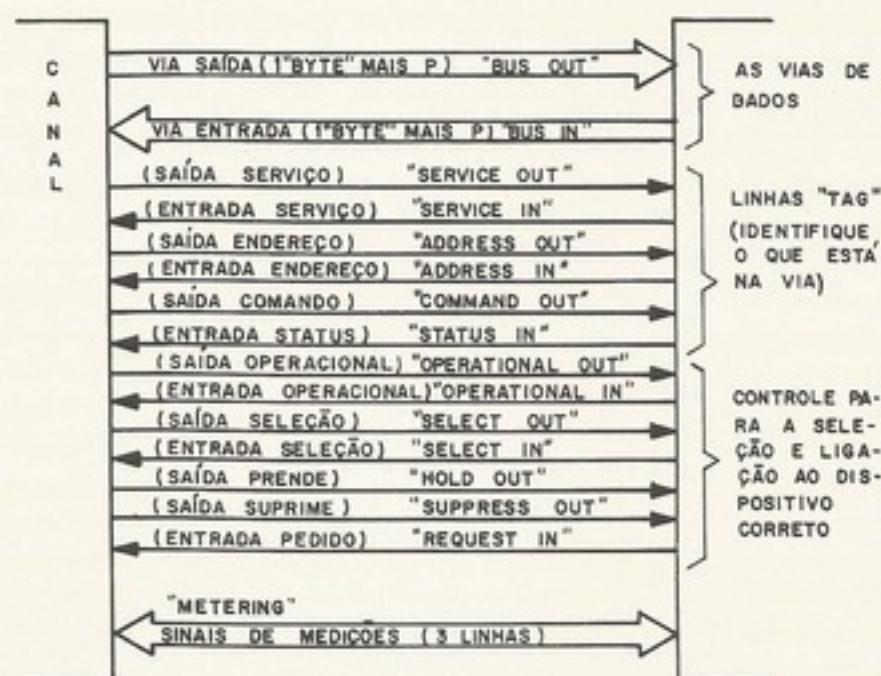


Figura 7-36. Os sinais da interface padronizada

tag, que identificam o tipo de informação (dados, comando, *status*, endereço); (3) as linhas de controle (*selection controls*), que selecionam e "travam" (*inter-lock*) o canal com um dispositivo; e (4) os controles, que operam as medições de utilização que o dispositivo tem. As linhas do tipo *in* e *out* são de entrada e saída respectivamente, em relação ao canal.

O sinal de *select out* passa serialmente através dos dispositivos. O dispositivo que o recebe primeiro tem a prioridade mais alta. O sinal funciona mais ou menos como o sinal *PRH* da interface do *HP 2116 B* (veja a Sec. 7-5).

O *S/370* tem mais uns dois sinais da interface, com a finalidade de poder efetuar a transferência de dados mais rapidamente. Na interface, os sinais de saída são unidirecionais, como na Fig. 7-11. É permitido um "toco" de 6 polegadas (14,5 cm) no máximo. A linha está terminada com resistor de $90\ \Omega$. As linhas de entrada ao canal também são unidirecionais, mas usam a técnica de *DOT-OR*, como visto na via de *skip* da Fig. 7-10. Os *drivers* devem

alimentar, pelo menos, dez receptores, e até dez *drivers* podem participar na *DOT-OR*. A resistência das linhas não podem superar $33\ \Omega$, um fator que limita o comprimento do cabo. A distância mínima entre os receptores é de 3 pés (99 cm).

Pelo fato de ter uma grande variação em *timing* entre canais e dispositivos na família S/360, a sincronização da interface está efetuada através de sinais da resposta, e não se usa relógio-mestre, como no *PDP-8* (Fig. 7-8) ou *HP 2116B*. O *timing* da interface é então "assíncrono".

Para mostrar como funciona uma interface desse tipo, vamos explicar uma sequência típica para a fase inicial de seleção de um dispositivo pelo canal. Normalmente o canal está sempre com *operational out* ativo.

1. O canal coloca o endereço (um byte) na via *out*.
2. O canal ativa o *address out*.
3. O canal ativa o *hold out*, que permite o *select out* passar.
4. O *select out* passa serialmente através dos dispositivos até o dispositivo que reconhece o seu endereço.
5. Esse dispositivo ativa o *operational in*.
6. O dispositivo coloca seu endereço na via *in* e ativa o *address in*.
7. O canal compara a via *out* com a via *in* para verificar que não houve erro. (Agora o dispositivo foi selecionado e podemos passar o comando a ele.) O canal desativa o *address out*.
8. O canal coloca o byte de comando (do *CCW*) na via *out* e ativa o *command out*.
9. O dispositivo desativa o *address in*, coloca seu byte de *status* na via *in* e ativa o *status in*.
10. O canal desativa o *command out*, acusa o *status* e responde com *service out*.

Dependendo do *status*, o canal pode continuar com a execução normal do comando. Como sempre, as dificuldades são encontradas com as condições excepcionais e não com o que "normalmente" acontece.

Podemos concluir essa explicação resumida bem simples dos canais do S/360 e S/370 com alguns comentários de Brooks⁽⁹⁾, um dos arquitetos do S/360. Antes do S/360, Brooks indica que os dispositivos de E/S resultam de um desenvolvimento apropriado da engenharia e que a atenção, casual e atrasada, é prestada ao operador (ligação com o sistema) e os problemas de programação. Ele sente que a maior contribuição do S/360 à arquitetura de E/S foi o desenvolvimento de interfaces padronizadas, elétricas e de programação, para dispositivos. A arquitetura do canal foi projetada para funcionar com um sistema operacional, levando-se em conta o sistema de programação de controle da E/S da época (1963). Mas foi justamente nesse ramo de programação que evoluiu muito, desde então. Brook previu que, quando as técnicas de controle de E/S (na parte do sistema operacional) se estabelecerem, dali a arquitetura (em hardware) do esquema de E/S também pode se modificar e consolidar.

7.7 O PAINEL

a. Função

Normalmente, quando se fala em entrada/saída, pensa-se logo nos canais e nos dispositivos periféricos. Mas existe também uma outra espécie de entrada/saída que é do painel de controle. Em termos de circuitos lógicos, a interface do painel e os controles manuais constam de uma pequena parte do sistema, mas, em termos de função, essa parte é muito importante: Os controles manuais são as chaves e os botões do painel, que, junto com as luzes, são usados pelos engenheiros para manter e depurar o hardware do sistema, pelo operador para controlar o sistema, e pelo programador para depurar seus programas. O painel serve como uma interface entre Homem e máquina, devendo então ser bem projetado. O programador sem experiência, aprendendo a trabalhar com a máquina, vai trocar os botões e tentar fazer coisas não-previstas, e a máquina precisa se defender dessas burrices.

entrada/saída

b. Controle

O painel (bounce) e outros circuitos especiais através de um deve-se tomar efeito de máscara

c. A leitura

A capacidade de leitura de um programa. As mostradas. Registrador de memória registrador de memória Cap. 5.

d. A carga

Quando a "regador" (load ou bounce)

1. Carrinho

pode-se carregar economia de memória

2. Memória

endereçar uma

fica um progra

3. Por inter

de um cartão pa

memória. De

e. Controle

A máquina tem outra. Para que geral, existem de

A máquina tem valores. É um grupo de sequências para instrução única, a parar o relógio, informações, que têm uma espécie de reações de E/S.

No momento não permitindo, mas permite a inserção dos sinais das funções. A máquina é programável T7.

b. Contatos mecânicos

O painel dispõe de chaves e botões, os quais trazem o problema de pulo de contatos (*bounce*) e sujeira ou desgaste dos contatos. Os contatos são selecionados com cuidado e circuitos especiais são usados para "limpar" o ruído dos sinais. Daí, os sinais entram no sistema através de um circuito que sincroniza o sinal com o ciclo da máquina (Cap. 3). Também, deve-se tomar cuidado com a ligação física do painel com a unidade central, para evitar o efeito de ruído acoplado entre fios que correm em paralelo.

c. A leitura e a escrita da memória principal

A capacidade da memória principal em ler e escrever é importante, tanto durante a fase de depuração do protótipo de uma máquina de computação, quanto na depuração do programa. Através das chaves, é desejável que as posições da memória sejam carregadas e mostradas. Para esse fim é necessário indicar a posição da memória, ou seja, carregar o registrador de endereço. Também a escrita e leitura dos registradores centrais (acumulador, registrador de índice) é uma exigência. Para uma indicação de operações desse tipo, veja o Cap. 5.

d. A carga do programa inicial (IPL)

Quando a memória está sem programa, como podemos carregar o programa "carregador" (*loader*) que carrega os outros programas? Essa tarefa chama-se *initial program load* ou *bootstrap*. Estudaremos em seguidas três maneiras de se fazer o *bootstrap*.

1. Carregar à mão. Através das chaves do painel e do comando de escrever na memória, pode-se carregar um programinha que pode carregar outros programas. Essa maneira é econômica, mas trabalhosa.

2. Memória especial de *bootstrap*. Através de uma chave especial no painel, pode-se endereçar uma memória especial do tipo "lê apenas". Guardada na memória "lê apenas" fica um programa carregador.

3. Por hardware. No IBM 1130, através do painel ou *console*, pode-se iniciar a leitura de um cartão perfurado, colocando-se o conteúdo do cartão a partir da localização "0" da memória. Daí, a unidade de controle desvia para localização "0".

e. Controles "roda", "partida" e "pare"

A máquina está projetada para rodar programas, executando instruções uma após da outra. Para que a máquina pare e depois recomece, precisa-se de controles especiais. Em geral, existem duas maneiras de parar a máquina; um botão no painel, e uma instrução "pare". A máquina pára de executar instruções, mas os registradores continuam a armazenar seus valores. É um problema muito interessante que, talvez, possa aproveitar a teoria de máquinas sequenciais para projetar os controles que permitem ao programa parar, rodar na base de instrução única e depois recomeçar rodando. Em alguns sistemas, parar a máquina equivale a parar o relógio-mestre. A desvantagem dessa técnica (*hard stop*) é que o sistema pode perder informações, que são transferidas através da via de entrada/saída. Então, muitos sistemas têm uma espécie de "espere" que é só para a *UCP*, mas que permite a continuação das operações de *E/S*.

No microcomputador do Cap. 5, o sistema tem o estado "pare", que pára a máquina não permitindo interrupções, e também possui o estado de "espere", que pára a máquina, mas permite a interrupção. Nessa máquina, ela pára através de um sinal que inibe a ativação dos sinais das fases. O relógio T_0, \dots, T_7 continua a rodar. Uma outra técnica de parar a máquina é parar o contador do relógio central, por exemplo, deixando a máquina no tempo T_7 .

7.8 SUMÁRIO E CONCLUSÕES

Neste capítulo, estudamos o problema de entrada/saída de computadores. As funções de E/S são: ligar o dispositivo periférico e suas informações à memória primária (armazenamento); e ligar o ser humano ao sistema computacional (E/S verdadeira). Foram expandidas as várias espécies de interface (radial, padronizado, multiplex) com a UCP e as várias técnicas de controlar a E/S (síncrono, assíncrono, programado, canal).

Um assunto importante em relação à E/S é a técnica de interrupção. Para melhor ilustrar os problemas de arquitetura e implementação de um sistema de interrupção, o sistema do minicomputador HP 2116 foi estudado em detalhe. Para se ter uma visão da arquitetura de E/S de um sistema projetado para o processamento de dados em multiprogramação, foi explicado o canal do S/360 da IBM. No final, tratamos brevemente do *console*, que, no entanto, é uma espécie de "dispositivo" de E/S.

EXERCÍCIOS

- 7-1. Qual a vantagem da sobreposição de E/S com processamento?
- 7-2. Projetar um registrador que aceite uma palavra de 8 bits em paralelo, e que coloca a palavra, bit a bit, para um único fio de saída. Use flip-flop tipo D e portas tipo AND-OR.
- 7-3. Explique: (a) interrupção; (b) prioridade; (c) as três fases importantes de E/S.
- 7-4. Na realização do canal num modelo de S/360 de pequeno porte, poderá ele ser implementado quase totalmente por microprograma, utilizando assim a UCP, embora o propósito da arquitetura do canal seja possibilitar a sobreposição de E/S com processamento?

BIBLIOGRAFIA

- (1) John S. Murphy, *Basics of Digital Computers*, Vol. 3, Hayden Book Co., New York, 1970
- (2) *Small Computer Handbook*, Digital Equipment Corp. Maynard, Mass., 1970
- (3) M. J. Mendelson e A. W. England, "The SDS Sigma 7: A Real-Time Sharing Computer", *Proc. FJCC*, pp. 51-64, 1966
- (4) *IBM 1800; Functional Characteristics*, IBM SRL, Form. A26-5918, White Plains, New York, 1966
- (5) *A Pocket Guide to Interfacing HP Computers*, Hewlett-Packard Co., Cupertino, CA, abril de 1970
- (6) *IBM System/360, Principles of Operation*, IBM SRL Form. A22-6821, White Plains, New York, 1967
- (7) *IBM S/360 I/O Interface, Channel to Control Unit, Original Equipment Manufacturer's Information*, IBM SRL, Form. A22-6843, White Plains, New York
- (8) A. Padegs, "Channel Design Considerations", *IBM Systems Journal*, Vol. 3, pp. 165-180, 1964
- (9) F. P. Brooks, Jr., "The Future of Computer Architecture", *IFIPS Congress 1965*, New York pp. 87-91

unidades. As funções primária (armação). Foram expostas UCP e as várias

Para melhor ilustrar a função, o sistema do bloco da arquitetura de multiprogramação, da console, que, no

utiliza a palavra, bit implementado quase a arquitetura do canal

New York, 1970

"Proc. FJCC, New York, 1966 CA, abril de 1970 New York, 1967 Data Processing Information, 1965-180, 1964 New York pp. 87-91

capítulo 8

UNIDADES DE CONTROLE

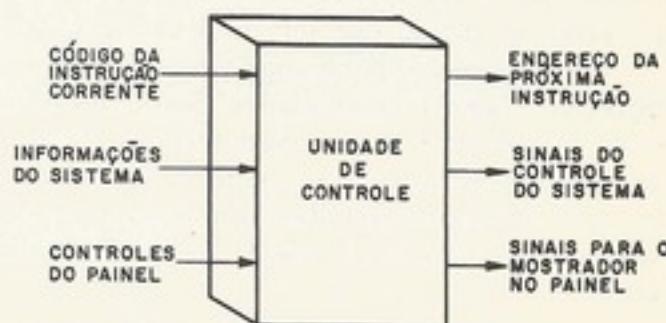
8.1 CONCEITOS GERAIS

De todas as unidades básicas de um computador, falta estudarmos as unidades de controle. Este capítulo é basicamente, o Cap. 3 da dissertação de mestrado de um dos autores^[4].

Como já dissemos, "o fluxo de dados é como uma marionete, com os fios e as linhas de controle disponíveis para quem quiser manobrá-la". É função da unidade de controle atuar sobre essas linhas de modo a gerenciar a execução de um programa em todos os detalhes. O fluxo de dados é uma ferramenta usada pela unidade de controle.

A Fig. 8-1 é um esquema que mostra as entradas e saídas da unidade de controle. Sua função é, conhecendo-se a instrução que está sendo executada, gerar todos os sinais para que essa instrução seja executada no fluxo de dados e, dando prosseguimento ao processo, providenciar a retirada da próxima instrução da memória. O operador do sistema pode alterar o funcionamento normal do sistema através das chaves do painel.

Figura 8-1. Esquema da unidade de controle



Uma instrução é levada a cabo com a execução de uma seqüência de operações elementares características da máquina. Dá-se o nome de microoperações a essas operações elementares. Para exemplificar, podemos dizer que são microoperações: a transferência de dados de um registrador para outro ($A \rightarrow C$); limpar (tornar zero) algum flip-flop ($0 \rightarrow B$); somar um ao conteúdo de um registrador ($CI \leftarrow CI + 1$); ler memória etc.

Uma microoperação é concluída com a conjunção ou seqüência de inúmeros sinais elétricos, corretamente ordenados. Por exemplo, para se executar a microoperação somar um ao conteúdo de um registrador, precisa-se de sinais elétricos que coloquem o conteúdo desse registrador em uma das entradas do somador, coloquem o número um na outra e a saída do somador na entrada do registrador e, depois de algum tempo, copiem a saída do somador no registrador.

Uma das responsabilidades da unidade de controle é gerar os sinais elétricos de modo a realizar a microoperação e, com o conjunto de microoperações na ordem certa, executar

^[4]Edson Fregni, "Projeto lógico da unidade de controle de um minicomputador", Laboratório de Sistemas Digitais, Escola Politécnica da USP, 1972

a instrução. Além disso, a unidade de controle deve estar pronta a responder a qualquer comando do painel ou de qualquer dispositivo de entrada e saída, iniciando, desviando ou parando seu trabalho.

A unidade de controle, em seu trabalho, toma como referência de tempo um intervalo chamado *ciclo da máquina*, que é usualmente estabelecido com referência ao ciclo de memória. Usualmente, divide-se o ciclo da máquina em vários segmentos e cada microoperação dura um ou mais segmentos. Esses segmentos são a menor unidade de tempo do sistema, ou seja, dois eventos não-simultâneos distam um múltiplo desses segmentos. Na Fig. 8-2 vê-se um exemplo de ciclo de máquina dividido em segmentos. O conceito de ciclo de máquina varia de situação a situação e de autor para autor.

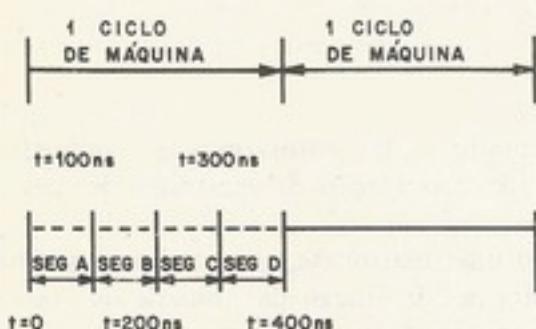


Figura 8-2. Exemplo de um ciclo de máquina dividido em segmentos de tempo

A execução de uma instrução de endereço simples gasta, sistematicamente, dois ciclos de máquina: o primeiro deles, usado para ler essa instrução da memória, é chamado de ciclo *I* (ciclo de instrução ou de *fetch*), e o segundo ciclo, durante o qual se lê o operando da memória e se realiza a operação especificada, é chamado de ciclo *E* (ciclo de execução). É claro que existem instruções com vários ciclos de execução (como a *multiplicação* em um sistema que tenha apenas somadores) e aquelas com nenhum ciclo de execução, como as instruções de desvio que servem apenas para endereçar a próxima instrução.

Por exemplo, suponhamos um computador com um acumulador que deve executar a instrução "soma no acumulador", onde se especifica um endereço cujo conteúdo é o operando que se deve somar. Suponhamos, ainda, que a memória seja magnética, portanto com leitura destrutiva, e necessite-se restaurar o que se lê. A Fig. 8-3 mostra um esquema das microoperações geradas, usando um bloco de atraso esquemático. Imaginemos um pulso entrando em *E* e, à medida que vai passando, vai gerando as microoperações no tempo certo.

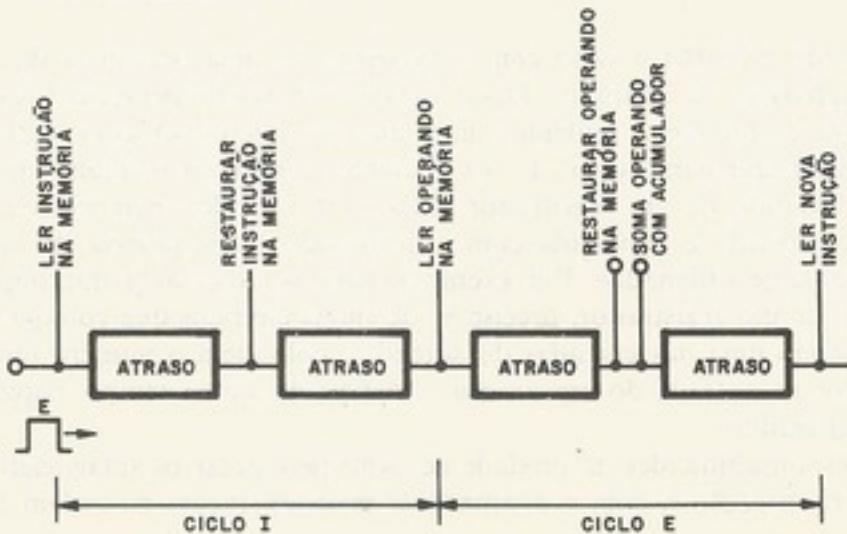


Figura 8-3. Diagrama esquemático das microoperações numa instrução de soma

Notar que o ciclo I é o mesmo para todas as instruções, havendo ramificação no gráfico (Fig. 8-3) apenas no ciclo E, depois da decodificação do código da instrução.

Um modo muito usado atualmente para se descrever a seqüência de microoperações na execução de uma instrução são as chamadas *cartas de microoperações (timing charts)*, onde se fornece um diagrama cuja referência são os ciclos I e E com seus segmentos de tempo. A Fig. 8-4 mostra a carta de microoperação do exemplo anterior.

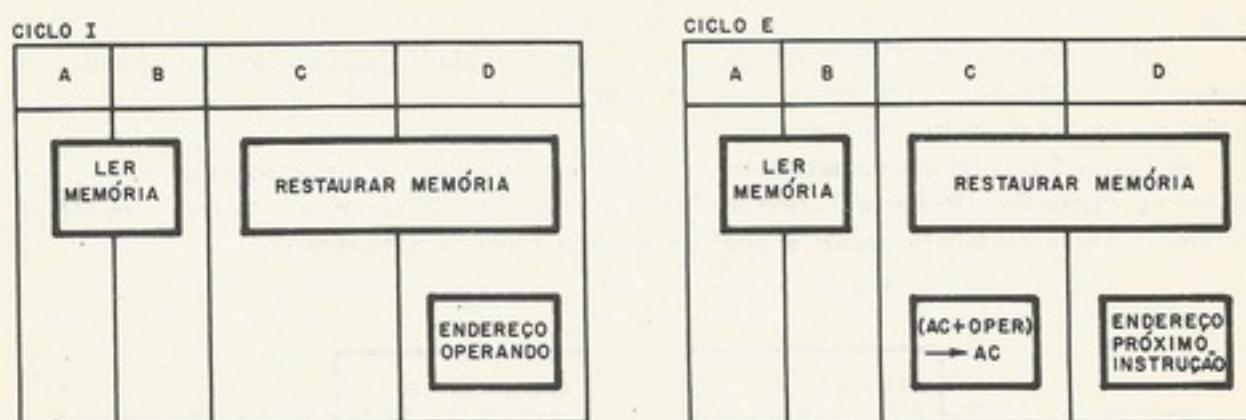


Figura 8-4. Carta das microoperações da instrução de soma do exemplo

Outra função da unidade de controle é zelar pelo estado do sistema. Se qualquer situação anormal ocorrer (erros de paridade, dispositivo de entrada e saída desligados etc.), ela deverá parar a execução, esperando a atuação do operador. Podem-se classificar os controles do painel em dois grupos: os que servem para carregar ou agir sobre um programa ("partida", "pare", "chaves de dados", "carrega endereço" etc.) e aqueles que servem para depurar (*debug*) um programa ("mostra posição", "ciclo único", "instrução única", etc.).

Resumindo, podemos dizer que unidade de controle é a parte do sistema que retira as instruções da memória, seguindo a seqüência do programa, e as executa através da combinação das microoperações.

a. Possíveis classificações das unidades de controle

As unidades de controle classificam-se em dois grupos, *síncrono* e *assíncrono*, e diz-se que, na máquina síncrona, o gerador dos sinais de segmento de tempo (*timing-pulse generator*) produz um conjunto de sinais de controle em cada ciclo e as microoperações são geradas tomando-se apenas esses sinais como referência, enquanto que, na máquina assíncrona, o fim de uma microoperação inicia a seguinte. Contudo muito poucas máquinas são inteiramente síncronas ou assíncronas. A maioria usa uma combinação de ambas as técnicas.

Existe ainda o conceito de unidade de controle *centralizada* e *descentralizada*.

Um sistema de controle centralizado é aquele onde existe uma única seção do sistema que gera todos os sinais de tempo, de controle e de *clock*. É uma organização normalmente usada em minicomputadores que tem a vantagem de ser direta, positiva e de controle pouco complexo. Nos sistemas de controle descentralizado, a unidade de controle é apenas monitora e dirige as várias subseções do computador. Esse método tem a vantagem de as várias porções do sistema poderem operar separadamente em sua própria velocidade ótima.

Quanto à sua implementação, classificam-se as unidades de controle em dois grupos: as de controle fixo (através de circuitos, *hardware*) e as microprogramadas. Essa classificação é importante, já que leva em conta a implementação física e, portanto, depende da tecnologia. Por isso, detalharemos o assunto a seguir.

8.2 CONTROLE FIXO

Em um computador de controle fixo, a unidade de controle é um circuito (*hard-wired*) encarregado da geração dos sinais de controle, a partir do código da instrução corrente. Projeta-se esse circuito para uma dada arquitetura e, quando se quiser modificá-la, será necessário reprojetar a unidade de controle. William Roberts⁽¹⁾ afirma que, "se a unidade de controle é fixada por circuitos (*hard-wired*), o computador funciona melhor em apenas uma das classes de aplicações e menos eficientemente em todas as outras". Outro conceito é o da unidade de controle microprogramada, onde a unidade de controle não é fixada por circuitos e é absolutamente geral, com seu comportamento ditado por sub-rotinas armazenadas na sua memória de controle.

A Fig. 8-5 mostra esquematicamente as partes principais de uma unidade de controle do tipo controle fixo. Roberts⁽¹⁾ apresenta um esquema parecido com esse, porém com menos detalhes. Apresentamos a seguir a análise de cada bloco separadamente.

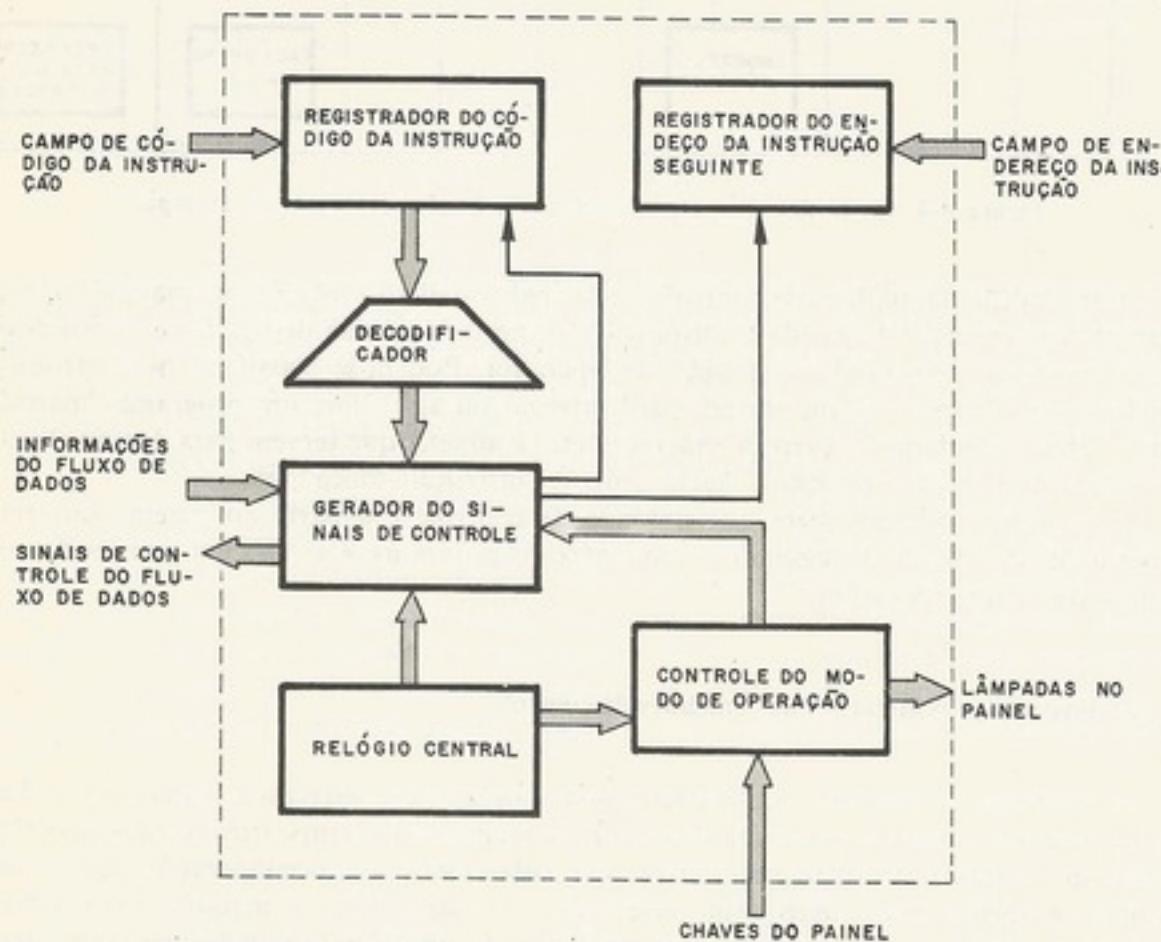


Figura 8-5. Esquema de uma unidade de controle do tipo controle fixo

a. Registrador do código da instrução

Sua função é apenas guardar o código da instrução, para alimentar o decodificador durante o tempo de execução da instrução.

b. Registrador de endereço de instrução

Muitas vezes chamado de contador de instruções (*instruction counter*) ou contador de programa (*program counter*), tem a finalidade de endereçar a instrução seguinte. Em instruções sem desvios (pulos), ele é incrementado de um e, no caso de desvios, é carregado com o campo de endereço da instrução.

c. Decodificador

É um circuito convencional, cujas saídas são as instruções. Existem tantas linhas de saída quantas forem as instruções, e sempre apenas uma das saídas fica no nível "um", indicando a instrução corrente.

O decodificador pode também fornecer saídas especiais que são energizadas para certos grupos de instruções, tais como instruções de testes e instruções de entrada/saída.

d. Gerador de sinais de controle

Tomando como referência os sinais de tempo (ciclo *I* ou ciclo *E* fornecido pelo controle de modo de operação e sinais de segmentos de tempo fornecidos pelo relógio central) e a instrução corrente, o circuito gerador de sinais de controle gera os sinais que irão controlar o fluxo de dados. Projeta-se essa parte da unidade de controle a partir da carta de micro-operações. Ware⁽²⁾ chama essa parte de seqüencializador (*sequencing*) e diz que os sinais gerados "são distribuídos nos lugares certos nas ocasiões corretas através de portas que são abertas pelo decodificador".

e. Relógio central

É um circuito com um oscilador básico que gera sinais que definem segmentos de tempo, conforme se vê na Fig. 8-6. É a referência de tempo para o sistema. Por exemplo, com o sinal *A*, controlam-se os eventos do início do ciclo; com o *B*, os do final da primeira metade, e assim por diante. Portanto pode-se dizer, que o relógio central especifica uma seqüência fixa de posições no tempo dentro do ciclo da máquina.

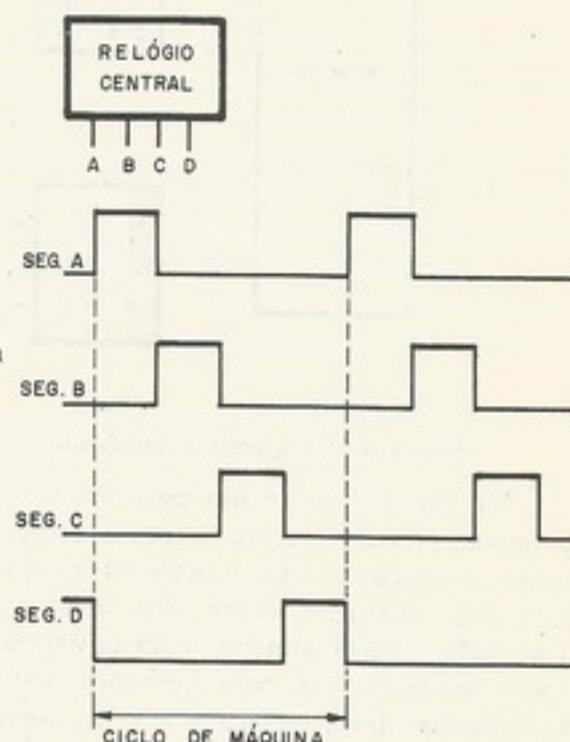


Figura 8-6. Sinais gerados por um relógio central com quatro segmentos de tempo

f. Controle do modo de operação

É a parte de interface entre o sistema e o operador. Por isso, o funcionamento da unidade de controle depende totalmente de sinais enviados por esse bloco. Quando, durante o funcionamento, o operador aciona a chave "pare", esse circuito espera que termine a instrução corrente e pára o sistema. É ele, também, que decide quando é ciclo de instrução ou quando

é de execução. Interrupções provocadas por dispositivos de entrada e saída são atendidas por ele. Esse bloco recebe o nome de controle do modo de operação, pois os modos especiais de funcionamento, como instrução única ou ciclo único, são executados por ele.

Resumindo, podemos dizer que o controle fixo é um circuito projetado para "manobrar" o fluxo de dados na seqüência correta das microoperações. E que recebeu o nome de *fixo* porque não pode ser mudado facilmente, exigindo que seja reprojeto, em caso de alguma alteração.

8.3 UNIDADE DE CONTROLE MICROPROGRAMADA

Em 1951, o matemático inglês M. V. Wilkes do Laboratório Matemático da Universidade de Cambridge apresentou a primeira idéia de microprogramação. Por isso, para introduzir esse assunto, nada melhor que usar as próprias palavras de Wilkes⁽³⁾.

"Em muitas máquinas, o projeto da unidade de controle tem sido obtido através de métodos semi-empíricos, e os circuitos resultantes, apesar de funcionarem, são complexos e não sistemáticos, e poderiam exigir alterações consideráveis se alguma modificação apreciável precisasse ser feita no código de operação da máquina. A finalidade era desenvolver um projeto simples em sua estrutura global e aplicável qualquer que fosse o código de operação da máquina".

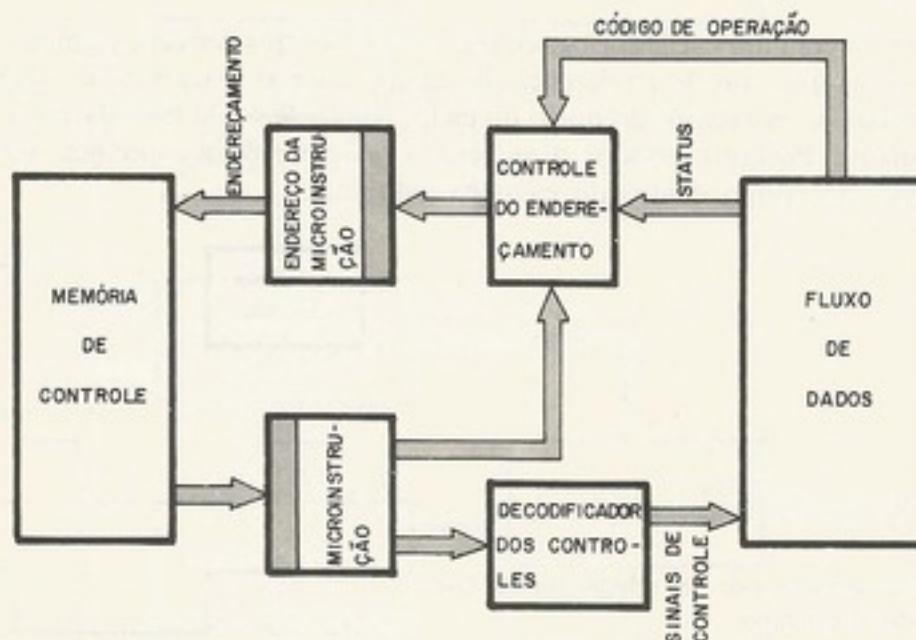


Figura 8-7. Esquema simplificado de uma unidade de controle microprogramada

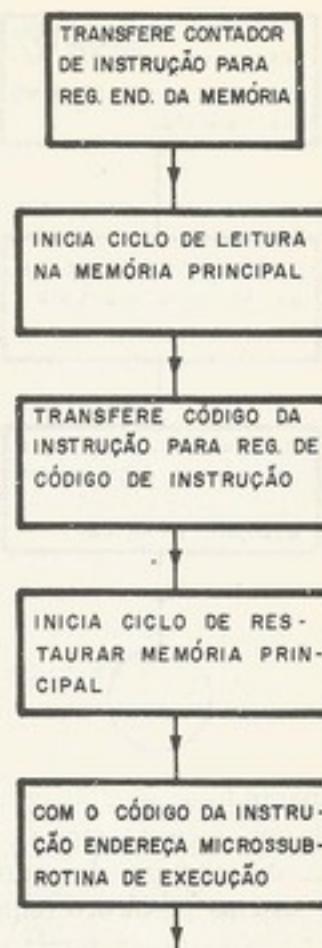
Na Fig. 8-7, vê-se um esquema simplificado de uma unidade de controle microprogramada. Nessa estrutura, usa-se o código de uma instrução, que não mais é decodificado como endereço de uma "microssub-rotina" (sub-rotina na memória de controle). Portanto, para cada instrução, existe uma "microssub-rotina" armazenada na memória de controle, que é lida seqüencialmente, microinstrução a microinstrução, e vai ditando todas as microoperações necessárias para a execução da instrução indicada pelo programador. A vantagem fundamental dessa técnica, é a versatilidade. Podem-se incluir no sistema instruções extremamente complexas, apenas mudando-se a microprogramação da unidade de controle.

Como um exemplo, imaginemos uma instrução de soma do operando no acumulador. O ciclo I, também conhecido como *fase de busca (fetch)*, corresponde a uma microssub-rotina na memória de controle em uma posição definida. Essa sub-rotina executa todos os ciclos I, encarregando-se de ler a instrução na memória principal, de extrair o código e endereçar a microssub-rotina de execução dessa instrução na memória de controle. A Fig. 8-8 apresenta o *fluxograma (flow-chart)* da microssub-rotina da fase de busca do exemplo.

são atendidas
os modos especiais
por ele.
tado para "manobrar"
o nome de fixo
em caso de alguma

único da Universidade
uso, para introduzir
obtido através de
são complexos
modificação apre-
e era desenvolver
o código de ope-

Figura 8-8. Fluxograma do ciclo I num computador micropogramado



Cada bloco nesse fluxograma corresponde a uma microoperação que o fluxo de dados precisa executar e, em nosso exemplo, corresponde à uma microinstrução. Portanto a unidade de controle lê a primeira microinstrução ("transferir CI para RE"), executa-a, e parte para ler a seguinte ("ler memória"); idem e assim por diante. Terminado o ciclo I a microinstrução seguinte já pertence à microssub-rotina de execução mostrada na Fig. 8-9. Poder-se-ia dizer que existem tantas microssub-rotinas quantas forem as instruções do sistema. Mas é claro que, assim como na unidade de controle fixa o decodificador gera sinais comuns a grupos de instruções, existem também trechos de microssub-rotinas comuns a várias instruções.

Note-se a total equivalência entre o fluxograma da microssub-rotina e as cartas de microoperações apresentadas no controle fixo.

Dada a extensão do assunto micropogramação, procurou-se fornecer os elementos essenciais dessa técnica. Um estudo mais detalhado consta do livro de Husson⁽⁴⁾.

Redfield⁽⁵⁾ sugere três medidas que caracterizam a micropogramação: monofásicos/polifásicos, paralelo/série e codificado/não-codificado.

a. Monofásicos e polifásicos

É prática comum, ao se projetar um computador micropogramado, organizar-se o fluxo de dados de tal modo que uma microoperação corresponda na maioria das vezes, a uma volta completa dos dados através da unidade aritmética. Nesse caso, chama-se de *ciclo da máquina* (às vezes, de ciclo do fluxo de dados) o tempo gasto nessa volta. Assim, toda microoperação é executada em um ciclo de máquina, conceito este que, aqui, desvincula-se da memória principal.

Sistemas *monofásicos* são aqueles em que cada microinstrução é executada durante um ciclo de máquina, isto é, para cada ciclo de máquina, é lida uma microinstrução na memória de controle. Nos sistemas *polifásicos*, as microinstruções especificam as microoperações para mais de um ciclo da máquina.

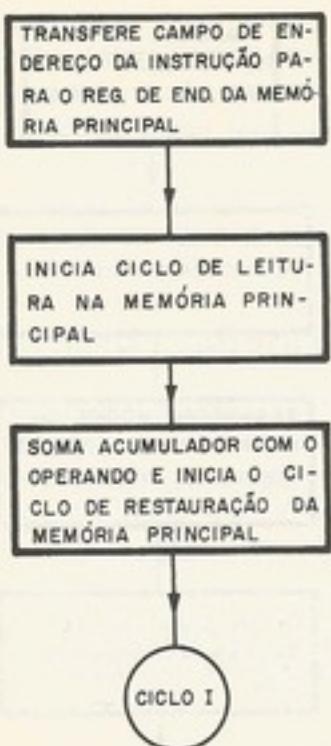


Figura 8-9. Fluxograma da microssubrotina de execução de soma do exemplo

Comparando-se os dois, pode-se dizer que o sistema monofásico exige uma memória de controle com menor tempo de acesso, já que a memória é lida em todo o ciclo da máquina e a palavra da memória de controle especifica as microoperações de um ciclo da máquina apenas. O sistema polifásico requer um menor número de microinstruções, já que cada microinstrução especifica a tarefa de vários ciclos da máquina.

b. Paralelo e série

É um conceito ligado ao tipo de endereçamento da memória de controle. No paralelo, o cálculo do endereço da microinstrução seguinte é feito simultaneamente à execução da microinstrução corrente. No série, é esperado o término da execução da microinstrução para se decidir sobre a próxima. A segunda técnica, apesar de mais lenta, tem a vantagem de não oferecer problemas nos casos de desvios condicionais na microssub-rotina, pois dali não se pode desviar em função do resultado da microoperação corrente.

c. Codificado e não-codificado

Controle não-codificado (horizontal⁽⁶⁾) é aquele onde cada ponto de controle do fluxo de dados corresponde a um bit da palavra de controle. No codificado, pontos de controle do fluxo de dados que nunca são energizados simultaneamente são agrupados e, para cada grupo, existe um campo codificado na palavra de controle que irá indicar qual elemento do grupo deverá ser energizado. Ligados a isso, existem os problemas de "encaixar" dentro do ciclo da máquina o ciclo da memória de controle, o ciclo da memória principal e o ciclo do fluxo de dados, já que eles formam grupos independentes de sinais. Husson⁽⁴⁾ afirma que "mesmo com o custo adicional dos decodificadores de campos, tal organização (codificada) mostrou ser a ótima, sob o aspecto custo e facilidade de programação".

d. Resumo

A unidade de controle microprogramada é uma técnica que substitui os circuitos que geravam os sinais que iriam compor as microoperações, por uma memória onde ficam armazenadas as seqüências de microoperações necessárias para executar cada operação. Vandling⁽⁷⁾ apresenta o exemplo de uma unidade de controle microprogramada, com detalhes.

8.4 COMPARAÇÃO ENTRE AS DUAS TÉCNICAS DE CONTROLE

As vantagens da microprogramação são citadas por inúmeros autores^(1, 5, 7), conforme se expõe a seguir.

a. Facilidade de projeto

É inerente à técnica, dada a sua regularidade. Foi essa característica que provocou seu desenvolvimento.

b. Flexibilidade

Pode ser modificada facilmente, bastando, para tanto, alterar-se o microprograma na unidade de controle.

c. Manutenção

As falhas são facilmente encontradas, provocadas pela regularidade. São possíveis ainda microprogramas de diagnose^(8, 9) para ajudar na manutenção do sistema completo.

d. Velocidade

Alguns autores⁽¹⁰⁾ admitem que, quando um usuário puder adaptar o seu repertório de instruções a suas necessidades próprias, ele conseguirá muito maior velocidade do sistema. O próprio Wilkes⁽¹¹⁾ mostrou-se receoso com essa técnica ao afirmar que "o uso de memória 'ler-escrever' no lugar de 'ler' somente para armazenar os microprogramas de tal forma que o programador possa montar seus próprios microprogramas foi mencionado como uma possibilidade interessante, mas eu duvido que um computador projetado dessa forma seja realmente necessário".

e. Emulação

A vantagem básica de um sistema microprogramado é ele poder *emular*^(4, 6) outras arquiteturas; isto é, um sistema microprogramado pode ter seu repertório de instruções adaptado de modo a simular totalmente uma outra arquitetura.

Mas, apesar de todas essas vantagens, deve-se ter em mente as palavras de Flynn⁽¹²⁾ ao afirmar que "a diferença significativa entre máquinas de controle convencional e microprogramadas está na *potência das instruções*". Isto é, sistemas com instruções bastante complexas devem ser microprogramados, ao passo que aqueles cujas instruções são bastante simples, como a maioria dos minicomputadores (instruções de endereço simples, com segundo operando implicado), devem ter sua unidade de controle do tipo controle fixo. É, economicamente, uma solução melhor.

Husson⁽⁴⁾ sugere que, quanto maior for o sistema, maior vantagem terá em ser microprogramado, e analisa as unidades de controle dos dois sistemas comerciais que pela primeira vez utilizaram a microprogramação, sistema IBM/360 e o RCA Spectra 70. Quanto ao sistema IBM/360, modelo 50, Schnabel⁽¹³⁾ não vacila ao concluir pelas vantagens, tanto em custo quanto em performance, da microprogramação. Mas esses sistemas são de grande porte. Redfield⁽⁵⁾, analisando um sistema de porte médio, o Univac C/SP concluiu pelas vantagens na *performance* da microprogramação, já que o custo era o mesmo. Isso mostra que nada tem de estranho o fato de a maioria dos minicomputadores terem controle fixo.

Langley⁽¹⁴⁾ procura mostrar que, com o uso de circuitos tipo *LSI* (*large scale integration*) a balança agora pendeu para o lado microprogramação. Ele mostra um minicomputador que utiliza *LSI* com a microprogramação como uma solução econômica. Isso se explica pelo fato de que se pode utilizar um único fluxo de dados em inúmeras funções diferentes:

controle da memória de controle, fluxo de dados propriamente dito, entrada/saída etc. Assim, o custo extra de circuitos do tipo fluxo de dados diminui. E mais, a memória de controle monolítica reuniu duas vantagens, baixo custo e a alta velocidade.

Porém, quando se pensa em uma pastilha de integrado contendo toda a unidade central, a microprogramação também não deixa de ser uma solução viável se a memória de controle consegue utilizar eficientemente a área disponível.

Sempre que existem duas alternativas com vantagens e desvantagens próprias, costuma-se questionar sobre uma solução intermediária, ou seja, uma unidade de controle que reúna as vantagens das duas técnicas. O modelo 145 do sistema IBM/370, o HP 2100 e outros, adotaram uma solução interessante: já que o ciclo *I* é sempre o mesmo, eles o fixaram fixo, deixando os ciclos de execução para serem microprogramados. Vê-se então que a seqüencialização do programa é feita com a técnica de controle fixo e a execução por microprogramação. Aqui, pode-se ir mais longe; pode-se pensar num computador sem um conjunto fixo de instruções, que seriam controladas por uma unidade de controle fixo; seriam reservados alguns códigos de instruções especiais, para serem microprogramados. Assim, poder-se-ia obter grande velocidade de execução das instruções comuns e uma memória de controle lenta para as instruções especiais.

Pode-se encontrar, além da referência⁽³⁾, um estudo bibliográfico organizado historicamente pelo próprio autor da idéia de microprogramação: Wilkes⁽¹¹⁾.

BIBLIOGRAFIA

- (1) William Roberts, "Microprogramming Concepts and Advantages as Applied to Small Digital Computers", *Computer Design*, Vol. 8, N.º 11, pp. 147-150, novembro de 1969
- (2) Willis H. Ware, *Digital Computer Technology and Design — Circuits and Machine Design*, Vol. 2, John Wiley and Sons, 1966
- (3) M. V. Wilkes et al., "The Design of the Control Unit of an Electronic Digital Computer", *Proceedings of the Institution of Electrical Engineers*, Vol. 105, parte B, N.º 20, pp. 121-128, março de 1958
- (4) Samir S. Husson, *Microprogramming — Principles and Practices*, Prentice-Hall, Inc., N.J., 1970
- (5) Stephen R. Redfield, "A Study in Microprogrammed Processors: A Medium Sized Microprogrammed Processor", *IEEE Transactions on Computers*, Vol. C-20, N.º 7, pp. 743-750, julho de 1971
- (6) Robert F. Rosin, "Contemporary Concepts of Microprogramming and Emulation", *Computing Surveys*, Vol. 1, N.º 4, pp. 199-212, dezembro de 1969
- (7) Gilbert Vandling e D. Waldecker, "The Microprogram Control Technique For Digital Logic Design", *Computer Design*, Vol. 8, N.º 8, pp. 44-51, agosto de 1969
- (8) A. M. Johnson, "The Microdiagnostics for the IBM System/360 Model 30", *IEEE Transactions on Computer*, Vol. C-20, N.º 7, pp. 298-303, julho de 1971
- (9) Ronald M. Guffin, "Microdiagnostics for the Standard Computer MLP-500 Processor", *IEEE Transactions on Computer*, Vol. C-20, N.º 7, pp. 803-808, julho de 1971
- (10) A. Grasselli, "The Design of Program-Modifiable Microprogrammed Control Units", *IRE Transactions on Electronic Computers*, pp. 336-339, junho de 1962
- (11) M. V. Wilkes, "The Growth of Interest in Microprogramming: A Literature Survey", *Computing Surveys*, Vol. 1, N.º 3, pp. 139-145, setembro de 1969
- (12) M. J. Flynn e R. F. Rosin, "Microprogramming: An Introduction and a Viewpoint", *IEEE Transactions on Computer*, Vol. C-20, N.º 7, pp. 727-731, julho de 1971
- (13) Dorothy L. Schnabel, "The Design of Processor Control Using a Read-Only Storage", *IBM Technical Report*, TROO. 1318, agosto de 1965
- (14) Frank J. Langley, "Small Computer Design Using Microprogramming and Multifunction LSI Arrays", *Computer Design*, Vol. 9, N.º 4, pp. 151-157, abril de 1970

apêndice

OS MICROCOMPUTADORES

A organização deste apêndice começa no genérico e se particulariza no final. Nas duas primeiras seções, procura-se descrever os microprocessadores em geral, suas qualidades e defeitos, tentando tirar o máximo das qualidades e superar os defeitos. A Sec. A.3 fornece um roteiro de projeto, mostrando quais as ferramentas e técnicas existentes para amparar o projetista.

A Sec. A.4 descreve, em detalhes, um microprocessador que tem sido muito usado: o INTEL 8080. A última parte do apêndice é dedicada à descrição de casos de aplicação, sob duas formas: exemplo e projetos. No exemplo, as soluções são apresentadas e, nos projetos, são propostos alguns problemas. Mesmo que o leitor não vá desenvolver os projetos, recomenda-se que os leia, já que tal material foi preparado para orientar o projetista na solução dos problemas.

A.1 INTRODUÇÃO⁽¹⁾

Em 1965, quando a família dos computadores eletrônicos digitais completava vinte anos de existência, uma empresa norte-americana, a DEC — Digital Equipment Corporation —, lançava no mercado o primeiro minicomputador, o PDP-5. Desde então, os minicomputadores desenvolveram-se, multiplicaram-se e seu mercado evoluiu de forma surpreendente.

Com os minicomputadores, aplicações nunca antes sonhadas tornaram-se realidade. Já não eram mais necessários investimentos vultosos para se dispor da rapidez, versatilidade e volume de trabalho de um computador. Mais que isso, a inovação ampliou os rumos da utilização do computador, que saiu do ambiente de ar condicionado dos *bureau* de processamento de dados e entrou nas oficinas, nos aviões, e mesmo nos supermercados, deixando de restringir seu diálogo com fitas magnéticas, discos e cartões, e passando a se comunicar também com sensores de temperatura, motores e relés.

O mundo técnico começou a encarar cada vez menos o computador como um “cérebro eletrônico”, e cada vez mais como uma máquina automática programável. Foi então que os computadores passaram a ser apenas um *componente* de um sistema de computação, já que seu custo era ínfimo comparado ao do sistema completo. Uma máquina automática controlada por um minicomputador era normalmente dezenas de vezes mais dispendiosa que o próprio computador.

Em 1970, a Intel Corporation, norte-americana, iniciava o desenvolvimento do 4004, o primeiro microprocessador. Era um processador completo, inserido em uma pastilha de circuito integrado, custando, na época, cerca de US\$ 100,00. O grande mercado que se previa então para o microprocessador era o das calculadoras. Tanto assim, que a Intel desenvolveu seu produto com o financiamento de uma empresa japonesa, chamada Busicom, que passou a fabricar uma calculadora com impressora, colocada no mercado em 1971.

Mas o enorme potencial dos microprocessadores não tardou a ser descoberto. Existia um sem-número de aplicações. O único problema do 4004 era sua limitada palavra, com 4 bits de largura, adequada para operar com algarismos decimais, porém, problemática para ser

usada em funções onde precisava manusear caracteres alfa-numéricos. Em 1972, a mesma empresa lançou o modelo 8008, muito parecido com seu predecessor, com a diferença de operar com 8 bits em paralelo. Foi logo encontrada uma aplicação imediata, até então inédita, para esse novo componente: os terminais ponto-de-venda "inteligentes" — uma caixa registradora que imprime a nota fiscal e calcula impostos, automaticamente.

O sucesso comercial dos microprocessadores foi tão grande que outras empresas, como a Fairchild, seguiram o exemplo da Intel. Atualmente, o número de empresas norte-americanas fabricantes de microprocessadores chega perto de uma centena, produzindo cerca de duzentos modelos diferentes.

aplicação dedicada

O ponto-chave na caracterização dos microcomputadores é a aplicação dedicada. Nos sistemas que os incorporam, os programas normalmente ficam numa memória fixa (*read only*) que não é alterada. Isso equivale a dizer que a tarefa (programa) que o microcomputador irá executar é definida no projeto do sistema e não é alterada.

A incorporação de um microprocessador aos circuitos de controle de elevadores, por exemplo (tradicionalmente projetados com relés, transistores ou circuitos integrados digitais convencionais), pode acrescentar ao controlador funções bem mais complexas, inclusive de monitoração e alarme, por um custo mais baixo. Nesse caso, a tarefa do microprocessador é fixa e determinada na hora do projeto (aplicação dedicada). Outro aspecto importante é a versatilidade: com a simples mudança dos programas, a mesma unidade pode ser usada para controlar um, dois ou mais elevadores.

Desde que surgiu pela primeira vez, o microprocessador vem adquirindo novas e importantes aplicações, que hoje se estendem a inúmeros setores de atividades. Com certeza, seu enorme potencial ainda está longe de esgotar-se. Ao contrário, sua utilização ganha, a cada dia que passa, renovada força.

Foi nas aplicações de *controle industrial* que os microprocessadores conseguiram penetrar causando maior impacto. O engenheiro de controle tem procurado se adaptar rapidamente à nova tecnologia, que lhe oferece os recursos de um computador convencional, sem apresentar, no entanto, a complexidade e também o alto custo, próprios desses sistemas.

Com efeito, o grande sucesso do controlador programável deve-se à redução drástica de seu custo. Os fabricantes especializados nesses equipamentos têm procurado configurar-lhes uma forma versátil, para permitir sua utilização em muitas aplicações de controle industrial: basta mudar o programa, que o controlador se comporta de forma diferente. Além disso, outro grande trunfo do controlador programável é sua programação, feita em uma linguagem adequada ao pessoal técnico familiarizado com o campo de controle industrial. Como exemplos específicos nessa área, podem ser mencionados os controles de elevadores, de perfis de temperatura em fornos, de processos industriais não muito sofisticados e de máquinas operatrizes.

A aquisição de dados é outra área da indústria que se beneficiou com a introdução dos microprocessadores. Nesse particular, as aplicações vão desde a coleta de dados para efeitos administrativos e de controle de produção (relógios de tarefa, por exemplo), que é feita de forma centralizada, com os dados sendo armazenados em uma fita magnética e impressos automaticamente em forma de relatório diário, até o controle de estoques e contabilidade.

Ainda no campo das atividades industriais, os microprocessadores estão sendo usados também em sistemas de controle de incêndio, de proteção contra intrusos (monitoração de portas e janelas) e de controle dos equipamentos ligados. Tais sistemas são "inteligentes", pois têm a capacidade de intervir em momentos críticos, fazendo uma avaliação da situação e tomando as providências cabíveis.

Nem mesmo do *processamento de dados* o novo componente ficou de fora. Ainda mais que a tendência atual desse setor é a de expandir na direção da "computação distribuída". Com os microprocessadores, tornou-se economicamente inviável ter-se, na unidade central,

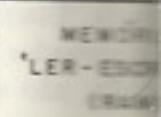
a fonte de todos os dados. Têm alguma "inteligência" em termos desse pré-processamento de texto, a verificação de dados, recebidos ao computador.

Os exemplos nos terminais point-of-sale, sistemas de encerramento de fechaduras em lojas e nas caixas bancárias.

A.2 A ARQUITETURA

Um microcomputador é formado por:

- microprocessador;
- memória central;
- etc. Em geral, o sistema é composto por:



a. Os componentes

1. Memória "inteligente"

A memória "inteligente" é aquela que tem alguma "inteligência" em termos desse pré-processamento de texto, a verificação de dados, recebidos ao computador.

Em 1972, a mesma
diferença de operar
em rede inédita, para
uma registradora
de empresas, como a
norte-americana
de duzentos

dedicada. Nos
fixa (*read only*)
microcomputador irá

elevadores, por
integridade digitais
inclusive de
microprocessador é
importante é a
ser usada para

novas e impor-
tante. Com certeza, seu
ganhos, a cada

conquistaram penetrar
rápidamente
sem apre-
sos sistemas.

redução drástica de
configurar-lhes
controle industrial:

Além disso, outro
linguagem ade-
m. Como exemplos

de perfis de tem-
operatrizes.
a introdução dos
dados para efeitos
que é feita de
impressos
contabilidade.

sendo usados
(monitoração de
“inteligentes”,
análise da situação

Ainda mais
“distribuída”.
a unidade central,

a fonte de todas as decisões e de controle. Na computação distribuída, as unidades periféricas têm alguma “inteligência”, e parte do processamento é realizada no local. Os terminais “inteligentes” em transmissão de dados realizam um pré-processamento antes de enviá-los à frente; esse pré-processamento envolve a formatação, a ajuda ao operador na forma de edição de texto, a verificação, etc. O microprocessador pode ainda realizar as funções de concentrador de dados, recebendo informações de várias fontes e transmitindo-as, após o pré-processamento, ao computador central, em um único canal.

Os exemplos de aplicações se multiplicam: na instrumentação, nas máquinas vendedoras, nos terminais ponto-de-venda, nos PABX (CPA), nos sistemas de entretenimento (jogos), nos sistemas de ensino (para escolas), nas balanças para supermercado, no controle de segredos de fechaduras em hotéis, na análise clínica, no controle de semáforos, nas máquinas para votar e nas caixas bancárias automáticas.

A.2 A ARQUITETURA DOS MICROCOMPUTADORES

Um microcomputador é um sistema eletrônico que incorpora um microprocessador. O microprocessador é a unidade central, ao redor da qual existem memórias, interfaces, relógios centrais, etc. Em geral, um microcomputador tem os elementos vistos na Fig. A.1.

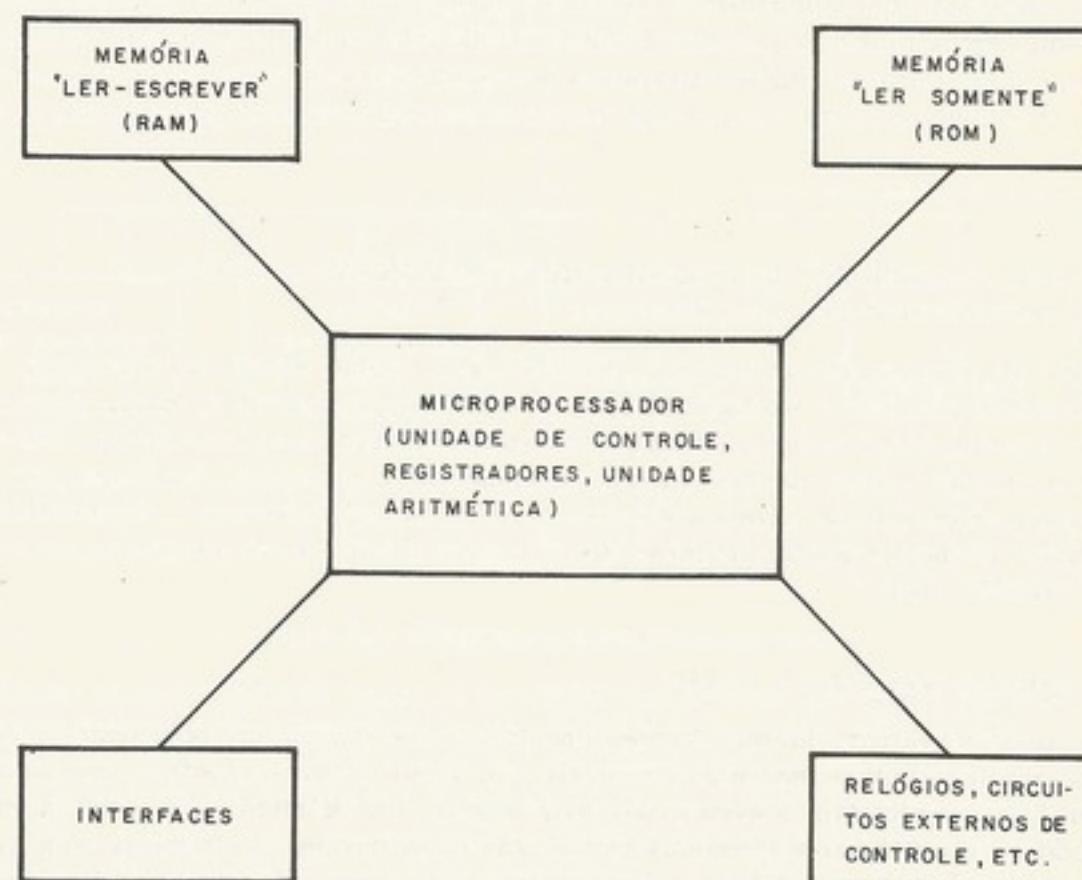


Figura A.1. Os componentes de um microcomputador

a. Os componentes

1. Memória “ler somente”

A memória “ler somente” (*ROM*) é o local onde se armazenam os programas e dados fixos. Como já foi dito, um microcomputador tem seu forte na aplicação dedicada. Dificilmente se vê um microcomputador de uso geral, pois essa função é normalmente realizada pelos mi-

nicomputadores. Na aplicação dedicada, os programas são fixos, já que o microcomputador passará o resto de seus dias executando a mesma tarefa. A vantagem de utilização de memórias tipo "ler somente", com tais programas é que o usuário não tem que se preocupar em carregar o programa cada vez que liga o sistema. Essas memórias não são voláteis, isto é, não perdem seu conteúdo quando a alimentação é desligada.

Existem três tipos de memória "ler somente": ROM, PROM e EPROM.

Memória "ler somente" (ROM — read only memory)

Nesse tipo, a programação da memória é feita durante sua fabricação. Seu uso só se justifica em produções de larga escala, e, por isso, são normalmente utilizadas em tarefas muito comuns. Os fabricantes dos microcomputadores fornecem memórias ROM, com programas padrões (pequenos monitores, controladores de TTY, etc.).

Memória "ler somente" programável (PROM — programmable read only memory)

As memórias programáveis saem da fábrica com "1" (ou zero) armazenado em todas as suas posições. O usuário (fabricante do microcomputador, por exemplo) consegue programá-la, utilizando um equipamento especial, o programador de ROM, que, sob controle do operador, envia pulsos de corrente à memória que está sendo programada. Esses pulsos de corrente irão "queimar" fuzíveis internos, que equivalerão a informações gravadas na memória. Nessa memória, uma vez gravado um certo conteúdo em uma posição, não se pode mudá-lo.

Memória "ler somente" programável e apagável. (EPROM — erasable programmable read-only memory)

Esse é um tipo especial de memória "PROM", que, mesmo depois de programada, pode ter seu conteúdo alterado. A operação de apagar a memória, normalmente, requer que se expõa seu circuito, através de uma janela existente na embalagem, à luz ultravioleta. Essa exposição recompõe os fuzíveis que haviam sido "queimados", e a memória fica limpa para nova programação. Este tipo de memória é o que deve ser utilizado durante o desenvolvimento da aplicação, já que, nessa fase, devem-se esperar muitas alterações nos programas, até se conseguir a solução final.

2. Memória "ler-escrever" (RAM)

Os dados que variam durante o processamento (tabelas de variáveis, por exemplo), devem ficar armazenados em uma memória "ler-escrever" de acesso aleatório (RAM — random access memory). Essa é a memória convencional, onde se grava e se lê informação. Com os microprocessadores, é usual haver memórias monolíticas (uma pastilha de 16 pinos pode conter alguns milhares de bits de memória). Nas aplicações onde não se tolera a volatilidade das memórias eletrônicas, ainda se usam memórias de núcleo magnético (Cap. 4).

3. Interfaces

As interfaces são sempre dedicadas aos equipamentos aos quais elas se ligam. Seu projeto, em geral, é feito pela equipe que desenvolve a aplicação. Para facilitar essa tarefa, os fabricantes de microprocessadores fornecem controladores de interface (em uma pastilha de circuito integrado), que, de um lado, ligam-se ao processador e, do outro, à interface. Esse componente facilita bastante o desenvolvimento das interfaces, pois ele é o responsável pela

aplicação...
comunicação...
Um aspecto...
ele realiza...

b. A estru...

A estrutura...
reço (Fig. A.1)...
recente na figura...
da via de dígitos...
A capacidade...
ditada pela...
Em estrutu...
1) o m...
a cada instan...
2) um dos...
o processador...

MEMÓRIA
"LER SOMENTE"
(ROM)

Como as...
são comunica...
das vias. Nas...
sem a partic...
trole. Ness...

Observe-se...
as partes sig...
genéricos nas...
para a adminis...
veio ao encon...
mília de comp...
memórias "ler...
etc. Esses comp...
de família é um...
cessador para...

comunicação com o processador e, eventualmente, com a memória (no caso de acesso direto). Um aspecto importante, que qualifica um microprocessador a uma aplicação, é a forma que ele realiza as funções de entrada e saída.

b. A estrutura

A estrutura mais comum de um microcomputador é organizada em vias de dados e endereço (Fig. A.2), às quais se ligam os vários componentes. As linhas de controle (que não aparecem na figura) são comandadas pelo processador, com a ajuda de circuitos externos. A largura da via de dados, que varia de 4 a 16 bits, determina o tamanho da palavra do microprocessador. A capacidade máxima de endereçamento direto (sem uso de bancos de memória) é, em geral, ditada pela largura da via de endereços.

Em estruturas como essa, existem dois aspectos básicos na administração geral:

- 1) o microprocessador controla as vias, informando aos demais elementos o que fazer a cada instante;
- 2) um dos elementos requisita o controle das vias para se comunicar com outro, que não é o processador.

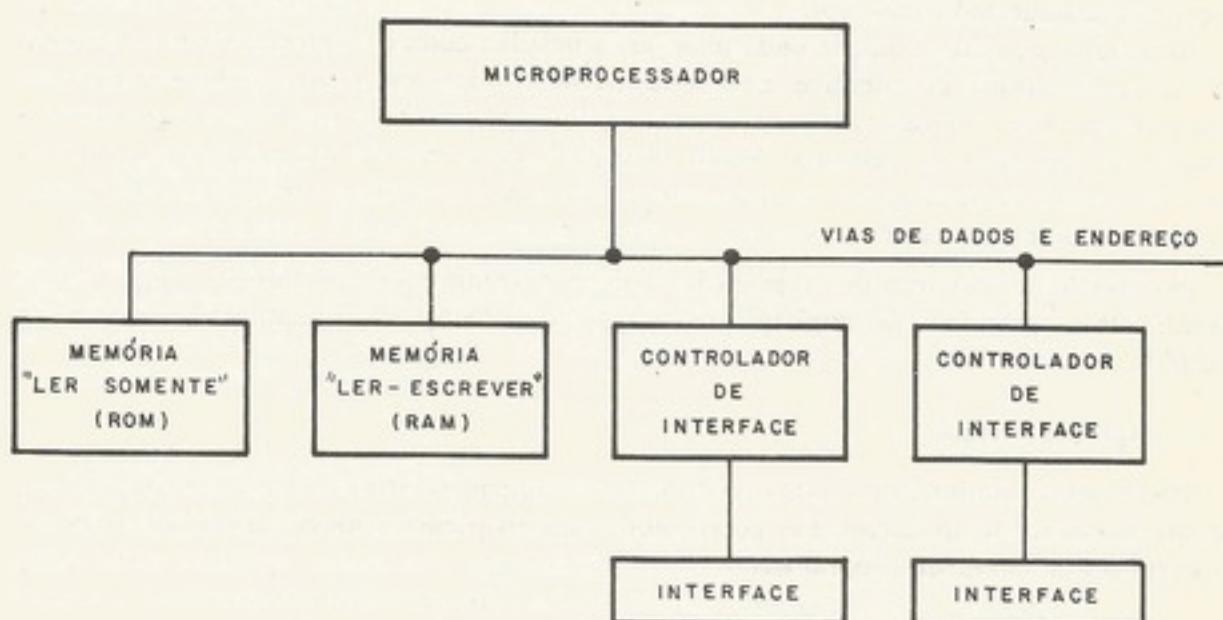


Figura A.2. A estrutura de um microcomputador

Como na maior parte do tempo as transações que ocorrem nas vias de endereço e dados são comunicações entre o processador e demais unidades, é natural que este tenha o controle das vias. Nas raras situações em que as vias são usadas para transações entre dois elementos sem a participação do processador, é necessário que um desses elementos requeira seu controle. Nessa situação, o processador sinaliza a permissão de uso das vias e as libera.

Observe-se, então, que, para que exista harmonia no conjunto, é necessário que todas as partes sigam um dado protocolo. É esse ponto que dificulta a utilização de componentes genéricos nas vias de dados e endereço. Isso porque, deve-se juntar a eles eletrônica necessária para a administração das vias e para comunicação com o processador. O conceito de "família" veio ao encontro da facilidade de projeto. É comum existir junto com o processador uma família de componentes que se interligam diretamente a ele. Essa família tem como membros memórias "ler escrever", memórias "ler somente", controladores de interfaces (vários tipos), etc. Esses componentes seguem o protocolo de comunicação ditado pelo processador. O aspecto de família é um ponto importante a ser examinado pelo projetista ao escolher um microprocessador para sua aplicação.

c. A partição

Nas diferentes alternativas de microcomputadores, um ponto importante que as distingue é a forma que suas partes são distribuídas em circuitos integrados (partição). Distinguem-se aqui três filosofias básicas.

1. A forma mais geral é aquela onde o processador é dividido em várias pastilhas. As partes que compõem o processador são, em geral, os registradores e a unidade aritmética (fluxo de dados) em uma pastilha, e a unidade de controle em outra. Essa solução permite que, pela associação de vários circuitos de fluxo de dados, tenham-se microcomputadores de largura de palavra variável. Por outro lado, a unidade de controle, separada do fluxo de dados, pode ser microprogramada, o que, em algumas aplicações, pode ser muito útil.

2. A alternativa que se encontra mais freqüentemente é aquela onde o microprocessador compõe uma única pastilha de circuito integrado. Essa é uma solução direta onde as variáveis sobre as quais se tem controle é o tamanho e o conteúdo da memória principal. Em geral, sua unidade de controle é fixa. Isso libera o projetista da tarefa de definir o conjunto de instruções. Na maioria das aplicações, as limitações impostas nesse caso (largura de palavra definida, e unidade de controle fixa) são toleráveis, podendo-se, portanto, usufruir as vantagens inerentes a ela (simplicidade na utilização).

3. O outro extremo é aquele onde uma única pastilha contém o processador, a memória e até os controladores de interface. Este seria, então, o microcomputador completo em uma única pastilha. Essa solução é disponível para sistemas pequenos, já que, nesse caso, existe a limitação de tamanho de memória. Maior quantidade de memória exige circuitos externos. Como um grande mercado para os microcomputadores é o controle de máquinas operatrizes, essa alternativa é promissora.

Em conclusão, no tocante à partição, tem-se o seguinte compromisso: quanto maior versatilidade e capacidade de ampliação se deseja, tanto maior será a quantidade de circuitos utilizados.

d. Aspectos internos

Em sua estrutura interna, o microprocessador é equipado com certos recursos que aumentam seu potencial de aplicação. Em geral, muitos desses pontos não são essenciais, mas seu uso simplifica a tarefa de programação.

1. Pilha

A utilização de uma *pilha* (*stack*) para armazenar dados, com instruções especiais para administrá-la, é um conceito que ganhou força com esse novo componente. Pilha é uma organização de memória onde o último dado armazenado é o primeiro a ser lido, em contraposição a *fila* (*queue*) onde o último dado armazenado é o último a ser lido.

Com a pilha, resolvem-se facilmente os problemas de desvios para sub-rotinas, com transmissão automática de parâmetros. O atendimento de interrupção também utiliza a pilha para armazenar o endereço de retorno e o conteúdo dos registradores (contexto). Essa pilha pode existir dentro do fluxo de dados, construída com registradores especiais, sendo, portanto, limitada. A pilha é chamada de "infinita" quando ela é parte da memória principal. No fluxo de dados existe um registrador especial, o ponteiro da pilha, que indica seu topo. Quando se armazena algo na pilha, o ponteiro é automaticamente incrementado. Ao se extrairem dados da pilha, o ponteiro é decrementado.

2. Memória local

Outro item encontrado nos microprocessadores é a memória local. Essa memória é composta de alguns registradores (4 a 64) de uso geral, com instruções especiais para manuseá-los.

Como o aces...
cessamento...
lentas), já q...
memória...
deve ser p...
salvar o co...

3. Acess...
O aces...
ração das v...

4. Interru...
A interru...
um amplo es...
que endereç...
ser decidid...
fontes não c...
interrupção...

5. A ap...
Os requi...
da aplicação...
trônica: com...
de vista das a...
em quatro cl...

Aquisição...
Nessa aq...
ração so...
rupção e fáci...

Comunica...
Aqui o p...
de procu...
linhas de dadi...

Interface...
Os poss...
alta confiabili...
pequeno s...
m...

Computa...
Nessa c...
rações numér...
com memória...

Como o acesso à memória local é rápido, seu uso permite que se ganhe muito tempo de processamento. Reduzem-se bastante as tramações entre o processador e a memória (que são lentas), já que os dados que estão sendo freqüentemente utilizados podem ser mantidos na memória local. Nas soluções onde existam interrupções freqüentes, o uso da memória local deve ser planejado, pois, a cada interrupção (que pode exigir baixo tempo de resposta), deve-se salvar o conteúdo dos registradores locais na memória (na pilha, se existir).

3. Acesso direto à memória

O acesso direto à memória é normalmente oferecido pelos processadores, através da liberação das vias de dado e endereço sob requisição externa.

4. Interrupção

A interrupção é um aspecto importante em inúmeras aplicações em tempo real. Existe um amplo espectro de alternativas. O mais comum é haver várias fontes (em uma única linha), que endereçam diretamente suas rotinas de tratamento (*vectored*). Também é usual a prioridade ser decidida por circuitos externos. Quando se tem uma única linha de interrupção, onde as fontes não endereçam suas rotinas, o tempo de resposta pode ser longo, pois o programa de interrupção tem que descobrir a fonte, testando cada uma delas.

5. As aplicações

Os requisitos para o bom funcionamento de um microprocessador são muito dependentes da aplicação. Como já foi dito, as aplicações são inúmeras, varrendo todo o campo da eletrônica: comunicação, instrumentação, controle, processamento de dados, etc. Sob o ponto de vista das qualidades que se busca em um processador, as aplicações podem ser agrupadas em quatro classes⁽²⁾, como segue.

Aquisição de dados e controle

Nessa aplicação, o processador deve ter: palavra de tamanho grande, habilidade de operação sobre números, velocidade, capacidade de reagir em tempo real, bom sistema de interrupção e facilidade de comunicação (interface) com fontes de sinais analógicos.

Comunicação em dados (*data communication*)

Aqui o processador deve oferecer: tratamento de dados em alta velocidade, bom esquema de procura em arquivos, geração e detecção de códigos corretores de erro, e interfaces com linhas de dados em série.

Interface com o ser humano (terminais "inteligentes")

Os pontos importantes nessa aplicação são: baixo custo, baixo número de componentes, alta confiabilidade, capacidade de aritmética decimal; baixa velocidade e palavra de tamanho pequeno são, em geral, toleráveis.

Computação

Nessa classe da aplicação, o processador deve oferecer: palavra de tamanho grande, operações numéricas mais elaboradas (multiplicação, compactação, etc.), baixo custo, interface com memória de massa, alta velocidade e linguagens de nível mais alto.

e. O conjunto de instruções

A avaliação da potência de um conjunto de instruções é um problema sem resposta genérica. Em geral, a avaliação é feita através da programação de tarefas especiais (*benchmark*), com diferentes conjuntos de instruções e posterior comparação dos resultados. O que acontece, em geral, é que um conjunto é melhor em certas tarefas e pior em outras. Como dizer, então, qual é o melhor conjunto de instruções?

Na maioria das aplicações, contudo, um conjunto de instruções padrão é suficientemente bom. Nesses casos, o projetista deve, preferencialmente, gastar seu tempo desenvolvendo um programa eficiente, ao invés de ficar procurando por outros conjuntos que melhor se adaptem a seu problema. O programa resultante é muito dependente da qualidade da codificação. Não se deve pensar que, escolhendo-se um conjunto de instruções adequado, ou até mesmo microprogramando-o, o problema está resolvido.

Tendo-se em vista a aplicação a que se destina, deve-se olhar para um conjunto de instruções analisando seu comportamento nas seguintes classes de tarefas expostas a seguir⁽³⁾.

Controle de programa

Essa tarefa depende fortemente dos seguintes itens: pulos condicionais, desvios para sub-rotinas (também condicionais), pilha controlada por programa, e capacidade de operação sobre endereços.

Movimentação de dados

Esse trabalho é influenciado por dois aspectos: largura da palavra e modo de endereçamento. O modo de endereçamento é um ponto de distinção entre os microprocessadores, já que, nesse detalhe, eles variam muito. Os tipos de endereçamento existentes são os descritos no Cap. 6.

Manipulação de dados

Para a tarefa de manipulação de dados, devem-se observar as instruções aritméticas e lógicas. Outro detalhe são os recursos que o processador oferece para operar sobre dados de tamanho múltiplo de uma palavra. Nesse caso, uma operação será feita, seqüencialmente, em "pedaços" do dado. Também influem aqui os formatos (binário, *BCD*, etc.) em que os dados podem ser operados.

Entrada e saída

Além da forma em que os dados caminham para dentro e fora do sistema, o esquema de interrupção é decisivo nessa tarefa. O conjunto de instruções deve prover recursos para atender interrupções, e controlar dispositivos externos.

A grande parte do conjunto de instruções é padrão, variando de um para outro apenas em função da estrutura do fluxo de dados.

As aplicações que requerem instruções especiais [instrução de procura (*search*) de um dado na memória, por exemplo] seriam melhor atendidas por um processador que, além da capacidade de ter as instruções especiais microprogramadas, já disponha de um conjunto padronizado de instruções.

f. Sumário

Nesta seção discutiram-se os aspectos da arquitetura dos microprocessadores e microcomputadores. Grande parte do que foi dito já havia sido comentado nos primeiros capítulos

deste livro. A
difere de um
ências e deve
tador é a apli
tanto seu pr

A.3 O DESE

A cri
muito ma
em projeto
de esboçar
ralelo), e pa
de etapas de

No dese
equipe, mui
importante, e
harmônios
Conseg
crição do pr
lhamento do

a. Descri

A tare
guagem. Qua
inicial na líng
Portugu

finida, em ou
deve ser tal q
Por exa
combinatóri
blocos lógicos
em portugu
e dai para fáci
existirem varia
faça a determi
Grande pa
Não basta dis
necessário que
operada? O q
Quais os víncu
Essa desc
de base para a

b. Esquem

Aqui é ons
é tarefa simples
As interfaces, e
importante aqu
Parte disso, pa

deste livro. A razão é que um microcomputador, em seu modo de funcionamento, em nada difere de um computador digital de uso geral. Ele apresenta as mesmas qualidades e deficiências e deve atender aos mesmos vínculos. O aspecto fundamental que distingue o microcomputador é a *aplicação dedicada*, isto é, ele irá funcionar rodando sempre o mesmo programa. Portanto seus programas, em geral, residem em memórias fixas.

A.3 O DESENVOLVIMENTO DE PROJETOS COM MICROPROCESSADOR

A criatividade no desenvolvimento de um projeto que utilize um microprocessador está muito mais no planejamento do *software* do que no do *hardware*. Um engenheiro experiente em projeto lógico deverá mudar um pouco sua forma de pensar. Isso porque ele deverá deixar de esboçar a solução de seu problema, juntando peças que operam simultaneamente (em paralelo), e passar a dar soluções seqüenciais, onde cada tarefa é realizada após outra, através de etapas de um programa.

No desenvolvimento de um projeto existem várias etapas. Dependendo do tamanho da equipe, muitas delas ocorrem simultaneamente, cada uma influindo nas demais. O aspecto importante, então, é ter-se uma forma de documentação de projeto que permita a organização harmoniosa do trabalho da equipe.

Consegue-se distinguir, num projeto de desenvolvimento, cinco grandes etapas: (a) descrição do produto, (b) esquematização da solução, (c) detalhamento do *hardware*, (d) detalhamento do *software*, e (e) teste integrado.

a. Descrição do produto

A tarefa de projetar, em última instância, é a de descrever o problema em uma outra linguagem. Quando um engenheiro projeta um circuito lógico, o que ele faz é descrever o problema inicial na linguagem de blocos lógicos, *flip-flops*, etc.

Portanto projetar é traduzir uma descrição de um sistema, que muitas vezes é pouco definida, em outra linguagem mais precisa, estabelecendo os pontos vagos. A descrição final deve ser tal que um profissional experiente possa entendê-la e construir uma máquina real.

Por exemplo, no Cap. 1, vimos que existem inúmeras formas de se descrever um circuito combinatório (descrição em português, tabela de verdade, mapa de Karnaugh, diagrama com blocos lógicos, etc.). Nesse caso, o projeto de tal circuito se inicia com a descrição do sistema, em português. O engenheiro, então, a traduz para a linguagem de Karnaugh (por exemplo) e daí pula facilmente para a dos diagramas com blocos lógicos. A criatividade está no fato de existirem várias "traduções" do mesmo texto, e deve-se chegar à que melhor entre elas, satisfaça a determinados vínculos.

Grande parte do sucesso final depende da correta descrição do objetivo a ser atingido. Não basta dizer-se que o que se quer é uma máquina de controle numérico, por exemplo; é necessário que se especifique, em todos os detalhes, sua arquitetura. De que forma ela será operada? O que ela deve acionar? Em que ambiente irá funcionar? Qual a confiabilidade? Quais os vínculos? Qual o custo final? Como será realizada a manutenção?

Essa descrição inicial poderá gerar o manual de especificação do produto, que servirá de base para o desenvolvimento seguinte.

b. Esquematização da solução em diagramas de blocos

Aqui é onde se cria a solução. A esquematização do *hardware*, exceto quando há detalhes, é tarefa simples. Em geral, a estrutura da unidade central é aquela ditada pelo processador. As interfaces, então, estão definidas pelos dois lados: do computador e dos periféricos. O mais importante aqui é a escolha dos equipamentos periféricos e a forma que o sistema será operado. Parte disso, porém, já está definida no manual de especificação do produto.

A escolha do microprocessador é uma tarefa difícil. Nela influem fatores intangíveis, como, por exemplo, o apoio que o fabricante dá ao desenvolvimento, a existência de segundas fontes, etc. Os fatores técnicos já foram discutidos na seção anterior.

Em paralelo com o esboço do *hardware*, deve-se planejar o *software*, que são trabalhos interdependentes. Sendo uma equipe pequena, toda ela deve participar da esquematização da solução (*software* e *hardware*).

É com os programas que o hardware ganha "vida". Até então, tem-se um conjunto de peças especificadas, com a topologia de ligação definida. Seu potencial de trabalho é genérico. Os programas lhe definirão o comportamento.

Tarefas

No planejamento dos programas, o problema é como definir sua estrutura. Em primeiro lugar, devem-se determinar as tarefas⁽⁴⁾. Tarefa (*task*) é um segmento do programa final com função bem definida e isolada das demais.

As primeiras tarefas a serem consideradas são as cíclicas, isto é, as que devem ser executadas de tempos em tempos. Em geral, esse caso ocorre na administração de entrada e saída. Outras tarefas são, normalmente, escravas dessas, isto é, elas são iniciadas sob comando das primeiras. Existem, também, as tarefas que ficam normalmente sendo executadas, quando nada mais há a ser feito.

Uma pergunta tem de ser respondida: como administrar todas essas tarefas?

Um primeiro fato importante é que todas as tarefas devem ser pequenas. Uma vez iniciada uma tarefa, ela só devolve o processador para outra, quando terminar, ou quando for interrompida por uma de maior prioridade. Nas soluções de problemas em tempo real, existe a exigência de o tempo de resposta ser curto. Os pontos que requerem resposta muito rápida devem atuar no sistema em sua linha de interrupção. Por outro lado, as tarefas cíclicas também são excitadas por um relógio externo que interrompe o processador periodicamente.

Vê-se, então, a necessidade de organizar as tarefas sob o ponto de vista de prioridade. As que exigem tempo de resposta mais curto devem ser as prioritárias e, para não elevar em demasia a demora na resposta para as outras, elas devem ser muito curtas.

Uma alternativa a ser examinada cuidadosamente é a colheita de dados: existe a opção de acesso direto à memória. Nesse caso, a interface irá carregar diretamente na memória o dado externo. Se o local de carga for fixo, não será necessário intervenção do programa.

Após se terem definido as tarefas, e a forma como são iniciadas, resta o problema de comunicação entre elas. Isso é feito através de uma base de dados comum.

Cada tarefa tem sua base de dados, ou área de trabalho, própria. Quando existem relações entre duas tarefas, a comunicação é feita através de uma base de dados comum. Nesse aspecto, existe o problema da concorrência: é preciso garantir que uma mensagem (ou dado), deixada por uma tarefa na base comum, seja retirada pela outra, antes que a primeira volte a carregar uma nova mensagem. Deve-se definir um número máximo de mensagens que podem ficar esperando tratamento, e colocá-las em um registro elástico: uma fila onde elas ficam aguardando o tratamento. Essa estrutura de organização de programas já existe há anos nos sistemas operacionais para tempo real.

Resumindo, deve-se organizar o *software* em tarefas, que são classificadas em função da forma como iniciam, nos seguintes grupos:

- tarefas iniciadas ao ligar-se o sistema (tarefas de *preparação*);
- tarefas iniciadas cicличamente, por meio de um relógio externo;
- tarefas iniciadas pelo processo externo;
- tarefas iniciadas por outras tarefas;
- tarefas iniciadas pelo operador.

Todas elas devem ser também classificadas sob o ponto de vista de prioridade. Observe-se que a técnica de interrupção é um fator muito importante na organização.

apêndice

Base de

A base de
dos dados m
da codificação

Os dados
dificuldade m
que devem ser
nizações de d
ligada

Listas

As listas
que possuem
uma lista de dadi

Uma lista

pode ser arm
endereço de
lista, $a[i]$, e a

Como o ender
o nome de fu

Por exem
deverá manu
somar m an

Tabelas

As matr
dimensional
índices: límit

A tabela apre
linha a límit

Dessa forma,
sição do item

onde N é a m
mente referida
de dimensões

Base de dados⁽⁵⁾

A base de dados de um programa é fundamental na eficiência do algoritmo. A estrutura dos dados manipulados pelo programa deve ser cuidadosamente planejada antes do início da codificação das tarefas.

Os dados isolados são simplesmente guardados na memória, em posições conhecidas. A dificuldade existe quando há dados agrupados em classes (vetores, matrizes, tabelas, etc.), e que devem ser armazenados de forma a permitir consulta rápida e fácil. As principais organizações de dados na memória são as seguintes: (1) lista, (2) tabela, (3) pilha, (4) fila, e (5) lista ligada.

Listas

As listas (*arrays*) são estruturas unidimensionais em que se armazenam vetores, ou dados que possam ser organizados de forma consecutiva. Por exemplo, um dicionário é uma grande lista de dados organizados seqüencialmente.

Uma lista de cinqüenta elementos,

$$a(1), a(2), \dots, a(49), a(50),$$

pode ser armazenada na memória, em endereços consecutivos. Dessa forma, pode-se saber o endereço de um elemento genérico, $a(i)$, conhecendo-se o endereço do primeiro elemento da lista, $a(1)$, e o tamanho (número de palavras), m , de cada elemento. A fórmula é a seguinte:

$$\text{end}[a(i)] = \text{end}[a(1)] + (i - 1) \cdot m.$$

Como o endereço de $a(1)$ é usado para o cálculo do endereço dos demais elementos, $a(1)$ recebe o nome de *base da lista*.

Por exemplo, um programa que retira, seqüencialmente, os itens da lista para processá-los deverá manter o endereço em um local especial. A cada acesso na lista, tal programa deverá somar m ao endereço, conferindo se o final da lista não foi ultrapassado.

Tabelas multidimensionais

As matrizes podem ser armazenadas na memória em tabelas multidimensionais (*multi-dimensional arrays*). Em uma tabela bidimensional, os elementos são vistos através de dois índices: linha e coluna. Exemplo:

$$\begin{array}{ccc} a(1,1) & a(1,2) & a(1,3) \\ a(2,1) & a(2,2) & a(2,3) \\ a(3,1) & a(3,2) & a(3,3) \\ a(4,1) & a(4,2) & a(4,3) \end{array}$$

A tabela apresentada no exemplo pode ser armazenada na memória, de maneira seqüencial, linha a linha,

$$a(1,1); a(1,2); a(1,3); a(2,1); a(2,2); a(2,3); a(3,1); \dots; a(4,3).$$

Dessa forma, sendo m_A o número de palavras ocupadas por um item de uma tabela A, a posição do item genérico $a(i,j)$ é dada por

$$\text{end}[a(i,j)] = \text{end}[a(1,1)] + [(i-1) \cdot N + j-1] \cdot m_A,$$

onde N é o número de elementos em cada linha. Essa técnica de cálculo de endereço é usualmente referida como "método polinomial". Podem-se derivar expressões análogas para tabelas de dimensões maiores que dois.

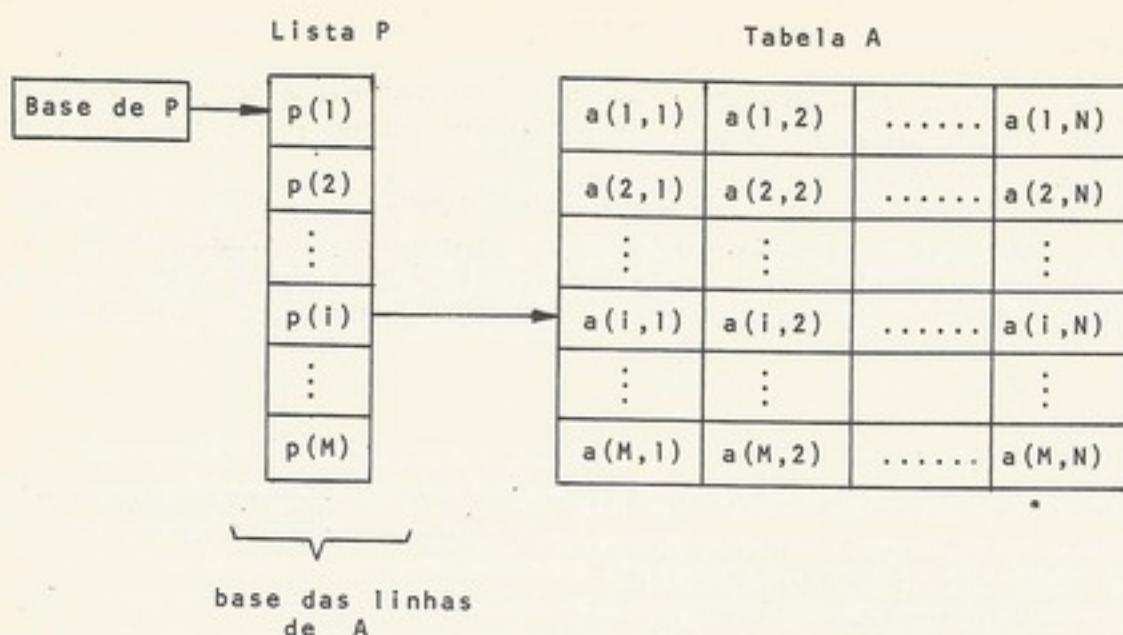


Figura A.3. Forma de acesso indireto a uma tabela bidimensional

Em microprocessadores, onde multiplicações são difíceis, pode-se utilizar a técnica de endereçamento indireto. Associa-se à tabela A uma lista P (Fig. A.3) contendo, para cada linha de A um ponteiro para seu primeiro item, ou seja, para a base de cada linha. Isto é, $p(i)$ é o endereço de $a(i,1)$. A lista P (lista de base da tabela A) é construída, e mantém seu conteúdo inalterado, no instante em que se aloca a tabela A na memória. Dessa forma, o acesso ao elemento $a(i,j)$ é feito nas seguintes etapas:

carrega-se o acumulador com o endereço de $p(1)$ (base da lista P);
 soma-se $(i-1) \cdot m_p$ ao acumulador (m_p é o tamanho de cada item da lista P);

(após essa soma, tem-se o endereço de $p(i)$, que aponta para o primeiro item da linha i da tabela A)

carrega-se o acumulador com o conteúdo da posição de memória cujo endereço foi calculado acima (base da linha i);
 soma-se $(j-1) \cdot m_A$ ao acumulador (m_A é o tamanho de cada item da tabela A)

[após essa última operação, tem-se, no acumulador, o endereço do item $a(i,j)$].

Esse procedimento foi simplificado pela hipótese de que, no acumulador, pode-se ter um endereço completo. Normalmente, um acumulador tem 8 bits, o que, na maioria das vezes, é insuficiente para armazenar um endereço. Nesse caso, as alterações são mais complicadas. Os microprocessadores de arquitetura mais avançada possuem registradores internos, de 16 bits, especiais para a manipulação de endereços, o que viabiliza essa técnica.

Pilhas

As pilhas (*stacks*) são listas com um procedimento de armazenamento e leitura característico. Quando se utiliza uma pilha, não há interesse por um item genérico em seu interior. O acesso à pilha é sempre feito em seu "topo", onde está o último item armazenado.

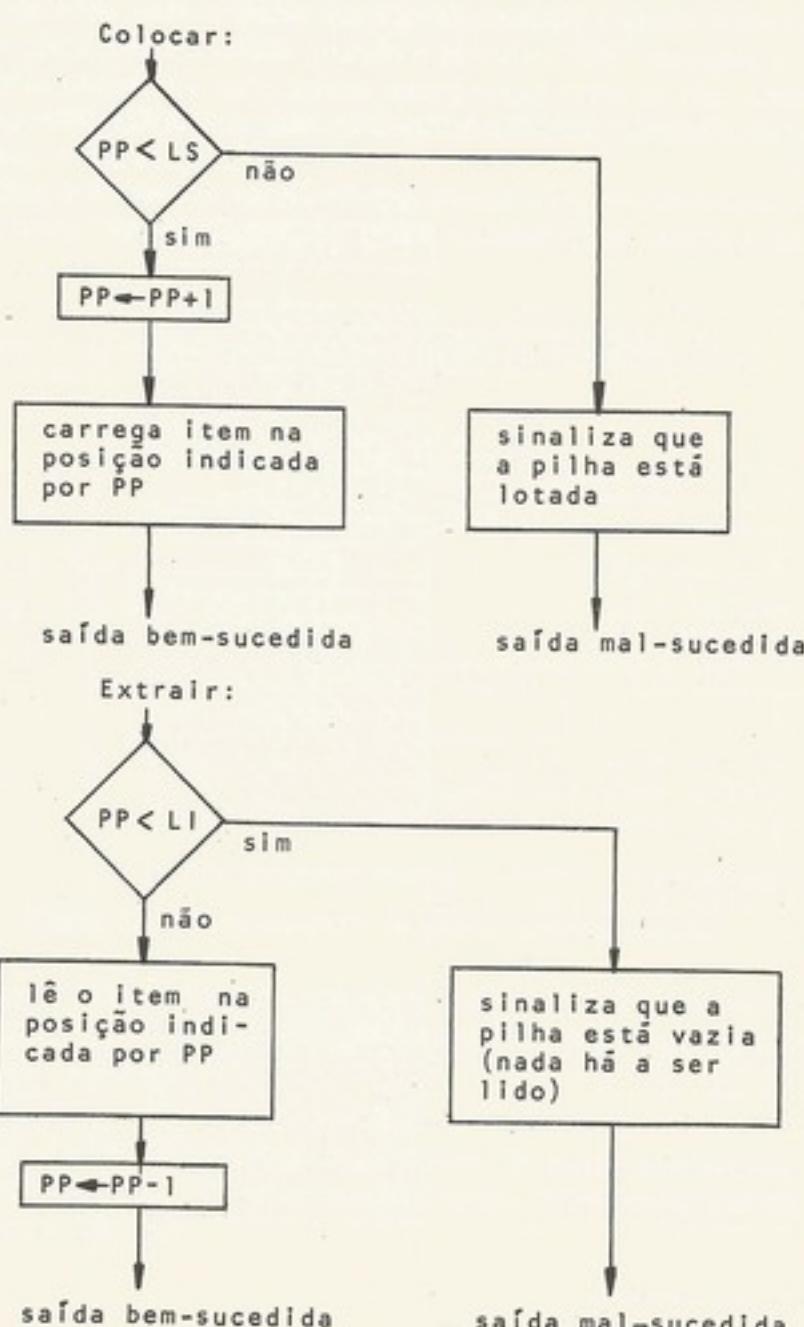
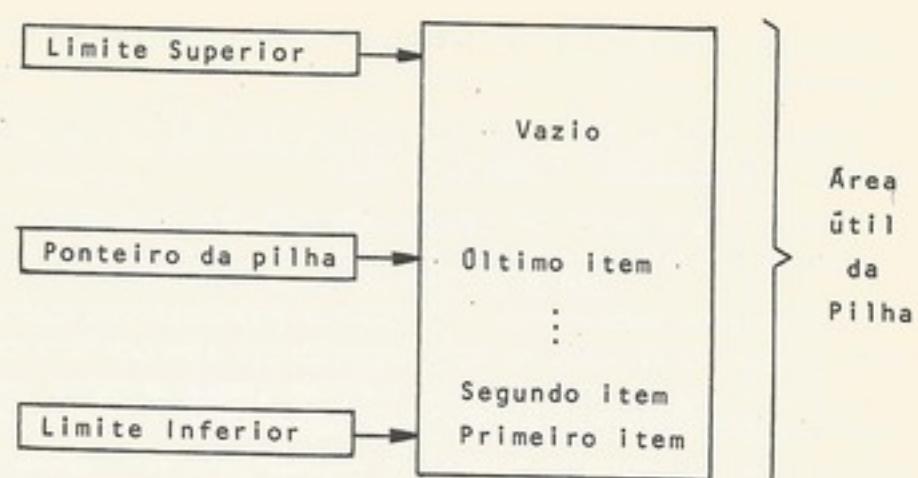
No instante de sua definição, uma pilha está vazia. Tudo o que se sabe dela é que seu topo está numa certa posição da memória. Uma pilha é caracterizada por três parâmetros (Fig. A.4):

ponteiro da pilha — endereço da última palavra armazenada (topo);
 limite superior — indica o limite máximo que a pilha pode atingir (controle da área útil);
 limite inferior — indica a parte inferior da pilha, e serve para controlar seu esvaziamento.

Figura A.4. ...

Figura A.5. ... superior da pilha

Figura A.4. A pilha

Figura A.5. As operações de colocar e extraír em uma pilha. PP = ponteiro da pilha; LS = limite superior da pilha; LI = limite inferior da pilha

Existem duas operações que são inerentes a essa organização de memória: colocar (*push*) e extraír (*pop*).

Colocar. A tarefa de colocar um elemento na pilha consiste em incrementar o ponteiro da pilha, e ali armazenar o item (Fig. A.5).

Extraír. Para se extraír um elemento da pilha, deve-se ler o conteúdo da posição indicada pelo ponteiro da pilha e, a seguir, decrementar o ponteiro (Fig. A.5).

A pilha estará vazia quando o ponteiro da pilha for menor que o limite inferior, e estará lotada quando ele for igual ao limite superior.

A pilha é uma estrutura de dados utilizada quando se requer que os itens sejam lidos na ordem inversa da que foram armazenados. Por exemplo: endereços de retorno de sub-rotinas, processamento de operações aritméticas, ou, em geral, nos algoritmos recursivos. As pilhas são, algumas vezes, chamadas de "armazenamento com inversão" (*reversion storage*).

Filas

Essa organização de dados também é uma lista, com procedimentos especiais de armazenamento e leitura.

Enquanto a pilha se caracteriza pelo fato de que "o último a entrar é o primeiro a sair" (*last in, first out*), na fila (*queues*), o "primeiro a entrar é o primeiro a sair" (*first in, first out*).

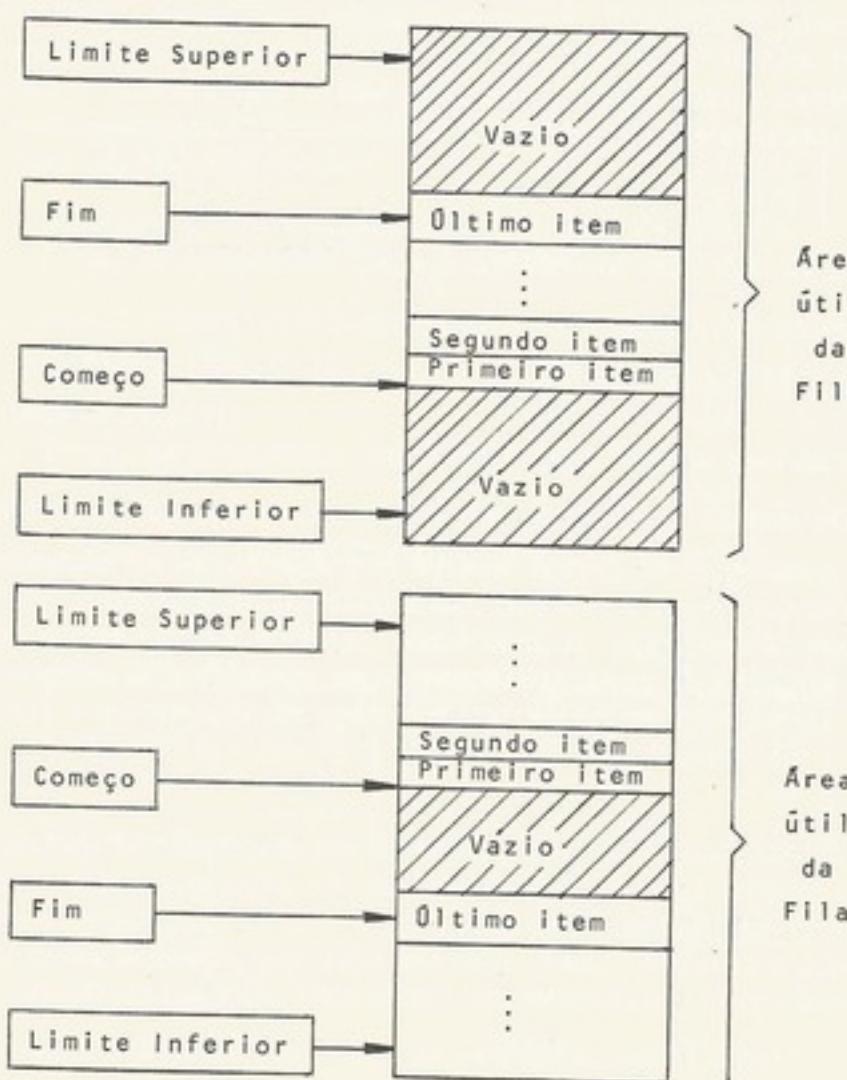


Figura A.6. Exemplo de filas

Uma fila é
armazenada de
zenamento. Vou
esta chegar ao fi
vazia devido a

Uma fila co
limites infer
fim — apen
começo —

Como na pilha
(*put*) e extraír

Colocar (*pu*
fazendo com q
ali o dado. Seja
tipo de comuni
apresentado m
valor de "fim".

dá a volta
o começo d
útil

Extrair. A
ocorre quando "i
e lê-se o item pa

Em aplicaçõ
damental. Enqu
para serem extra
tada conforme

Listas ligadas

Essa é a estru
turas vistas são

Uma fila é inicialmente definida por uma área útil, vazia, na memória. Os itens vão sendo armazenados de forma sucessiva na área útil. A extração deles é feita na mesma ordem de armazenamento. Vê-se, então, que a fila é uma estrutura que se move dentro de sua área útil. Quando esta chegar ao limite superior da área útil, deverá continuar na parte inferior, que já deve estar vazia devido às extrações ocorridas.

Uma fila é definida pelos seguintes ponteiros (Fig. A.6):

- limites inferior e superior — indicam os limites da área útil da fila;
- fim — aponta para o último item colocado na fila;
- começo — aponta para o último item extraído da fila.

Como na pilha, a estrutura da fila é definida juntamente com dois procedimentos, colocar (*put*) e extrair (*get*).

Colocar (Fig. A.6). Para se colocar um item na fila, deve-se incrementar o ponteiro “fim”, fazendo com que ele volte para o início quando ultrapassar o limite superior, e armazenar ali o dado. Sabe-se que a fila está lotada quando o ponteiro “fim” encontra o “começo”. Esse tipo de controle torna obrigatória pelo menos uma posição vazia na fila. Analise o algoritmo apresentado na Fig. A.7. Ali se usou uma palavra de rascunho, RASC, para não se perder o valor de “fim”, caso a fila esteja cheia.

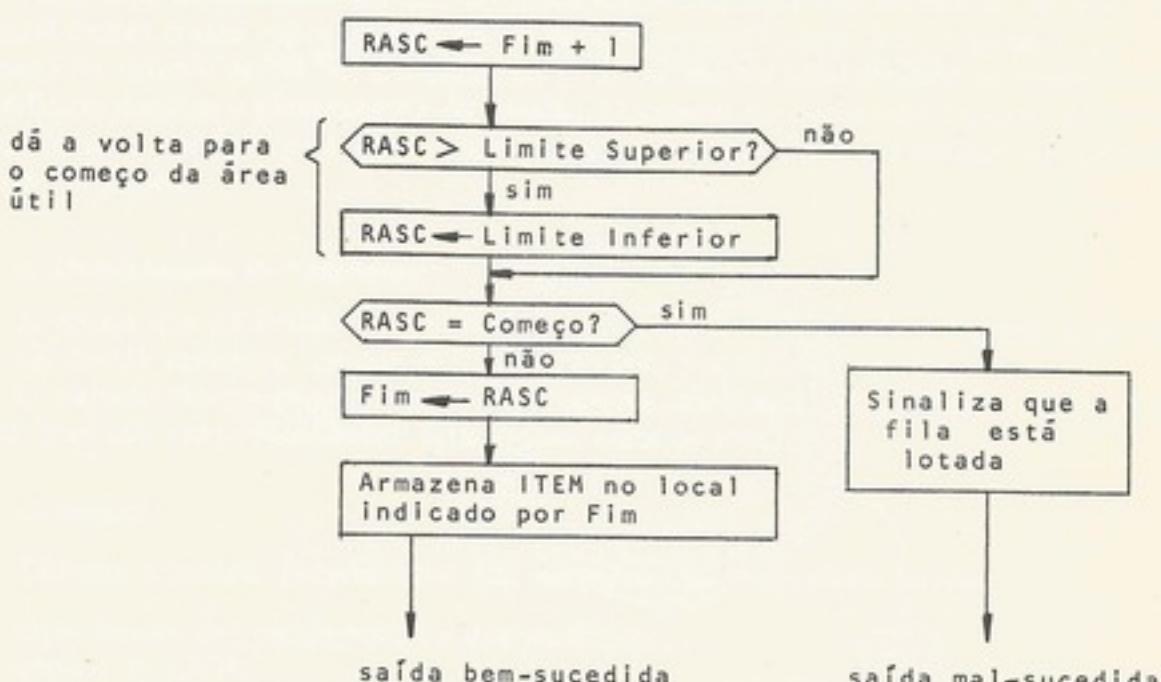


Figura A.7. O procedimento de colocar um item na fila

Extrair. A extração começa com um teste para verificar se a fila está vazia (Fig. A.8), que ocorre quando “começo” = “fim”. Caso não esteja vazia, incrementa-se o ponteiro “começo”, e lê-se o item por ele indicado.

Em aplicações em tempo real ou em controle de entrada/saída, a fila é uma estrutura fundamental. Enquanto não forem processados, os dados vão sendo armazenados em uma fila, para serem extraídos mais tarde. O aspecto circular de uma fila faz com que ela seja representada conforme indica a Fig. A.9.

Listas ligadas

Essa é a estrutura de dados mais refinada que será apresentada neste texto. Todas as estruturas vistas são seqüenciais: os dados são armazenados na memória, em posições consecutivas,

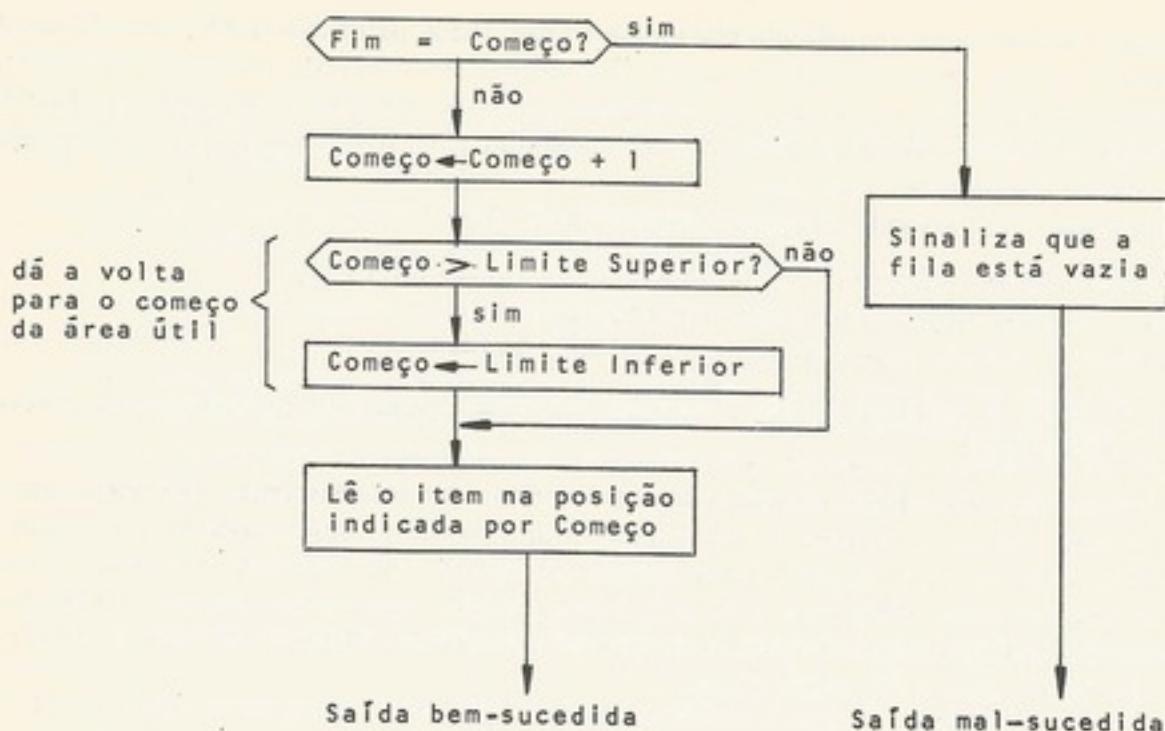
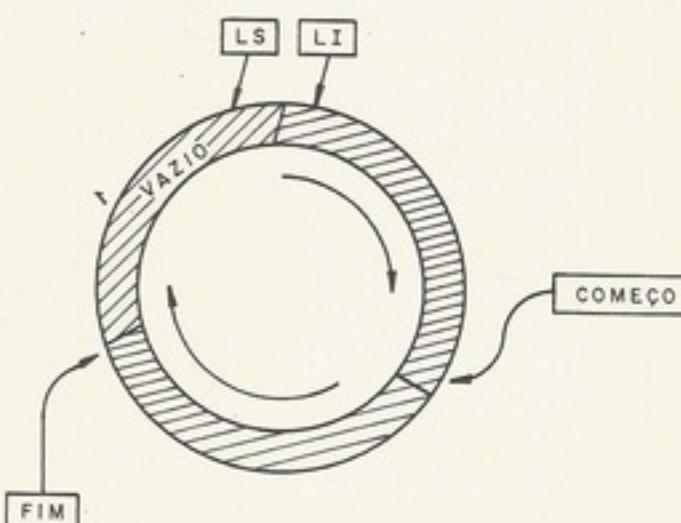
Figura A.8. O procedimento de *extrair* um item da fila

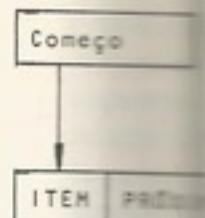
Figura A.9. A representação da fila em forma circular

formando blocos sólidos. O que aconteceria, nessas estruturas, se se retirasse um item do interior do bloco? Seria necessário mover todo o restante do bloco para não deixar "buracos", que seriam impossíveis de controlar.

A lista ligada (*linked list*) permite a fácil inserção e extração de itens no interior da estrutura, sem a necessidade de movimentar os itens restantes. Isso é conseguido anexando-se a cada item um ponteiro indicando o item seguinte (Fig. A.10). Esses ponteiros não precisam estar fisicamente perto dos dados, isto é, eles podem formar uma lista isolada.

Uma lista ligada tem seu inicio indicado por um ponteiro ("começo") apontando para o primeiro item da lista. Cada item aponta para o seguinte através de seu ponteiro próprio ("próximo"). No último item da lista, o ponteiro é nulo (indicando o fim da lista).

Uma dificuldade a ser sanada em uma lista ligada é a de controle da área útil. Ao se extrair um item do interior da lista, esse local se torna vazio. A parte vazia da área útil é formada, então,



por vários pedaços
a lista disponibilizada.

Uma lista ligada
cando o começo.

A adição e remoção
ponível, para se
ligado à lista.

Conclusão

Após esquematizar
manual devendo
manual do projeto
modificado para
redundante se
partes do projeto.

c. Detalhamento

A equipe de
Ali, ela devendo
O trabalho
essas fases são:

O projeto

Durante o

especificando
projeto da

Esse trabalho é
ticular. As tâmas
uma seleção das
jetista produzindo
O trabalho de pro
para pequenos pr

No projeto
periférico a ser

A montagem

O protótipo
tarefa, deve-se

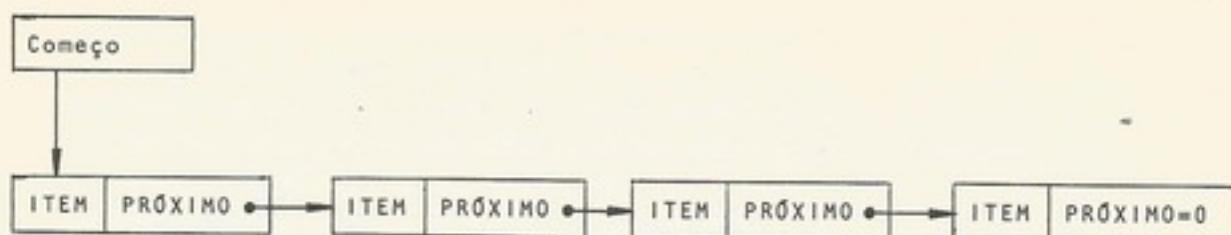


Figura A.10. Uma lista ligada

por vários pedaços isolados. O controle dessa parte vazia é feito por uma segunda lista ligada: a lista disponível.

Uma lista ligada é iniciada com o ponteiro "começo = 0", e o ponteiro disponível indicando o começo da área útil. Essa área tem todos os locais ligados da forma da lista ligada.

A adição de um item na lista é uma tarefa que envolve a extração de um local da lista disponível, para ser utilizado na lista ligada. Quando se extrai um item da lista ligada, o local é ligado à lista disponível, para uso futuro.

Conclusão

Após esquematizar-se a solução do problema, deve-se gerar o manual do projeto. Esse manual deverá conter todos os detalhes da solução esboçada nessa fase. A primeira parte do manual do sistema deverá ser o manual do produto, escrito anteriormente e, possivelmente, modificado durante o planejamento da solução. O manual do sistema deve conter informação, redundante se possível, para que a equipe possa, agora, dividir-se e trabalhar nas diferentes partes do projeto.

c. Detalhamento do "hardware"

A equipe que irá detalhar o *hardware* terá como ponto de partida o manual do projeto. Ali, ela deverá encontrar um esboço do sistema com descrição das partes.

O trabalho deverá passar pelas fases de projeto, montagem de protótipo e testes. Todas essas fases são inter-relacionadas, e cada uma afeta as demais.

O projeto

Durante o projeto propriamente dito encontram-se as seguintes tarefas:

- especificação dos componentes e dos periféricos;
- projeto da *UCP*;
- projeto das interfaces.

Esse trabalho requer experiência nas diversas opções existentes para a solução do caso particular. As técnicas de projeto são as normalmente utilizadas em sistemas digitais: é basicamente uma seleção das alternativas, interligando blocos, segundo recomenda o fabricante. Um projetista produzirá melhores resultados, quanto maior for sua experiência e melhor sua memória. O trabalho de projeto lógico é fortemente influenciado pelo conhecimento prévio de soluções para pequenos problemas, que são agrupados para formar o todo.

No projeto das interfaces é necessário que se disponha de uma boa documentação do periférico a ser utilizado.

A montagem do protótipo

O protótipo deverá ser montado à medida que as soluções forem sendo criadas. Nessa tarefa, deve-se ter em mente que haverá muitas alterações.

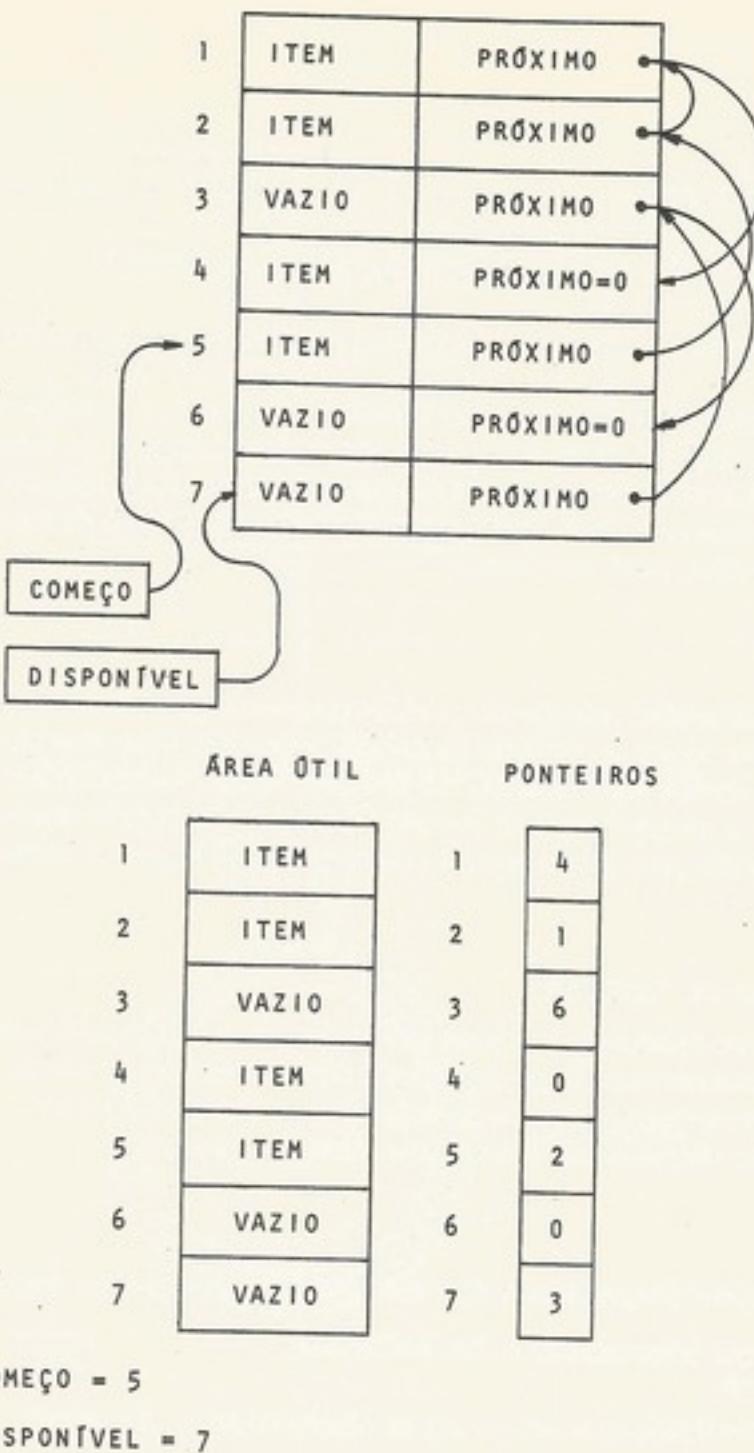


Figura A.11. Organização da área útil em uma lista ligada

Os testes

A realização dos testes finais do protótipo exige que se escrevam pequenos programas, que irão exercitar todas as suas partes. Mesmo que, nessa altura, o *software* esteja pronto, deve-se começar o teste com programas simples. A adição dos erros existentes no protótipo com os do *software* aumentaria muito a complexidade do teste.

Os resultados

O produto gerado no detalhamento do *hardware* é composto dos seguintes itens:

protótipo testado;
diagramas lógicos dos circuitos;
esquemas dos circuitos das interfaces, fonte de alimentação, etc.;
relação dos componentes utilizados (com alternativas para substituição);
esquema de fiação dos conectores;
manual de operação.

d. O detalhamento do "software"

O ponto de partida para o detalhamento do *software* é a organização das tarefas e a descrição das bases de dados. Para cada tarefa, deve-se dispor dos seguintes elementos:

fluxograma;
base de dados local;
base de dados global.

De posse desses elementos, o desenvolvimento do *software* será feito, para cada tarefa, em duas etapas:

programação do código e alocação relativa da base de dados;
testes e correções.

A programação

Dependendo da complexidade dos programas, o engenheiro deverá dispor de certos recursos. Quanto maior e mais intrincadas forem as tarefas, tanto menor será a probabilidade de o projetista conseguir gerar bons programas, codificando em linguagem de máquina.

O uso de *linguagem de máquina*, para a programação das tarefas deve ficar restrito às aplicações muito pequenas (poucas dezenas de instruções). Nesse caso, deve-se detalhar o programa, com cada linha de código explicada e documentada. O programa é, então, carregado diretamente em PROM apagável, e testado no protótipo do sistema. As eventuais correções serão feitas apagando-se toda a memória, e reprogramando seu conteúdo.

As aplicações mais complexas exigem que se use alguma linguagem de programação. Existem aqui duas alternativas:

programação em linguagem de montagem (*assembly language*);
programação em linguagem de alto nível.

A *linguagem de montagem* é uma forma de se escrever o programa na qual códigos das instruções são seus mnemônicos, e os operandos são simbólicos. Isto é, cada operando recebe do programador um nome. Em todo o programa, esse dado é referido pelo seu nome. No final do programa, deve-se especificar o tipo e o local onde os operandos, que receberam nomes, se situam. O texto gerado nessa linguagem alimenta um programa especial, o montador (*assembler*), que, a partir dele, irá gerar o código de máquina correspondente.

Os fabricantes de microprocessadores fornecem sistemas de desenvolvimento, que são construídos com os próprios microprocessadores, cuja função é assistir o projetista no detalhamento do *software*. Esses sistemas contêm programas montadores, que são usados para realizar a tradução descrita acima.

Observe-se que a linguagem de montagem é extremamente dependente da linguagem de máquina. Para cada instrução em linguagem de máquina, tem-se, em geral, uma correspondente na linguagem de montagem. Por isso, programar em linguagem de montagem é uma tarefa difícil, cheia de detalhes e truques. A maioria dos programadores prefere afastar-se da máquina utilizando linguagem de alto nível.

Com as linguagens de *alto nível*, a programação fica muito próxima do fluxograma. O programador não desvia sua atenção para detalhes. O texto gerado nessa linguagem deve ser

traduzido para linguagem de máquina, o que é feito por programas chamados de *compiladores*. Os compiladores são muito complicados e extensos para serem rodados em um sistema de desenvolvimento. Os fabricantes de microprocessadores fornecem compiladores que, por sua vez, são escritos em linguagem de alto nível, entendida pela maioria dos computadores existentes (Fortran IV, por exemplo). Esses compiladores podem ser rodados em qualquer computador, e traduzirão os programas para a linguagem de máquina do microprocessador.

Na programação em linguagem de alto nível, deve-se ter em mente que se perde um pouco da eficiência do *software* resultante. Isso é devido ao fato de os comandos escritos em linguagem de alto nível serem transcritos, pelo compilador, em blocos de instruções em linguagem de máquina. Daí ocorre algum desperdício. Programando em linguagem de montagem, o texto resultante poderá ser otimizado pelo programador.

Os testes e as correções

Após a programação, há as tarefas descritas em linguagem de máquina. Essa descrição deve ser testada.

A maneira trivial, se bem que não a melhor, é testar os programas no protótipo. O que resultará dessa tentativa será apenas a certeza de que o *software* não funciona. Pouco mais que isso se saberá. A existência de muitos erros dificultam o seu teste no protótipo. Por isso, a menos que a programação seja muito simples, o programador precisa de ajuda para testar e corrigir seus programas. Essa ajuda pode vir em duas formas:

sistema de desenvolvimento com monitor/editor para testar e corrigir programas; minicomputador de uso geral com um interpretador/monitor/editor para a linguagem do microprocessador.

Um *interpretador* é um programa que recebe as tarefas codificadas em linguagem de máquina do processador; define alguma área de memória como sendo os registradores do fluxo de dados do microprocessador; e se comporta como este. O interpretador é, portanto, uma espécie de "simulador" do microprocessador. Essas formas de ajuda oferecem ao programador dois serviços: teste e correção.

Durante o *teste*, os programas são armazenados em memória "ler-escrever". Sob controle do programador, o programa monitor insere marcas no texto. Essas marcas, chamadas de pontos de quebra (*break points*), são pontos onde o monitor irá parar o programa para que o programador possa examinar os conteúdos do fluxo de dados. Durante o teste, o programador se comunica com o monitor, orientando-o quanto à inserção de pontos de quebra, listagem de conteúdos de memória, de registradores, etc. Com essa ferramenta, o teste ocorre através do processamento e do exame de pequenos trechos do programa.

A correção do programa é feita por uma parte do monitor chamada *editor*. Esse programa facilita a inserção, mudança e eliminação de linhas de programa.

Normalmente o programador se comunica com o monitor, durante os testes, através de um terminal vídeo. Através da digitação de comandos especiais, ele instrui o monitor sobre o que fazer, e este responde com mensagens na tela.

O teste realizado dessa forma não é definitivo. Ele roda em velocidade diferente da final, e não está ligado aos periféricos. Apesar de se poder simular o meio ambiente da aplicação final, dificilmente se consegue reproduzir com exatidão tal situação. Portanto, enquanto o *software* não for juntado ao protótipo e ambos ligados aos periféricos, não se poderá dizer que não existem erros.

Os resultados

Como produto dessa fase do projeto, obtém-se listagens dos programas testados convenientemente documentadas. Os programas deverão estar armazenados em fitas de papel per-

apêndice

furadas ou fitas para "queimar"

e. O teste

Esta é a etapa de problemas.

Não existem e os periféricos e tarefas são fáceis

É claro que de quebra nos sistema integrado inicial, sem poss

Devido à em memórias PEs construídos direto integrados, e o s

Quando a única. Essa é umasíveis falhas. Com necessidade de

O resultado manual deverá de uma descrição

f. Sumário

Apesar de problemas, sua utilização compõe

Neste item o protótipo funciona ponto de partida placas de circuitos

A.4 O MICRO

Em desempenho de processador é o que tem provado. Devido a sua portabilidade, objetivando

Esse processador endereçando seu geral. Possibilita para armazenamento conjunto de instruções na memória local, instruções armazenadas de interrupção que mostra que sua

furadas ou fita magnética cassete. No teste integrado, esses programas deverão ser usados para "queimar" as memórias "ler somente" do protótipo.

e. O teste integrado

Esta é a última etapa do desenvolvimento, porém a menos previsível e com muitos problemas.

Não existem ferramentas, nem formas de ajuda. Aqui se juntarão o *software*, o *hardware* e os periféricos, e se ligará o sistema. A experiência e o bom senso da equipe encarregada dessa tarefa são fatores decisivos.

É claro que se podem usar alguns truques. Havendo problemas, podem-se inserir pontos de quebra nos programas, e remontá-los (ou recompilá-los) de forma que sejam testados no sistema integrado. Uma vez resolvidos os problemas, os programas têm de voltar à sua forma inicial, sem pontos de quebra.

Devido à previsão de muitas alterações nos programas, estes devem estar armazenados em memórias *PROM* apagáveis, ou mesmo em memórias "ler-escrever". Possivelmente seriam construídos dois protótipos: o primeiro com memória "ler-escrever", para ser usado nos testes integrados, e o segundo com memória "ler somente" (versão final).

Quando a aplicação permite, o protótipo pode dispor de recurso de operação em instrução única. Essa é uma boa forma de se acompanhar a evolução do processamento e detectar as possíveis falhas. Contudo, nas aplicações em tempo real, é impossível usar esse recurso, dada a necessidade de tempos de resposta curtos.

O resultado do teste integrado é o produto final do projeto: o manual do produto. Esse manual deverá conter todas as informações para operação e manutenção do sistema, além de uma descrição completa de todos os seus componentes: *software*, *hardware* e periféricos.

f. Sumário

Apesar de o microprocessador ser uma solução economicamente viável para inúmeros problemas, sua utilização está longe de ser simples. Esse é um componente sofisticado e de utilização complexa.

Neste item descreveram-se as etapas de um projeto (Fig. A.12). Seu resultado é um protótipo funcionando, com a documentação necessária para descrevê-lo e operá-lo. Esse é o ponto de partida para a equipe que irá industrializá-lo: definir o empacotamento final, com placas de circuitos impressos, etc.

A.4 O MICROPROCESSADOR INTEL 8080

Em dezembro de 1973 a Intel Corporation (norte-americana), começou a produzir o microprocessador 8080, tipo canal *N*. Esse processador está sendo muito usado em todo o mundo, o que tem provocado o aparecimento de outras empresas que fabricam o mesmo componente. Devido a sua popularidade, o 8080 foi escolhido para ser usado como exemplo básico neste texto, objetivando dar ao estudante uma visão mais clara da potencialidade dos microprocessadores.

Esse processador, com um ciclo de máquina da ordem de $2\ \mu s$, opera em dados de 8 bits, endereçando até 64K bytes de memória. Tem uma memória local com seis registradores de uso geral. Possibilita o uso de uma pilha de dados na memória principal, que também é utilizada para armazenar os endereços nos desvios a sub-rotinas e atendimento de interrupções. Seu conjunto de instruções é bastante potente, permitindo manipulação direta dos registradores na memória local, transferências de dados e endereços para a memória, controle da pilha, instruções aritméticas e lógicas, e instruções de entrada e saída. O processador tem recursos de interrupção por até oito fontes diferentes, de mesma prioridade. Uma análise superficial mostra que suas características se aproximam bastante das dos minicomputadores.

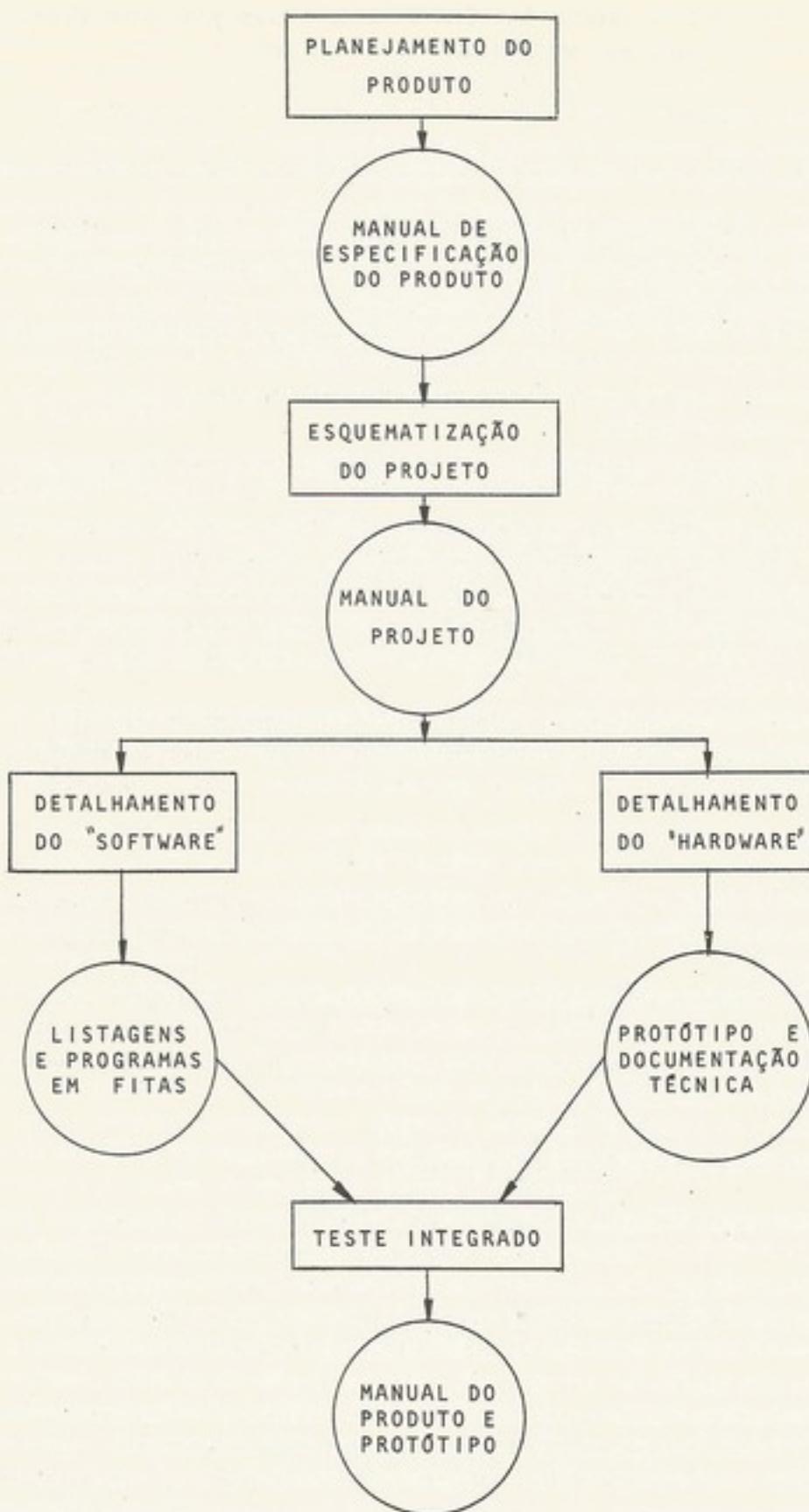
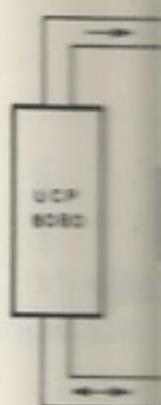


Figura A.12. O desenvolvimento de projeto com microprocessadores



a. A unida

Dentro da
dade central de
de controle. Es-
liga-se à memória
com sinais de se-
recional, para
Muitas vezes a
nentes especifi-
via bidireccional.
Mesmo quando
inserir um con-

A unida
deve também o
ração de tous



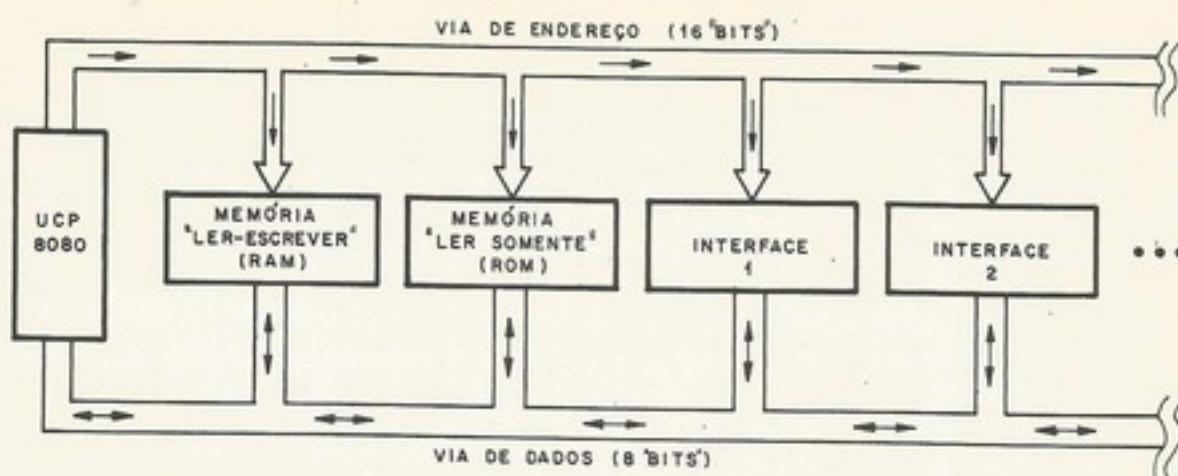


Figura A.13. Interligação do 8080 a memórias e interfaces

a. A arquitetura do microprocessador 8080⁽⁶⁾

Dentro de uma pastilha de circuito integrado, com quarenta terminais, existe uma unidade central de processamento completa, com unidade aritmética, memória local e unidade de controle. Esse processador, através de suas vias de dados (bidirecional) e de endereço, interliga-se à memória e interfaces, da forma vista na Fig. A.13. Além dessas vias, existem as linhas com sinais de controle, gerados pela unidade de controle interna à UCP. A via de dados é bidirecional, para se economizarem terminais do circuito integrado, que já tem quarenta pinos. Muitas vezes a administração dessa via é problemática, apesar da disponibilidade de componentes específicos para realizar a interface com ela. Nesses casos, opta-se pela bifurcação dessa via bidirecional em duas vias de 8 bits cada, uma em cada sentido, como mostra a Fig. A.14. Mesmo quando se mantém a via de dados bidirecional, por problemas de *fan-out*, costuma-se inserir um circuito excitador (*driver*) entre o processador e a via bidirecional externa.

A unidade central de processamento 8080, além dos sinais das vias de endereço e dados, deve também conter vários sinais de controle e sincronismo, de forma a supervisionar a operação de todos componentes (memórias, interfaces) que a ela se interligam.

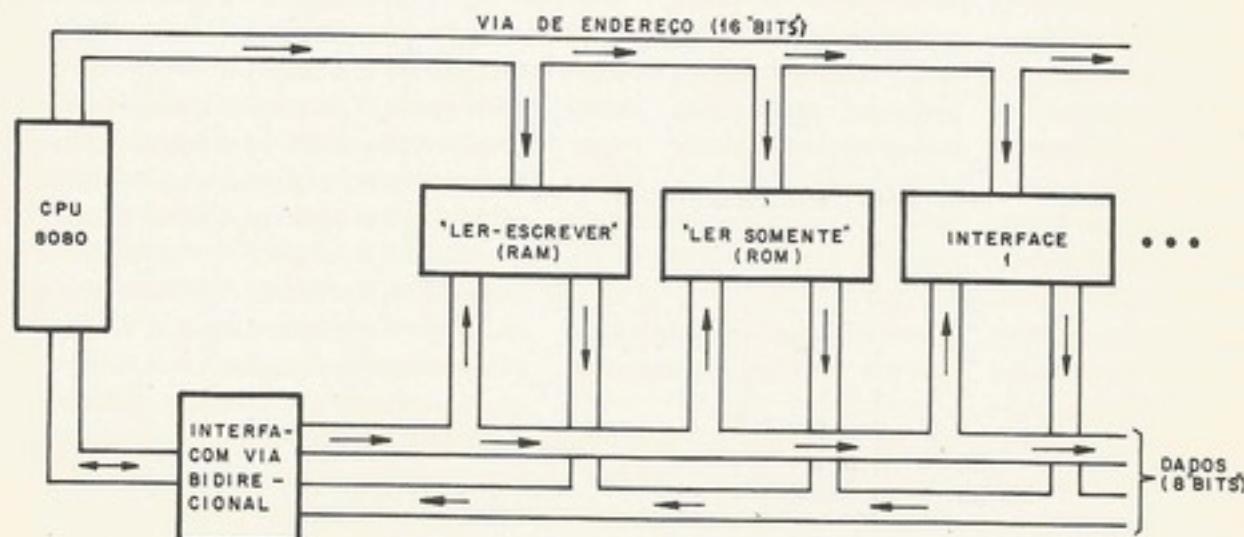


Figura A.14. Separando a via bidirecional em duas unidirecionais

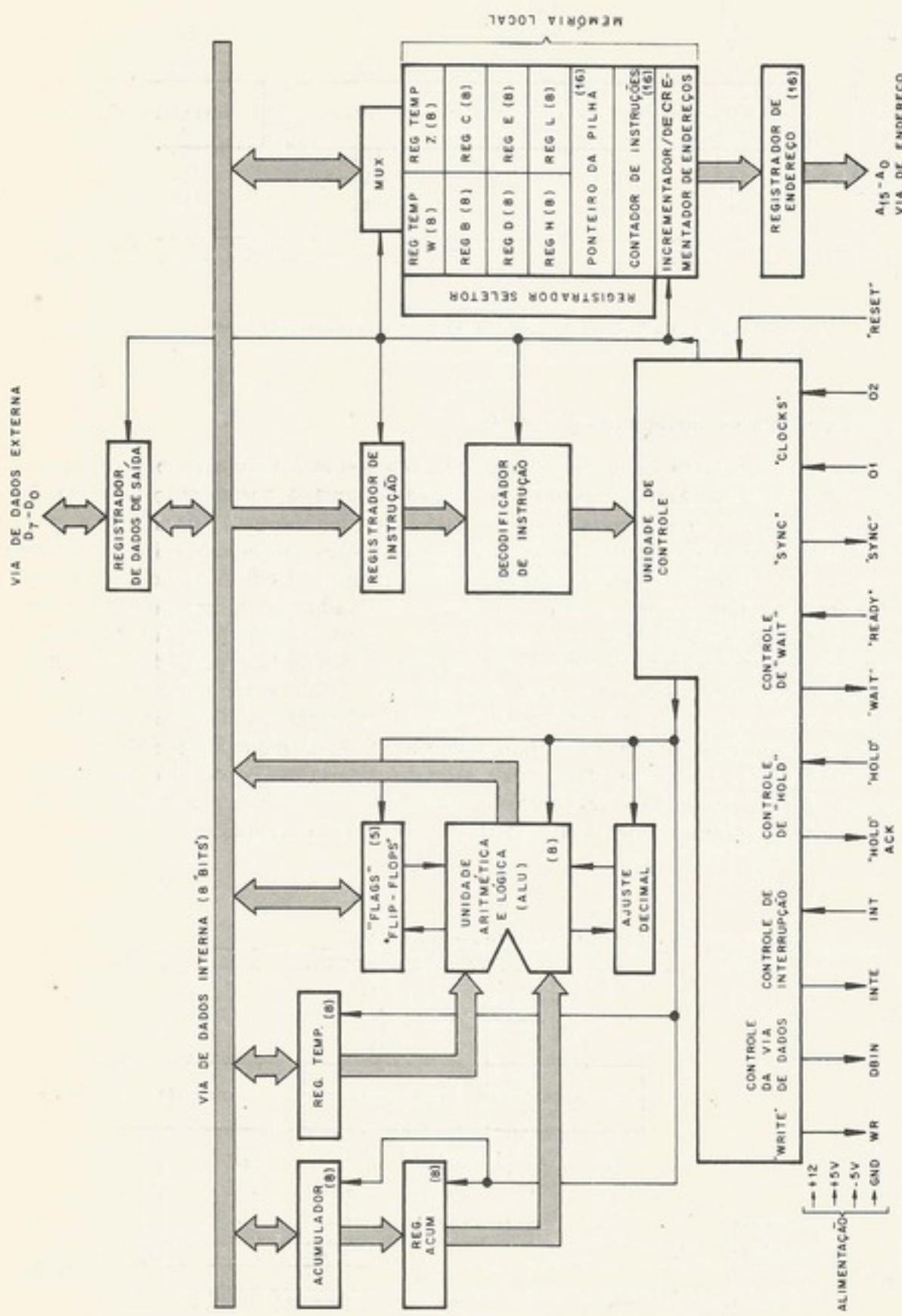


Figura A.15. Diagrama funcional da UCP

Figura A.15.

Diagrama

Esse pr
de entra
e D0 a D7
de relógio

A UCP
e lógica de

Rodapé

A memó
da segund
seis regis
pilha (naci

O com
(fetch), e ini
para opera
no começo d
o ponteiro é d
Dessa forma
confere ao R

Os sei
instruções,
16 bits pa
durante a

A com
plexador,
mória loca
sob contr
de incremen



Figura A.15. Diagrama funcional da UCP

Figura A.16. Relação dos terminais

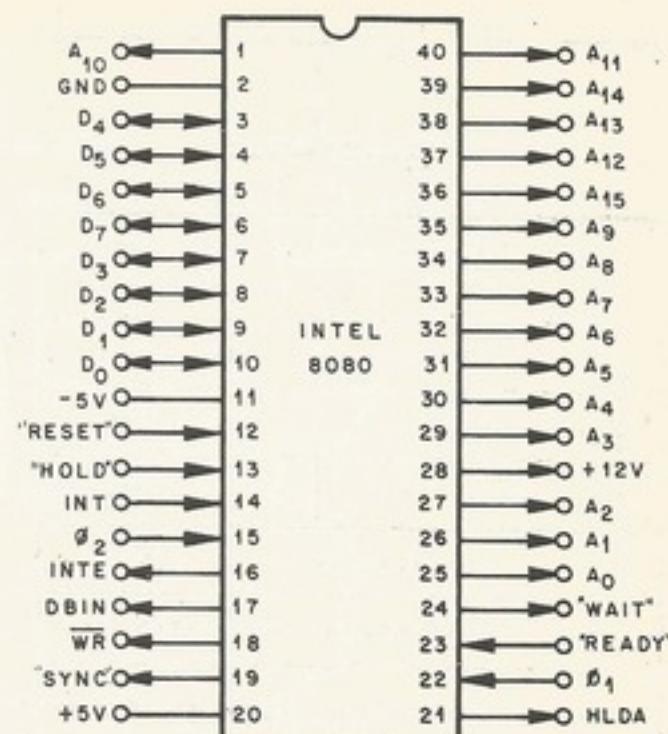


Diagrama funcional

Esse processador é organizado internamente como mostra a Fig. A.15, cujos terminais de entrada e saída estão na Fig. A.16, onde os terminais A₀ a A₁₅ formam a via de endereço, e D₀ a D₇ a de dados. O restante dos sinais são alimentação (GND, +5V, +12V, -5V), sinais de relógio (ϕ_1 , ϕ_2) e os sinais de controle.

A UCP 8080 é constituída de quatro unidades funcionais distintas: (1) rede de registradores e lógica de endereçamento, (2) unidade aritmética e lógica, (3) unidade de controle e (4) via de dados.

Rede de registradores e lógica de endereçamento

A memória local da UCP 8080 é constituída de seis registradores de 16 bits, organizados da seguinte forma: um par de registradores para armazenamento temporário de 8 bits cada; seis registradores de 8 bits para uso geral; um registrador de 16 bits usado como ponteiro da pilha (*stack pointer*), e um registrador de 16 bits como contador de instrução (veja a Fig. A.17).

O contador de instrução é automaticamente incrementado durante cada fase de busca (*fetch*), e indica o endereço da próxima instrução a ser executada. Existem instruções especiais para operar uma parte da memória organizada como pilha (*stack*). O ponteiro da pilha define, no começo do programa, o início da pilha. Cada vez que se introduzem dados na pilha (*push*), o ponteiro é decrementado. Quando se retiram dados da pilha (*pop*), o ponteiro é incrementado. Dessa forma a pilha cresce para baixo. A capacidade de administrar uma pilha na memória confere ao 8080 um ótimo recurso para controlar desvios a sub-rotinas, como será visto adiante.

Os seis registradores de uso geral (B, C, D, E, H e L) são diretamente endereçáveis por instruções, e podem ser utilizados ou isoladamente, ou em pares, formando registradores de 16 bits para uso geral. Os registradores temporários são utilizados pela unidade de controle durante a execução das instruções, e não são acessíveis ao programador.

A comunicação entre a memória local e a via interna de dados é feita através do multiplexor, em blocos de 8 bits. As transferências de 16 bits são realizadas internamente à memória local, podendo-se usar o decrementador/incrementador de endereços. Isto é, podem-se, sob controle de programa, transferir 2 bytes de um par de registradores para outro. O circuito de incrementação/decrementação é acionado durante o ciclo de busca, para incrementar o

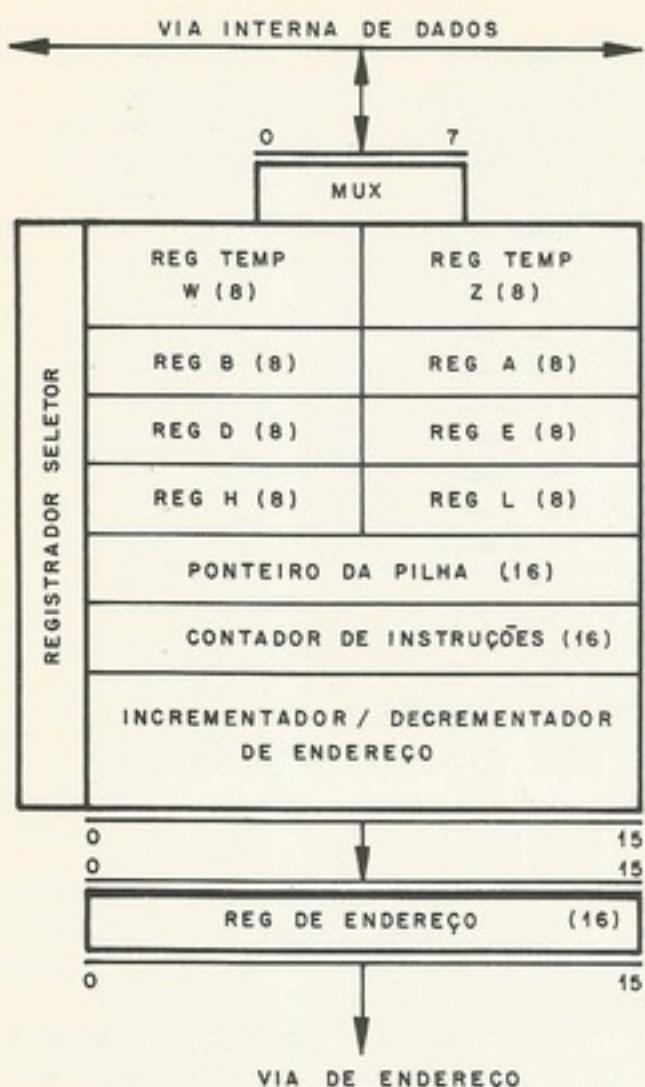


Figura A.17. Rede de registradores e lógica de endereçamento

contador de instrução e, durante os ciclos de execução, nas instruções que especificam tal operação em pares de registradores.

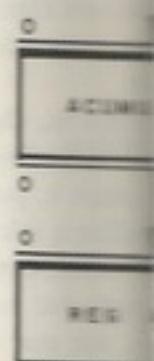
Unidade aritmética e lógica

A Fig. A.18 mostra a unidade aritmética e lógica em mais detalhes. As operações lógicas e aritméticas são realizadas na *ULA*, cujas entradas são os registradores *REG TEMP*, *REG ACUM* (*accumulator buffer*) e o *flip-flop* de "vai um" (um dos *flags*). O *REG TEMP*, por sua vez, recebe dados da via interna de dados, que podem ser provenientes da memória local, ou externos à *UCP*. As instruções aritméticas são, em geral, realizadas com o acumulador. Por exemplo, uma instrução *ADD R* move o conteúdo do registrador *R* para *REG TEMP*, transfere o acumulador para *REG ACUM*, e realiza a soma dos dois. No final da operação, armazena o resultado no acumulador.

O registrador de *flag*, que em muitos computadores recebe o nome de *status*, é composto pelos seguintes *bits*: zero (*Z*), "vai um" (*CY*), sinal (*S*), paridade (*P*) e "vai um" auxiliar (*AC*). O conteúdo desse registrador é carregado por instruções aritméticas e lógicas, em função do resultado da operação. As instruções de pulo condicional utilizam o registrador de *flag* para tomar a decisão lógica requerida.

Unidade de controle

Essa unidade de controle nada tem de especial quando comparada às já estudadas. É uma unidade de controle fixa. Durante a fase de busca, o primeiro *byte* da instrução é armazenado



no registrador de controle, em seguida os sinais de controle. O funcionamento

Via de controle

Devido à necessidade de dados comuns para serem utilizados

O circuito é isolada da interface, no modo de

b. Função

Os fabricantes neste texto. As designações

O tempo

Um ciclo de instrução depende da duração

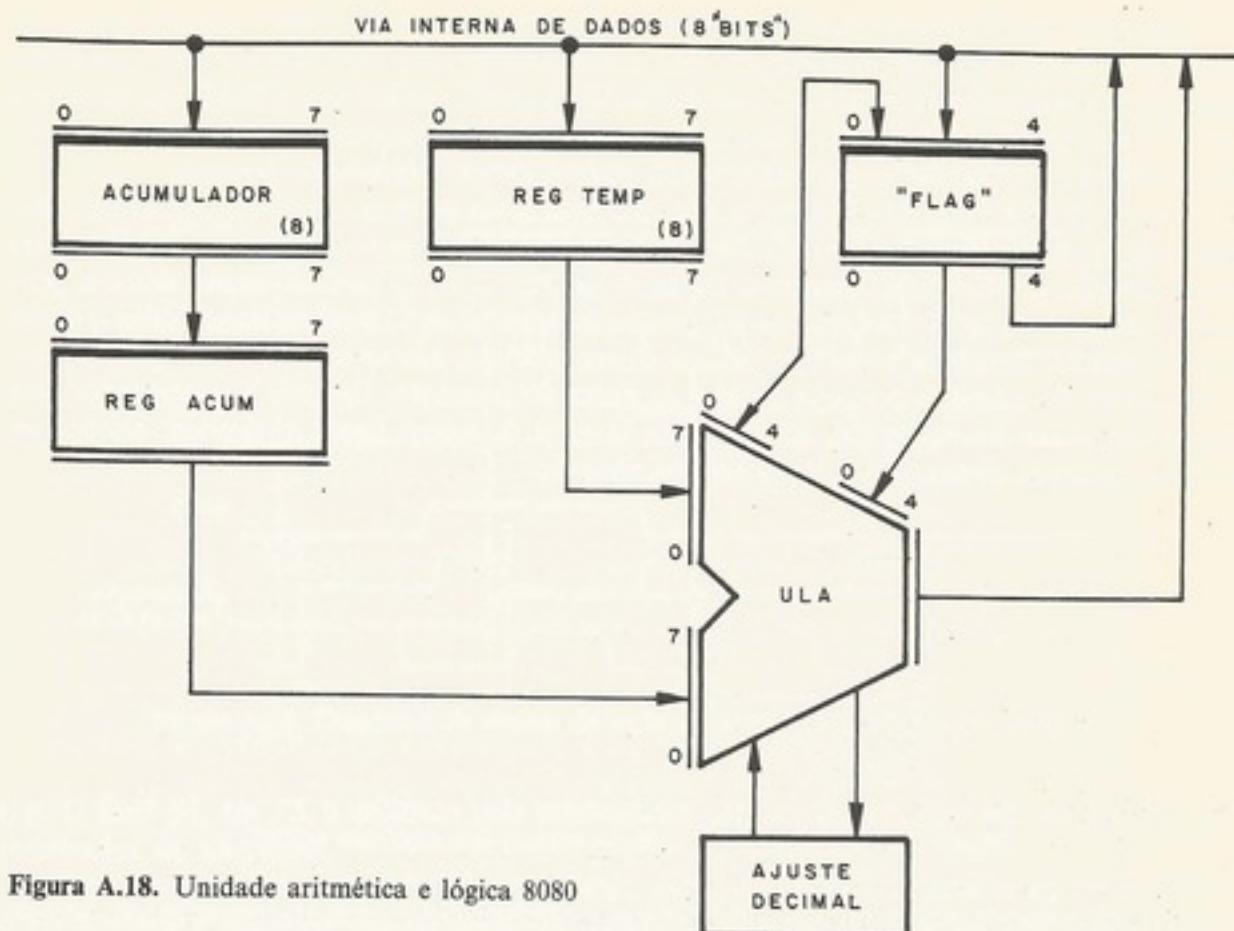


Figura A.18. Unidade aritmética e lógica 8080

no registrador de instrução — esse registrador contém o código de operação. A unidade de controle, em função do código de operação e tendo como referência sinais de relógio, fornece os sinais de controle do fluxo de dados (interno), além de gerar os sinais para controle externo. O funcionamento mais detalhado da unidade de controle será apresentado a seguir.

Via de dados

Devido à característica bidirecional dessa saída, existem excitadores (*drivers*) implementados com a tecnologia *tri-state*. Dessa forma, a via interna de dados fica isolada da via externa, para ser utilizada também na circulação de dados entre as unidades da UCP.

O excitador de saída tem três modos de operação: no modo *desligado*, a via externa fica isolada da interna; no modo *saída*, os dados existentes na via interna são forçados na externa; e, no modo *entrada*, os dados da via externa são colocados na via interna.

b. Funcionamento do microprocessador

Os fabricantes desse sistema utilizam uma nomenclatura um pouco diferente da definida neste texto. Para facilidade dos que vão estudar esse microprocessador mais detalhadamente, as designações constantes dos manuais serão mantidas.

O “timing”

Um *ciclo de instrução (instruction cycle)* é definido como o tempo requerido para retirar a instrução da memória e executá-la. O ciclo de instrução, cujas duração e composição dependem da instrução, é dividido em ciclos de máquina.

Um *ciclo de máquina* é caracterizado por um acesso à memória ou a uma interface de entrada/saída. Por exemplo, o primeiro ciclo de máquina de todo ciclo de instrução é utilizado para retirar a instrução da memória. Cada ciclo de máquina, por sua vez, é composto de alguns (3 a 5) ciclos de relógio, chamados de *estados (state)*. Durante um ciclo de máquina, os estados vão se sucedendo, de forma ordenada e sob supervisão da unidade de controle.

Um estado, por sua vez, é dividido em dois segmentos de tempo, não-superponíveis, chamados de ϕ_1 e ϕ_2 (fase 1 e fase 2). Esses tempos são definidos diretamente por osciladores que alimentam a UCP 8080, conforme indicado na Fig. A.19. O período desses sinais é, portanto, a duração de cada estado. Enquanto o relógio central estiver ativado, um estado sucede o outro, compondo ciclos de máquina que irão formar os ciclos de instrução. Quando o processador está parado, suspenso ou em espera, ele entra em estados de espera (*wait*) sucessivos, até que os circuitos externos comandem a mudança de situação.

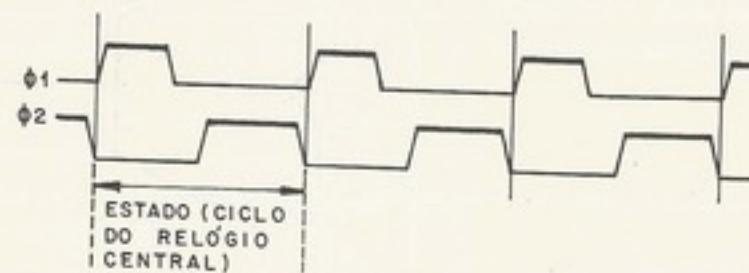


Figura A.19. Os sinais do relógio central que alimenta o 8080

A análise de alguns exemplos clarificará a organização do *timing* do processador. Inicialmente se analisará a instrução do tipo *ADD R* que irá somar o conteúdo do registrador *R* ao acumulador. O ciclo dessa instrução terá apenas um ciclo de máquina, já que o único acesso à memória é o de busca da instrução (*fetch*). Os operandos se encontram armazenados dentro da *UCP*. Esse único ciclo de máquina é iniciado por um estado *T1*, onde o endereço da instrução é colocado na via de endereço.

O estado seguinte, *T2*, é utilizado pelo processador para fornecer ao circuito externo informações (*status*) sobre o tipo de ciclo de máquina que está ocorrendo. Essa informação, o *status* — que não deve ser confundido com estado (ciclo do relógio principal) — é utilizada para controlar os circuitos externos.

O estado seguinte, *T3*, é utilizado para carregar o processador com o conteúdo da posição da memória endereçada no *T1* anterior. Esse conteúdo já deve estar presente na via de dados. Caso se utilize uma memória mais lenta, esta deverá acionar a linha de entrada do 8080 denominada *ready* (pronto), de forma a obrigá-lo a aguardar a leitura na memória, por um tempo conveniente.

O último estado do único ciclo de máquina que compõe o ciclo de instrução, neste exemplo, é o *T4*. Esse estado é utilizado para executar a instrução que acabou de ser lida na memória e que, no estado *T3* anterior, fora carregada no registrador de instrução. Sua execução é simples, e tudo ocorre internamente ao 8080; portanto um só estado é suficiente. O conteúdo do registrador *R* (endereçado pela instrução) é lido na memória local e transferido, através da via interna de dados, para o *REG TEMP*. Esse byte é então somado ao acumulador.

O diagrama de tempo dos principais sinais envolvidos na descrição precedente é visto na Fig. A.20. Nessa figura vemos que o endereço fica estável na via de endereço, desde a metade de *T1* até a metade do estado que sucede *T3*. O sinal ϕ_2 , em *T1*, liga o sinal *SYNC*, que é desligado com o ϕ_2 seguinte. O sinal *SYNC*, que caracteriza o primeiro estado de todo ciclo de máquina, é usado, por exemplo, para carregar a informação de *status* que, durante esse tempo, está presente na via de dados [região (1)]. No início de *T3*, os dados da memória, sob controle do sinal *DBIN* (*data bus in*), gerado pelo 8080, são colocados na via de dados [região

(2)]. O sinal *registrador* é ligado quando o processador é realizado para *ready = 1*.

ENDEREÇO

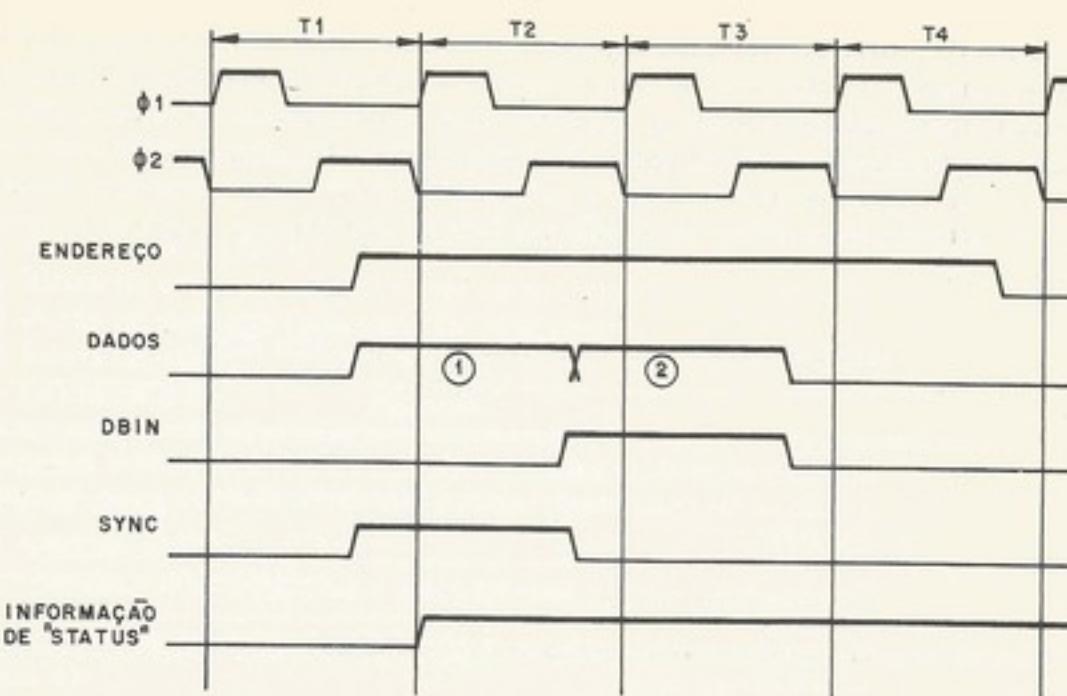
DADOS

DBIN

READY

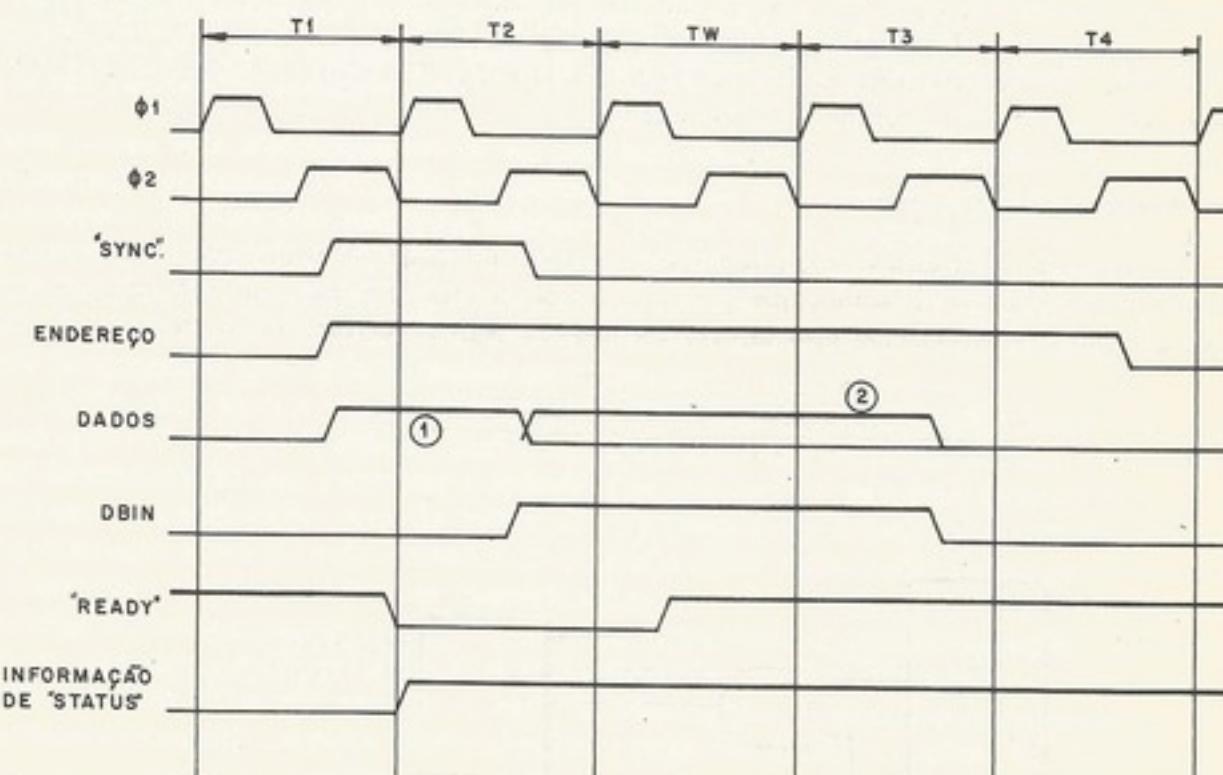
INFORMAÇÃO DE STATUS

Figura A.20. Diagrama de tempo dos principais sinais envolvidos na descrição precedente

Figura A.20. Diagrama de *timing* do ciclo de instrução da instrução *ADD R*

(2)]. O sinal ϕ_1 , em T_3 , é utilizado pelo processador para carregar o byte lido na memória no registrador de instrução.

Quando se utiliza uma memória mais lenta, esta terá de gerar o sinal *ready* para obrigar o processador a aguardar alguns ciclos de relógio, enquanto a mesma termina a leitura. Isso é realizado pelo processador através da inclusão de alguns estados de espera (TW), enquanto $ready = 0$. Nos demais estados, os sinais são como os da Fig. A.20 (veja a Fig. A.21).

Figura A.21. Diagrama de *timing* da instrução *ADD R* com o sinal *ready* gerando um estado de espera TW

Todo ciclo de instrução é formado por alguns ciclos de máquina — tantos quantos forem os acessos à memória ou a interfaces de entrada e saída. Cada ciclo de máquina tem, no mínimo, três estados (T_1 , T_2 , T_3), podendo ser ampliado com alguns estados TW (entre T_2 e T_3). Outros estados, T_4 e T_5 ocorrem, dependendo das operações que são executadas com o byte que, em T_3 , é carregado no processador. O processador utiliza os três primeiros estados de cada ciclo de máquina para enviar ou extraír um byte da memória ou interface de entrada e saída (Tab. A.1).

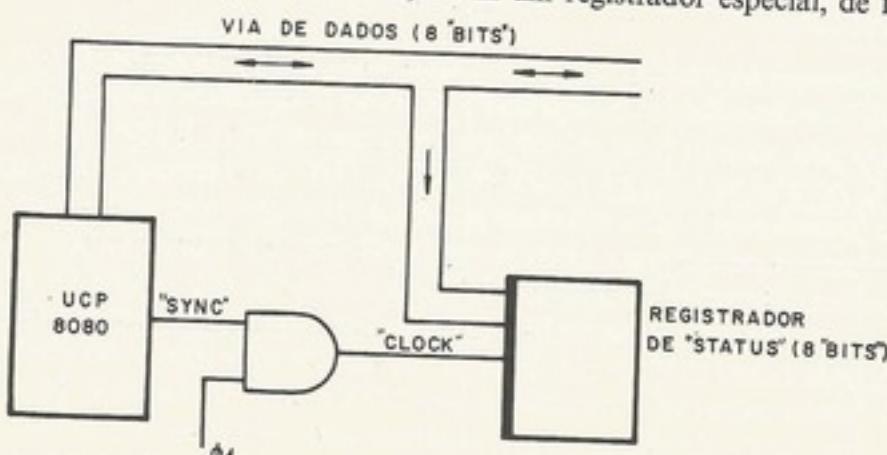
Tabela A.1. Descrição dos estados do processador 8080

Estado	Descrição
T_1	Coloca o endereço de memória, ou o número de um dispositivo de entrada e saída na via de endereço; coloca a informação de <i>status</i> na via de dados; liga <i>SYNC</i>
T_2	O processador testa as entradas <i>ready</i> e a instrução <i>halt</i> ; desliga <i>SYNC</i>
TW (opcional)	O processador entra no estado de espera se <i>ready</i> for baixo, ou se a instrução de <i>halt</i> foi executada
T_3	Nesse estado, ou a informação da via de dados ($DBIN = $ alto) é carregada no processador, ou o processador comanda a escrita dessa via na memória ou interface de entrada/saída ($WR' = $ baixo)
T_4 e T_5 (opcionais)	Esses estados são utilizados pelo processador para executar as operações internas que a instrução requer

Uma instrução de soma que requer dois ciclos de máquina é a *ADD M*, com dois acessos à memória. Essa instrução faz com que um byte armazenado na memória, cujo endereço se encontra nos registradores *H* e *L* (16 bits), seja somado ao acumulador. No primeiro ciclo de máquina (estados T_1 , T_2 e T_3), o processador lê a instrução na memória e a coloca no registrador de instrução. No segundo ciclo de máquina, transfere o conteúdo dos registradores *H* e *L* para a via de endereço, e lê o operando (estados T_1 , T_2 e T_3); no último estado (T_4) desse ciclo, ele soma o operando ao acumulador.

A informação de "status"

No meio de cada estado T_1 , que inicia todo ciclo de máquina, o processador coloca na via de dados a palavra de *status*, que especifica o tipo de ciclo que está sendo iniciado. O circuito externo deve armazenar essa informação em um registrador especial, de forma a poder

Figura A.22. O registrador de *status*

moldar seu comportamento a ele. Apesar disso, o processador fornece ainda, como se mostrou anteriormente, outros sinais de controle direto como *DBIN*, *WR* (para escrever na memória o conteúdo da via de dados), *SYNC*, *wait*, etc. (Fig. A.16). A informação de *status*, armazenada em circuitos externos (Fig. A.22), define, portanto, o ciclo de máquina, que será um dos seguintes:

- 1) ciclo de busca (*fetch*);
- 2) leitura de memória (operando ou endereço);
- 3) escrita de memória (operando ou endereço);
- 4) leitura da pilha;
- 5) escrita na pilha;
- 6) entrada;
- 7) saída;
- 8) interrupção;
- 9) suspenso (*hold*);
- 10) parado e interrompido.

O tipo do ciclo de máquina é codificado nos 8 bits de *status* (para maiores detalhes, veja a referência 6).

Esquema de interrupção

Qualquer dispositivo externo pode interromper o processador, desde que a interrupção seja permitida, o que é indicado pelo sinal *INTE* (*interrupt enable*). Provoca-se a interrupção pela elevação do sinal *INT* para o nível "1". Quando isso acontece, após o fim da instrução corrente, o processador, ao invés de executar um ciclo de busca (*fetch*), iniciará o ciclo de interrupção.

No ciclo de interrupção, o dispositivo que causou a interrupção força, na via de dados, uma instrução de *RST* (*restart*). Essa instrução provocará as seguintes ações:

- 1) armazenamento do conteúdo do contador de instrução na pilha;
- 2) pulo para uma posição de memória especificada, pelo dispositivo, em um campo de três bits existente na instrução *RST*.

Dessa forma, uma interrupção simula um pulo para uma sub-rotina. O conteúdo do contador de instrução armazenado na pilha é recuperado quando se retorna do programa tratador de interrupção. Existem oito endereços diferentes que podem ser forçados pelo dispositivo que causa a interrupção; 0, 8, 16, 24, 32, 40, 48 e 56. Nesses endereços existem os começos de rotinas tratadoras de interrupção, que irão salvar o conteúdo dos diversos registradores do processador, e proceder à execução da tarefa requerida pela interrupção. Tais rotinas são escritas pelo programador para cada aplicação específica.

A Fig. A.23 mostra um diagrama de sinais existentes no início do atendimento de uma interrupção. O sinal *INT*, subindo para "1", causa, no próximo ciclo de busca, a queda para "0" do sinal *INTE*. Isso provocará a inibição dos pedidos de interrupção subsequentes. Esse ciclo de busca será muito parecido com o ciclo de busca normal, onde o processador inibe a incrementação automática do contador de instrução. Nesse ciclo, o dispositivo que provocou a interrupção bloqueia a leitura na memória e força, na via de dados, a instrução *RST* com o endereço da rotina tratadora correspondente. Para o processador, tudo se passa como se tivesse lido na memória a instrução *RST*. Essa instrução, cujos ciclos de execução aparecem na Fig. A.23, nos dois ciclos que sucedem o ciclo de interrupção, provocará o armazenamento do contador de instrução na pilha. O primeiro desses ciclos decrementa o ponteiro da pilha e armazena nessa posição da memória o byte mais significativo do contador de instrução. O segundo ciclo decrementa mais uma vez o ponteiro da pilha, que indicará a posição onde é armazenado o byte menos significativo do contador de instrução.

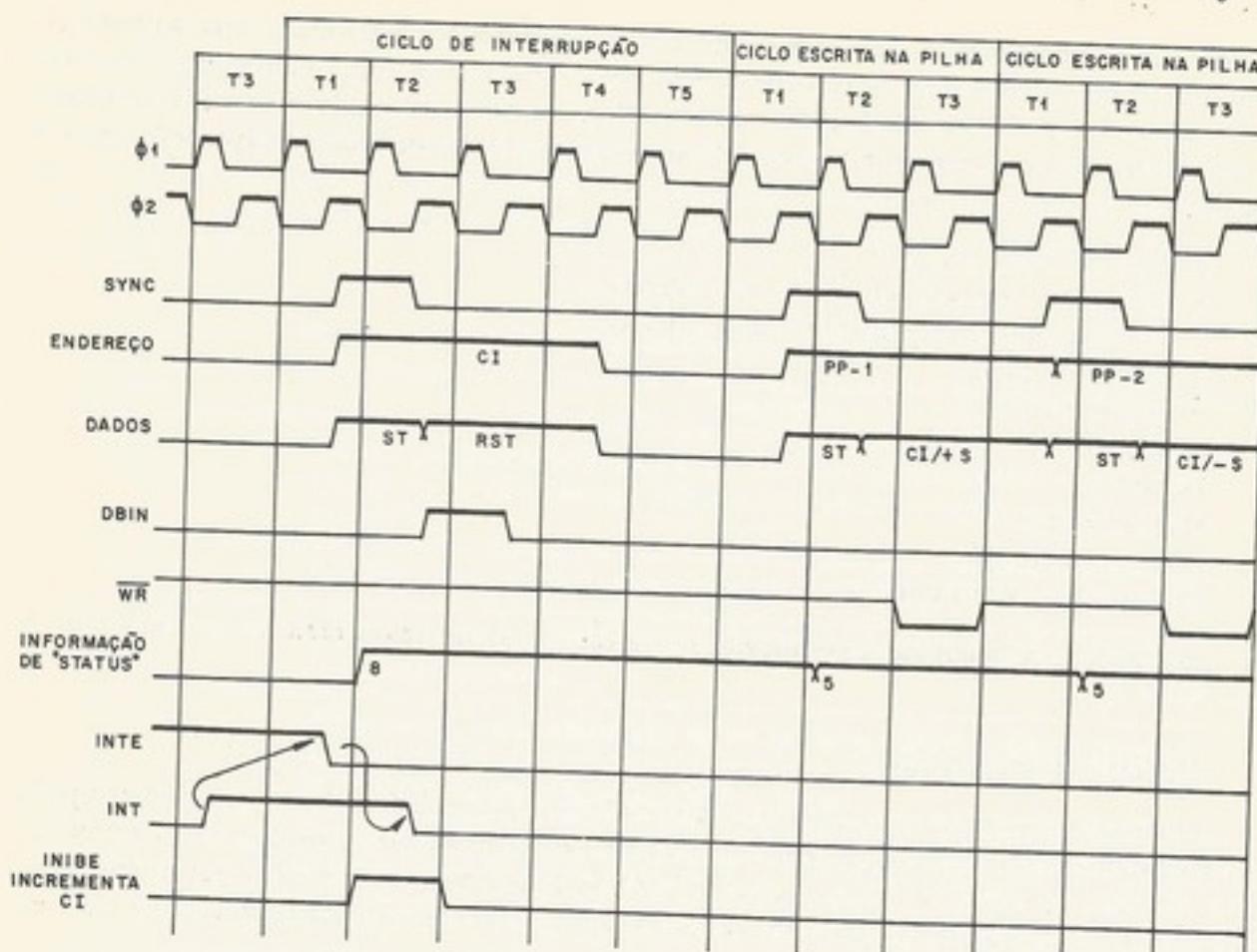


Figura A.23. Diagrama de timing da interrupção. CI = contador de instrução; PP = ponteiro de pilha; ST = status; RST = instrução que inicia o tratamento de interrupção; $CI+S$ = byte mais significativo do CI ; $CI-S$ = byte menos significativo do CI .

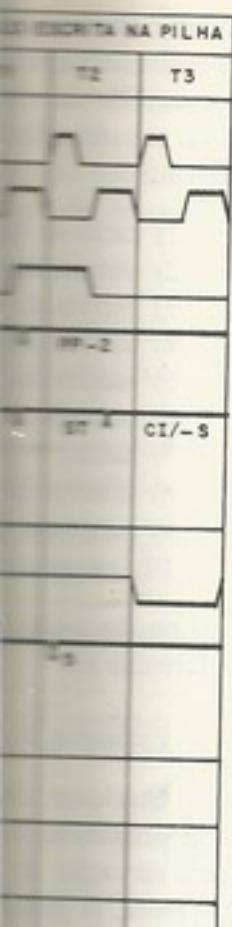
No final do segundo ciclo de escrita na pilha, o processamento é desviado para a instrução armazenada na posição de memória especificada pela instrução RST . Nos oito endereços possíveis (0, 8, 16, 24, 32, 40, 48 e 56), existem desvios para oito rotinas de tratamento de interrupção diferentes.

Esquema de *hold* (suspende)

Realiza-se o acesso direto à memória do 8080 ao forçar-se o sinal *hold* para o nível “1”. Essa ação faz com que o processador termine o ciclo de máquina corrente e libere as vias de endereço e dados para serem usados pelo circuito externo. Enquanto a memória é operada pelos circuitos externos (acesso direto), o processador fica suspenso, gerando estados de “suspenso” (*hold*) sucessivos. O sinal de saída *HLDA* (*hold acknowledge*) informa os circuitos externos que o processador está em suspenso. Com a liberação do sinal *hold*, o processador retoma o processamento normal. Esse esquema mostra claramente o acesso direto à memória feito por ciclos “roubados”.

Consegue-se parar o processamento através de uma chave no painel, atuando-se na linha de *hold*, mesmo que não se realizem acessos diretos à memória. O controle externo desse sinal permite também que se opere o sistema no modo “instrução única”. Isso é obtido da seguinte forma: colocada a máquina no modo suspenso (*hold*), a cada vez que se pressiona um botão “instrução única”, o sinal *hold* cai para zero durante um tempo suficiente para executar a instrução seguinte. O inconveniente é que, uma vez executada a instrução de *halt* (veja a seguir), não se consegue ir para a instrução seguinte.

apêndice	... m
A instru...	
A execuç...	
de estados de	
ficam liberados	
quando:	
1) um ...	
2) um ...	
O sinal R...	
O RST	
sinal armazena...	
existente no en...	
A esse sinal, pa...	
o trabalho. O...	
Sinal	
A0 a A15	
D0 a D7	
GND	
-5 V	
+5 V	
+12 V	
reset	
hold	
INT	
φ1, φ2	
INTE	
DBIN	
WR	
SYNC	
HLDA	
ready	
wait	



PP = ponteiro de
S = byte mais

para a instrução
de endereços
de tratamento de

nível "1".
libere as vias de
é operada
estados de "sus-
circuito exter-
processador retoma
memória feito

seja na linha
termo desse sinal
mão da seguinte
um botão
executar a instru-
seguir),

A instrução "pare" (*halt*)

A execução da instrução de pare (*halt*) faz com que o processador entre em uma sequência de estados de espera (T_{WH}), onde não existe processamento, e as vias de dados e endereços ficam liberadas para uso por dispositivos externos. O processador escapa dessa situação apenas quando:

- 1) um sinal na linha *RESET* o obriga a iniciar um ciclo de busca na posição zero; ou
- 2) um sinal de interrupção que, se atendido (*INT* = 1), irá forçar a instrução *RST*.

O sinal *RESET*

O *RESET* é um sinal cuja função é a de impor condições iniciais ao processador. Esse sinal armazena zero no contador de instrução, obrigando o processamento iniciar na instrução existente no endereço zero. Além disso, esse sinal deve forçar o início de um ciclo de busca. A esse sinal, portanto, deve-se ligar uma chave do painel, para permitir que o operador reinicie o trabalho. Outro sinal que deve acionar essa entrada é o *LAL* ("limpa ao ligar").

Tabela A.2. Sinais nos pinos do processador Intel 8080

Sinal	Pino	Descrição
A0 a A15	1, 25, 26, 27, 29 a 40	Via de endereço
D0 a D7	3 a 10	Via de dados (bidirecional)
<i>GND</i>	2	Terra
-5 V	11	-5 V
+5 V	20	+5 V
+12 V	28	+12 V
<i>reset</i>	12	Entrada, limpa o contador de instrução, reiniciando processamento no endereço zero
<i>hold</i>	13	Entrada, para o processamento — acesso direto à memória pode ser realizado
<i>INT</i>	14	Entrada, interrompe o processamento normal e permite que seja forçado o código da instrução <i>RST</i> na via de dados
ϕ_1, ϕ_2	15, 22	Externo, sinais de relógio
<i>INTE</i>	16	Saída, informa que interrupções estão sendo aceitas
<i>DBIN</i>	17	Saída, sinalização para a memória ou interface colocar informação na via de dados
<i>WR</i>	18	Saída, sinalização para a memória ou interface armazenar o conteúdo da via de dados
<i>SYNC</i>	19	Saída, sinal gerado em todo inicio do ciclo de máquina para sinalizar que a informação de <i>status</i> está na via de dados
<i>HLDA</i>	21	Saída, informa que o processador está no estado de <i>hold</i>
<i>ready</i>	23	Entrada, força o processador a aguardar o término da leitura, ou escrita, de dados na memória
<i>wait</i>	24	Saída, sinaliza que o processador está esperando término de leitura na memória

Os sinais nos terminais do processador

A Tab. A.2 apresenta um resumo das funções dos sinais nos quarenta pinos do processador Intel 8080.

c. O conjunto de instruções

As instruções do 8080 são organizadas em *bytes* de 8 bits. O primeiro byte de cada instrução é chamado de código de operação, embora, muitas vezes, contenha também endereços de registradores. Muitas instruções são formadas por apenas um byte: são aquelas que não referenciam a memória ou têm referência implícita. As instruções de 2 bytes de comprimento utilizam o segundo byte para dados (nas imediatas) ou o número de dispositivo de entrada e saída. As instruções de referência direta à memória têm 3 bytes, onde os dois últimos formam o endereço (16 bits para endereçar até 64K bytes). Veja a Fig. A.24.

Modos de endereçamento

O processador tem quatro modos distintos de endereçamento: direto, registrador, indireto por registrador e imediato.

Direto

No modo direto, os bytes 2 e 3 da instrução contêm o endereço final do operando.

Registrador

O operando está localizado na memória local, e seu número está especificado no código de operação.

Indireto por registrador

O código de operação especifica um par de registradores locais que contém o endereço do operando.

Imediato

O operando é o segundo byte da instrução.

São esses os modos gerais de endereçamento. Existem casos especiais, como a instrução *RST*. Nessa instrução, o endereço na memória é especificado por 3 bits, cujo conteúdo binário é multiplicado por oito, no cálculo do endereço final, que pode ser 0, 8, 16, 24, 32, 40, 48 ou 56. Outra instrução de endereçamento especial é a *RET*, usada no retorno de sub-rotinas. Essa instrução extrai da pilha o endereço da instrução seguinte.

A relação das diferentes instruções está apresentada na Tab. A.3. Maiores detalhes do funcionamento dessas instruções podem ser encontrados na Referência 6. Essa tabela, cujas instruções foram agrupadas da forma usada na Referência 7, apresenta uma coluna, com nome em inglês, para facilitar a memorização do mnemônico.

d. O microprocessador Intel 8080: sumário

O microprocessador descrito tem seus pontos altos na capacidade de endereçamento (até 64K bytes), e na sua forma de administração da pilha — de profundidade infinita —, onde se armazenam os endereços de retorno de sub-rotinas.

registros do processador

o byte de cada instrução também endereços sólidos aquelas que não possuem de comprimento fixo de entrada e saída os últimos formam

register, indireto

al do operando.

specificado no código

contém o endereço

, como a instrução cujo conteúdo binário é 16, 24, 32, 40, 48 bytes de sub-rotinas.

Mais detalhes do Tabela, cujas coluna, com nome

de endereçamento infinita —, onde

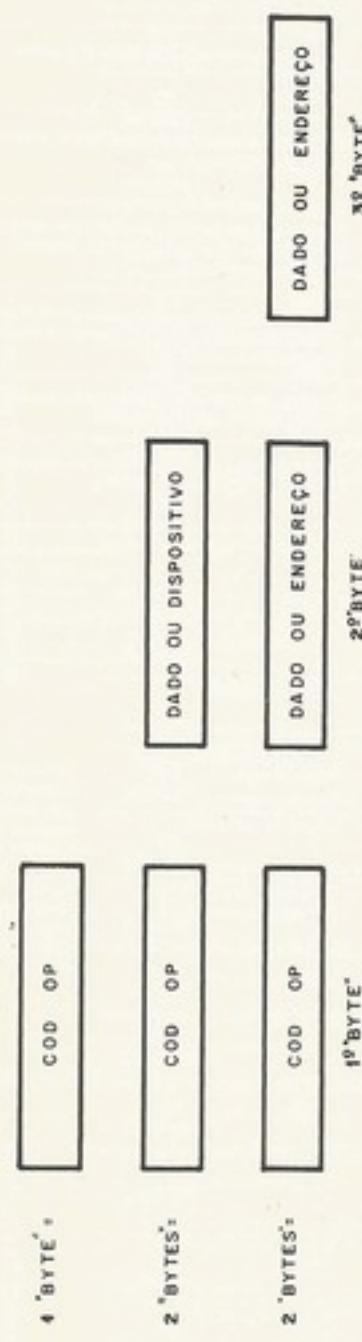


Figura A.24. O formato das instruções do 8080

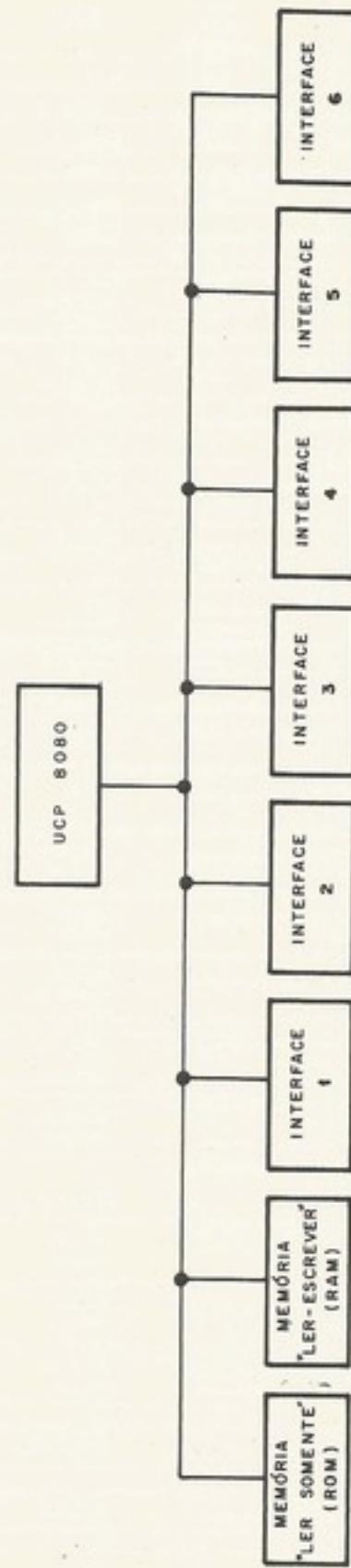


Figura A.25. Organização do controlador

Tabela A.3. Conjunto de instruções do microprocessador Intel 8080

Tipo	Cod. inst. 19 byte	Outro(s) byte(s) da instrução	Nome	Quant. de bytes da instrução	Número de ciclos de relógio	Operação	Tipo
E/S	IN 11011011	DISP	Input	2	10	Carrega no acumulador dados do dispositivo DISP (DISP = 0 a 255)	
	OUT 11010011	DISP	Output	2	10	Descarrega o acumulador no dispositivo de número DISP (DISP = 0 a 255)	
	STA 00110010	END	Store accumulator direct	3	13	Armazena o acumulador na posição da memória endereçada	
	LDA 00111010	END	Load accumulator immediate	3	13	Carrega o acumulador com o conteúdo da posição da memória endereçada	
	STAX, RP 00RP0010		Store accumulator indirect	1	7	RP especifica um par de registradores que contém o endereço de posição da memória onde armazena o conteúdo do acumulador	
	LDAX, RP 00RP1010		Load accumulator indirect	1	7	RP especifica um par de registradores que contém o endereço da memória de onde é extraído o conteúdo para carregar no acumulador	
	MOV, M, R 01110RRR		Move to memory	1	7	Armazena o registrador R na posição da memória endereçada pelos registradores H e L	
	MOV, R, M 01RRR110		Move from memory	1	7	Carrega o registrador R com o conteúdo da posição endereçada pelos registradores H e L	
	LHLD 00101010	END	Load H and L direct	3	16	Carrega registradores H e L com o conteúdo das posições de memória endereçadas por END	
	SHLD 00100010	END	Store H and L direct	3	16	Armazena os registradores H e L nas posições de memória endereçadas por END	
	ADD, M 11000110		Add memory	1	7	Soma no acumulador	
	ADC, M 10001110		Add memory with carry	1	7	Soma no acumulador com bit "vai um" (CY)	Todas estas instruções utilizam como operando o conteúdo da posição da memória endereçada pelos registradores H e L
	SUB, M 10010110		Subtract memory	1	7	Subtrai do acumulador	
	SBB, M 10011110		Subtract memory with borrow	1	7	Subtrai do acumulador com bit "empresta um" (CY)	
	ANA, M 10100110		And memory	1	7	AND com acumulador	
	XRA, M 10101110		Exclusive or memory	1	7	exclusive-or com acumulador	
	ORA, M 10110110		Or memory	1	7	OR com acumulador	
	CMP, M 10111110		Compare memory	1	7	Compara com acumulador	
	INR, M 00110100		Increment memory	1	10	Incrementa a memória	
	DCR, M 00110101		Decrement memory	1	10	Decrementa a memória	
	LXI, RP 00RP0001	Dado 16	Load register pair immediate	3	10	Carrega 2 bytes de dados (immediatos) no par de registradores indicado por RP	
INSTRUÇÕES IMEDIATAS E PULOS	MVI, M 00110110	Dado	Move to memory immediate	2	10	Armazena 1 byte de dados (imediato) na posição da memória endereçada pelos registradores H e L	
	MVI, R 00RRR110	Dado	Move immediate	2	7	Carrega 1 byte de dados (imediato) no registrador R	
	JMP 11000011	END	Jump	3	10	Pula para a instrução na posição END	

REFERÊNCIA PRIMÁRIA À MEMÓRIA

INSTRUÇÕES LÓGICAS E ARITMÉTICAS COM REFERÊNCIA À
MEMÓRIAINSTRUÇÕES IMEDIATAS
E PULOS

CHAMADAS E RETORNOS DE SUBROTINAS

OPERAÇÕES IMEDIATAS (ARITMÉTICAS E LÓGICAS)

Tabela A.3. (continuação)

Tipo	Cod. inst. 1º byte	Outro(s) byte(s) da instrução	Nome	Quant. de bytes da instrução	Número de ciclos de relógio	Operação
CHAMADAS E RETORNOS DE SUB-ROTIÑAS	CALL 11001101	END	Call	3	17	Pula para sub-rotina na posição END
	CC 11011100	END	Call on carry	3	11/17	Pula para sub-rotina na posição END se "vai um" = 1 (CY = 1)
	CNC 11010100	END	Call on no carry	3	11/17	Pula para sub-rotina na posição END se "vai um" = 0 (CY = 0)
	CZ 11001100	END	Call on zero	3	11/17	Pula para sub-rotina na posição END se zero (Z = 1)
	CNZ 11000100	END	Call on not zero	3	11/17	Pula para sub-rotina na posição END se não-zero (Z = 0)
	CP 11110100	END	Call on plus	3	11/17	Pula para sub-rotina na posição END se positivo (S = 0)
	CM 11111100	END	Call on minus	3	11/17	Pula para sub-rotina na posição END se negativo (S = 1)
	CPE 11101100	END	Call on parity even	3	11/17	Pula para sub-rotina na posição END se paridade par (P = 1)
	CPO 11100100	END	Call on parity odd	3	11/17	Pula para sub-rotina na posição END se paridade ímpar (P = 0)
	RET 11001001		Return	1	10	Retorna da sub-rotina (o endereço de volta é extraído da pilha)
OPERAÇÕES IMEDIATAS (ARITMÉTICAS E LÓGICAS)	RC 11011000		Return on carry	1	5/11	Retorna da sub-rotina se "vai um" = 1 (CY = 1)
	RNC 11010000		Return on no carry	1	5/11	Retorna da sub-rotina se "vai um" = 0 (CY = 0)
	RZ 11001000		Return on zero	1	5/11	Retorna da sub-rotina se zero (Z = 1)
	RNZ 11000000		Return on not zero	1	5/11	Retorna da sub-rotina se não-zero (Z = 0)
	RM 11111000		Return on minus	1	5/11	Retorna da sub-rotina se negativo (S = 1)
	RP 11110000		Return on plus	1	5/11	Retorna da sub-rotina se positivo (S = 0)
	RPE 11101000		Return on parity even	1	5/11	Retorna da sub-rotina se paridade par (P = 1)
	RPO 11100000		Return on parity odd	1	5/11	Retorna da sub-rotina se paridade ímpar (P = 0)
	ADI 11000110	Dado	Add immediate	2	7	Soma dado imediato no acumulador
	ACI 11001110	Dado	Add immediate with carry	2	7	Soma com "vai um" (CY) o dado imediato no acumulador
OPERADORES DE COMPARAÇÃO	SUI 11010110	Dado	Subtract immediate	2	7	Subtrai dado imediato do acumulador
	SRI 11011110	Dado	Subtract immediate with borrow	2	7	Subtrai, com bit empréstimo (CY) o dado imediato do acumulador
	ANI 11100110	Dado	And immediate	2	7	AND dado imediato com acumulador
	XRI 11101110	Dado	Exclusive or immediate	2	7	exclusive-or dado imediato com acumulador
	ORI 11110110	Dado	Or immediate	2	7	OR dado imediato com acumulador
	CPI 11111110	Dado	Compare immediate	2	7	Compara dado imediato com acumulador
	SHL 11111100	Dado	Shift left immediate	2	7	Shift left dado imediato

Tabela A.3. (continuação)

Tipo	Cod. inst. 1º byte	Outro(s) byte(s) da instrução	Nome	Quant. de bytes da instrução	Número de ciclos de relógio	Operação
PULOS CONDICIONAIS	JC 11011010	END	Jump on carry	3	10	Pula para posição END se "vai um" = 1 ($CY = 1$)
	JNC 11010010	END	Jump on no carry	3	10	Pula para posição END se "vai um" = 0 ($CY = 0$)
	JZ 11001010	END	Jump on zero	3	10	Pula para posição END se zero ($Z = 1$)
	JNZ 11000010	END	Jump on not zero	3	10	Pula para posição END se não-zero ($Z = 0$)
	JP 11110010	END	Jump on plus	3	10	Pula para posição END se positivo ($S = 0$)
	JM 11111010	END	Jump on minus	3	10	Pula para posição END se negativo ($S = 1$)
	JPE 11101010	END	Jump on parity even	3	10	Pula para posição END se paridade par ($P = 1$)
TRANSFERÊNCIAS ENTRE REGISTRADORES	JPO 11100010	END	Jump on parity odd	3	10	Pula para posição END se paridade ímpar ($P = 0$)
	MOV, R ₁ , R ₂ 01R ₂ R ₃ R ₁ R ₃	Move register		1	5	Transfere registradores R_1 para registradores R_2
	XCHG 11101011		Exchange H and L with D and E.	1	4	Troca registradores DE com HL.
	SPHL 11111001		Move HL to SP	1	5	Transfere registradores HL para o ponteiro da pilha
	INR, R 00RRR100		Increment register	1	5	Incrementa registrador R
	DCR, R 00RRR101		Decrement register	1	5	Decrementa registrador R
	CMA 00101111		Complement accumulator	1	4	Complementa acumulador
OPERAÇÕES SOBRE REGISTRADORES	DAA 00100111		Decimal adjust accumulator	1	4	Ajusta decimal no acumulador
	RLC 00000111		Rotate left	1	4	Gira acumulador à esquerda, anota "vai um" em CY
	RRC 00001111		Rotate right	1	4	Gira acumulador à direita, anota "vai um" em CY
	RAL 00010111		Rotate left through carry	1	4	Gira acumulador à esquerda com "vai um" (CY)
	RAR 00011111		Rotate right through carry	1	4	Gira acumulador à direita com "vai um" (CY)
	DAD, RP 00RP1001		Add register pair to H and L	1	10	Soma par de registradores RP a H e L
	INX, RP 00RP0011		Increment register pair	1	5	Incrementa par de registradores RP
	DCX, RP 00RP1011		Decrement register pair	1	5	Decrementa par de registradores RP
	STC 00110111		Set carry	1	4	Liga "vai um"
	CMC 00111111		Complement carry	1	4	Complementa "vai um"

OPERAÇÕES ENTRE REGISTRADORES

OPERAÇÕES COM PILHA

MISCÉLANEA

TIPO

Tabela A.3. (continuação)

	Tipo	Cod. inst. 19 byte	Outro(s) byte(s) da instrução	Nome	Quant. de bytes da instrução	Número de ciclos de relógio	Operação
adicionar "vai um" = 1 (CY = 1)		ADD, R 10000RRR		Add register	1	4	Soma registrador <i>R</i> no acumulador
adicionar "vai um" = 0 (CY = 0)		ADC, R 10011RRR		Add register with carry	1	4	Soma registrador <i>R</i> , com "vai um" (CY), no acumulador
subtrair (Z = 1)		SUB, R 10010RRR		Subtract register	1	4	Subtrai registrador <i>R</i> do acumulador
subtrair (Z = 0)		SBB, R 10011RRR		Subtract register with borrow	1	4	Subtrai registrador <i>R</i> , com "empresta um" (CY), do acumulador
multiplicar (Z = 0)		ANA, R 10100RRR		And register	1	4	AND registrador <i>R</i> com acumulador
multiplicar (Z = 1)		XRA, R 10101RRR		Exclusive or register	1	4	exclusive-or registrador com acumulador
dividir por (P = 1)		ORA, R 10110RRR		Or register	1	4	OR registrador <i>R</i> com acumulador
dividir por (P = 0)		CMP, R 10111RRR		Compare register	1	4	Compara registrador <i>R</i> com acumulador
armazenar <i>R</i> , para registrador <i>R</i>	OPERAÇÕES ENTRE REGISTRADORES	PUSH, RP 11RP0101		Push	1	11	Armazena conteúdo do par de registradores <i>RP</i> na pilha
carregar <i>R</i> , para o ponteiro da pilha	OPERAÇÕES COM PILHA	POP, RP 11RP0001		Pop	1	10	Carrega conteúdo do topo da pilha no par de registradores <i>RP</i>
interrupção <i>I</i>	INTERRUPÇÃO	XTHL 11100011		Exchange stack top with H and L	1	18	Troca topo da pilha com registradores <i>H</i> e <i>L</i>
interrupção <i>I</i>		EI 11111011		Enable interrupts	1	4	Permite (enable) interrupção
interrupção <i>I</i>		DI 11110011		Disable interrupts	1	4	Proíbe (disable) interrupção
reiniciar		RST, N 11NNNN11		Restart	1	11	Restart, endereço 8 x (NNN)
	MISCELÂNEA	NOP 00000000		No op	1	4	Sem operação
reseta, soma "vai um" em CY		HLT 01110110		Halt	1		Pausa
reseta, soma "vai um" em CY		PUSH PSW 11110101		Push processor status word	1	11	Salva o acumulador e o registrador de flag na pilha
reseta com "vai um" (CY)		POP PSW 11110001		Pop processor status word	1	10	Restaura o valor do acumulador e do registrador de flag com valores extraídos da pilha
reseta com "vai um" (CY)							
reseta com <i>H</i> e <i>L</i>							
reseta com <i>RP</i>							
reseta com <i>RP</i>							
"Reset"							

O esquema de interrupção, apesar de simples, tem sua "potência" aumentada quando se inserem circuitos externos para gerenciar as prioridades. O atendimento às interrupções tem sua programação simplificada com o uso das instruções *push PSW* e *pop PSW*, que realizam o salvamento e a restauração automática do contexto do programa interrompido. Seu conjunto de instruções é bastante balanceado, sendo adequado em aplicações "tempo real".

Um de seus pontos fracos é a forma de endereçamento⁽⁸⁾: a falta de endereçamento indexado causa problemas em algumas aplicações. O fluxo de dados desse processador é bastante versátil, com a disponibilidade de uma memória local, além do acumulador. A colocação do contador de instrução como um registrador endereçado pelo programa aumenta sua potência, permitindo, contudo, que programadores "espertos" consigam realizar tarefas impossíveis de ser entendidas por terceiros.

Para evitar que o projetista codifique em linguagem de máquina, o fabricante desse sistema fornece meios de desenvolver programas sob assistência de outro computador.

Essa ajuda existe na forma de um sistema de desenvolvimento — o Intellec 8 — com software para montagem de texto (*assembler*), monitoração e edição. Além disso, existe também um compilador *PL/M*, escrito em Fortran IV, para ser rodado em qualquer computador de porte médio ou grande. Nesta última forma, os programas são compilados em um computador de porte maior, que gera o código objeto para o microprocessador.

A.5. EXEMPLOS E PROJETOS COM MICROPROCESSADORES

Essa é a parte mais importante deste apêndice. Os autores acreditam que só se aprende a projetar analisando projetos e, mais tarde, executando-os.

Este item inicialmente apresenta um exemplo de aplicação de microprocessadores. A seguir, são propostos seis projetos. A descrição dos projetos procura orientar o leitor para a solução. Mesmo que não se tenha a intenção de desenvolvê-los, é importante que se leia a sua descrição. Esse material mostrará, em linhas gerais, como se organiza uma solução para cada aplicação.

a. Exemplo — controle de elevadores

Uma das aplicações que normalmente ocorre quando se pensa em microprocessadores é o controle de elevadores. Sobre as soluções eletromecânicas, ele tem as vantagens e desvantagens da eletrônica: maior confiabilidade, por um lado, e, por outro, maior sensibilidade a ruidos (interferências).

O microprocessador, no controle de elevadores, tem a vantagem de permitir a utilização de algoritmos mais elaborados. Pode-se, por exemplo, ter um sistema de controle adaptativo. Quando o tráfego é pequeno, o elevador sempre sobe até o andar mais alto em que exista uma requisição e desce atendendo as demais chamadas. Na hora de maior movimento, isso pode fazer com que, em andares mais baixos, haja espera muito longa. Nessas horas, o elevador atinge sua lotação máxima nos andares superiores, e passa direto sem parar nos andares inferiores. Nessa situação, pode-se ter um algoritmo mais elaborado que, na hora de movimento, comece não pelo andar mais alto, mas sim pelo primeiro andar que fora "pulado" na descida anterior. Neste exemplo, trataremos do controle de um único elevador. O microprocessador se torna uma solução melhor quando centralizamos nele o controle de vários elevadores. Ou, então, quando temos um microprocessador para cada elevador, interligados entre si, otimizando o serviço do conjunto. Não é o objetivo deste texto analisar os algoritmos de controle do elevador. Isso será feito por uma rotina que não será detalhada. O que se pretende aqui é mostrar como se organiza o controle, sob os pontos de vista de hardware e software.

A organização do controlador, vista na Fig. A.25, é como a da maioria dos sistemas que utilizam microprocessadores: *UCP*; memória "ler-escrever", *RAM* (onde ficam os dados); memória "ler somente", *ROM* (com os programas); e interfaces com sensores, chaves, atuadores,

res, lâmpadas, motores, etc. É na estrutura dos programas e na forma como as interfaces funcionam que se vêem as qualidades e defeitos de um sistema. Nessa organização, as interfaces estão definidas da seguinte forma:

- interface 1 — interface de entrada de dados vindos das chaves e dos controles que se encontram dentro do elevador (carro);
- interface 2 — interface de entrada e saída de dados das chaves, lâmpadas e sensores das portas, situados nos andares;
- interface 3 — interface de saída de dados para acionar os motores que comandam as portas;
- interface 4 — interface de entrada e saída para acionamento e monitoração dos motores que movimentam o carro;
- interface 5 (opcional) — interface de saída ligada às lâmpadas indicativas da posição do elevador, situadas no carro e nos andares;
- interface 6 (opcional) — interface de entrada e saída para comunicação com um *console* da portaria (quadro geral de controle dos elevadores).

Organização do software

A tarefa do *software*, como na maioria das aplicações em "tempo real", é analisar as entradas em uma freqüência suficientemente alta, manipular esses dados e gerar os sinais de saída. Do ponto de vista de alocação de memória, o *software* pode ser representado da forma que se vê na Fig. A.26.

Os programas de *aquisição de dados* são aqueles que exercitam as interfaces de entrada de forma a ler as informações de entrada para o sistema (técnica de *polling*) (Fig. A.27). Esse programa deve "rodar" periodicamente, sob comando de um relógio externo. O relógio, de tempos em tempos, interrompe o processador, e requer que o programa de aquisição de dados seja rodado. Esse programa carrega a base de dados de entrada, que é usada pelo programa principal para gerar dados de saída.

O programa principal analisa os dados de entrada e molda seu comportamento a eles. Este utiliza uma base de dados interna, com tabelas e parâmetros, onde ficam armazenados os pedidos de serviço e o roteiro de viagem até então estabelecido. O programa principal gera dados de saída, que são armazenados em uma região da memória.

O programa de saída realiza o comando das interfaces de saída em função do conteúdo da base de dados de saída. Esse programa pode rodar periodicamente, sob controle de um relógio externo, ou sob requisição do programa principal. Este último sinaliza, na base de dados de saída, que existe tarefa a ser executada nas interfaces de saída.

A organização dos programas, sob o ponto de vista de prioridades, é como se vê na Tab. A.4. Observe nessa tabela que apareceram novos programas, indiretamente associados ao problema.

No nível de mais alta prioridade, de atendimento imediato garantido, estão os programas de emergência. A função deles é atender pedidos de serviços prioritários emitidos por: botão de emergência no carro (elevador deve parar e abrir a porta), falta de energia elétrica (se o processador for alimentado por bateria, este deve sinalizar na portaria o local onde se encontra o carro), sinais de alarme (quebra de cabo, porta aberta, etc.), que obrigam o carro a parar imediatamente.

Quando o processador é ligado para início de trabalho, o sinal de *LAL* (limpa ao ligar) aciona a entrada *RESET* do processador. Isso obriga o processamento a iniciar na posição zero da memória. A rotina de *preparação* (cujo início está naquele endereço) informa, ao circuito externo de prioridade, que está se iniciando uma tarefa de prioridade 1. A seguir, limpa o conteúdo da memória e leva o carro para um ponto inicial (andar térreo, por exemplo). Após isso, limpa as interrupções de nível mais baixo para que o sistema comece a funcionar.

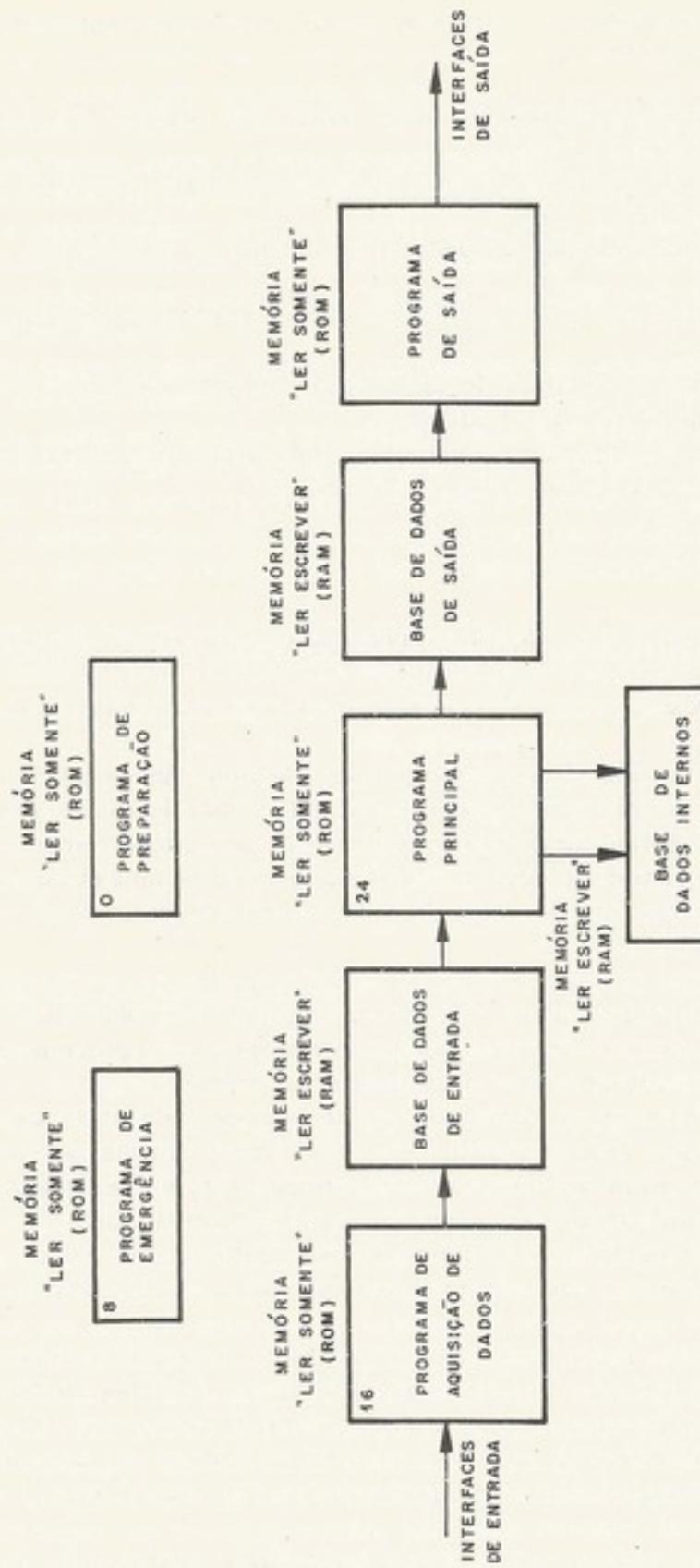


Figura A.26. Organização dos programas na memória do controlador

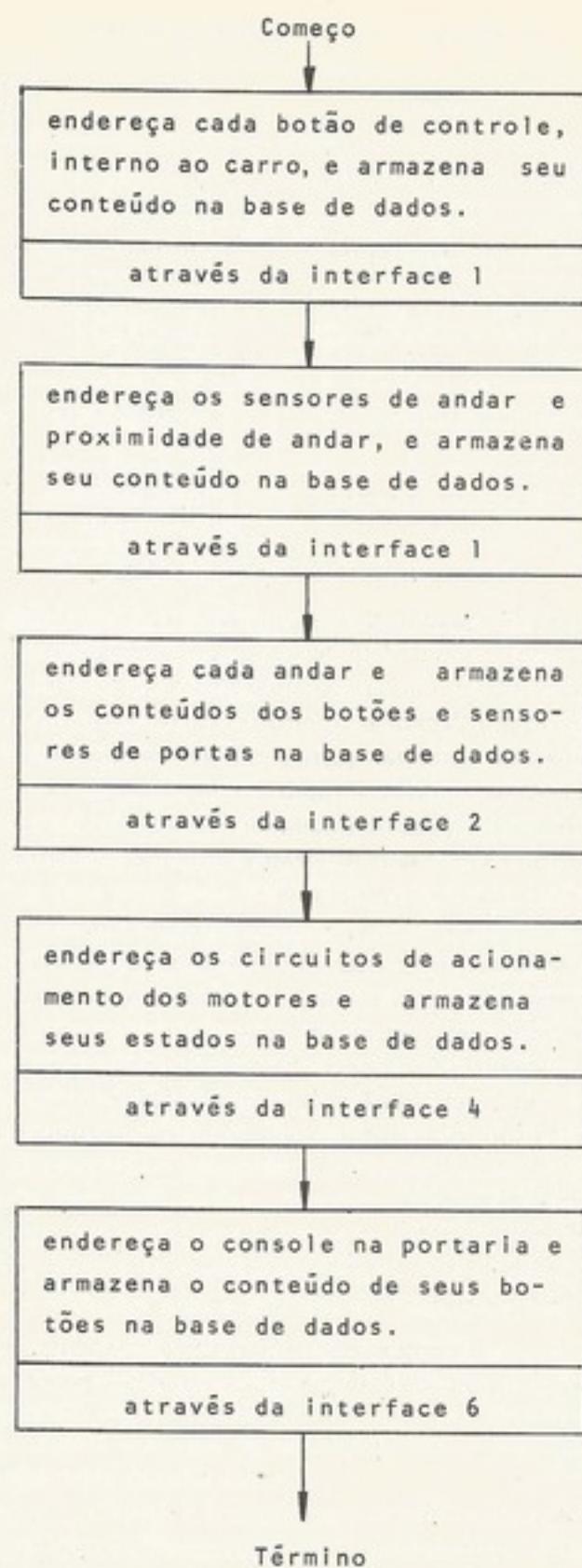


Figura A.27. O programa de aquisição de dados

Tabela A.4. Organização dos programas

Nível de prioridade	Programa	Forma de início
0 (maior prioridade)	Programa de emergência	Interrupção por: botão de emergência, falta de força, detetores de alarme, etc.
1	Programa de preparação	Quando se liga o controlador
2	Programa de aquisição de dados e programa de saída	Interrupção por meio de relógio externo (tempo t_1)
3	Programa principal	Interrupção por meio de relógio externo (tempo t_2)
Processamento normal	Programas de autoteste	Fica rodando enquanto não houver interrupção

Com um período t_1 , o processador roda os programas de aquisição de dados e de saída. A cada tempo t_2 , o programa principal inicia o funcionamento. A freqüência de funcionamento deste último não precisa ser tão rápida quanto as dos primeiros, já que sua função é apenas executar o algoritmo para decidir o roteiro do carro. Os primeiros têm de ser suficientemente rápidos para conseguir amostrar com a rapidez necessária as chaves e controlar a rotação dos motores. Por isso, o valor de t_2 pode ser algumas vezes maior que o valor de t_1 .

O fato de $t_2 = nt_1$ obrigará a base de dados de entrada a ter de suportar a chegada de n varreduras nas entradas a cada vez que o programa principal descarregar esses dados. O uso de filas nessa base de dados resolve o problema.

No tempo livre, isto é, quando todas as tarefas requisitadas por interrupções já terminaram, o processador pode rodar programas de autoteste (*self-testing*). Esses programas são úteis para descobrir, a tempo, possíveis falhas no sistema. A descoberta dessas falhas, antes de elas causarem erro, permite uma operação mais segura do controlador.

Observe-se que, nessa organização, interrupções provocadas por alarmes e sinal de emergência forçam endereço 8 na instrução *RST*. A interrupção dos relógios t_1 e t_2 forçam endereços 16 e 24, respectivamente. Portanto as rotinas começam nas seguintes posições:

- posição 0 — rotina de preparação;
- posição 8 — rotina de emergência;
- posição 16 — rotina de aquisição de dados seguida pela rotina de saída;
- posição 24 — programa principal.

Convém ressaltar que o endereço de início não define a prioridade da interrupção. Essa tarefa, como já foi dito, deve ser realizada por circuitos externos ao processador.

Aspectos do hardware

O hardware da unidade central de processamento é o padrão, ou seja, o microprocessador ligado a memórias da forma recomendada pelo fabricante⁽⁶⁾. Neste exemplo, deve-se voltar a atenção para os circuitos de interface, do lado dos sensores e atuadores.

Um dos problemas sérios enfrentados no controle eletrônico de elevadores é a alta dose de ruídos existentes nos cabos que interligam o controlador ao carro e andares. Cuidados especiais devem ser tomados para se atingir alta imunidade a ruidos. A escolha dos cabos blindados é importante. A forma de transmitir os sinais (sinais diferenciais em pares de fios, por exemplo) também deve ser considerada em detalhe, juntamente com os circuitos do controlador, os quais irão receber esses sinais. Devido a isso, deve-se economizar ao máximo o número de cabos

Figura A.28. Ex-
ecução das di-
versas tarefas

interligando os
seletores locais

Tomemos

um conjunto de

de andar e númer

de se ligarem m-

fios de endereço

(fechada ou abri-

variar o conten-

A.28). Note-se

recomendação e se

amostragem da

loga. A única d-

Suponhamos q-

gerados pelo m-

-cima (↑) e cua-

endereço, que p-

Este, quando a

porta), o conte

O comando

endereço o end

sobre flip-flop

deste exemplo.

Sumário

Neste ex-
cessor. Na
dos program-
definam funções
economizar co-

Recomenda-

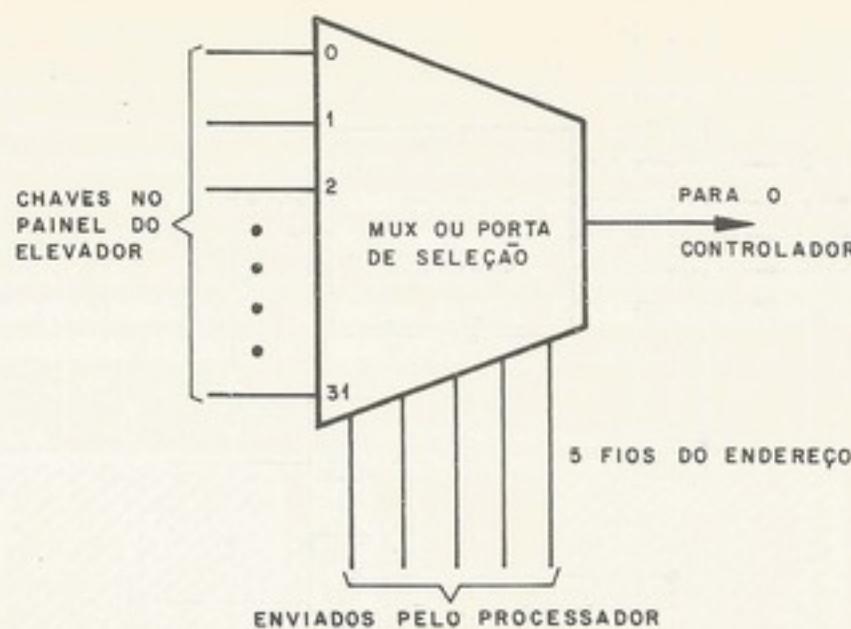


Figura A.28. Forma de se economizarem cabos entre o controlador e o elevador, através da multiplexação das chaves dentro do painel no carro

interligando o processador às chaves e sensores. Uma forma de se conseguir isso é através de seleção local (*polling*) desses elementos, e usando-se um único fio para cada conjunto de variáveis.

Tomemos o caso em que os cabos vão para o carro, saindo da interface 1. No carro, há um conjunto de botões que devem ser amostrados, juntamente com os sensores de proximidade de andar e número de andar. Suponha que existam trinta botões no painel do carro. Ao invés de se ligarem trinta cabos (um para cada botão) ao controlador, pode-se ter um cabo de cinco fios de endereço, que o controlador comanda, e um cabo que transmite a posição da chave (fechada ou aberta). O processador, sob comando do programa de aquisição de dados, irá variar o conteúdo dos cinco fios de endereço, de forma a ler a posição das trinta chaves (Fig. A.28). Note-se que, para um controlador que utilize um microprocessador, a técnica de endereçamento e seleção (multiplexação) é natural, barateando, inclusive, o custo da interface. A amostragem dos botões e dos sensores de portas nos andares pode ser resolvida de forma análoga. A única diferença é que o multiplexador é distribuído da forma indicada na Fig. A.29. Suponhamos que haja 25 andares — novamente, 5 bits é suficiente para endereçá-los (endereços gerados pela interface 2). Em cada andar, devem-se amostrar duas chaves [chamando-paracima (\uparrow) e chamando-parabaixo (\downarrow)], e o interruptor que acusa porta aberta. Os cinco fios de endereço, que percorrem todo o prédio, alimentam, em cada andar, um detetor de endereço. Este, quando o endereço coincide com o número do andar, coloca, nas vias de dados (\uparrow , \downarrow e porta), o conteúdo das respectivas variáveis naquele andar.

O comando das lâmpadas nos andares deve também ser endereçável. Isto é, a interface 2 endereça o andar e comanda sinais de acender e apagar as lâmpadas. Esses comandos atuam sobre *flip-flops* locais. Outros detalhes de *hardware*, muito específicos e fora dos objetivos deste exemplo, não serão aqui considerados.

Sumário

Neste exemplo analisou-se um controlador de elevadores construído com um microprocessador. Na solução apresentada, percebe-se a importância de uma organização planejada dos programas, em função dos níveis de interrupção. O software “tempo real” requer que se definam tarefas bem específicas, que são executadas periodicamente. Consegue-se também economizar circuitos através de uma disposição planejada dos elementos do controlador. Recomenda-se que o leitor elabore em mais detalhes a solução apresentada.

b. Projeto

Nesta seção desenvolvemos de exercitá-lo a estrutura de um edifício — mais esses projetos são importantes; o leitor faça os

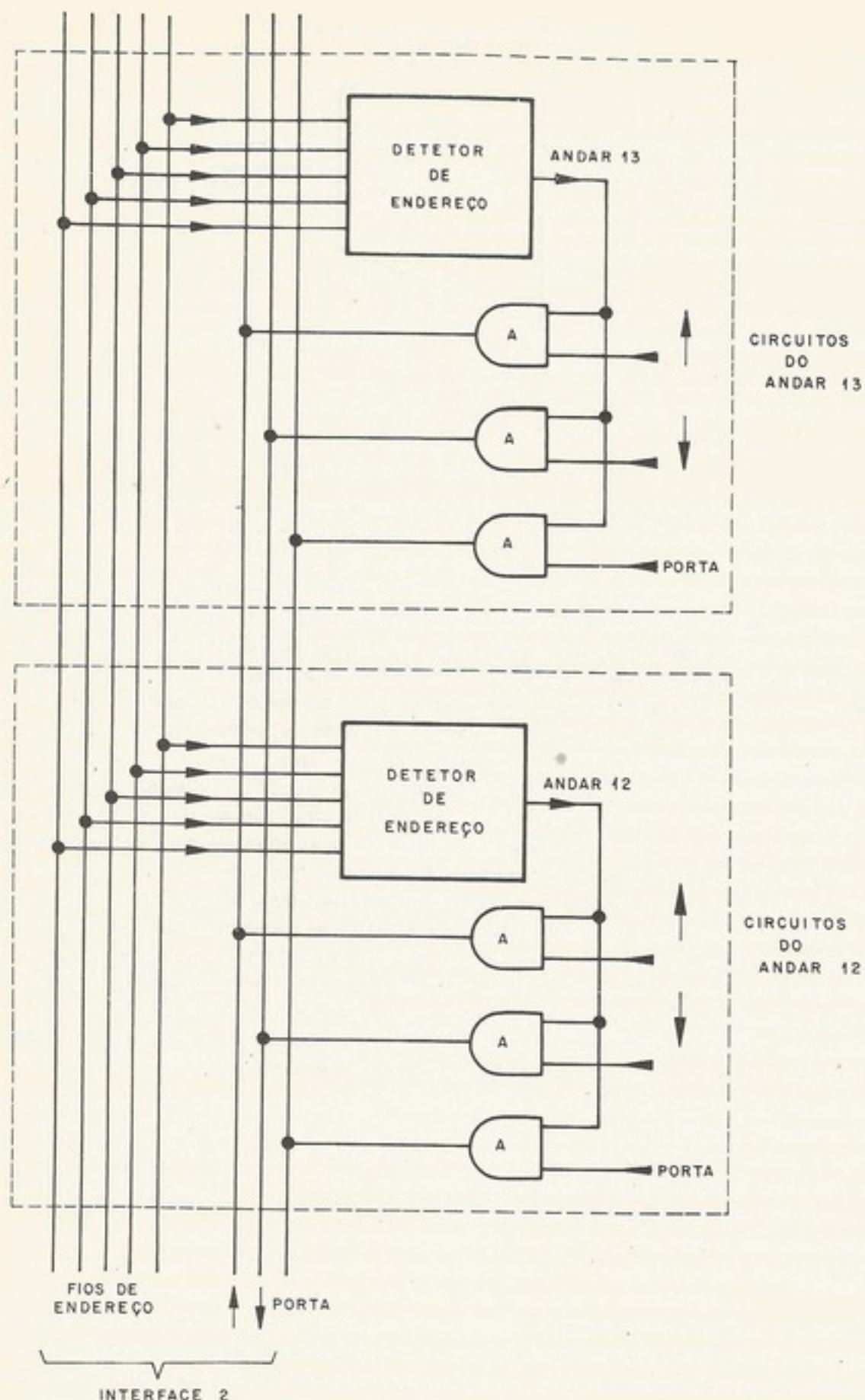


Figura A.29. Circuitos de amostragem dos sinais nos andares

b. Projeto 1 — controle de faróis de trânsito⁽⁹⁾

Nesta seção, bem como nas seguintes, existem várias sugestões de projetos para serem desenvolvidas, utilizando microprocessadores. O objetivo é oferecer ao leitor a possibilidade de exercitarse no planejamento do *hardware* e do *software*. Muitos desses casos exigem uma estrutura de *software* em “tempo real”, onde inúmeras tarefas devem ser executadas periodicamente — recomenda-se especial atenção a esses detalhes. Como projeto final de cursos, esses projetos podem até ser detalhados. Em todos eles, procuramos definir os pontos mais importantes; contudo, ainda assim, inúmeros detalhes ficaram em aberto. Espera-se que o leitor faça as necessárias hipóteses e defina todos esses pontos.

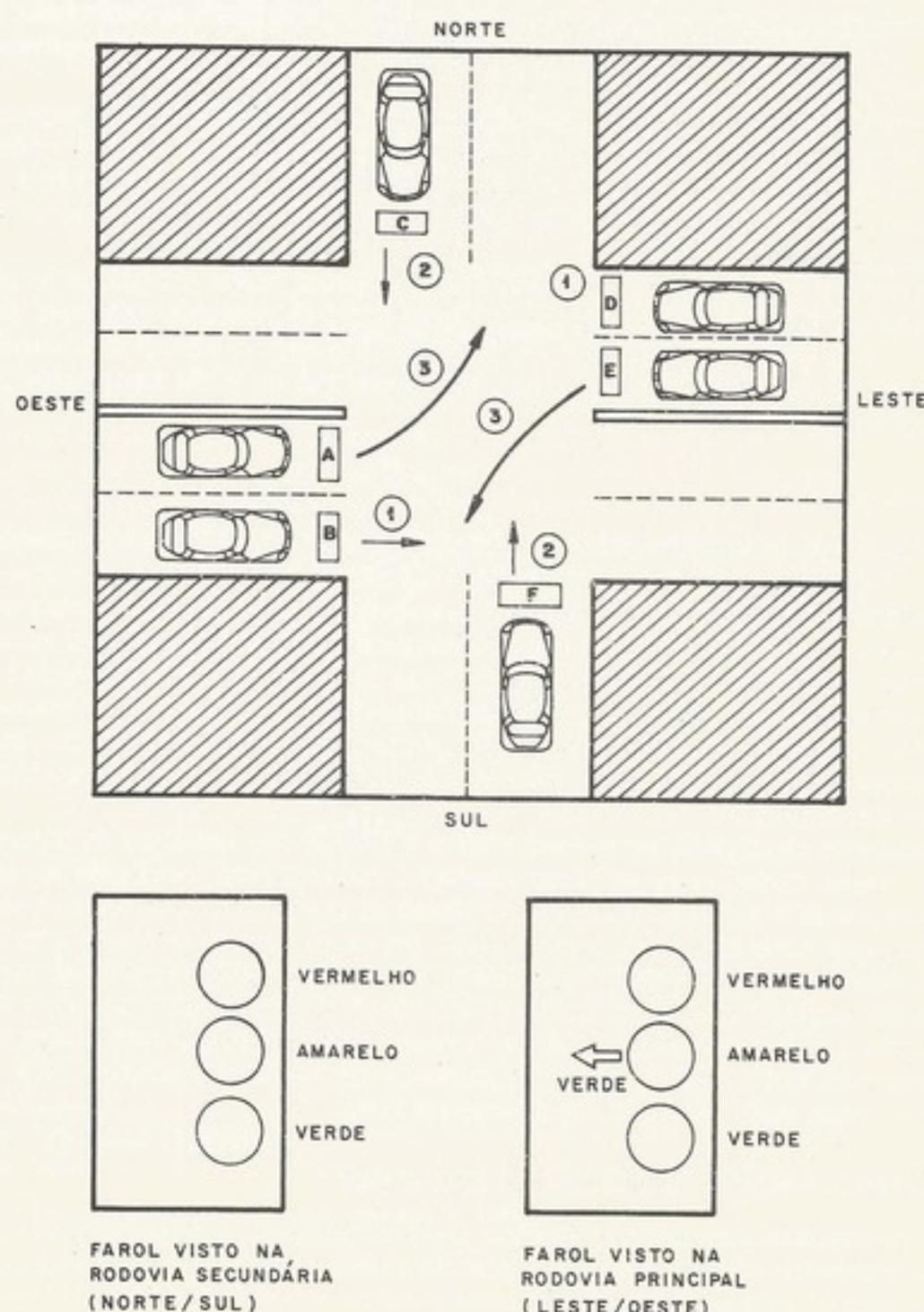


Figura A.30. Interseção com seis rotas distintas organizadas em três configurações

Primeira parte

Este projeto consiste num controlador de um farol de trânsito adaptativo ao volume de tráfego. O farol deve controlar uma interseção com seis rotas distintas, organizadas em três configurações (Fig. A.30): (1) leste↔oeste, (2) norte↔sul e (3) leste→sul/oeste→norte. O objetivo do controlador é permitir a passagem de um número máximo de carros em qualquer período. Por outro lado, o tempo de espera (rota com pouco tráfego) não pode ser maior que um limite especificado.

Para que se possa ajustar a duração de cada ciclo de verde ao volume de tráfego, instalam-se, nos pontos de chegada da interseção (A, B, C, D, E e F), placas, no solo, com uma chave acoplada: toda vez que as rodas de um veículo passam por ela, o contato se fecha, informando o controlador. Cada veículo, portanto, irá comandar dois sinais ao controlador (pneus dianteiros e traseiros), e veículos com mais de dois eixos serão considerados equivalentes a mais que um carro.

Esse controlador tem, portanto, possibilidade de contar os veículos que passam pelos diferentes pontos da interseção. Essa informação é colhida durante um ciclo de verde, e ela irá influir na duração do ciclo de verde seguinte. Para avaliar a densidade de tráfego, o controlador deve contar o número de veículos que passam em um determinado ciclo de verde e dividir pela sua duração.

Em resumo, o controlador recebe, como entradas, pulsos vindos dos pontos de chegada na interseção (A, B, C, D, E e F) (Fig. A.31). Como saída, ele comanda três ciclos distintos do farol (1, 2 e 3), cuja duração é função da densidade de tráfego. O vínculo a ser respeitado é que a duração do vermelho, em qualquer rota, seja menor que um limite especificado (mesmo com tráfego zero).

Segunda parte

Observe-se que, na especificação desse controlador, tem-se por objetivo a otimização local do tráfego. Admita que a via leste-oeste da Fig. A.30 seja uma rota importante e deve ter seu tráfego otimizado globalmente. Em cada uma de suas interseções importantes existe um controlador, como descrito. A solução, então, é interligar esses controladores locais entre si, de forma a criar, na via leste-oeste, "ondas verdes" nos dois sentidos. Essa interligação é feita com sinais de sincronismo (Fig. A.31), que cada controlador gera para os dois vizinhos. Uma "onda verde" é entendida como uma sucessão de ciclos verdes em um certo sentido, de forma que um veículo, entrando na via, e seguindo a uma dada velocidade, encontrará todos os faróis abertos em sua trajetória.

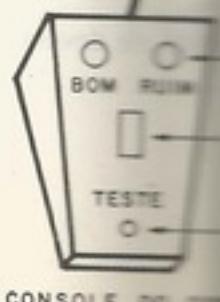
O importante na segunda parte desse projeto é planejar um algoritmo de controle que gere essas "ondas verdes", sem deteriorar (muito) o controle local conseguido na primeira parte. Defina também a forma física (tipos de sinais) do sincronismo entre os controladores.

É importante notar que, nesta segunda parte, haverá um sistema de controle de trânsito na via leste-oeste com "inteligência" distribuída, ou controle descentralizado. É necessário que a solução proposta se auto-sincronize (o que acontecerá se momentaneamente faltar a força em todo o sistema?) e, mais ainda, que nenhum controlador, por mau funcionamento, consiga fazer com que os demais deixem de funcionar corretamente. Lembre-se, mais importante que a "onda verde" é o bom funcionamento local.

c. Projeto 2 — testador automático de circuito digital⁽⁹⁾

Para as empresas que utilizam circuitos integrados digitais, é importante que possam testar os componentes antes de soldá-los na placa de circuito impresso. O teste isolado de todos os componentes é, portanto, um problema que tem de ser resolvido. O objetivo deste projeto é desenvolver um testador automático de circuitos integrados digitais, de fácil operação e alta produtividade (Fig. A.32).

CHAVES NO
LEITO DA RUA



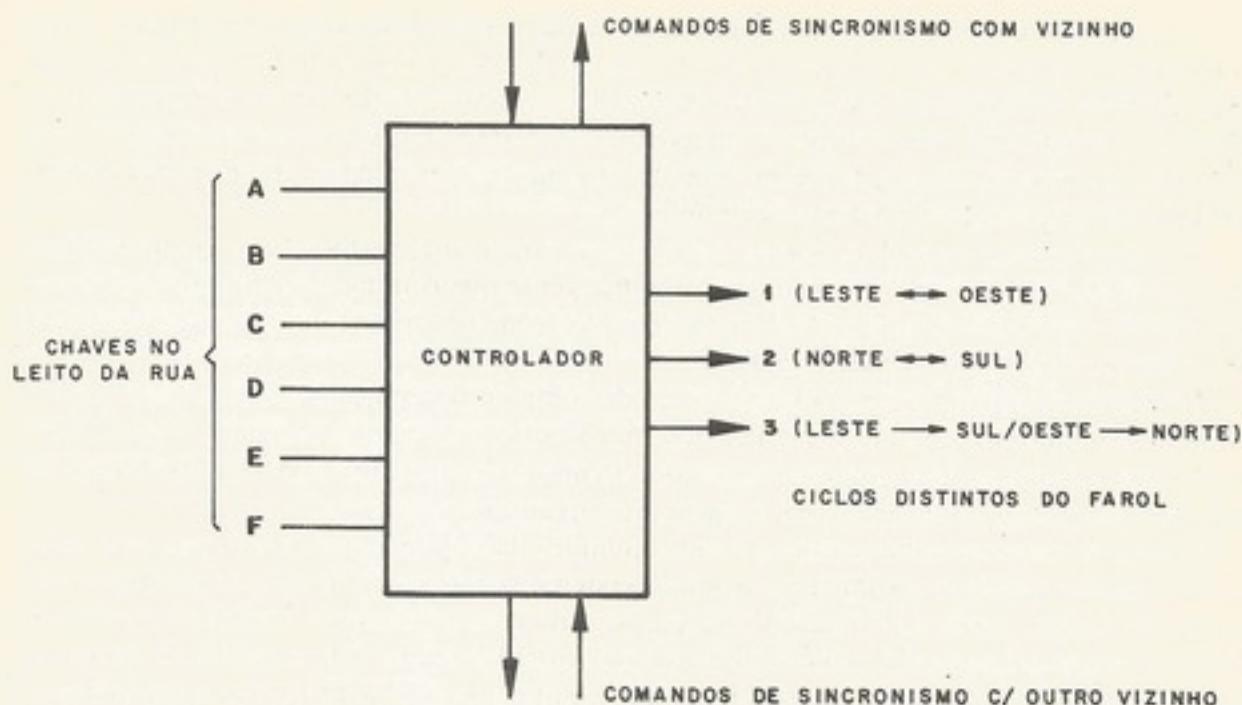


Figura A.31. O controlador de farol

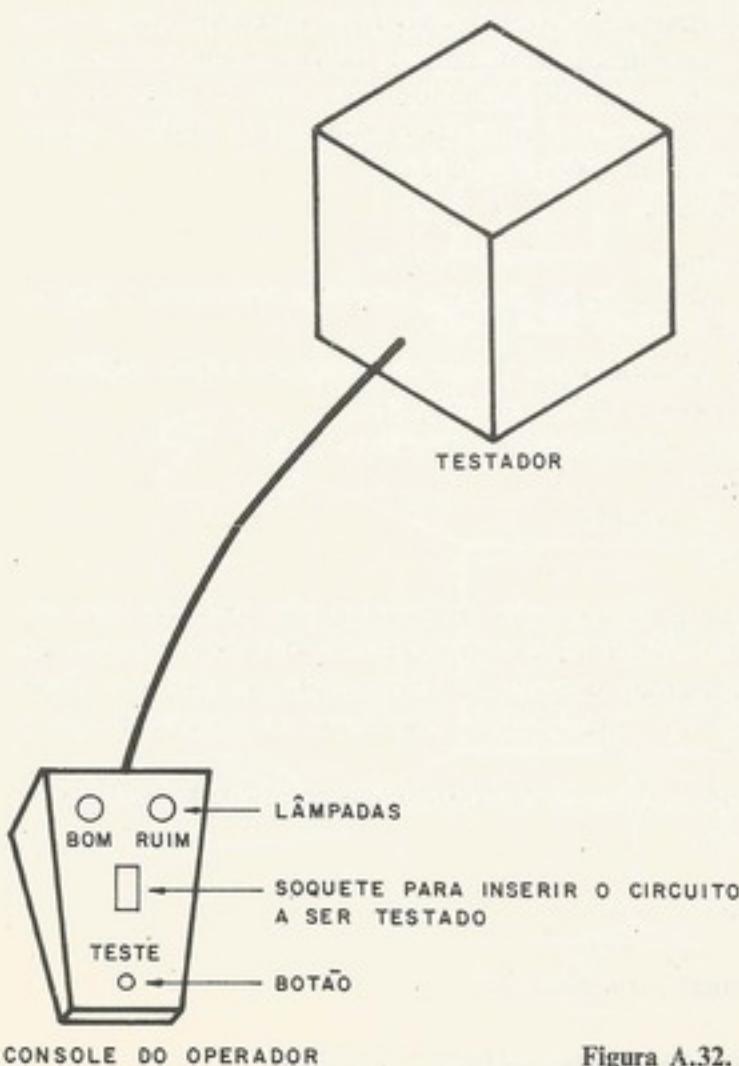


Figura A.32. O testador automático de circuito digital

A utilização do sistema deve ser simples. Após os preparativos iniciais, o operador insere no testador o circuito a ser testado e pressiona um botão. No painel do testador haverá a indicação de falha ou operação correta. Os preparativos iniciais deverão ser os necessários para "informar" a máquina sobre o tipo de integrado a ser testado e qual o teste a realizar. Como o testador deve ser operado para testar lotes do mesmo tipo de circuito, as restrições de simplicidade na preparação inicial são mais folgadas.

Por simplicidade, considere apenas testes funcionais, isto é, verificar se o circuito realiza a função lógica especificada no catálogo. Suponha ainda que o testador será feito para testar circuitos de uma única família, TTL, por exemplo, com um número máximo de terminais (16 pinos, na configuração *dual-in-line*).

O teste será, portanto, realizado por meio de comparação com um modelo. Esse modelo pode ser: (1) um outro circuito que é inserido numa posição no painel, chamada "referência", ou (2) uma sub-rotina que simula a lógica do circuito a ser testado. Essa escolha deverá ser feita pelo projetista, tendo em vista a particular organização adotada.

O teste de circuitos combinatórios é muito simples: basta alimentar as entradas com *todas* as combinações possíveis e comparar as saídas com as da referência. Esse teste exaustivo não leva muito tempo. Dada a limitação de 16 pinos, o número máximo de entradas é, portanto, 13, o que requer 2^{13} testes. Se cada combinação das entradas requer 100 µs (média de 25 instruções do 8080), o teste completo levará 0,8 s, o que é um tempo bom para um operador humano.

O teste dos circuitos seqüenciais já é mais problemático, pois não basta alimentá-los com todas as combinações das entradas. A saída dependerá também do estado interno. O testador deverá, portanto, ter uma outra política de teste, onde controlará as entradas de *clock* e *reset* de forma especial. Uma parte importante nesse projeto é planejar o algoritmo de teste dos circuitos seqüenciais. A solução, contudo, deve ser prática e viável para um equipamento nesse porte. Não se recomenda a utilização de algoritmos, de deteção de falhas, desenvolvidos para testar placas de circuitos impressos (sensibilização de caminhos, algoritmo D, algoritmo de Roth, etc.). Tente uma solução do tipo "variam-se as entradas e testam-se as saídas", exaustivamente se necessário, mesmo que não detete 100% das falhas.

Este projeto requer, portanto, o seguinte:

- a) planejar a forma de teste (algoritmo);
- b) definir como instruir o testador sobre a forma de teste, caso esta varie com o tipo de circuito sob teste;
- c) definir a maneira de informar o testador sobre o tipo do integrado sendo testado, e quais os pinos que são entradas, saídas, *clock* e *reset*, e alimentação;
- d) estruturar o *hardware* do testador, com memória, interfaces, etc.;
- e) detalhar a organização do *software*;
- f) (opcional) detalhar os circuitos do sistema;
- g) (opcional) codificar os programas.

Algumas soluções viáveis para a entrada de informações no testador são: teclados tipo calculadora, cartões perfurados ou chaves no painel. Uma solução mais sofisticada utilizaria um terminal (vídeo ou teletipo) com uma linguagem de controle interpretada por *software*. Neste último caso, o projetista deverá definir essa linguagem, e considerar os programas que a interpretam.

À medida que o projetista for criando soluções para os problemas apresentados, ele deverá responder a algumas perguntas, tais como:

- como alimentar o circuito sob teste?
- como acionar as entradas?
- os preparativos iniciais não estão muito complicados?
- a velocidade está boa?
- qual o requisito para o operador que irá realizar a preparação inicial?

e. Projeto 3 — um PABX controlado por microprocessador⁽¹¹⁾

Um PABX (*private automatic branch exchange*) é uma central telefônica privada, existente em empresas, que permite a ligação interna de dois telefones sem “incomodar” a central pública. Os telefones recebem o nome de *ramais*, cada um com sua numeração própria. Discando, em um ramal, o número de outro, o PABX realiza, automaticamente, a conexão entre os dois.

Em um PABX, chegam também linhas telefônicas públicas (troncos), que permitem aos ramais a realização de chamadas externas. Discando o dígito zero em um ramal, consegue-se acesso automático a essas linhas externas. Uma chamada telefônica, originada em um aparelho externo e dirigida a um ramal, “caia” no *console* do operador do PABX, que a atende, verifica com quem o interlocutor quer falar, e realiza a transferência para o ramal correspondente. (Fig. A.33).

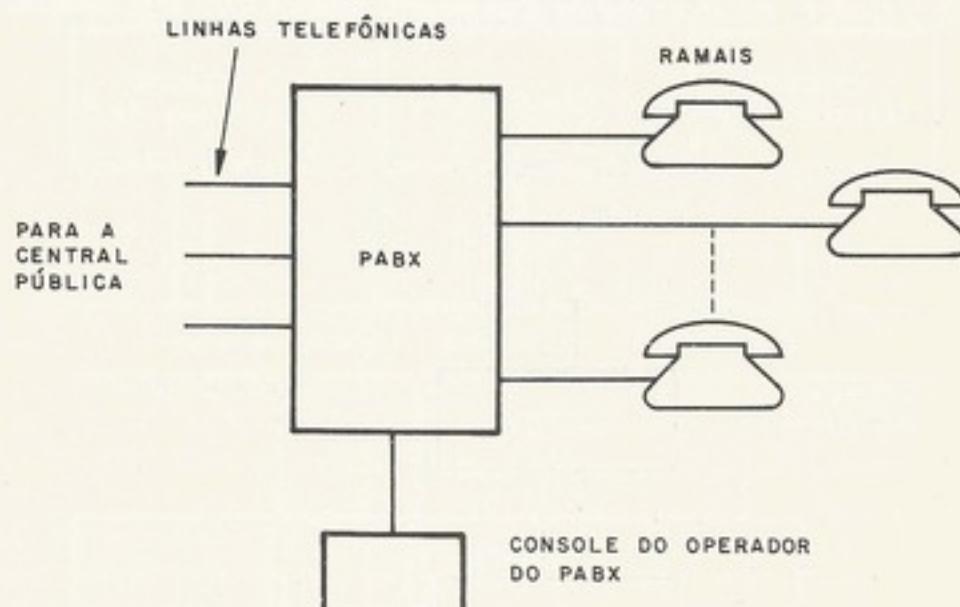


Figura A.33. Estrutura do PABX

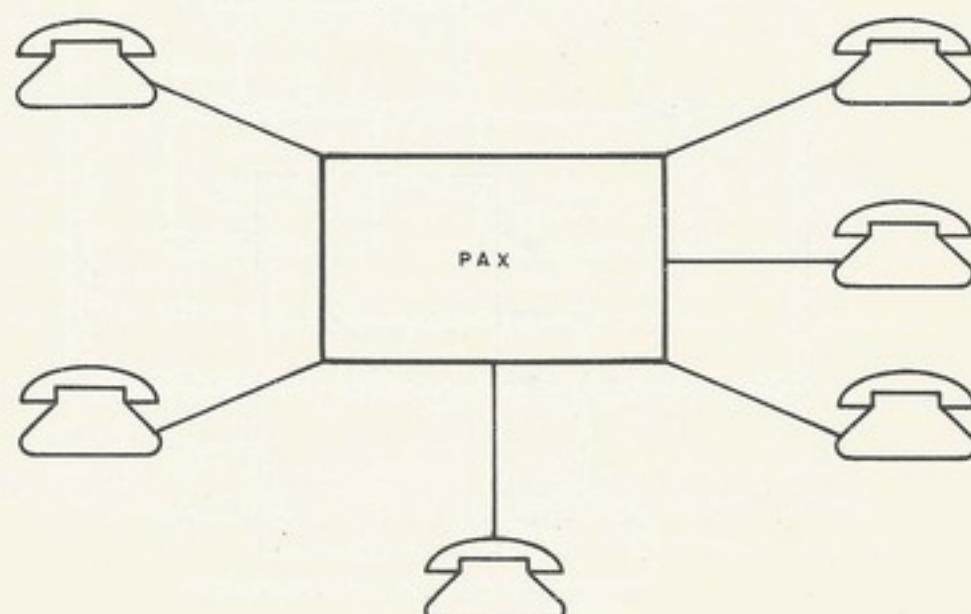


Figura A.34. Estrutura do PAX

Um caso particular do PABX é o PAX (*private automatic exchange*) (Fig. A.34), que apenas interliga ramais. Não tem *console* de operador, nem acesso a troncos externos. Este projeto, por simplicidade, considerará, de início, esse caso.

Primeira parte

O que se propõe aqui é projetar um PAX com 64 ramais, controlado por um microprocessador. Essa técnica recebe o nome de *CPA*, isto é, controle por programa armazenado. Os ramais, nessa central, recebem os números de 11 a 74. Cada ramal deverá ter sua interface com o processador. Nessa interface, além dos circuitos que monitoram a linha, existe um módulo chamado de "híbrida", o qual transforma o par de fios bidirecionais do telefone em dois pares unidirecionais (Fig. A.35).

As vias de áudio (fios unidirecionais) são as entradas e saídas da matriz de interconexão (Fig. A.35). Esse módulo, sob comando do processador, interliga uma dada entrada, em qualquer saída.

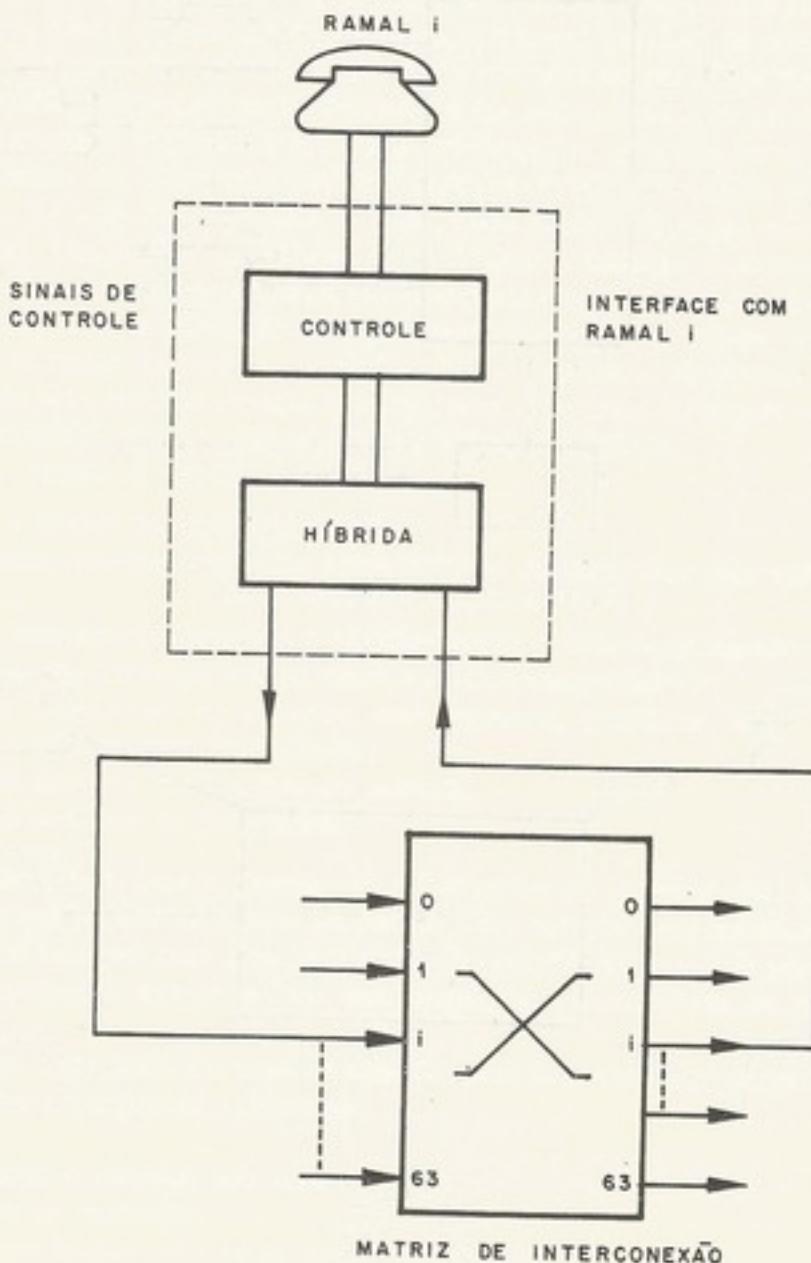


Figura A.35. Conexão de ramais

(Fig. A.34), que apenas

Este projeto,

por um micropro-

gramma armazenado.

ter sua interface

línea, existe um mó-

vel de telefone em dois

de interconexão

entrada, em qual-

apêndice os microcomputadores

Existem inúmeras formas de se construir uma matriz de interconexão⁽¹⁰⁾. A primeira é a matriz *espacial*, onde um conjunto de multiplexadores analógicos (relés ou transistores *MOS*), convenientemente endereçados, realizam a conexão. Outra técnica é a *multiplexação por divisão temporal*. Aqui, existe um único multiplexador na entrada ligado a um demultiplexador na saída. O tempo é dividido em segmentos, onde cada saída amostra sua entrada. Um relógio (pode ser o microprocessador) comanda a amostragem, de forma que sua freqüência não seja menor que 8 kHz para não distorcer o sinal de áudio.

A última técnica que se pode considerar é a matriz de *comutação* (ou interconexão) *temporal*. Os sinais de áudio nas interfaces são codificados em binário. Ciclicamente, o processador atende a cada interface, retirando o código binário correspondente ao sinal de áudio gerado no ramal, e envia a ela o código binário de seu interlocutor. A interconexão é, portanto, realizada por um programa que armazena os códigos lidos e mais tarde os envia ao destinatário. A freqüência de varredura também deve ser maior que 8 kHz.

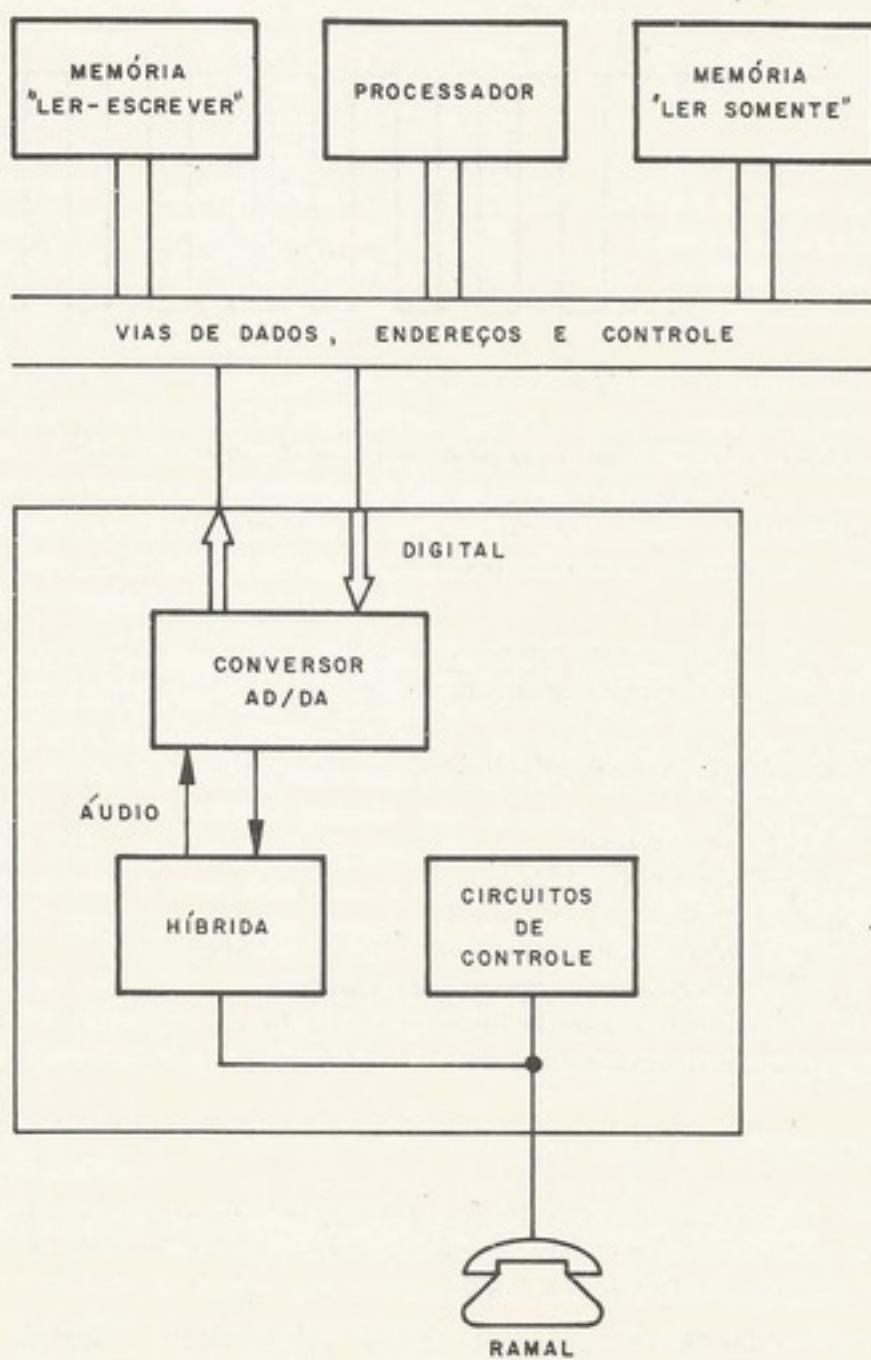


Figura A.36. A estrutura do PAX

Apesar de ser a mais cara, sugere-se aqui, que o leitor utilize a última técnica, que, do ponto de vista do processador, é a mais natural (Fig. A.36). Quando o telefone está no "gancho", sua linha está aberta; fora do "gancho", está fechada através dos circuitos locais do aparelho telefônico. Quando se disca um número, o aparelho desconecta os circuitos locais e curto-circuta a linha por 50 ms (simplificação) para cada pulso. Quando se disca o dígito 8 (por exemplo), a volta do rotor, coloca na linha oito pulsos de 50 ms cada, distanciados 100 ms entre si. Isto é, o trem de oito pulsos dura 800 ms no total (Fig. A.37).

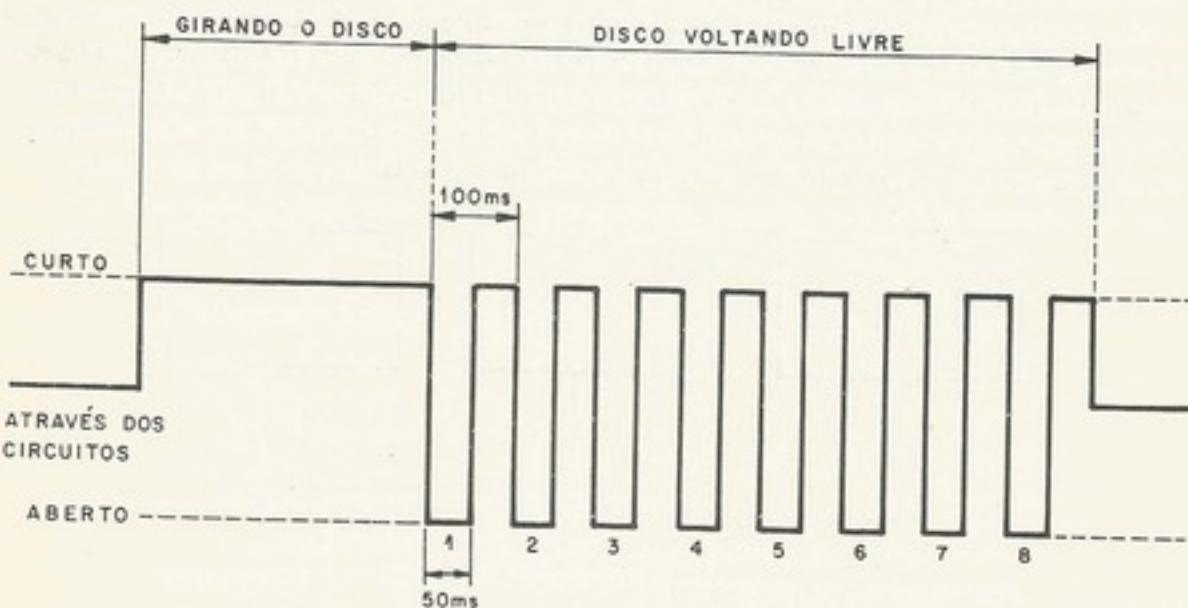


Figura A.37. Os sinais na linha do ramal ao discar-se o número 8

O processador deverá varrer os ramais, em uma freqüência suficientemente rápida, para conseguir detetar os pulsos e contá-los por programa. Esse projeto consiste, portanto, no seguinte:

- estruturar, em diagramas de blocos, o *hardware*;
- projetar a interface, especificando quais os sinais lidos pelo processador e quais os gerados por ele;
- organizar os programas, considerando as tarefas básicas
preparação inicial,
aquisição de dados de discagem,
interpretação dos números discados,
controle das interconexões,
aquisição e geração de informações de áudio (na forma digital),
geração dos sinais de linha, de ocupado e campainha;
- (opcional) detalhar o circuito das interfaces e da UCP;
- (opcional) detalhar os programas.

Segunda parte

Uma central telefônica controlada por um microprocessador pode ter inúmeros recursos que um sistema convencional não tem. Adaptar o *software* desenvolvido na primeira parte para incluir os serviços enumerados a seguir.

- Chamada de volta.* Se um ramal tenta chamar outro e recebe o sinal de ocupado, este, então, disca o número 0 ("em cima" do tom de ocupado) e instrui o processador para chamá-lo de volta quando o ramal desejado estiver livre.

apêndice

2. Confinada numa caixa, o ouro. O prata, três aparelhos

3. Interrupções onde existe uma ligação de seu n.º seguido de n.º parte, espera-se que será mais afetuosa

Terceira parte

Esta terceira capacidade de 64 canais detetam se está ou não atendível). Como o

As ligações exteriores

A interface

com os ramais, o ("atende") que as externas. Pressionar

tema e, a seguir, fica por conta do

Os serviços

apenas de redimensionamento

A sequência

e das interfaces de

descrito.

f. Projeto 4 —

Esta descrição

berdade nas espécies

para controle de

dada por dois motores

perpendicular (y),

movimento para baixo

Existem, acopladas

denadas x e y. O

POS x —

POS y —

z/"alto" —

da peça);

z/"baixo" —

(broca dentro da

De posse desses

VEL x —

VEL y —

VALV —

(para furar a peça).

2. *Conferência.* Discando o dígito 9, o processador fica instruído de que está sendo iniciada uma conferência. Em seguida ao 9, disca-se o número de dois ramais, em série, um após o outro. O processador então toca a campainha dos dois telefones chamados, e interliga os três aparelhos (chamador e chamados) em uma única conversa.

3. *Instrução de "mude".* Se um usuário que tem um certo ramal tiver de mudar de sala, onde existe outro ramal, ele pode instruir o processador a transferir, automaticamente, as ligações de seu ramal para o novo. Isso é feito da seguinte forma: em seu aparelho, ele disca 8, seguido do número do ramal para o qual deverão ser transferidas as chamadas. Nessa segunda parte, espera-se que o projetista detalhe o programa de controle das interconexões, já que este será mais afetado pela inclusão desses serviços especiais.

Terceira parte

Esta terceira parte consiste no projeto e detalhamento de um PABX completo, com capacidade de 64 ramais e 10 troncos. Deverão existir interfaces especiais para troncos, que apenas detetam se está chamando ou não (o tom de campainha é tensão alta (90 V), facilmente detetável). Como o tronco não disca número de ramal, sua interface, nesse aspecto, é mais simples. As ligações externas "caem" na mesa do operador.

A interface com a mesa do operador deve levar em conta que esta, além de se comunicar com os ramais, tem o controle sobre as chamadas externas que chegam. Existe uma tecla ("atende") que o operador pressiona para ir atendendo, na ordem de chegada, as chamadas externas. Pressionando "atende", o operador conversa com o interlocutor que chamou o sistema e, a seguir, pressiona a tecla de "transfere" e disca o número do ramal. A transferência fica por conta do processador.

Os serviços especiais podem funcionar da forma descrita anteriormente, cuidando-se apenas de redefinir os números para requisitá-los. Reserve o dígito zero para pedir linha externa.

A sequência de projeto pode seguir a da primeira parte, incluindo-se o projeto do *console* e das interfaces de tronco. A Referência 13 descreve um PABX real, com a estrutura do aqui descrito.

f. Projeto 4 — controle numérico

Esta descrição de projeto e as seguintes serão mais sucintas, de forma a dar ao leitor liberdade nas especificações do sistema. O que se propõe aqui é utilizar um microprocessador para controle de uma furadeira elétrica. A peça a ser perfurada é fixada a uma mesa comandada por dois motores: o primeiro movimenta-a em uma direção (*x*), e o segundo, na direção perpendicular (*y*). A broca, sempre girando, está acoplada a um cabeçote pneumático que a movimenta para baixo, sob controle de uma válvula.

Existem, acoplados à mesa, dois sensores, que informam sua posição, fornecendo as coordenadas *x* e *y*. O microcomputador deverá receber como entrada os seguintes sinais:

POS x — sinal analógico que indica a posição da mesa no eixo *x*;

POS y — sinal analógico que indica a posição da mesa no eixo *y*;

z/“alto” — sinal digital que indica estar o cabeçote em sua posição superior (broca longe da peça);

z/“baixo” — sinal digital que indica estar o cabeçote na posição inferior máxima (broca dentro da peça).

De posse desses sinais, o controlador deverá comandar as seguintes saídas:

VEL x — velocidade do motor que movimenta o carro no eixo *x* (positiva ou negativa);

VEL y — velocidade do motor que movimenta o carro no eixo *y* (positiva ou negativa);

VALV — sinal digital que aciona a válvula pneumática do cabeçote, fazendo-o baixar (para furar a peça).

O objetivo do controlador é permitir que se faça, automaticamente, uma seqüência de furos na peça, posicionados corretamente. O comando do controlador pode ser feito através de um teclado (acionado pelo operador) ou por um dispositivo que "lê" uma fita de papel. Esse comando deverá ser realizado através de uma linguagem de controle com as seguintes instruções:

"vá para $x = 999$ e $y = 999$ ",
"fure",

onde os campos 999 indicam as coordenadas do furo.

Durante o funcionamento, o operador posiciona a peça na máquina e passa a fornecer ao controlador uma seqüência de instruções, de forma a obter a peça perfurada corretamente.

Projete o controlador, dando ênfase aos seguintes aspectos:

- a) arquitetura;
- b) organização dos programas;
- c) algoritmo de controle de velocidade dos motores;
- d) detalhamento em diagramas de blocos dos programas;
- e) forma de alimentar o controlador com a seqüência de instruções;
- f) aspectos de carga e descarga da peça;
- g) fixação da peça na sua posição inicial;
- h) controles manuais da máquina.

g. Projeto 5 — controle de portas em hotéis

Projete um microcomputador que controle a fechadura das portas dos quartos de um hotel. Seu funcionamento deverá ser como segue.

Quando um hóspede se registra no hotel, o funcionário da portaria insere um cartão magnético em um equipamento de gravação (acoplado ao computador), e datilografa em um teclado os dados do hóspede. O computador realiza o registro, e grava no cartão magnético um código especial. Esse cartão magnético pode ser então utilizado pelo hóspede para abrir a porta de seu quarto. Ao lado de cada fechadura deve existir um leitor de cartão, que envia ao computador o código lido. Este confere se o código é o correto, e abre a porta. Quando o hóspede deixa o hotel, o funcionário da portaria, pelo mesmo teclado, desabilita o código de seu cartão.

Esquematize a solução para o problema, dando atenção aos seguintes detalhes:

- a) arquitetura;
- b) forma de interligar as portas ao computador (economizar fios);
- c) organização do *software*;
- d) forma de geração de códigos (o sistema tem de ser seguro);
- e) detalhamento dos programas;
- f) outras funções que o sistema pode realizar.

h. Projeto 6 — instrução programada

A instrução programada é uma técnica de ensino que consiste em apresentar ao estudante uma frase e uma pergunta, dirigindo sua resposta para a correta. Se a resposta for correta, então informa ao estudante seu sucesso, e passa para a pergunta seguinte. Quando a resposta for errada, a seqüência normal de perguntas será interrompida para uma série que irá corrigir a falha.

Projete uma máquina de instrução programada, onde as perguntas são armazenadas em uma fita magnética cassete. Sob comando do computador, um grupo de perguntas é transferido do cassete para a memória. O estudante se comunica com o processador através de um

BIBLIO

LEITUR

Uma introdução
Am R. P.
Vol. 7, n.
Am R. P.
Computa

Thomas R. H.
N.Y. 1975

terminal com vídeo. As perguntas aparecem escritas na tela, e suas respostas são feitas pelo teclado.

Um programa (o supervisor), que deve ser independente do assunto que está sendo ensinado, deve controlar as operações. O supervisor decide as perguntas a serem feitas em função das respostas dadas pelo estudante. Detalhe esse sistema, dando atenção a:

- forma em que as perguntas serão armazenadas na fita cassette (deve haver indicação da resposta correta e das perguntas seguintes, nos casos de resposta incorreta);
- arquitetura do hardware;
- esquematização do programa supervisor;
- esse sistema pode ser utilizado para aplicações como jogos Exemplifique.

BIBLIOGRAFIA

- Edson Fregni, "A Revolução dos Microprocessadores", *Revista Eletricidade Moderna*, n.º 32, pp. 14 a 23, Ed. Abril, São Paulo, setembro de 1975
- D. N. Kaye, "How to Pick a Microprocessor, or a Mini or anything in between", *Electronic Design*, pp. 26 a 30, 2 de agosto de 1975
- C. D. Weiss, "Analysis of Microprocessor Instruction Sets", *Microprocessors: New Direction for Designers*, Hayden Book, N.J., 1975
- K. Rothmüller, "Task Partitioning in Programmable Logic Systems", *IEEE Computer*, pp. 19 a 23, janeiro de 1976
- H. S. Stone, *Introduction to Computer Organization and Data Structures*, McGraw Hill, 1972
- Intel Corporation, *Intel 8080 Microcomputer Systems User's Manual*, julho de 1975
- Adam Osborn & Ass., *An Introduction to Microcomputers*, Adam Osborne & Ass., 1975
- Microcomputer Technique Inc., *New Logic Notebook*, Vol. 1, n.º 1, setembro de 1974
- C. Dennis Weiss, "Traffic-Light Controller: an Example". *Microprocessors: New Directions for Designers*, pp. 51 a 55, Hayden Book Company, N.J., 1975
- T. A. Laliotis e T. D. Brumett, "A Microprocessor — Controlled DIC Test System", *IEEE Computer*, Vol. 8, n.º 10, pp. 60 a 67, outubro de 1975
- M. Gohl e H. P. van Roosmalen, "Electronic Private Automatic Branch Exchange EBX 100", *Philips Telecommunication Review*, Vol. 33, n.º 3, pp. 105 a 112, setembro de 1975
- A. A. Collins e R. D. Pederson, *Telecommunications: a Time for Innovation*, Merle Collins Foundation, Dallas, Texas, 1973
- H. M. Straube, "Chestel 'Uniloop' PABX", *National Telecommunication Conference — NTC 73*, pp. 10E-1 a 10E-6

LEITURA COMPLEMENTAR

Uma relação bibliográfica com um total de 518 entradas aparece em

Ann R. Ward — "LSI Microprocessors and Microcomputers: a Bibliography", *IEEE Computer*, Vol. 7, n.º 7, julho de 1974, pp. 35 a 39

Ann R. Ward — "LSI Microprocessors and Microcomputers: a Bibliography Continued", *IEEE Computer*, Vol. 9, n.º 1, janeiro de 1976, pp. 42 a 53

Thomas R. Blakeslee, "Digital Design with standard MSI and LSI", Wiley — Interscience Publication, N.Y. 1975

ÍNDICE

- Acesso direto, 260
- Adição binária, 42
- Álgebra booleana, 5
- Amplificador sensor, 1
- Análise de circuitos, 1
- AND*
 - função, 110
 - lógico, 5
 - porta, 6
- Aritmética
 - BCD*, 53
 - decimal, 51
 - complemento à 1
 - complemento à 2
 - representação à 1
 - binária de números
 - adição, 42
 - divisão, 59
 - multiplicação, 52
 - subtração, 45
 - em complemento à 1
 - em complemento à 2
 - em sinal e algarismos
- Armazenamento, 139
 - secundário, 3, 139
- Arquitetura
 - da UCP, 226
 - do sistema de computador, 226
- Arquivo, 159
- Astável de pressão, 1
- Atraso, 28
- Babbage, Charles, 1
- Bandwidth, 138
- Baudot, código de, 3
- Bidirecional, 260
- Binário refletido, 110
- Bit, 36, 140
 - de paridade, 36
- Bloco, 159, 228
- Blocking, 160
- Boole, George, 5

ÍNDICE

- Acesso direto, 260, 262
- Adição binária, 42
- Álgebra booleana, 5
- Amplificador sensor, 147
- Análise de circuitos combinatórios, 12
- AND*
 - função, 110
 - lógico, 5
 - porta, 6
- Aritmética
 - BCD*, 53
 - decimal, 51
 - complemento de dez, 51
 - complemento de nove, 51
 - representação de negativos, 51
 - binária de números positivos, 42
 - adição, 42
 - divisão, 59
 - multiplicação, 59
 - subtração, 43
 - em complemento de dois, 47
 - em complemento de um, 44
 - em sinal e amplitude, 48
- Armazenamento, 138
 - secundário, 3, 138, 139, 159
- Arquitetura
 - da *UCP*, 226
 - do sistema de computação, 2
- Arquivo, 159
- Astável de precisão, 128
- Atraso, 28
- Babbage, Charles, 1
- Bandwidth, 138
- Baudot, código de, 36
- Bidirecional, 260
- Binário refletido, código em, 37
- Bit*, 36, 141
 - de paridade, 36
- Bloco, 159, 228
- Blocking, 160
- Boole, George, 5
- Booleana álgebra, 5
 - expressões, 10
- Bootstrap*, 283
- Bounce, 283
- Buffer*
 - circuito integrado, 129
 - de memória, 139
- Buffering*, 160
- Busca, 193
- Byte*, 244
- Cache-backing*, 154
- Campo, 159
- Canal de entrada/saída, 244, 263
- Capacidade de memória, 147
- Caráter, 36
- Carregador, 283
- Cartas de microoperação, 287
- Ciclo
 - básico da máquina, 61
 - completo, 143
 - da memória, 124, 140
 - de execução (*E*), 126, 286
 - de instrução (*I*), 126, 286
 - de máquina, 125, 126, 286
 - do fluxo de dados, 124
 - rachado, 144
 - único, 205
- Circuitos
 - combinatórios, 9, 10, 61
 - análise, 12
 - custo, 13
 - síntese, 13
 - de paridade, 63, 64
 - de prioridade, 64, 65
 - digitais, 9
 - integrados digitais, 28
 - atraso, 28
 - famílias
 - DTL*, 29
 - ECL*, 30
 - HTL*, 30

- MOS-FET**, 31
RTL, 28
TTL, 29
fan-in, 28
fan-out, 28
“meia-soma”, 98
seqüências, 9, 19
diagrama de estado, 21
estado interno, 20
“soma completa”, 42, 99
Clock, 69
Códigos, 36
Bandot, 36
BCD, 38
binário refletido, 37
distância unitária, 36
EBCDIC, 36
Gray, 37
Hollerith, 36
XS-3, 38
Complementação, 109
Complemento
de dez, 51
de dois, 40
de nove, 51
de um, 39
Computador
IAS, 1, 138
IBM S/360, 242
IBM S/370, 242
mini, 4
Patinho Feio, 170
PDP-8, 237
Contador, 85
binário, 85
código Gray, 92
com propagação do “vai um” (assíncrono), 85
com “vai um” simultâneo (síncrono), 86
em anel, 92
não-binários, 90
módulo, 85
“para baixo”, 85
“para cima”, 85
Contatos mecânicos, 132
pulos nos contatos (*bounce*), 283
Controle
do modo de operação, 289
fixo, 288
microprogramado, 288, 289
Conversão de números, 55
BCD-binário, 58
binário-decimal, 56
decimal-binário, 58
Corrente coincidente, 141
Cross-bar, 152
Cycle-steal, 263
D, flip-flop, 68
Deblocking, 160
Decodificador, 62, 289
Demultiplexador, 117
Descriptor de dados, 228
Diagrama de estado, 21
Digitais, circuitos, 9
Digito binário, 36
Diodos, 23
em circuitos lógicos, 23
Distância unitária, código, 36
Distribuição, porta, 117
Divisão binária, 59
DOT-OR, 28
Driver, 129
DTL, 29
Dualidade, 9
Dupla palavra, 227
EBCDIC, código, 36
ECL, 30
Emulação, 226, 293
Endereçamento
abreviado, 233
direto, 231
imediato, 230
implicado, 230
indexado, 231
indireto, 231
indireto pós-indexado, 233
relativo, 233
Endereço, 141
duplo, 230
efetivo, 231, 233
indireto, 3
simples, 228
Entrada, 254
Estado interno, 19
Expandable, 131, 132
Falta, 155
Fan-in, 28
Fan-out, 28
Fase de busca, 290
Fetch (veja Busca)
Flip-flop, 20, 66
D (sensível à borda), 68
JK, 71
JK master-slave, 73
PH (polarity hold), 67
RS (set-reset), 66
Fluxo de dados, 1, 61, 122
Fluxo de sinais, 115
Fio inhibit, 143
Firmware, 111
Fonte de alimentação, 130
Fragmentação, 164
Funções lógicas
AND, 130
NAND, 130
NOR, 130
OR, 130
Gerações
características, 3
primeira, 3
quarta, 4
segunda, 3
terceira, 3
Gerador de sinal
Gray, código, 37
Hardware (termo)
Hard-wired, 123
Hollerith, código, 36
HTL, 30
IAS, computador
IBM
S/360, computador
S/370, computador
Informação, 140
Instrução
aritmética, 238
booleana, 238
curta, 176
de convenção, 238
de deslocamento, 238
de desvio, 238
de edit (edição), 238
de entrada e saída, 238
longa, 172
referentes a memória, 238
única, 238
Interface, 254
multiplexador, 254
sinais de, 254
Interferência de, 254
Interrupção, 3, 289
causa de, 289
classes, 289
de programa, 289
nível simples, 289
níveis múltiplos, 289
no HP-2100, 289
no IBM S/360, 289
Inversor, 6
JK, flip-flop, 71
Karnaugh, mapa, 23
“Ler-destrutivo”, 164
“Limpa ao ligar”, 164

- Funções lógicas
 AND , 110
 $NAND$, 110
 NOR , 110
 OR , 110
- Gerações
 características, 5
 primeira, 3
 quarta, 4
 segunda, 3
 terceira, 3
- Gerador de sinais de controle, 289
- Gray, código, 37
- Hardware* (circuitos), 1
- Hard-wired*, 123
- Hollerith, código, 36
- HTL*, 30
- IAS, computador, 1, 138
- IBM
 S/360, computador, 242
 S/370, computador, 242
- Informação, 141
- Instrução
 aritmética, 234
 booleana, 236
 curta, 176
 de conversão, 237
 de deslocamento, 236
 de desvio, 235
 de *edit* (redação), 237
 de entrada e saída, 237
 longa, 172
 referentes à memória, 235
 única, 206
- Interface, 254
 multiplexada, 257
 sinais de, 257
- Interferência de memória, 140
- Interrupção, 3, 263
 causa de, 269
 classes, 263
 de programa, 248
 nível simples, 264
 níveis múltiplos, 265
 no *HP-2116B*, 270
 no IBM S/360, 249
- Inversor, 6
- JK*, *flip-flop*, 71
- Karnaugh, mapa de, 10
- "Ler-destrutivo", 141
- "Limpa ao ligar", 129
- Linha de atraso, 141
- Localidade de referência, 154
- Lotes, processamento em, 3
- Mapa de Karnaugh, 10
- Máquina Turing, 233
- Máscaras, 236
- Master-slave, flip-flop*, 73
- "Meia soma", circuito, 98
- Memória
 capacidade, 147
 ciclo, 140
 monolítica, 147
 bipolar, 148
 MOS dinâmica, 150
 MOS estática, 149
 MOS-FET, 31
 núcleo de ferrite, 141
 $2\frac{1}{2}D$, 145
 $2D$, 145
 $3D$, 3 fios, 144
 $3D$, 4 fios, 144
 primária, 138, 140, 152
 sistema *cache-backing*, 154
 sistema *cross-bar*, 152
 sistema multiporto, 153
 sistema particionado, 153
 proteção, 248
 virtual, 162
 volátil, 141
- Microinstrução, 176, 290
- Microoperação, 115, 126
- Microprogramação, 123
 codificada, 292
 monofásica, 291
 não-codificada, 292
 paralelo, 292
 polifásica, 291
 série, 292
- Microssub-rotina, 290
- Multiplexador, 117
- Multiplicação binária, 59
- Multiprogramação, 3, 242
- Multiporto, 153
- Minicomputadores, 4
- Monolítica, memória, 147
- NAND (NE)*
 função, 110
 porta, 13
- NOR (NOU)*
 função, 110
 porta, 14
- Núcleo de ferrite, 141
- Números, 37
 fracionários, 39
 inteiros, 39
 negativos, 39

Operando
 autodescrito, 228
 decimal empacotado, 227
 de tamanho variável (*VFL*), 227
OR (OU)
 função, 6, 110
 porta, 7
Osciladores, 127
 Colpits, 127
 com portas *TTL*, 128
 com saída para *TTL*, 127
Overflow, 41
Overlap, 140
Overlay, 162
Overrun, 140

Paginação, 164
Painel de controle, 282
Palavra, 36, 141
 tamanho de, 226
Particionado, 153
PDP-8, computador, 237
Periféricos, 255
 processador, 263
PH, flip-flop, 67
Ponto
 fixo, 41
 flutuante, 41
Porta
 de distribuição, 117
 de seleção, 117
 lógica
 inversor, 6
 AND (*E*), 6
 NAND (*NE*), 13
 NOR (*NOU*), 14
 OR (*OU*), 6
 XOR (*OUX*), 7
Prime implicant (p.i.), 18
Processador periférico, 263
Processamento em lotes, 3
Produto fundamental, 12
Programa, 2
 carregador, 283
Projeto lógico, 2
Proteção de memória, 248
Pulo
 condicional, 235
 incondicional, 235

Registrador, 78
 contador, 85
 de dados da memória, 141, 143
 de endereço da memória, 141
 deslocador, 79
 do código de instrução, 288
 do endereço de instrução, 288

Registro
 físico, 160
 lógico, 160
Relocação dinâmica, 163
Relógio central, 188, 289
Reset, 66
Retorno de transporte, 45
RS, flip-flop, 66
RTL, 28

Saída, 254
Salto, 235
Segmento, 286
Seleção, porta de, 117
Sense amplifier, 141
Sensível à borda, 68
Sensível ao nível, 67
Seqüenciador, 289
Set, 66
Sistema
 de computação, 2
 compromissos, 2
 organização, 1
 de programação, 2
IOCS, 160
 close, 160
 get put, 160
 open, 160
 procura, 160
 numérico, 57
 operacional, 3
Soma
 completa, circuitos, 42, 99
 canônica, 12
 mínima, 15, 18
Somador, 98
 binário em complemento de um, 105
 binário em complemento de dois, 104
 com "propagação do vai um", 101
 com "vai um antecipado", 102
 decimal, 106
 de palavras, 100
Software, 3
Subcampo, 159
Subtração
 binária em complemento de um, 44
 binária em complemento de dois, 47
 binária em sinal e amplitude, 48
 completa, 108
Subtratores, 107
 de palavras, 109
 empréstimo antecipado, 109
 propagação do empréstimo, 109

Tabela
 de combinações, 10
 de estados (*flow-table*), 22

estados
 ponto de
 Tamanho de
 Tecnologia
 LSI, 147
Tempo
 de acesso, 11
 de transporte
 Timing, 66, 103
Transistor
Transistor
Transformador
Transistor
 como chave
 estudo em
 estudo base
 em circuitos
Tri-state logic
TTL, 29

Underflow, 40
Unidade
 aritmética, 1
 central de p

- estado interno, 22
ponto de operação, 22
Tamanho de palavra, 226
Tecnologia
 LSI, 147
Tempo
 de acesso, 138
 de transporte, 139
Timing, 66, 124
Transbordamento, 41
Transcodificação dinâmica de endereço, 162
Transformada numérica, 11
Transistores, 23
 como chave, 24
 estudo em regime, 24
 estudo transitório, 25
 em circuitos lógicos, 26
Tri-state logic, 131
TTL, 29
Underflow, 41
Unidade
 aritmética, 110
 central de processamento, 1
de controle, 61, 123, 285
 assincrona, 287
 centralizada, 287
 descentralizada, 287
 síncrona, 287
Unidirecional, 260
Univia, 259
“Vai um” (*carry-out*), 42
“Vem um” (*carry-in*), 42
“Vem um quente”, 48
Velocidade de dados, 139
Vias (*bus*), 61, 121
 bidirecional, 261
 características, 260
 unidirecional, 260
Virtual, memória, 162
XS-3, código, 38
XOR (OUX)
 lógico, 7
 porta, 7
Wilkes, M. V., 290

... de um, 105
... de dois, 104
... um", 101
... ", 102

... de um, 44
... de dois, 47
... óptima, 48

... 109
... número, 109

... 22

74 - QD - 15-474

Este trabalho foi elaborado pelo processo de **FOTOCOMPOSIÇÃO**
Monophoto - no Departamento de Composição da Editora
Edgard Blücher Ltda. - São Paulo - Brasil

H.D. - U.S.A. 1971



impresso na
planimpress gráfica e editora
rua anhaia, 247 - s.p.

74 - QD - 15-474

W.M.C. - G.O. - 4



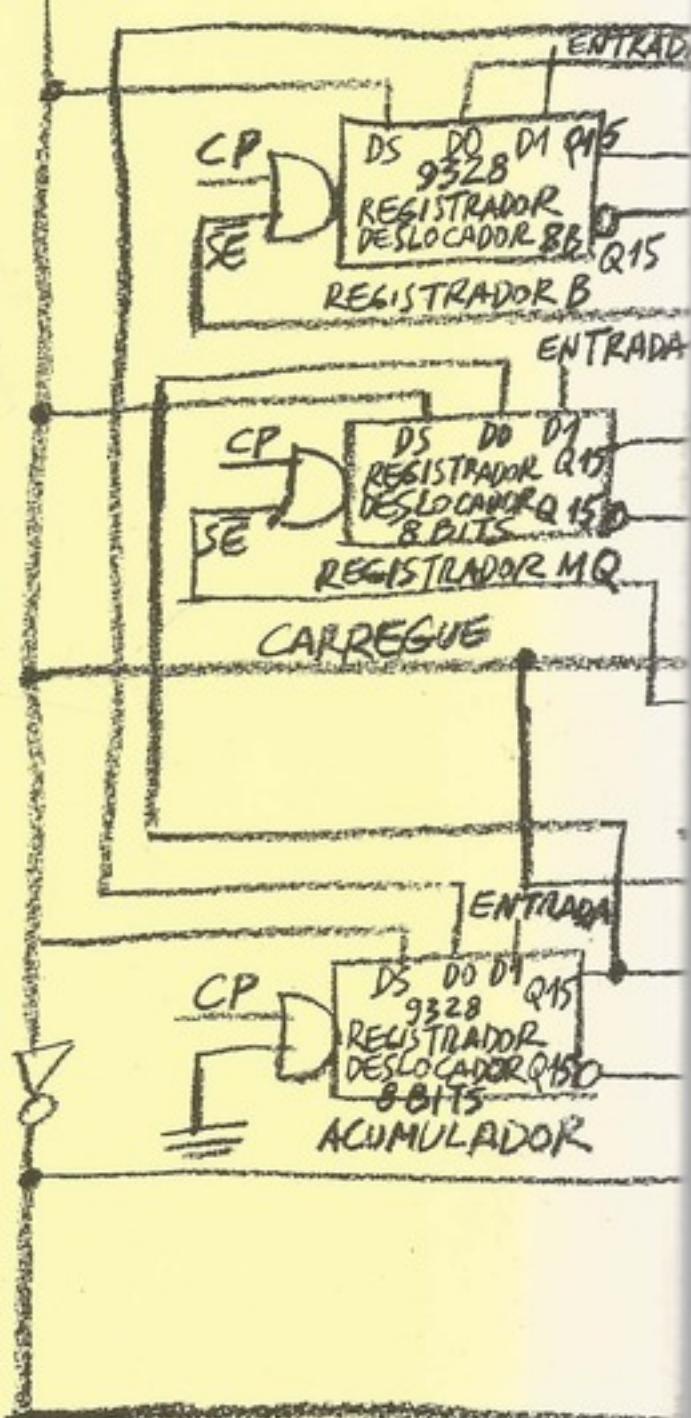
EDITORAS EDGARD BLÜCHER LTDA.

PROTEÇÃO DE

INFORMÁTICA

www.edgarblucher.com

CARREGUE



05372