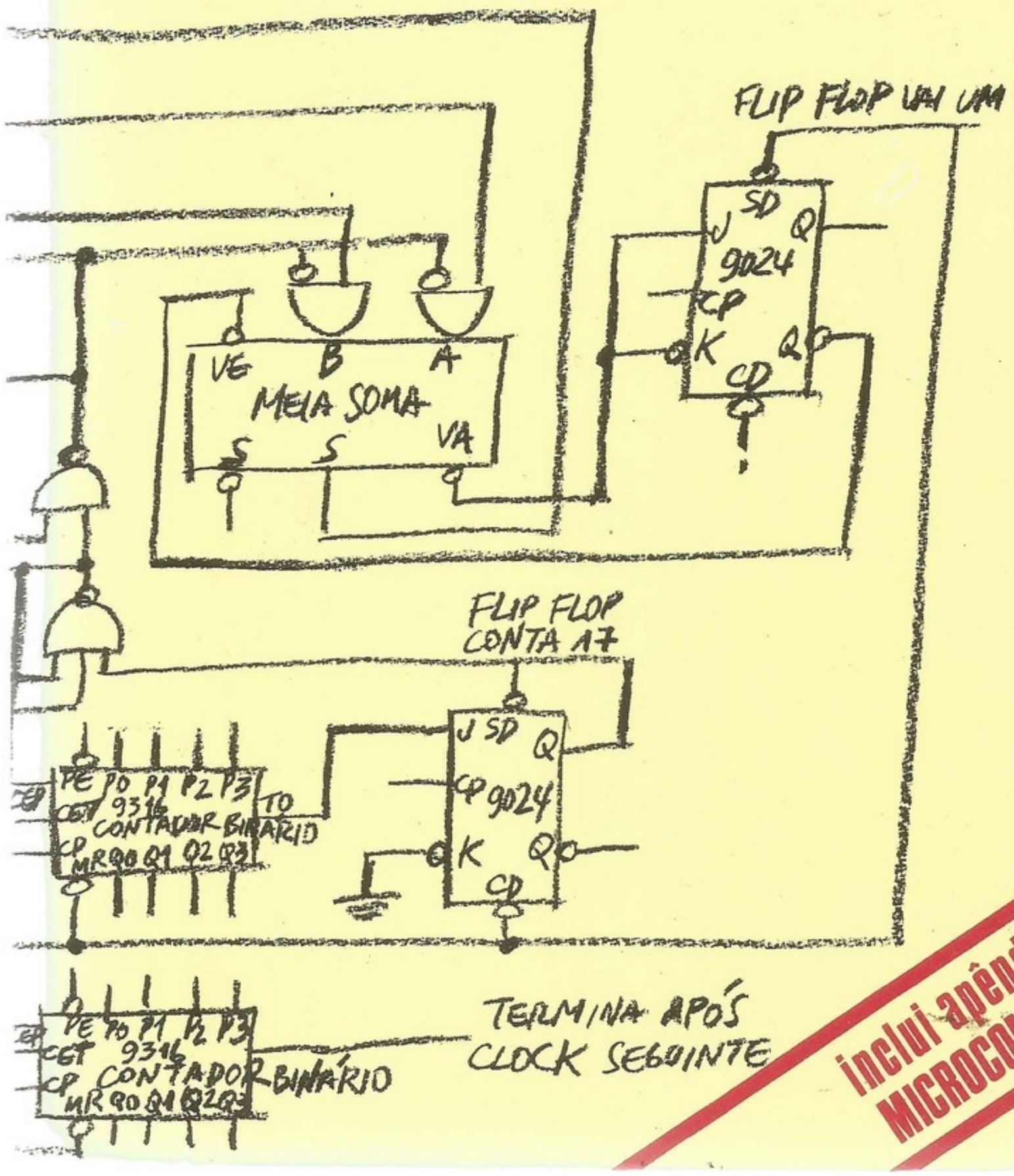
Glen George Langdon Jr.
Edson Fregni

PROJETO DE COMPUTADORES DIGITAIS

2.ª edição



inclui apêndice sobre MICROCOMPUTADORES

PROJETO DE COMPUTADORES DIGITAIS

FICHA CATALOGRÁFICA

Langdon, Glen George, 1937–

L263p 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 construção 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

2ª edição

EDITORA EDGARD BLÜCHER LTDA.

©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

EDITORA EDGARD BLÜCHER LTDA.
0 1000 CAIXA POSTAL 5450
END. TELEGRAFICO: BLUCHERLIVRO
SÃO PAULO — SP — BRASIL

Impresso no Brasil *Printed in Brazil*

CONTEÚDO

Prefácio XV
Prefácio dos autores XVI

1 *Introdução e conhecimentos básicos* 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 díodos 23
a. Circuitos lógicos com díodos 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 *RTL* (*resistor transistor logic*) 28
c. Família *DTL* (*diode transistor logic*) 29
d. Família *TTL* (*transistor transistor logic*) 29
e. Família *HTL* (*high threshold logic*) 30
f. Família *ECL* (*emitter coupled logic*) 30
g. Circuitos *MOS-FET* 31

Exercícios 33
Bibliografia 34

2 *Códigos, números e aritmética* 36

2.1 Códigos 36
2.2 Números 37
a. Sistemas 37
b. Números inteiros e fracionários 39

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 43
c. Aritmética usando complemento de um do subtraendo 44
d. Aritmética usando complemento de dois 47
e. Aritmética com números em sinal e amplitude 48
f. Variações 50
2.4 Aritmética decimal 51
a. Representação dos negativos 51
b. Exemplos de aritmética decimal 51
c. Aritmética em *BCD* usando representação sinal e amplitude 53
2.5 Conversão entre números binários e decimais 55
a. Conversão binária a decimal 56
b. Conversão decimal a binária 58
2.6 Sumário 59
Exercícios 59
Bibliografia 60

3 *Elementos do projeto lógico* 61
3.1 Introdução 61
3.2 Circuitos combinatórios 61
a. Decodificadores 62
b. Circuitos de paridade 63
c. Circuitos de prioridade 64
Exercícios 64
3.3 *Flip-flop* 66
a. *Flip-flop RS* 66
b. *Flip-flop* tipo *PH* 67
c. *Flip-flop* tipo *D*, sensível à borda 68
d. *Flip-flop* tipo *JK* 71
e. *Flip-flop master-slave* 73
f. Outros tipos de *flip-flops* 74
Exercícios 75
3.4 Registradores 78
a. Registrador deslocador 79
b. Registrador-contador 85
c. Contadores binários 85
d. Contadores não-binários 90
Exercícios 93
3.5 Unidades aritméticas 98
a. Somadores 98
b. Subtratores 107
c. Complementação 109
d. Funções lógicas 110
e. Unidades aritméticas 110
f. Exemplo detalhado de uma unidade aritmética 111
Exercícios 114

39
39
40
41
42
42
43
44
47
48
50
51
51
51
53
55
56
58
59
59
60
61
61
61
62
63
64
64
66
66
67
68
71
73
74
75
78
79
85
85
90
93
98
98
107
109
110
110
111
114

3.6 Fluxo de sinais 115
a. Transferência de dados para registradores 115
b. Transferência em paralelo 118
c. Transferências por vias 121
d. Fluxo de dados 122
Exercícios 123
3.7 O *timing* e o ciclo da máquina 124
a. Exemplo 124
b. A determinação do ciclo da máquina 125
Exercício 126
3.8 Circuitos especiais 126
a. Osciladores 127
b. "Limpa ao ligar" 129
c. Entradas do sistema 129
d. *Driver* 129
e. Fontes de alimentação 130
f. Outros circuitos 131
g. Circuitos de interface com contatos mecânicos 132
3.9 Sumário 136
Bibliografia 137

4 *Memória e armazenamento* 138

4.1 Introdução 138
4.2 O *bandwidth* e suas implicações 138
4.3 Memória primária 140
4.4 Memória de núcleo de ferrite 141
4.5 Memória monolítica 147
a. Bipolar 148
b. Memória *MOS* 149
4.6 Sistemas de memória primária 152
4.7 O sistema *cache* 154
a. Motivação 154
b. O sistema *cache-backing* 154
c. Localidade de referência 154
d. A *performance* do esquema 155
e. A viabilidade do *cache* 156
f. A administração do *cache* 157
g. Assuntos relacionados ao *cache* 158
4.8 Armazenamento secundário 159
a. Composição de arquivos 159
b. A técnica de *blocking* 160
c. O sistema *IOCS* 160
d. Exemplo 161
4.9 Memória virtual 162
a. Motivação 162
b. Transcodificação dinâmica de endereço 162
c. O mecanismo de transcodificação 163
d. O tamanho dos blocos ou páginas 164
e. Tabelas de paginação de um nível 164
f. Tabelas de paginação de dois níveis 165
g. Implementação de relocação dinâmica no *S*/360 modelo 67 167
h. Experiência com memória virtual 167

4.10 Sumário 167
Exercícios 168
Bibliografia 168

5 *Exemplo de um minicomputador* 170

5.1 Introdução 170
5.2 Esboço da arquitetura 170
5.3 Instruções principais 172
a. Instruções longas 172
a.1. Grupo principal 172
a.2. Grupo de entrada/saída 173
a.3. Grupo das imediatas 175
b. Instruções curtas 176
b.1. Microgrupo 1 (*MICG*1) 176
b.2. Microgrupo 2*a* (*MICG*2*a*) 177
b.3. Microgrupo 2*b* (*MICG*2*b*) 178
5.4 Fluxo de dados 179
a. Análise dos circuitos 181
a.1. Memória 181
a.2. Registradores 181
a.3. Unidade aritmética 185
a.4. Portas de seleção 185
a.5. Ciclo da máquina 188
b. Implementação dos circuitos 189
b.1. *FR*1-1 e *FR*2-2 190
b.2. *FS*1-3 e *FS*2-4 190
b.3. *FAC*-5 190
b.4. *FM*1-6 191
b.5. *FCM*-7 191
b.6. *FIN*-8 192
5.5 Fases e microoperações 193
a. Fases 193
b. Microoperações 194
c. Exemplo 194
5.6 Unidade de controle 195
a. Considerações gerais 195
b. Exemplo 199
c. Decodificador 199
d. Controle de estado 199
5.7 Interrupção 204
5.8 Modos de operação especiais 205
a. Grupo 1 205
b. Grupo 2 206
5.9 Sumário 207
5.10 Pranchas 209

Exercícios 225

6 *Arquitetura da UCP* 226

6.1 Introdução 226
6.2 Formatos de informação 226
a. Tamanho de palavra 226

167
168
168
170
170
170
172
172
172
173
175
176
176
177
178
179
181
181
181
185
185
188
189
190
190
190
191
191
192
193
193
194
194
195
195
199
199
199
204
205
205
206
207
209
225
226
226
226
226

b. Relacionamento operando/instrução 226
c. Formato dos operandos 227
d. O formato das instruções 228
6.3 Especificação do operando 230
a. Imediato ou implicado 230
b. Direto 231
c. Indireto 231
d. Indexado 231
e. Relativo 233
f. Abreviado 233
g. Combinações 233
6.4 A potência do conjunto de instruções 233
a. Instruções aritméticas 234
b. Instruções referentes à memória 235
c. Desvios 235
d. Deslocamento e operações booleanas 236
e. Supervisão e entrada/saída 237
f. Outras instruções 237
6.5 O *PDP*-8 237
a. Introdução 237
b. As instruções 238
c. Estados principais e interrupção 241
d. Comentário 242
6.6 Arquitetura da terceira geração e os sistemas IBM *S*/360 e IBM *S*/370 242
a. Motivação 242
b. As direções da arquitetura 242
6.7 A arquitetura do IBM *S*/360 243
a. Ponto de vista do programador 244
b. Formato dos dados 244
c. Formatos da instrução e do endereçamento 245
6.8 Feições do *S*/360 e *S*/370 para multiprogramação 248
a. Introdução e o sistema *Stretch* 248
b. Proteção da memória 248
c. Relocação e transcodificação dinâmica do endereço 249
d. O *PSW* e o mecanismo de interrupção 249
e. Comentário sobre o *PSW* 250
f. O estado "supervisor" 251
g. A instrução "*test and set*" (*TS*) 251
6.9 Sumário 251

Exercícios 252
Bibliografia 252
Leituras complementares 253

7. *Entrada/saída* 254

7.1 Introdução 254
7.2 Dispositivos e suas interfaces 255
a. Funções dos periféricos 255
b. A essência do controle dos periféricos 255
c. A ligação entre a *UCP* e o dispositivo 255
d. Os sinais da interface 257
e. O *timing* da via de *E/S* 258

e.1. Caso síncrono 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 (*DMA*) 262
c. Canal 263
d. Processador periférico (*CDC* 6600) 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. *Hardware* 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 *HP* 2116*B* 270
a. Assuntos gerais 270
b. Os níveis e a seqüência da interrupção 271
c. A rede de prioridade 271
d. Os *flip-flops control* e *flag* 271
e. O projeto da parte da interrupção do cartão de interface 272
f. O subsistema de interrupção da *UCP* 274
g. Carta de *timing* da interrupção 275
7.6 A arquitetura de *E/S* no *S*/360 277
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 (*IPL*) 283
e. Controles "roda", "partida" e "pare" 283
7.8 Sumário e conclusões 284
Exercícios 284
Bibliografia 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

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

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 conseqüência, 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. Freqüentou 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 início 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 início 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 interatuam. 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 possuírem 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 aí 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 *HP*2116*B* 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 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.

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

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 relês 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 relês 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

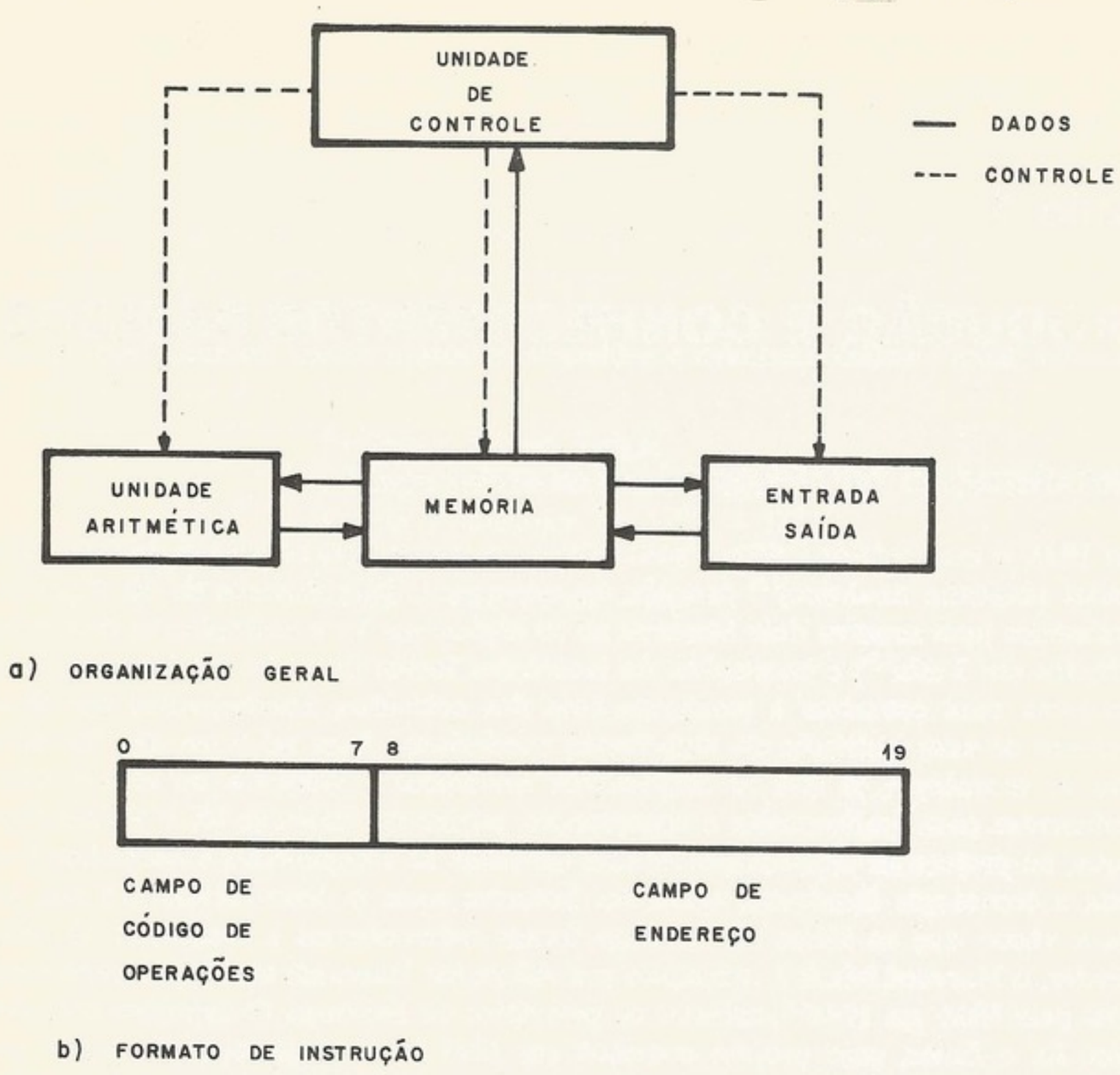


Figura 1-1. O computador *IAS*

(*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.

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

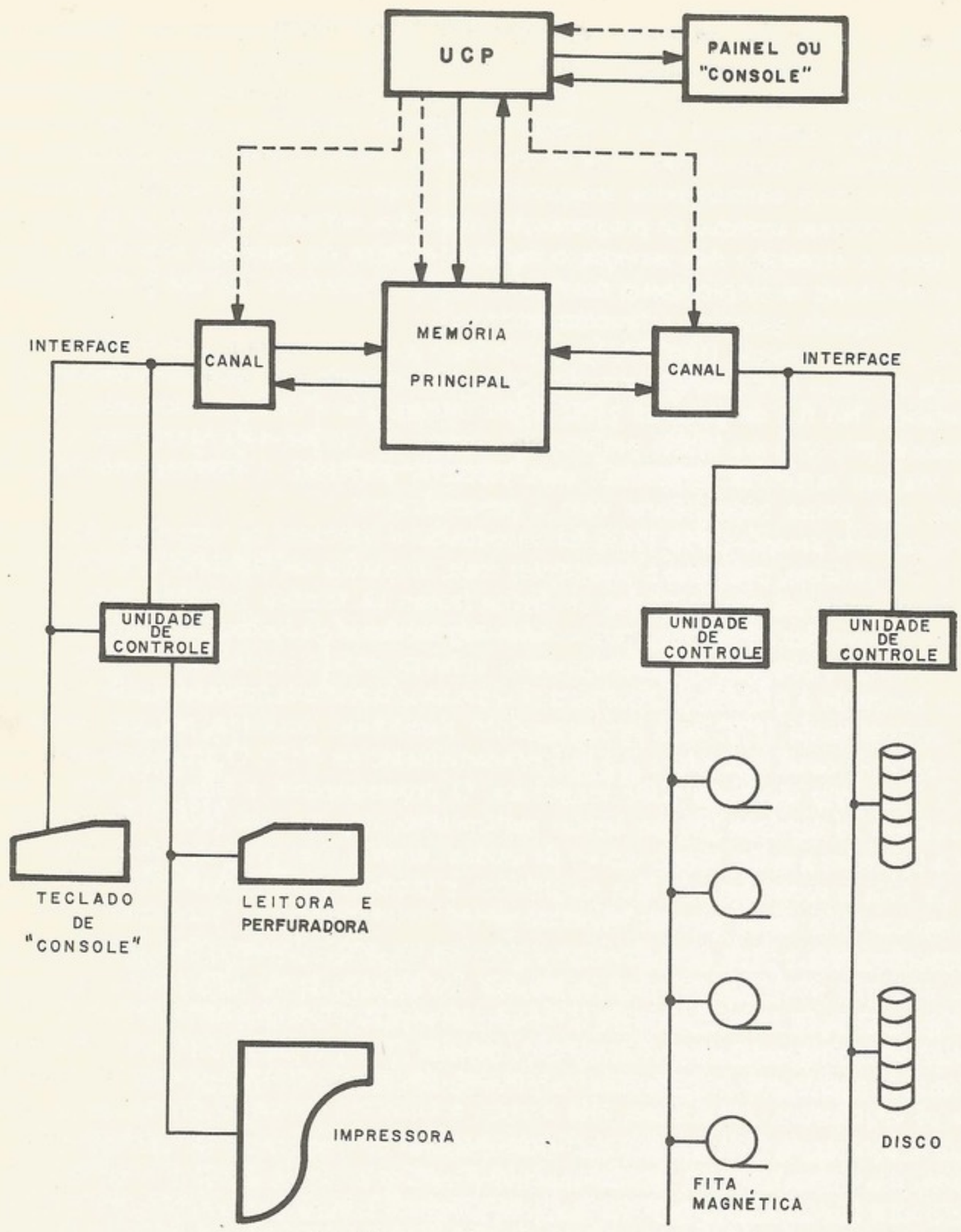


Figura 1-2. Um computador de terceira geração

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

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

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

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:

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

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:

$$\text{(P.5)} \quad 0 \cdot 0 = 0;$$
$$\text{(P.6)} \quad 0 \cdot 1 = 0;$$
$$\text{(P.7)} \quad 1 \cdot 0 = 0;$$
$$\text{(P.8)} \quad 1 \cdot 1 = 1.$$

Operação OR

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

(P.9) $0 + 0 = 0$;
(P.10) $0 + 1 = 1$;
(P.11) $1 + 0 = 1$;
(P.12) $1 + 1 = 1$.

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 $\veebar$), e pode ser definida pelos seguintes postulados:

(P.13) $0 \oplus 0 = 0$;
(P.14) $0 \oplus 1 = 1$;
(P.15) $1 \oplus 0 = 1$;
(P.16) $1 \oplus 1 = 0$.

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),

$$A\,B = B\,A,$$
$$(A\,B)\,C = A\,(B\,C) = A\,B\,C;$$

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'".

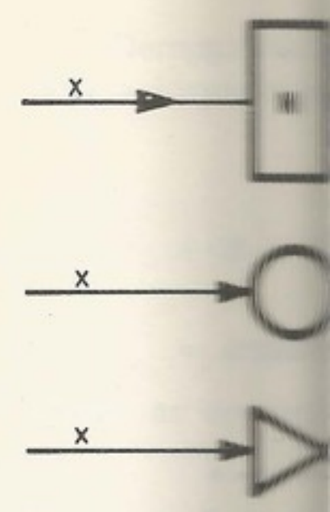
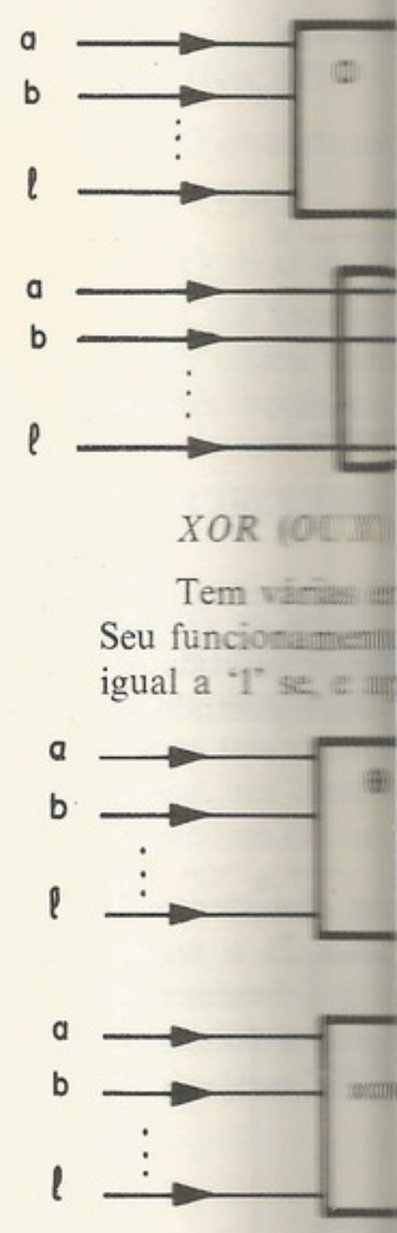


Figura 1-3. Símbolos lógico *NOT*



XOR (OU

Tem
Seu funcio
igual a '1' se

c. Teorema

Sejam A, B e

(T.1) $A \cdot 0 =$
(T.5) $A + 0 =$
(T.9) $A \oplus 0 =$
(T.13) $(A')' =$
(T.14) $A \cdot B +$
(T.15) $(A + B) \cdot$

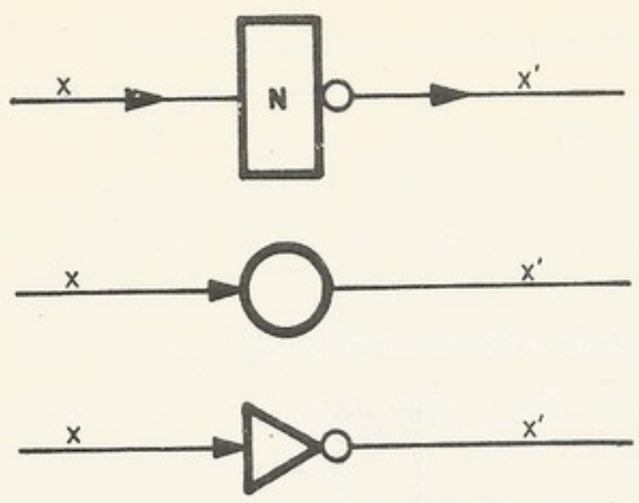


Figura 1-3. Símbolos mais comuns do bloco lógico *NOT*

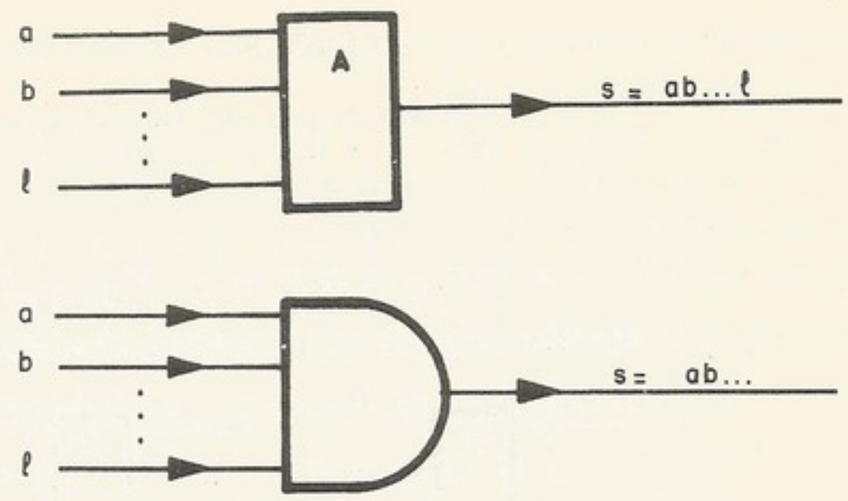


Figura 1-4. Símbolos mais comuns do bloco lógico *AND*

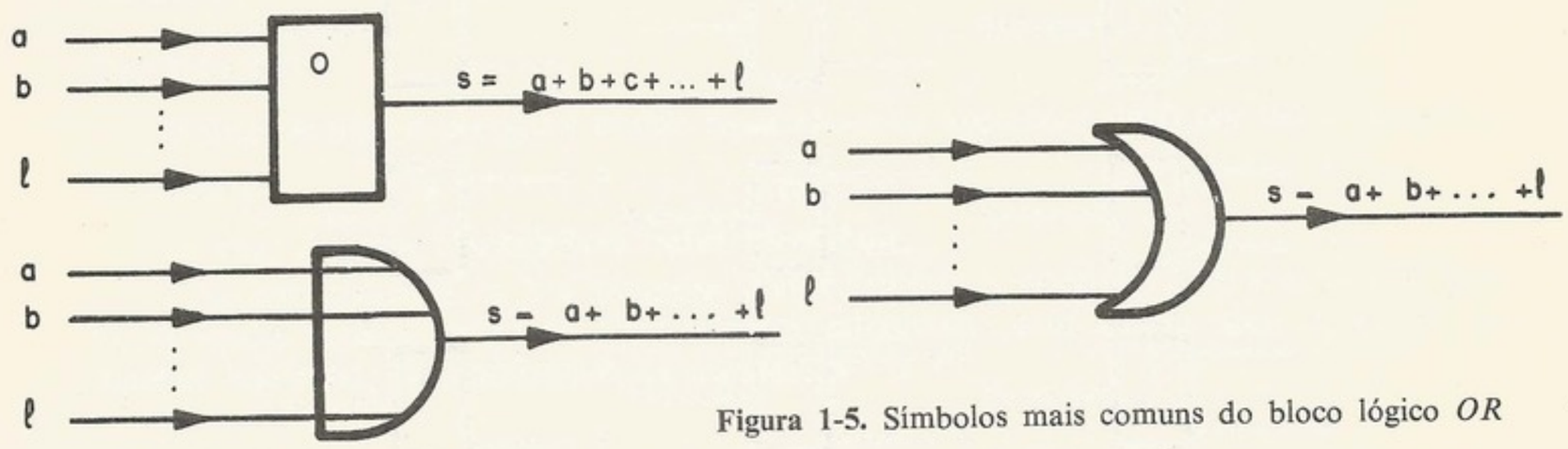


Figura 1-5. Símbolos mais comuns do bloco lógico *OR*

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'".

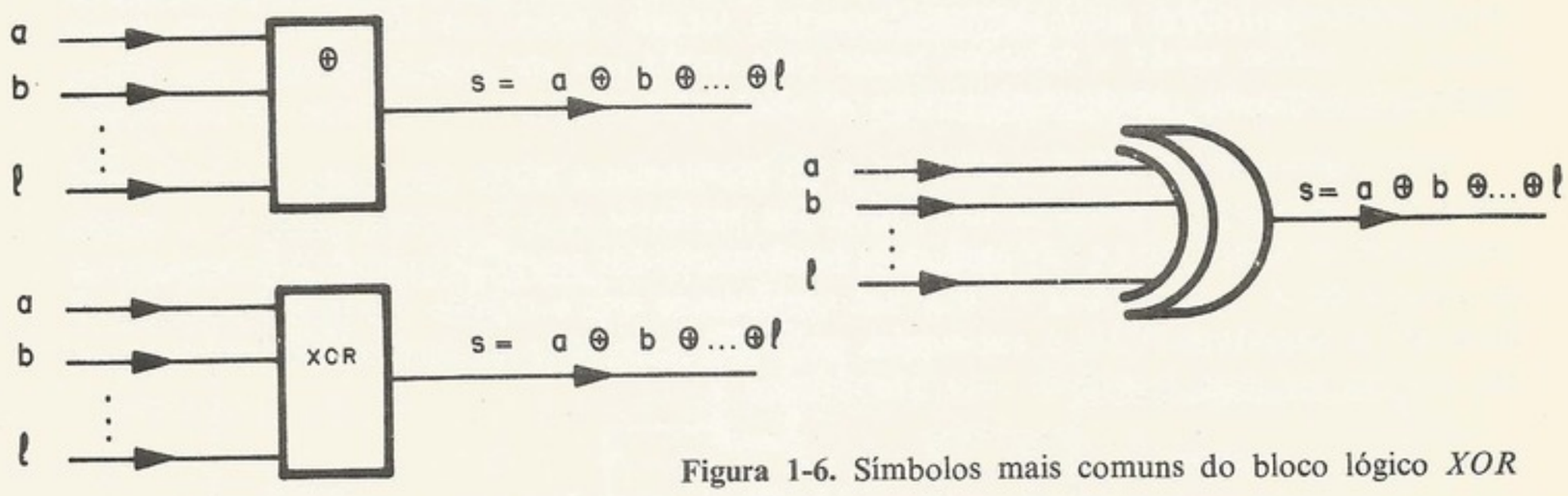


Figura 1-6. Símbolos mais comuns do bloco lógico *XOR*

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) $(A')' = A$;

$$\left.\begin{array}{l}\text{(T.14) } A \cdot B + A \cdot C = A \cdot (B + C) \\ \text{(T.15) } (A + B) \cdot (A + C) = A + B \cdot C\end{array}\right\} \text{(fatoração);}$$

(T.16) $A \cdot B + A \cdot B' = A$; (T.17) $A + A \cdot B = A$; (T.18) $A + A' \cdot B = A + B$;

$$\left.\begin{array}{l}\text{(T.19) } (A \cdot B)' = A' + B' \\ \text{(T.20) } (A + B)' = A' \cdot B'\end{array}\right\} \text{(teoremas de De Morgan);}$$

(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.

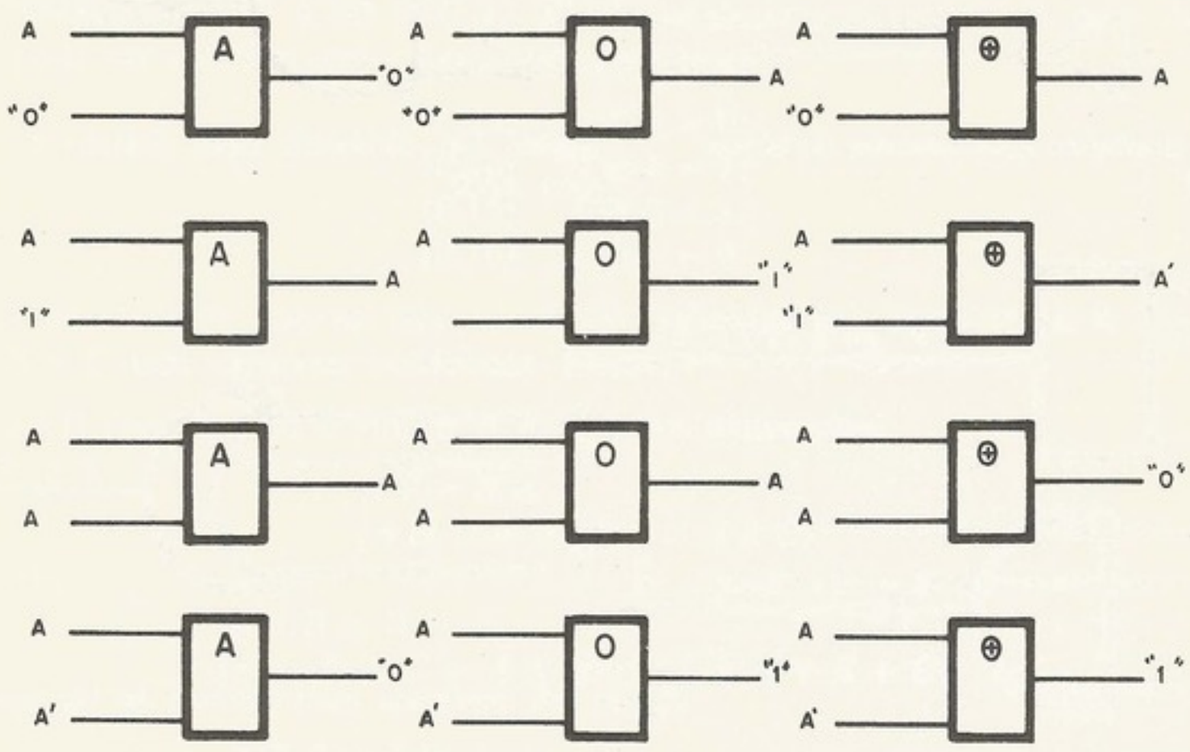


Figura 1-7. Significado físico das doze primeiras identidades

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:

$$A \cdot B + A \cdot B' \overset{\text{(T.14)}}{=} A \cdot (B + B') \overset{\text{(T.8)}}{=} A \cdot 1 \overset{\text{(T.2)}}{=} A.$$

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

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

$$A \cdot B + A' \cdot C + B \cdot C \overset{\text{(T.2)}}{=} A \cdot B + A' \cdot C + B \cdot C \cdot 1 \overset{\text{(T.8)}}{=}$$

$$= A \cdot B + A' \cdot C + B \cdot C \cdot (A + A') \overset{\text{(T.14)}}{=} A \cdot B + A' \cdot C + B \cdot C \cdot A + B \cdot C \cdot A'.$$

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

$$A \cdot B + A' \cdot C + B \cdot C \overset{\text{(T.15)}}{=} A \cdot B + A \cdot B \cdot C + A' \cdot C + A' \cdot C \cdot B = A \cdot B + A' \cdot C.$$

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".

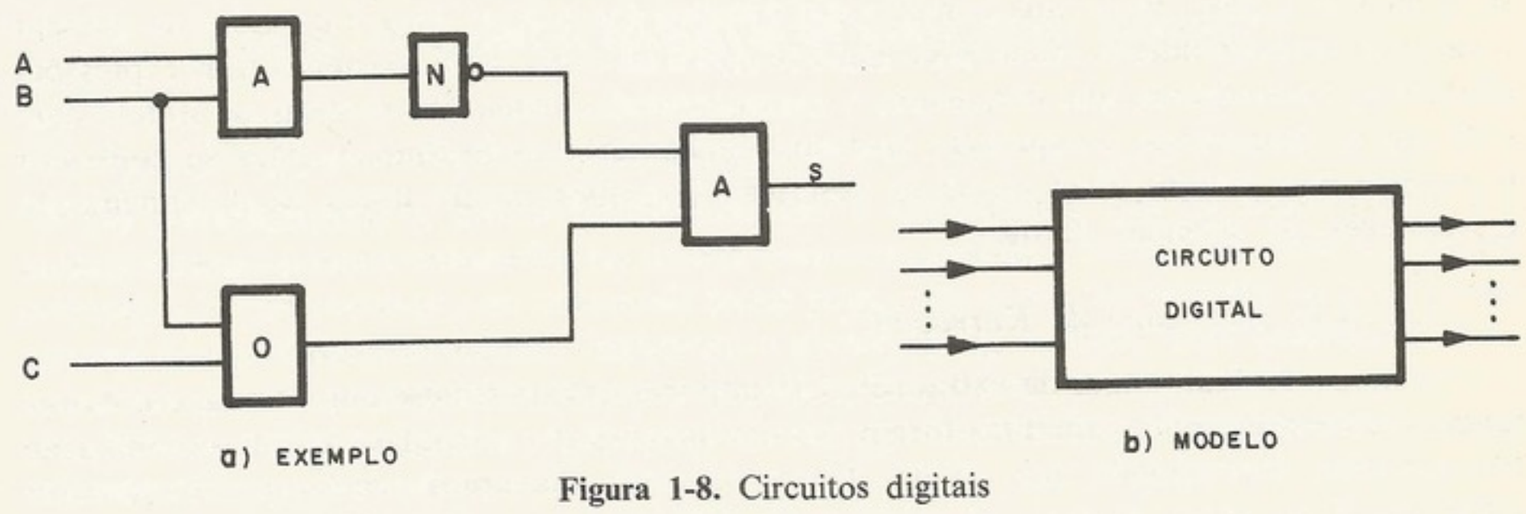


Figura 1-8. Circuitos digitais

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.

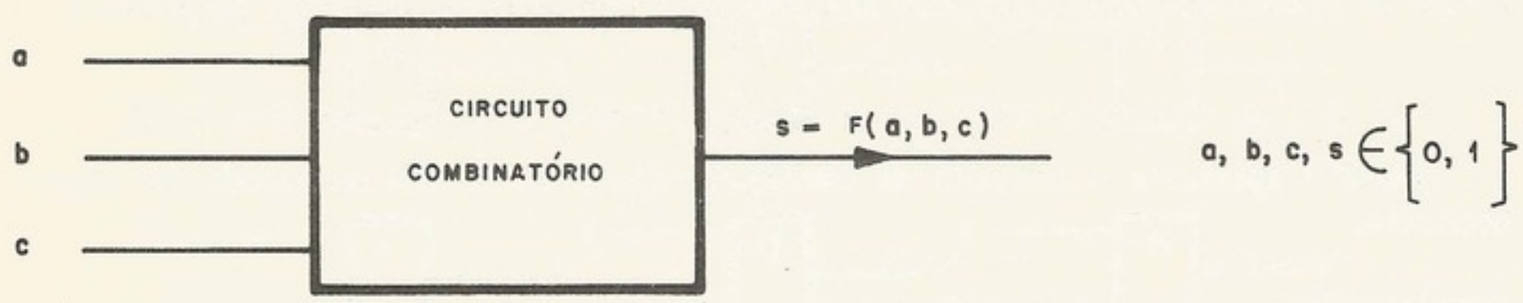


Figura 1-9. Esquema de um circuito combinatório

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.

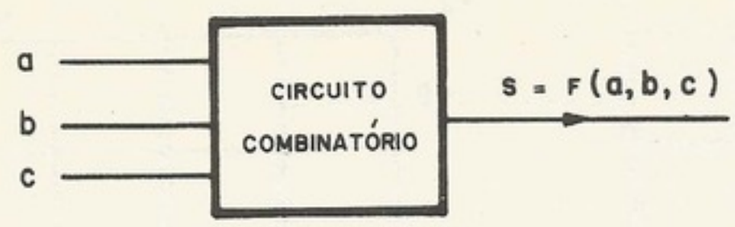
Descrição por uma expressão booleana

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) MODELO

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

b) TABELA DE COMBINAÇÕES

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

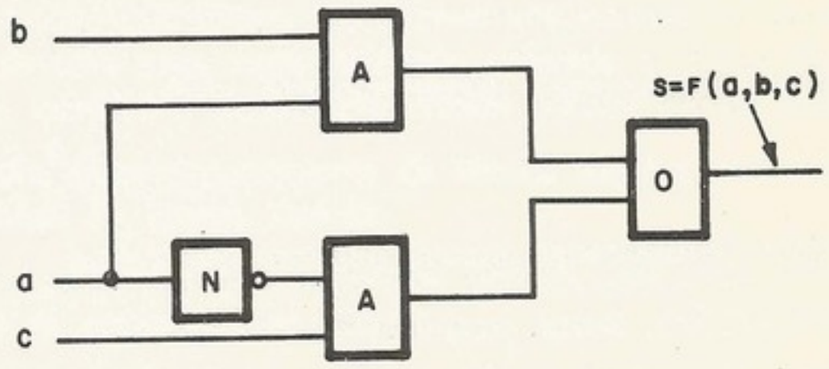
c) EXPRESSÃO BOOLEANA

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



f) DIAGRAMA LÓGICO

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.

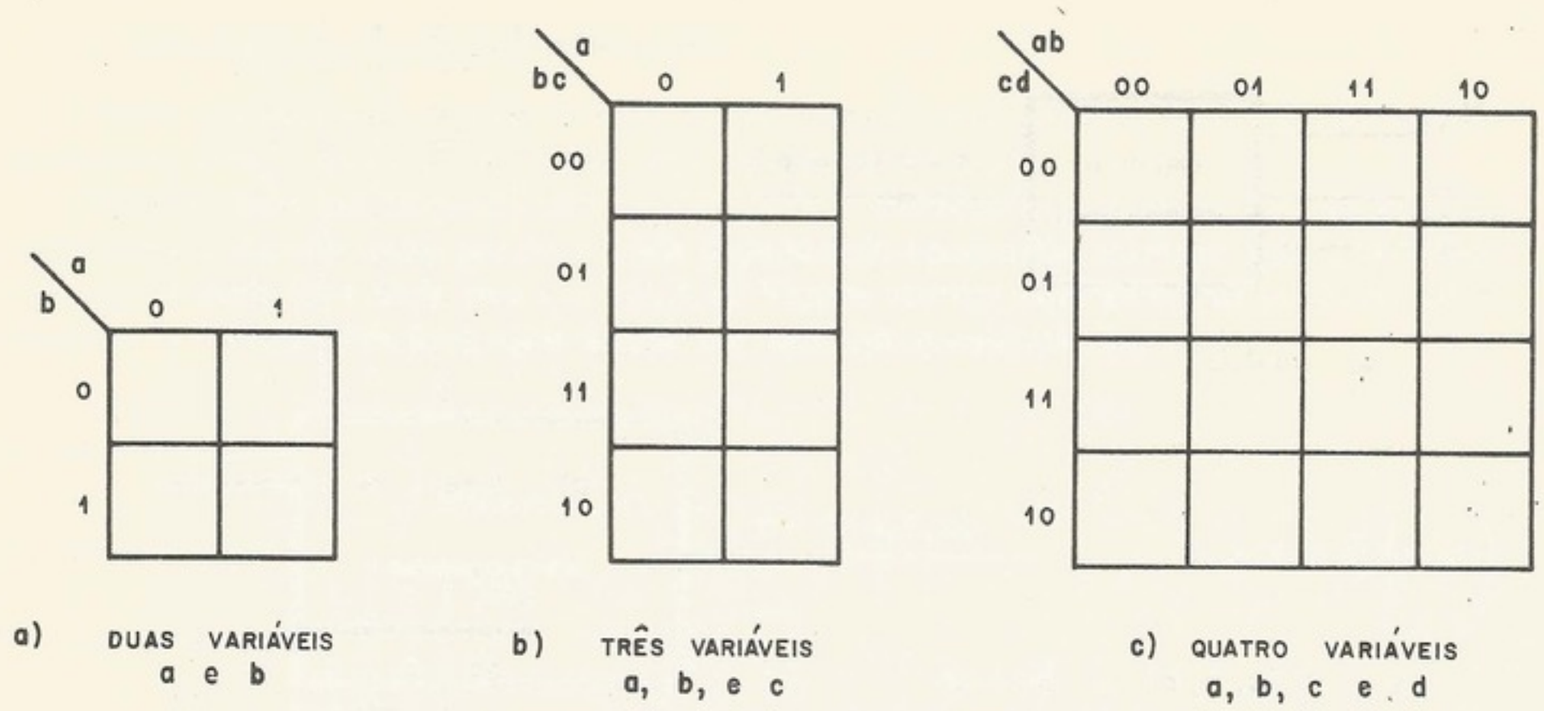


Figura 1-11. Mapas de Karnaugh

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

$$\text{entrada} \left| \begin{array}{ccc} a & b & c \\ 0 & 1 & 1 \end{array} \right.$$

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

$$\text{entrada} \left| \begin{array}{cccc} a & b & c & d \\ 1 & 0 & 0 & 1 \end{array} \right.$$

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

DEFINIÇÃO. Soma canônica[9]

Dada uma tabela de 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.

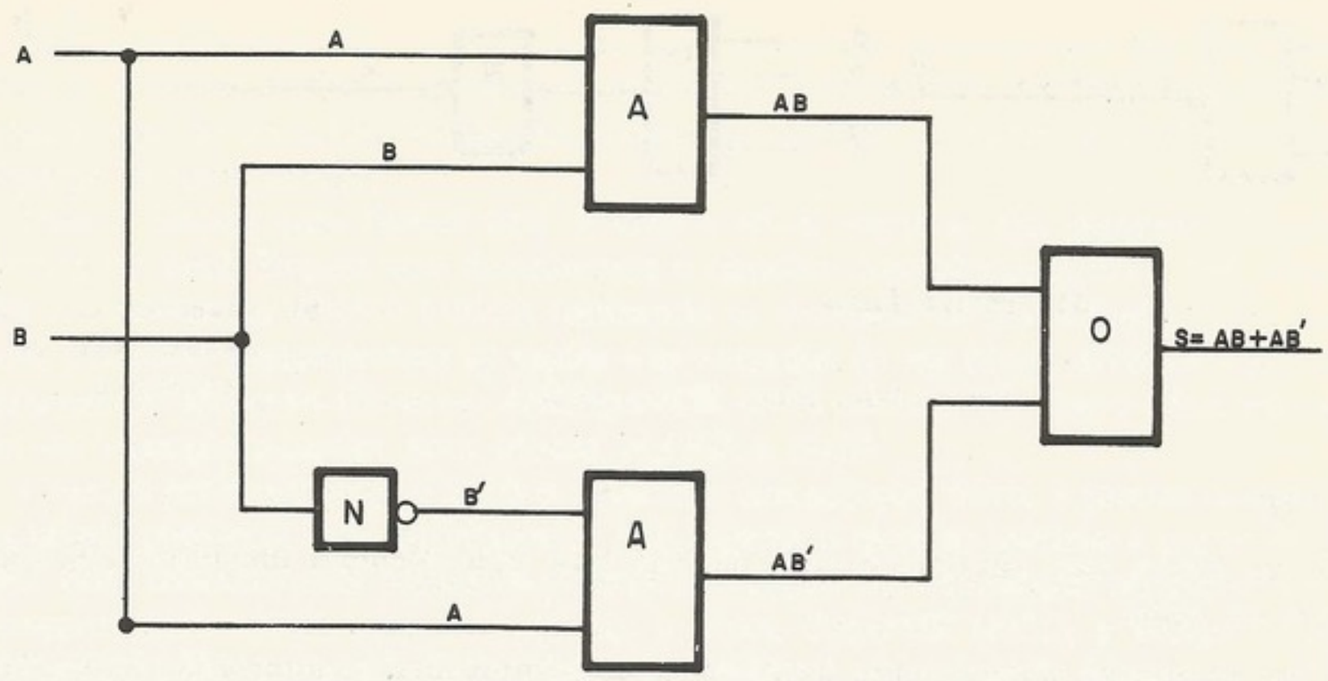


Figura 1-12. Exemplo de análise de um circuito combinatório

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.

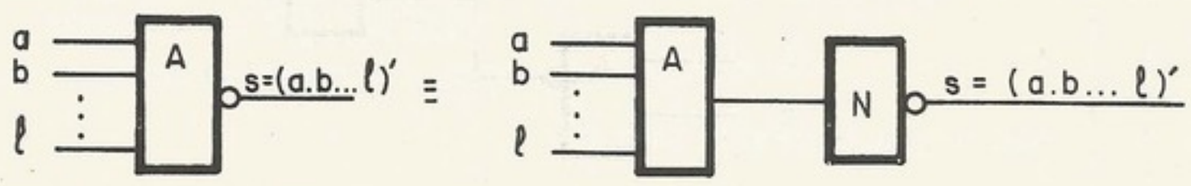
Custo (2). Entre dois circuitos de mesmo número de blocos lógicos, será mais barato aquele que tiver menor número total de entradas dos blocos.

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.

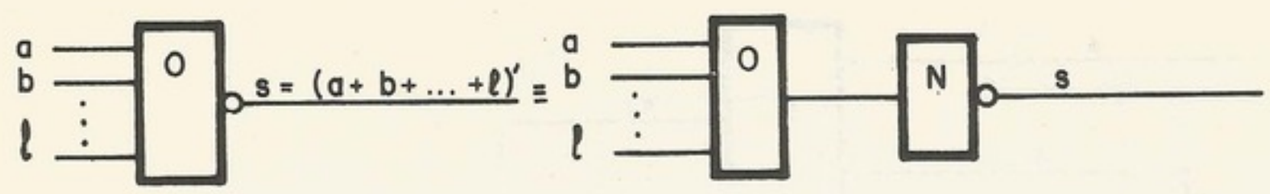


a	b	s
0	0	1
0	1	1
1	0	1
1	1	0

a) EQUIVALÊNCIA

b) TABELA DA VERDADE PARA NAND DE DUAS ENTRADAS

Figura 1-13. Bloco lógico *NAND*



a) EQUIVALÊNCIA

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

b) TABELA DA VERDADE PARA NOR DE DUAS ENTRADAS

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:

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

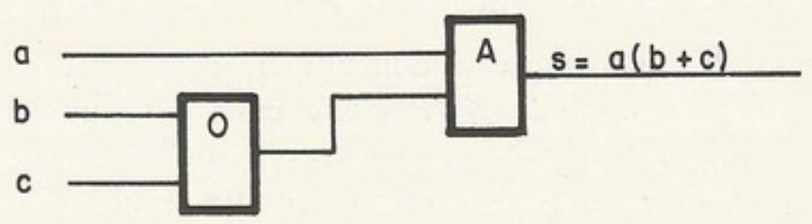
A seguir, tentamos obter uma expressão mais simples cuja síntese deverá ser mais barata. Então, simplificando

$$s = abc + abc' + ab'c,$$
$$s = ab + ab'c,$$
$$s = a(b + b'c),$$
$$s = a(b + c).$$

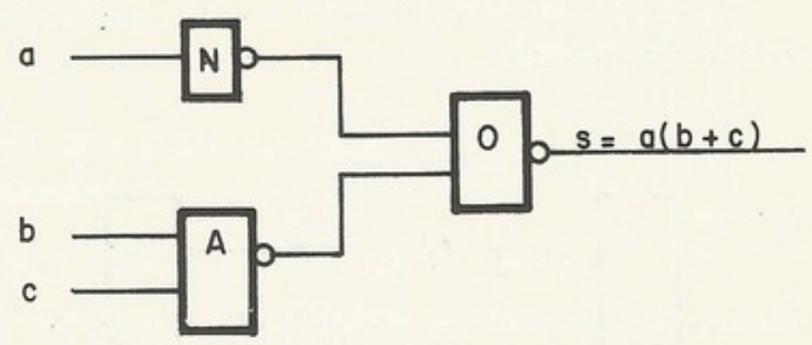
O circuito sintetizado com a expressão mais simples é visto na Fig. 1-15(b).

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

a) DESCRIÇÃO PELA TABELA DA VERDADE



b) DIAGRAMA LÓGICO DO CIRCUITO SINTETIZADO COM AND E OR



c) DIAGRAMA LÓGICO DO CIRCUITO SINTETIZADO COM NAND E NOR

Figura 1-15. Circuito detetor de código *BCD* inválido

Esse método, que exige muita arte do projetista, é um processo que não leva necessariamente ao mínimo, podendo-se obter expressões irredutíveis 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-los 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

$$\text{(T.16)}\quad A \cdot B + A \cdot B' = A,$$
$$\text{(T.21)}\quad A \cdot B + A' \cdot C + B \cdot C = A \cdot B + A' \cdot C$$

e, eventualmente, o teorema

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

simplificamos a soma canônica até chegarmos à soma mínima.

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

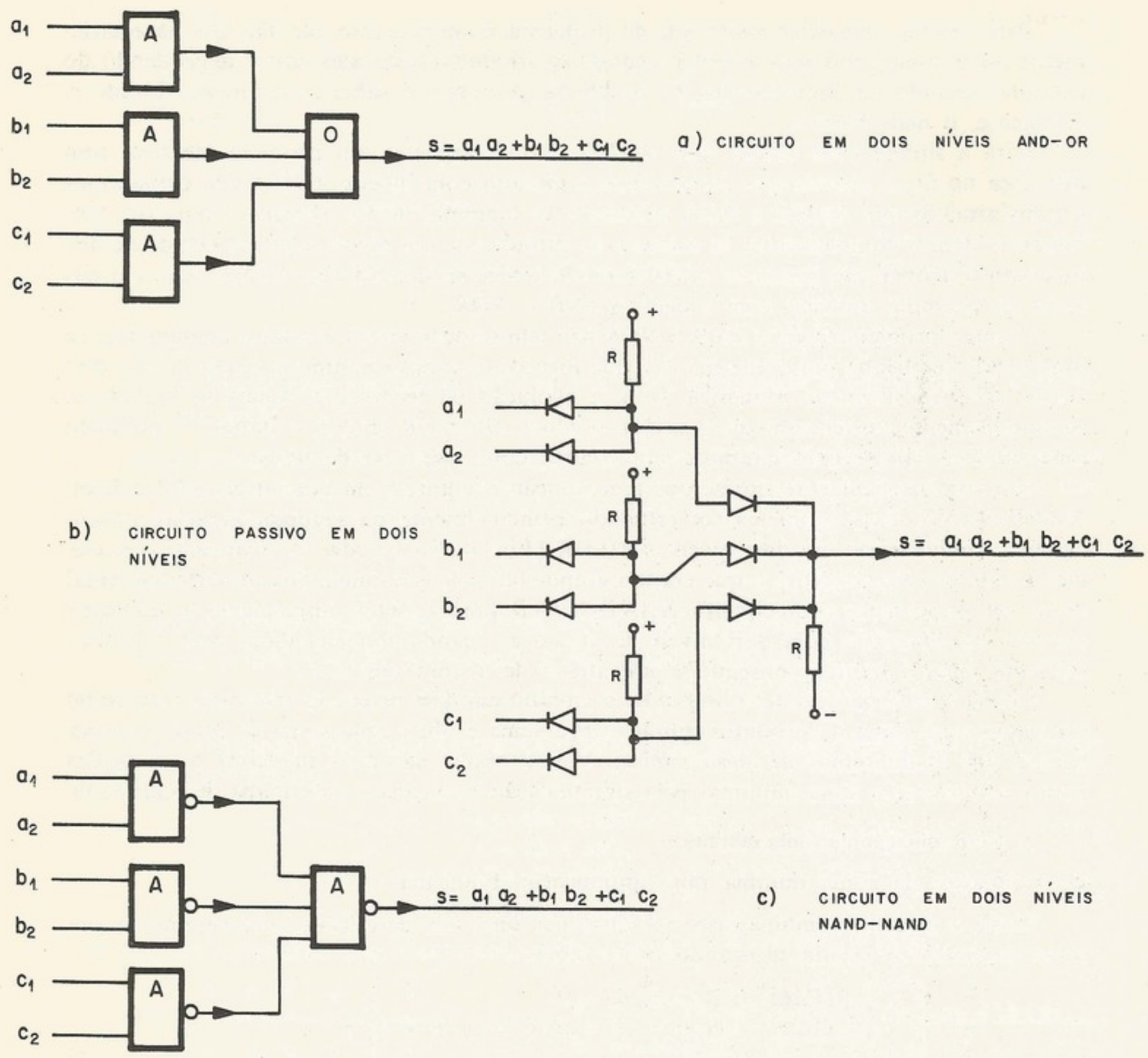
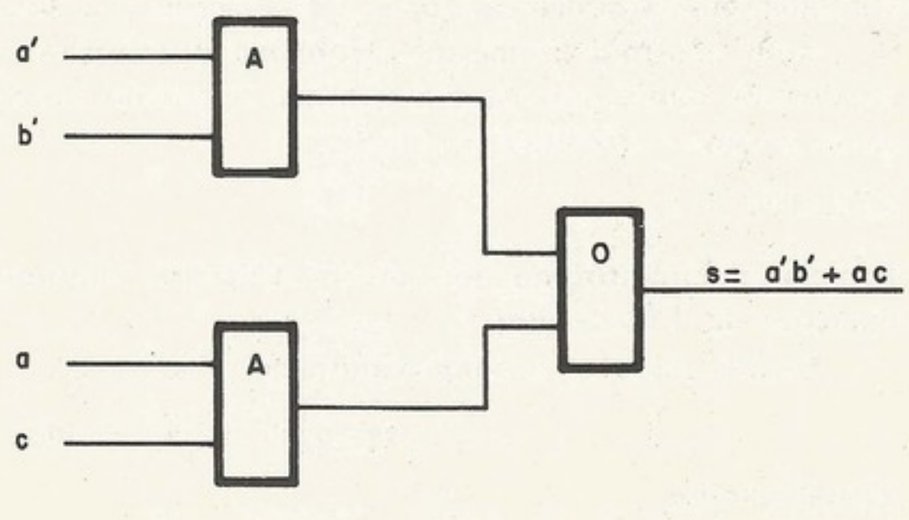


Figura 1-16. Várias implementações do mesmo circuito em dois níveis

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



b) DIAGRAMA LÓGICO

Figura 1-17. Circuito combinatório do Exemplo 1

EXEMPLO 2

Vamos ilustrar o problema de caminhos diferentes levarem a somas irredutíveis 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,

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

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

Segunda maneira. Simplificando por outro caminho,

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

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

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

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.

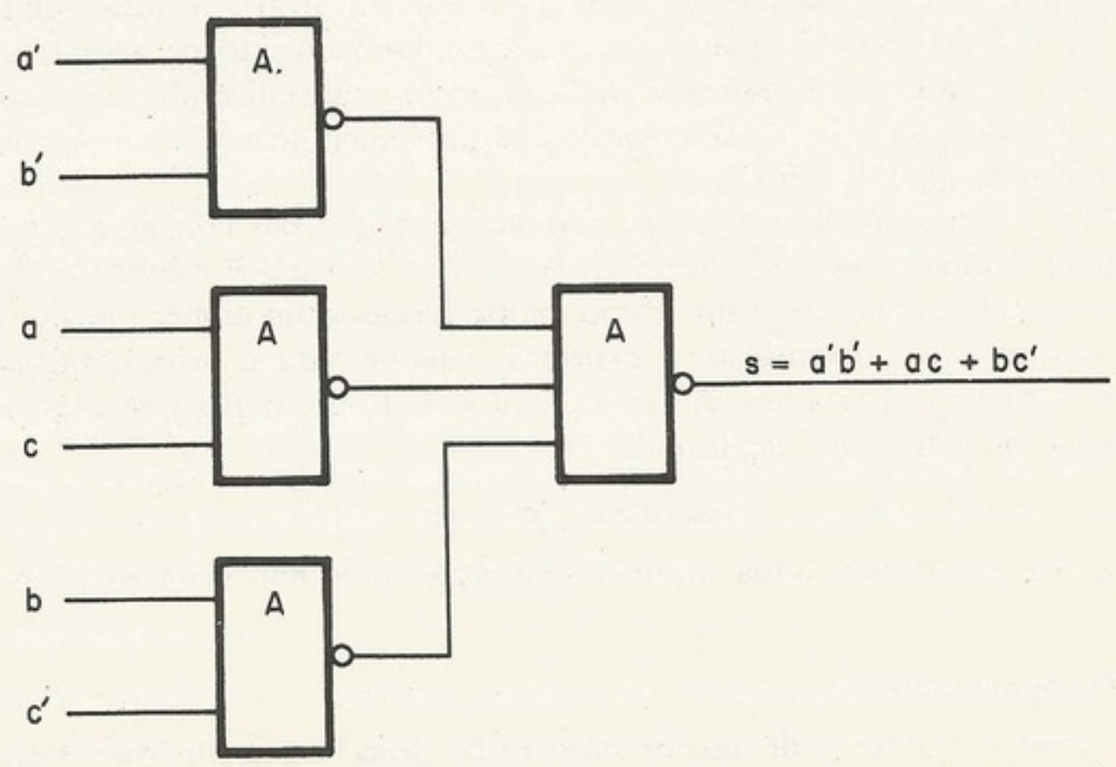


Figura 1-18. Circuito mínimo em dois níveis *NAND-NAND* do Exemplo 2

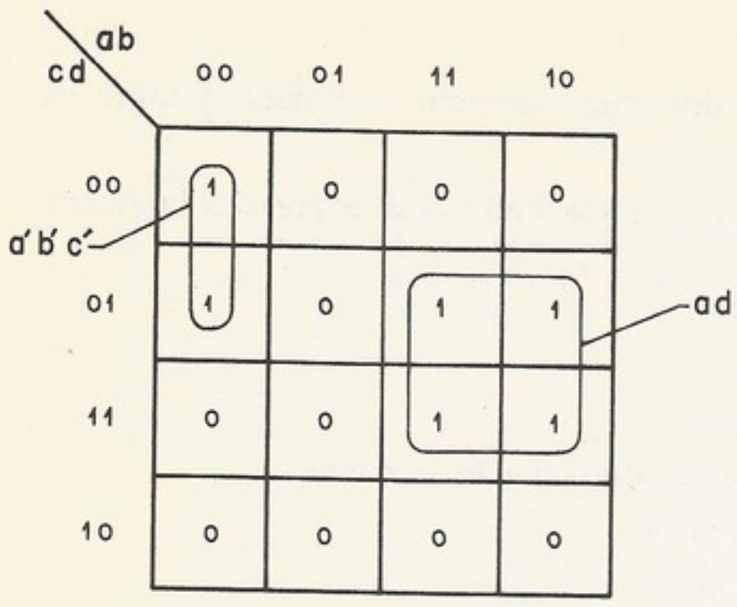


Figura 1-19. *Prime implicant* no mapa de Karnaugh

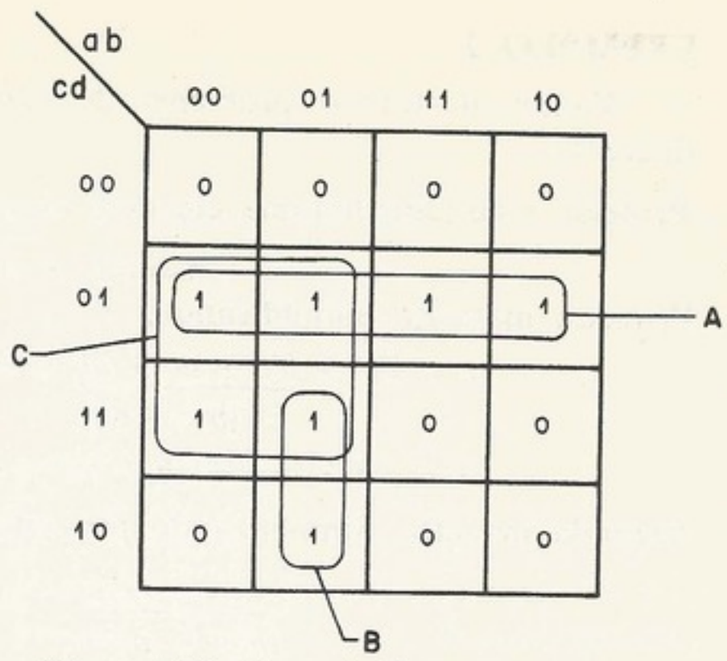


Figura 1-20. Mapa de Karnaugh do Exemplo 3

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):

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

EXEMPLO 3

No mapa de Karnaugh da Fig. 1-20, indicamos três grupos retangulares com 2^n células preenchidas com "1". Portanto essa função tem três *prime implicants*, que escreveremos a seguir.

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

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 circuladas. (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

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

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).

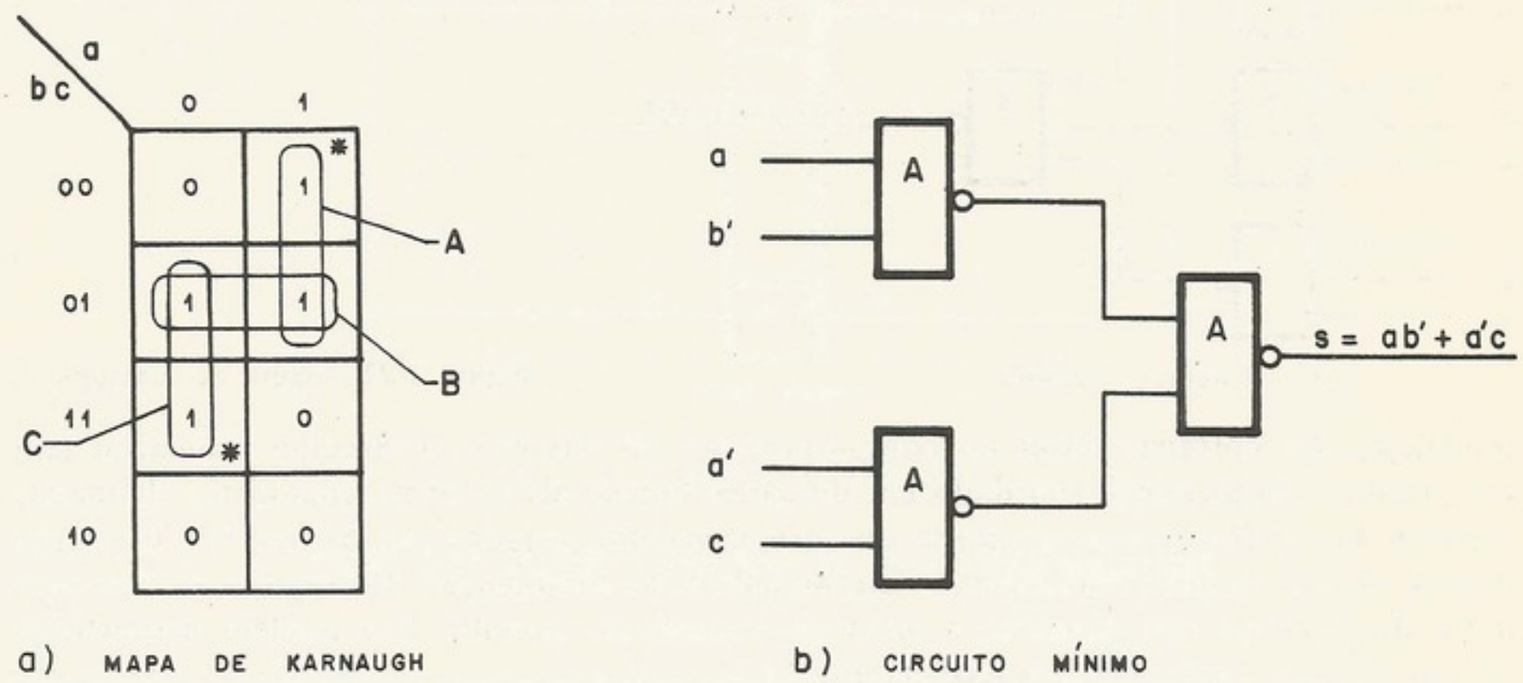


Figura 1-21. Figura do Exemplo 4

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*,

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

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

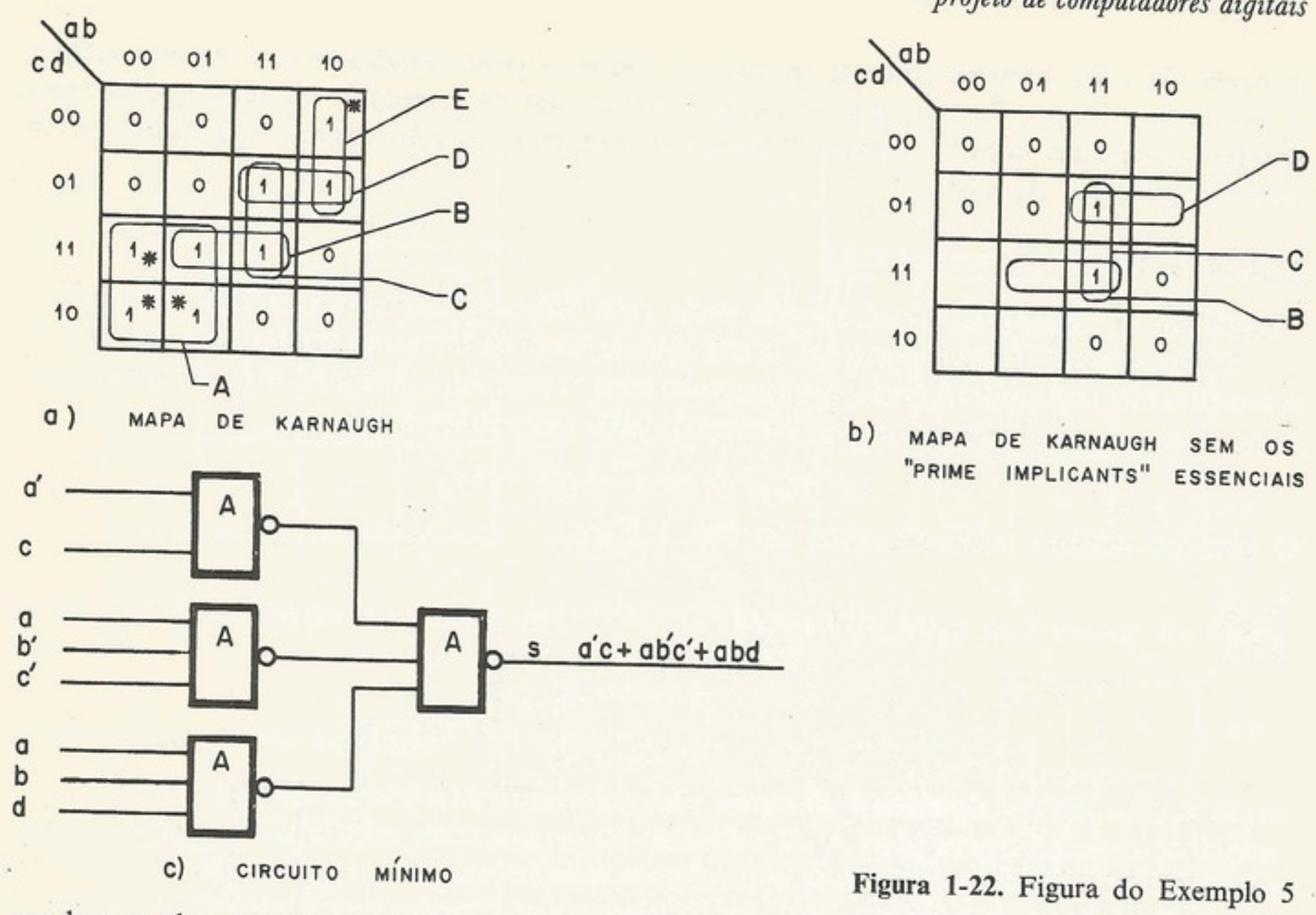


Figura 1-22. Figura do Exemplo 5

mudanças das entradas influi 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 $(Z_1, Z_2, \ldots, Z_m)$, que são as saídas do circuito seqüencial, e o conjunto $(y_1, y_2, \ldots, y_p)$, que determina o estado seguinte do sistema $(Y_1, Y_2, \ldots, Y_p)$. 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

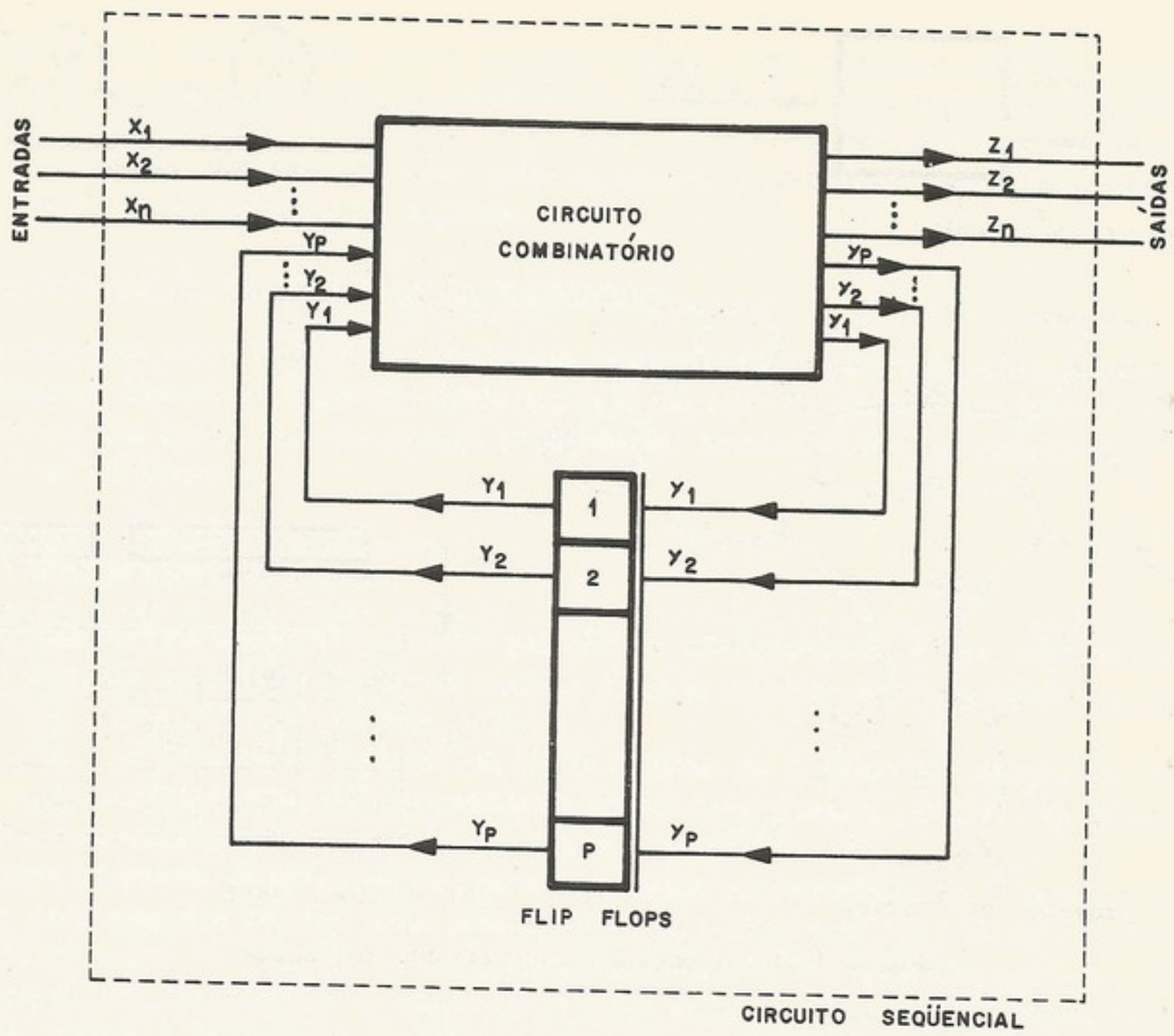


Figura 1-23. Modelo matemático de um circuito seqüencial

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.

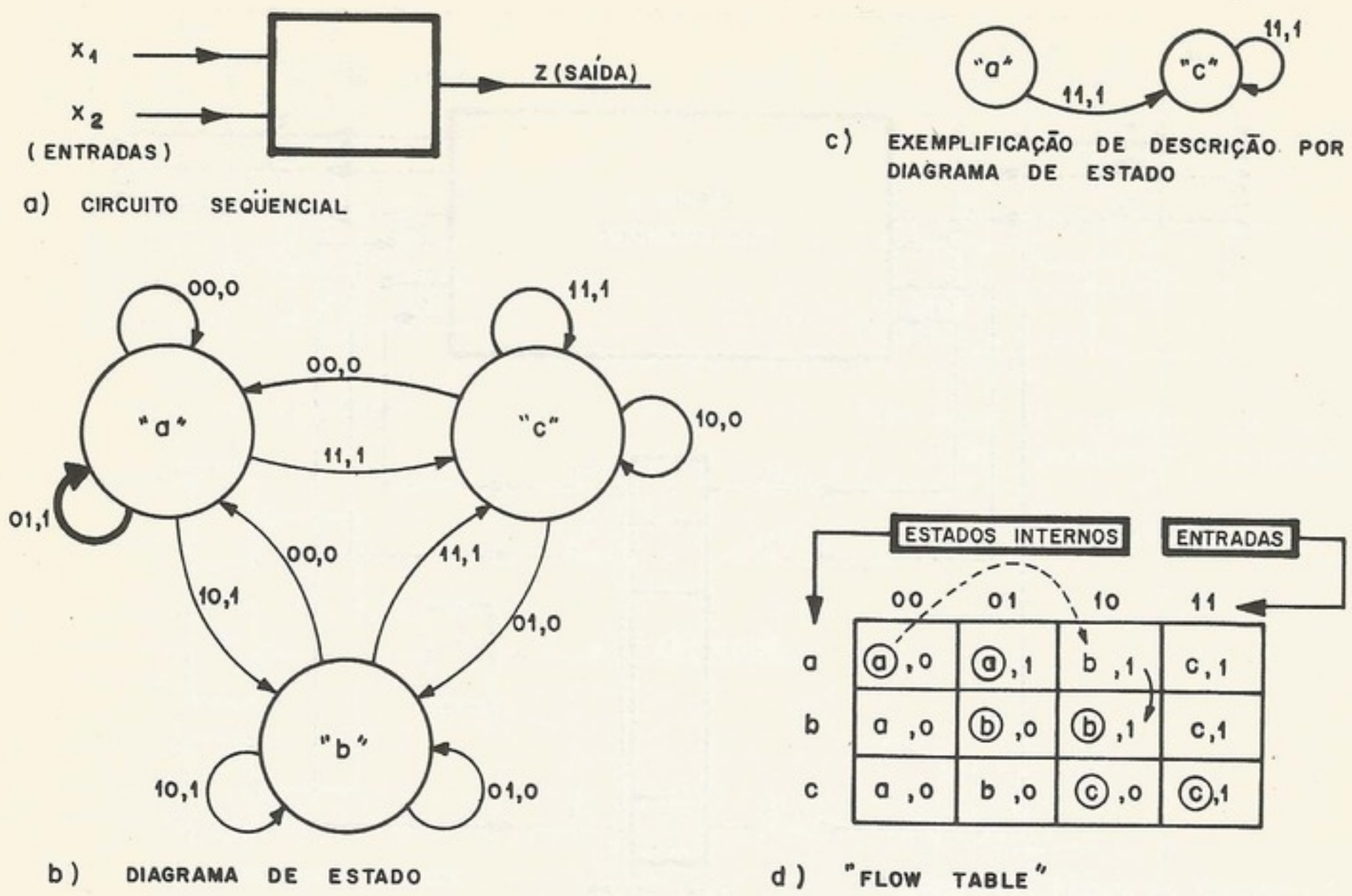


Figura 1-24. Descrições de um circuito seqüencial

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 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 díodos

Um dispositivo usado na primeira geração de computadores, e que continua sendo usado, é o díodo. Junto com resistores, o díodo 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 encadearem-se indefinidamente os *gates AND* e *OR* sem amplificação. Na primeira geração, os circuitos com díodos, 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.

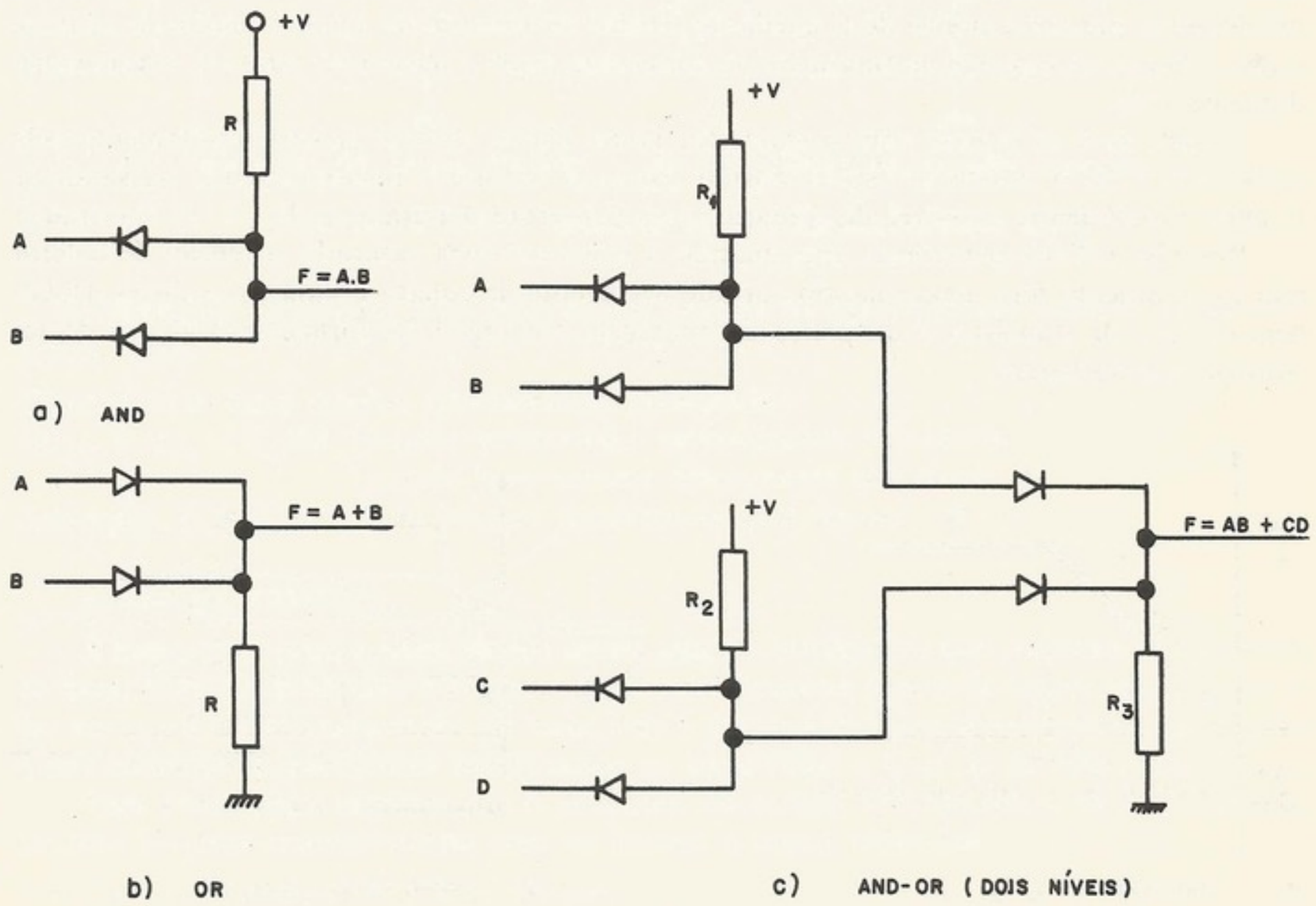


Figura 1-25. Circuitos lógicos com díodos

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.

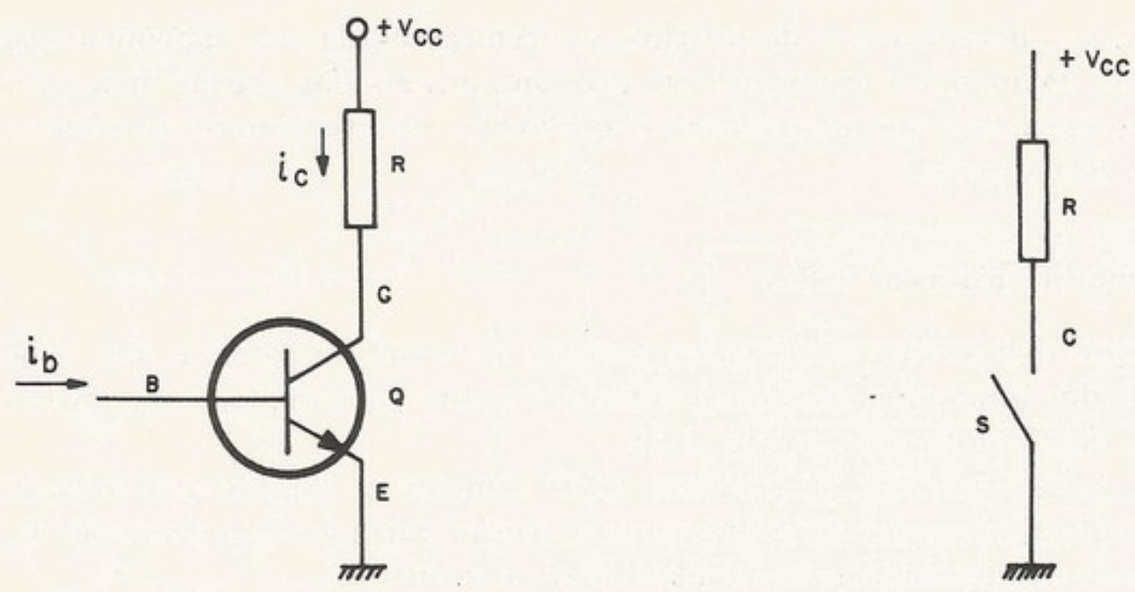


Figura 1-26. O transistor como chave

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 ($i_c = \beta i_b$) 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 $i_c \cdot R = V_{cc}$, 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 $i_b \geqslant V_{cc}/(R \cdot \beta)$ (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.

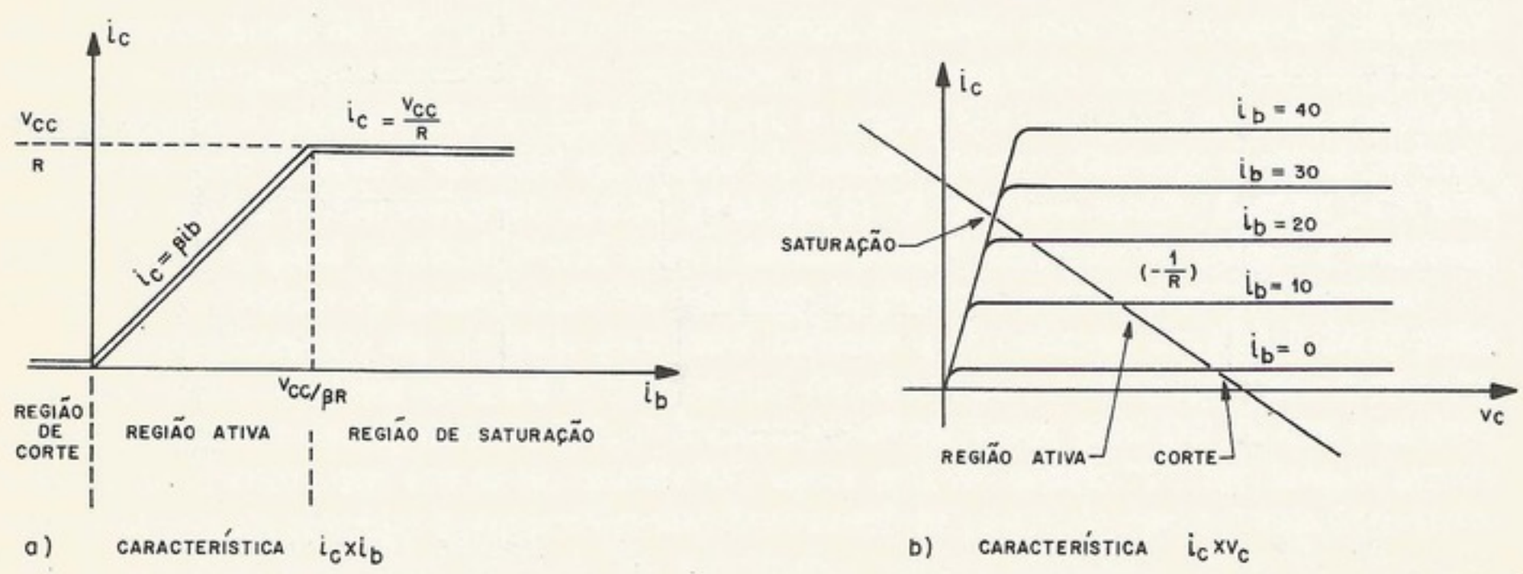


Figura 1-27. Curvas características do transistor como chave

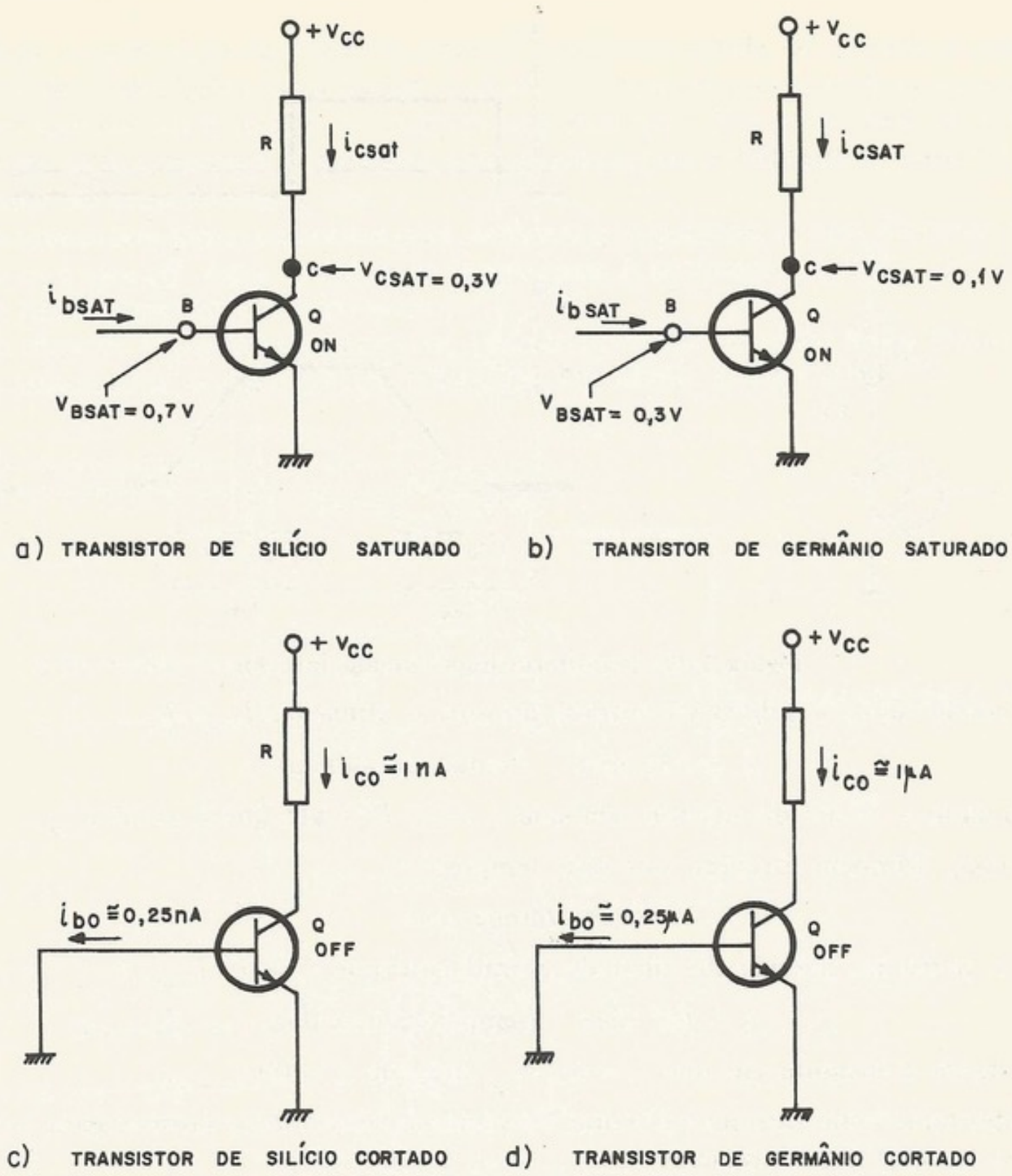


Figura 1-28. Valores característicos dos transistores como chave

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

Corte. $i_b = 0$, duas junções reversamente polarizadas.
Ativa. $0 < i_b < V_{cc}/\beta R$ $\begin{cases}\text{junção coletor-base reversamente polarizada,}\\ \text{junção base-emissor diretamente polarizada.}\end{cases}$
Saturação. $i_b \geqslant V_{cc}/\beta R$, 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,

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

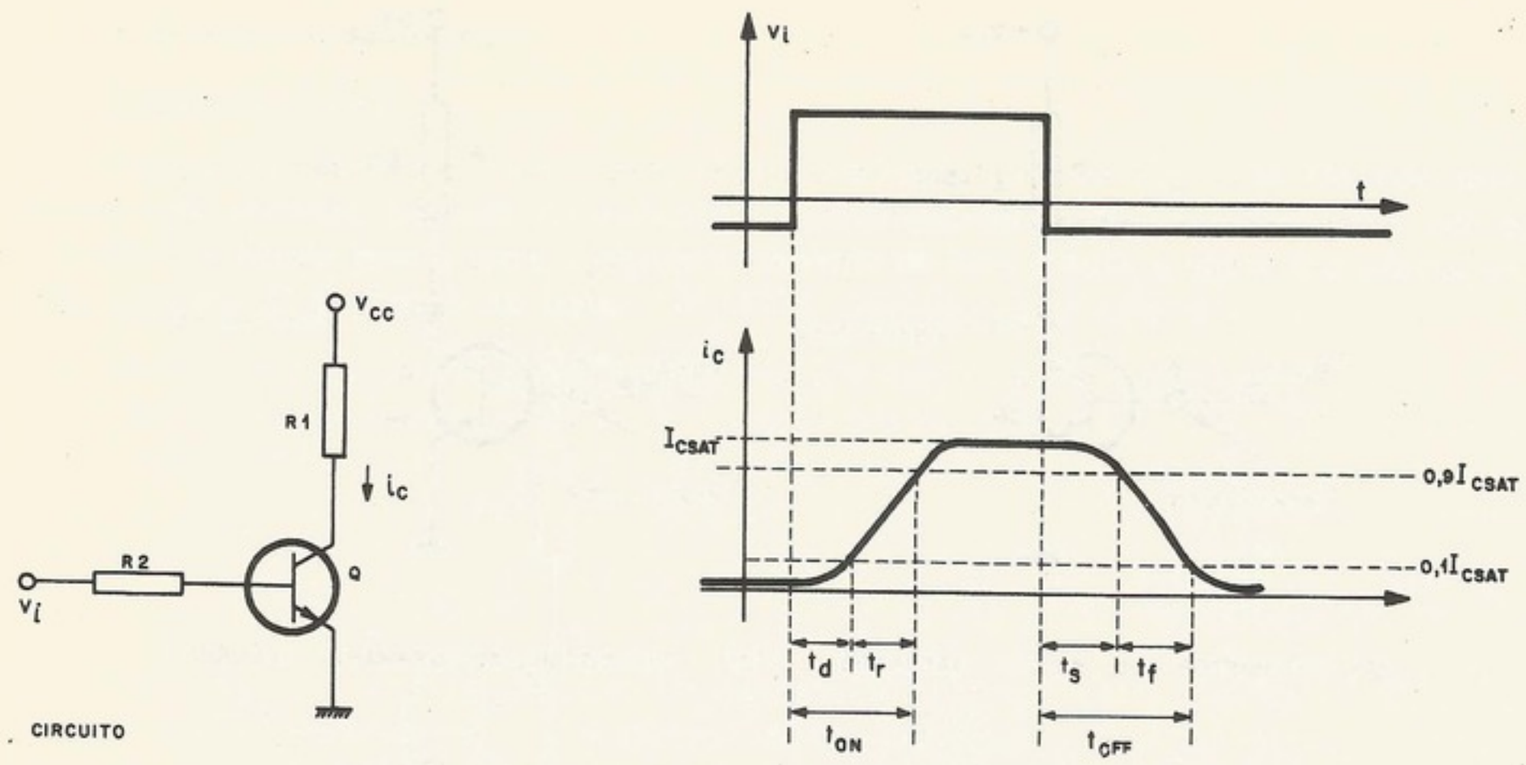


Figura 1-29. Transitório num circuito inversor

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

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

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

Turn-off time. Também formado por dois tempos,

$$t_s = \textit{storage time},$$

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

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

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

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 inversor [Fig. 1-29(a)] na saída de um *AND* ou de um *OR* passivos (veja a Fig. 1-30). O funcionamento do inversor é entendido sem

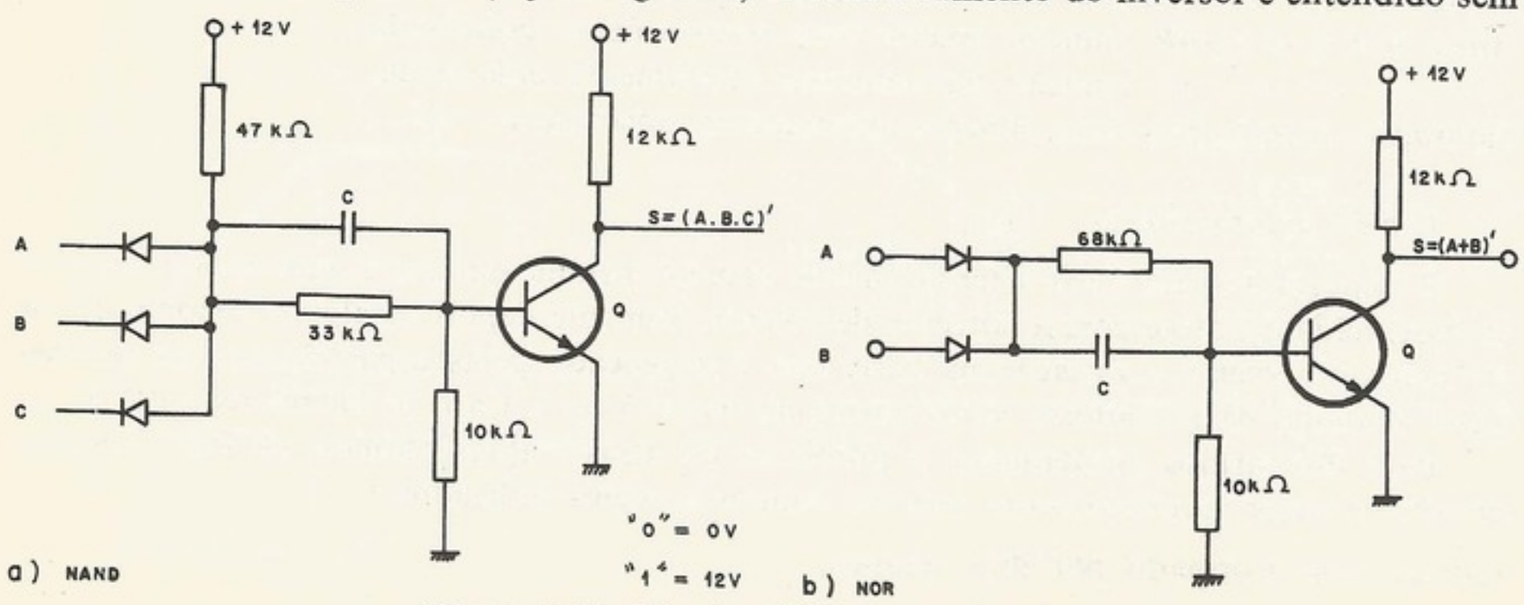


Figura 1-30. Circuitos lógicos transistorizados

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].

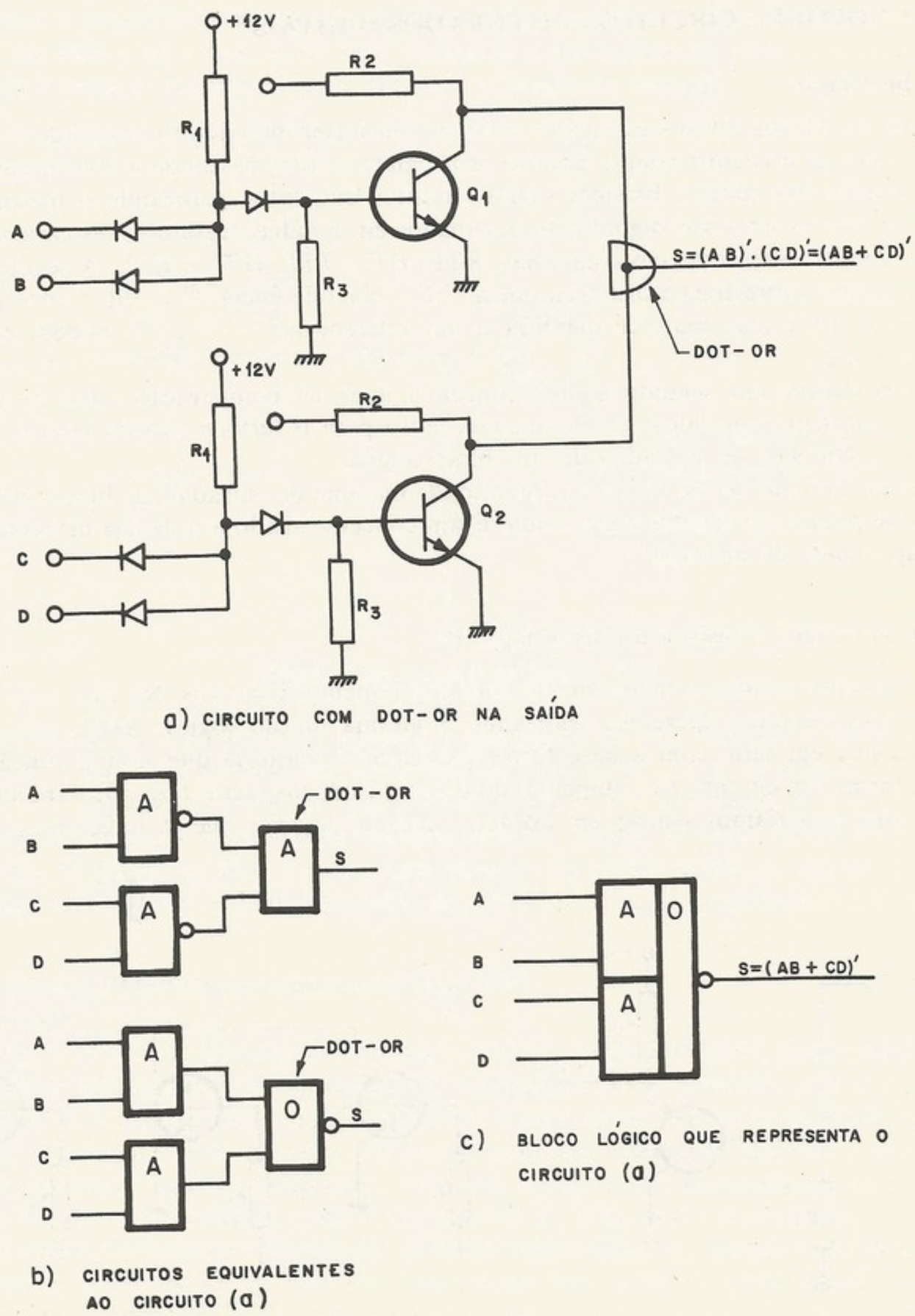


Figura 1-31. O *DOT-OR*

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,

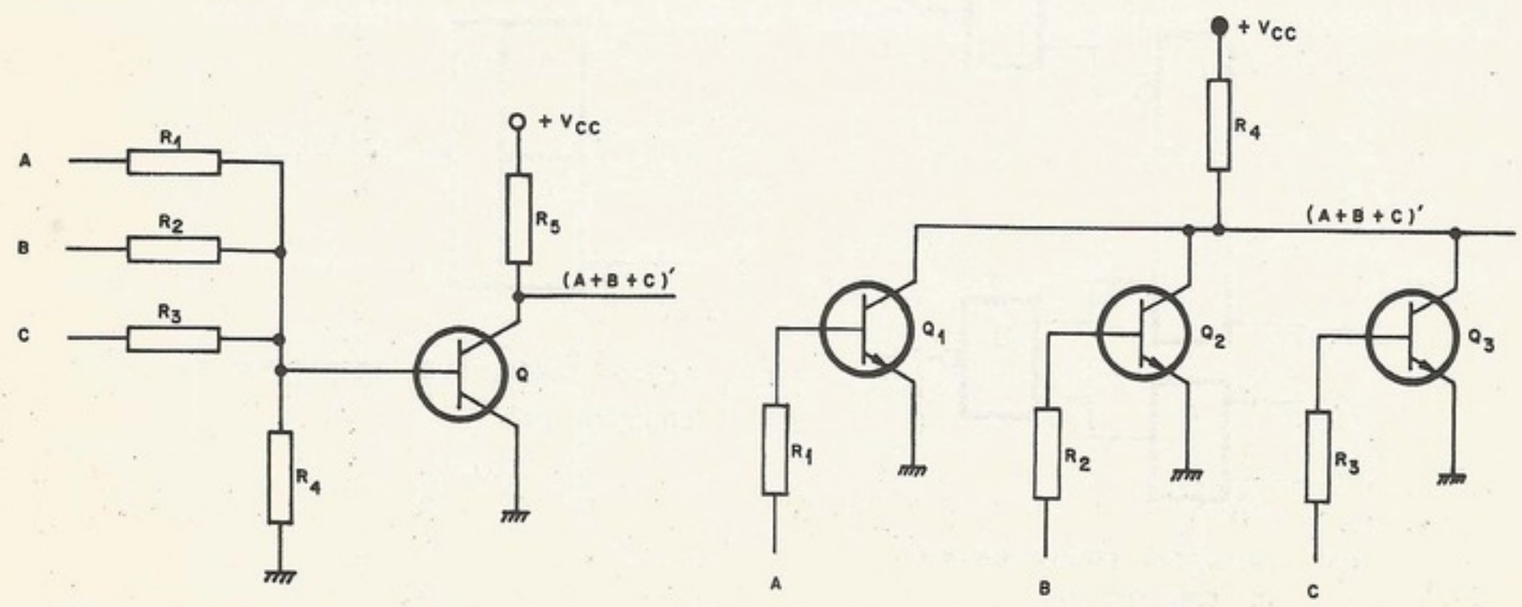


Figura 1-32. Circuitos *RTL*

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 díodos vistos na figura. O díodo 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*.

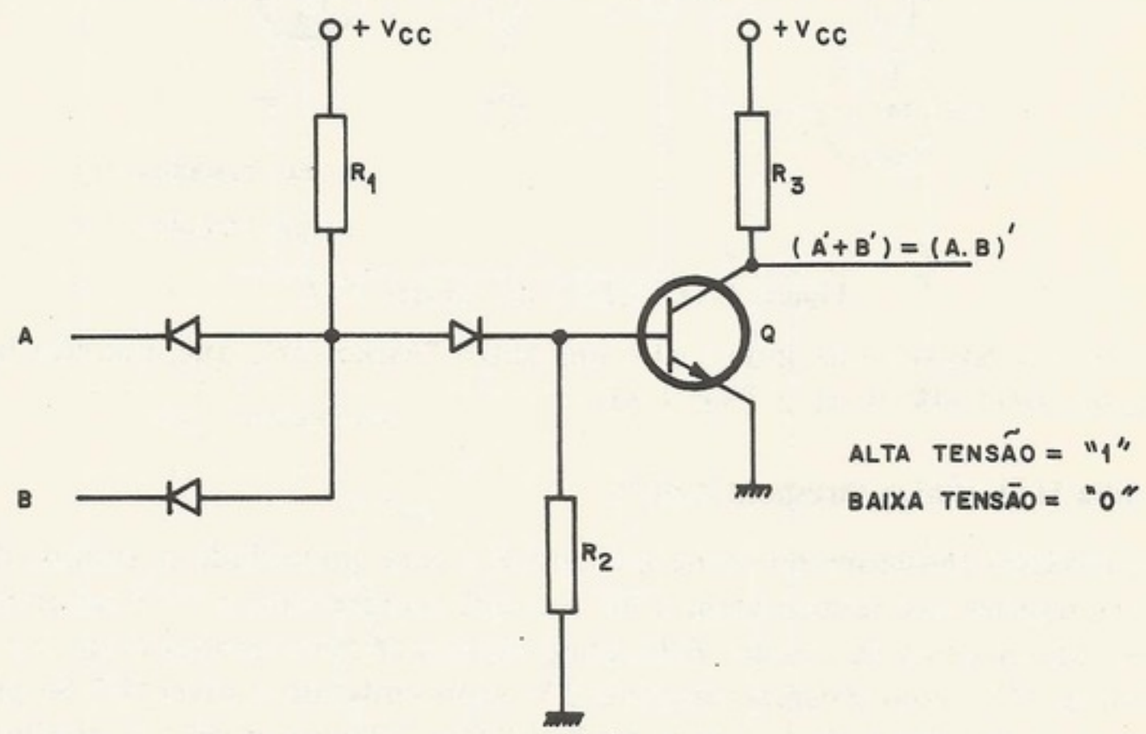


Figura 1-33. *NAND* da família *DTL*

O problema da imunidade ao ruído é o que segue. Se não existisse o díodo 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 díodo 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 ruídos (*glitches*), muitas vezes indesejáveis.

Além de gerar ruídos 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-

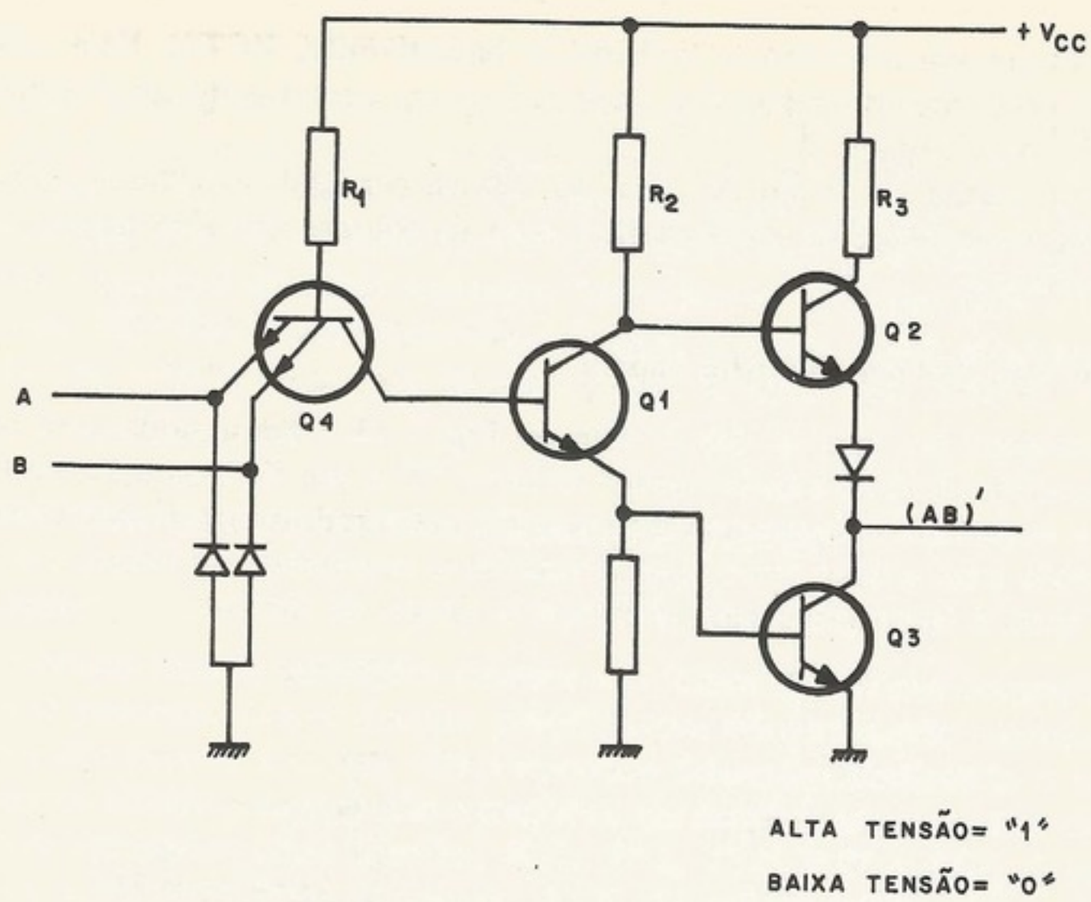


Figura 1-34. *NAND* da família *TTL*

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 díodo 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 $k\Omega$ 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 família, por problemas de acoplamento de impedâncias e níveis.

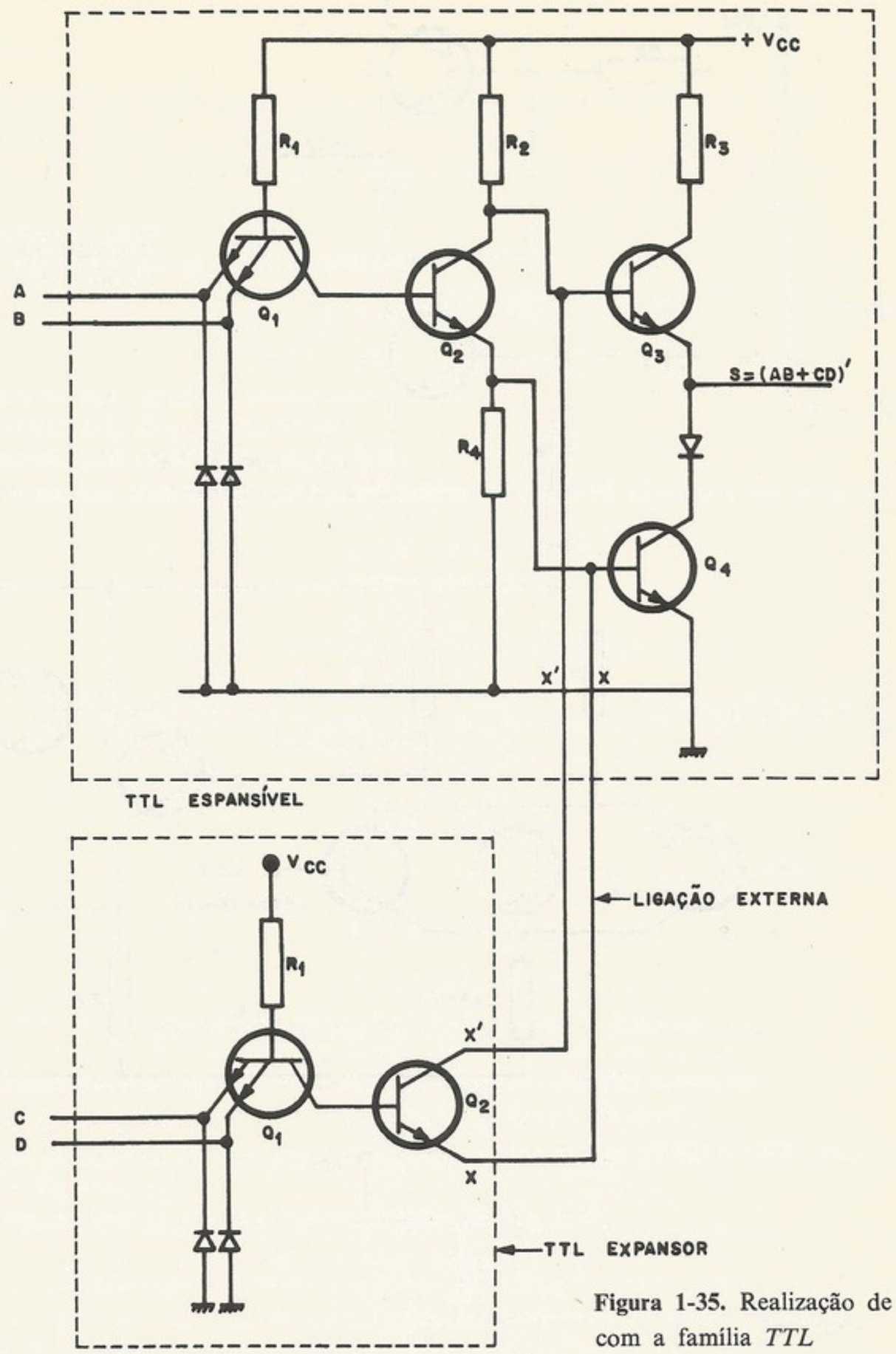


Figura 1-35. Realização de *DOT-OR* com a família *TTL*

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 obtêm-se *gates* tipo *N A N D*. Na figura, o *FET* ligado à fonte funciona como o resistor de carga.

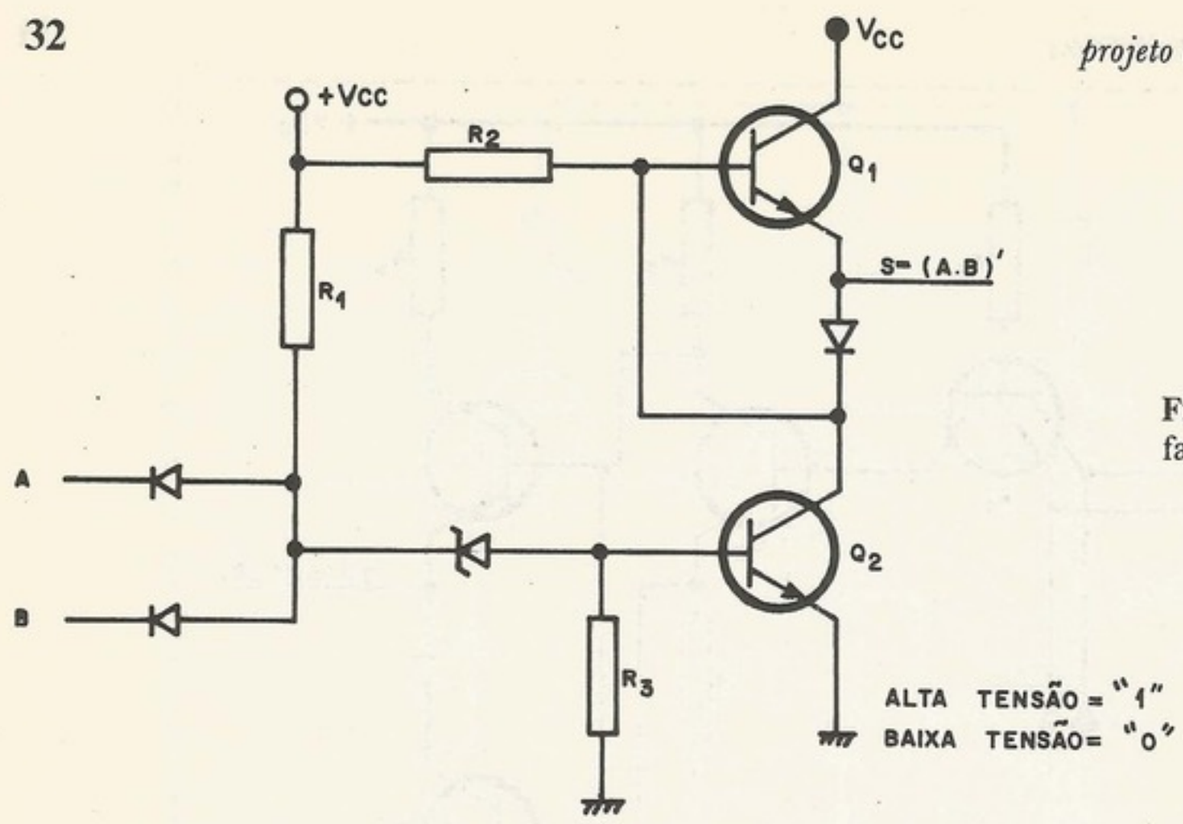


Figura 1-36. *NAND* da família *HTL*

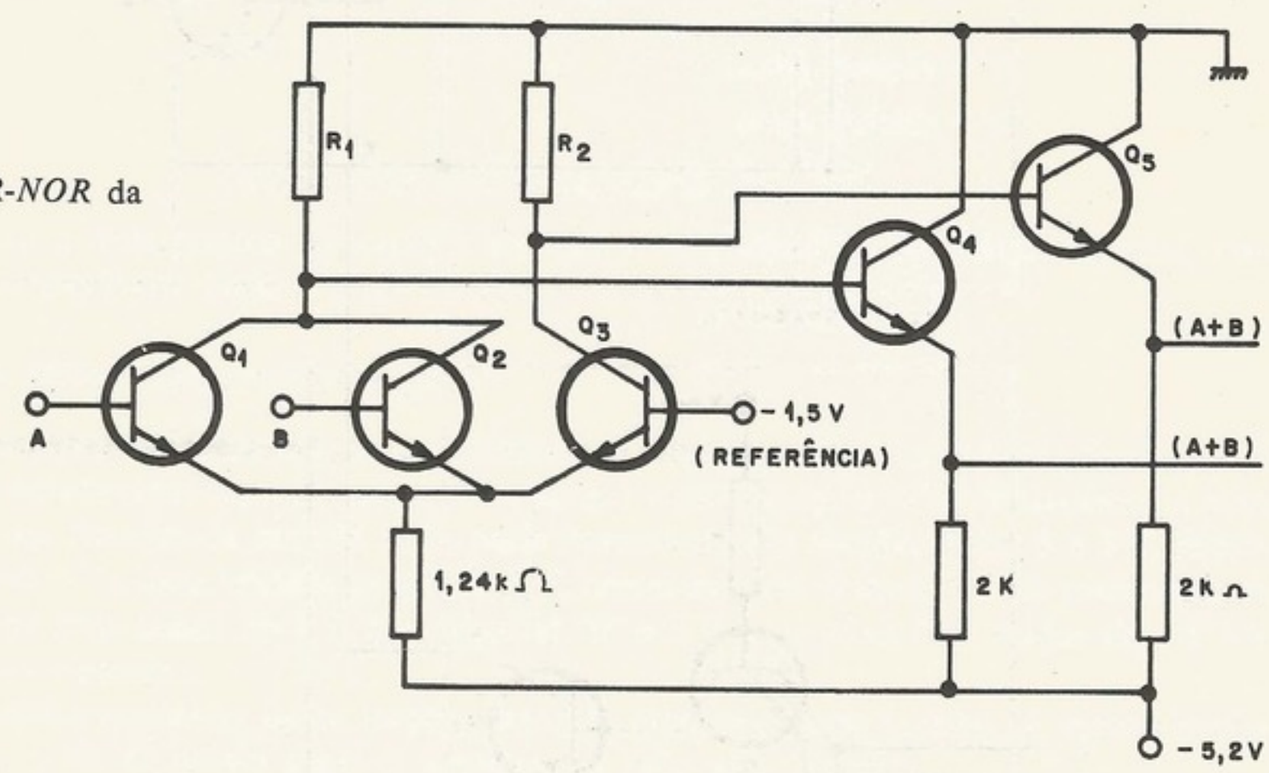
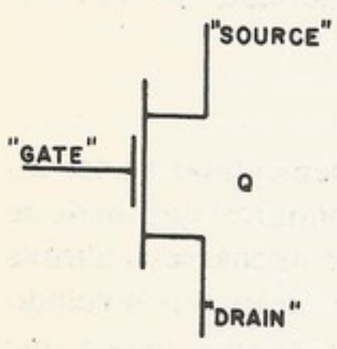
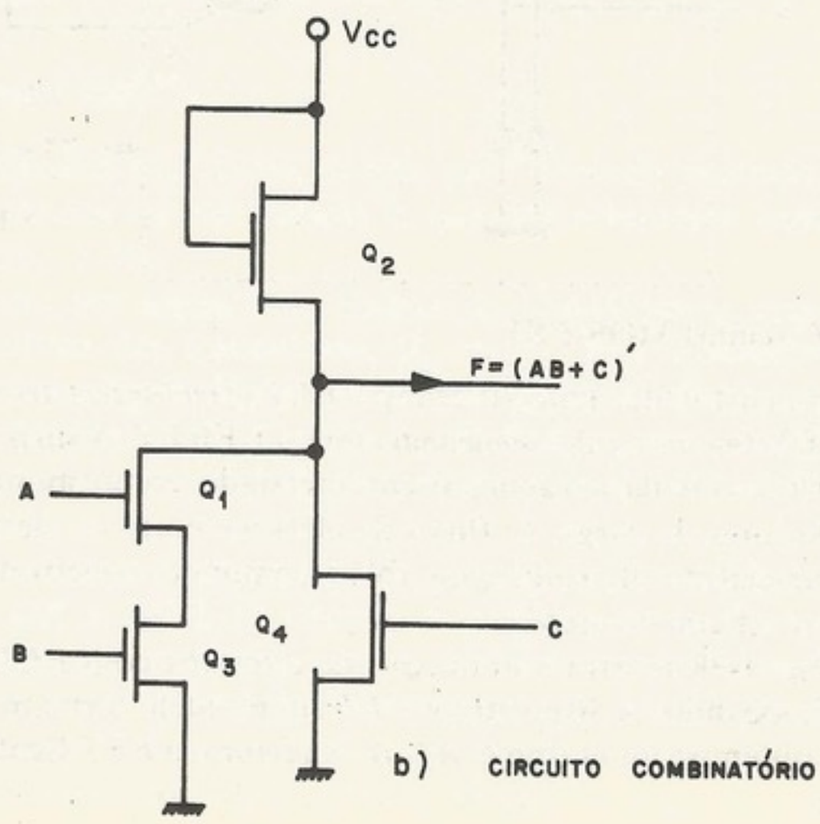


Figura 1-37. *OR-NOR* da família *ECL*



a) MODELO



b) CIRCUITO COMBINATÓRIO

Figura 1-38. *Field effect transistor (FET)*

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

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

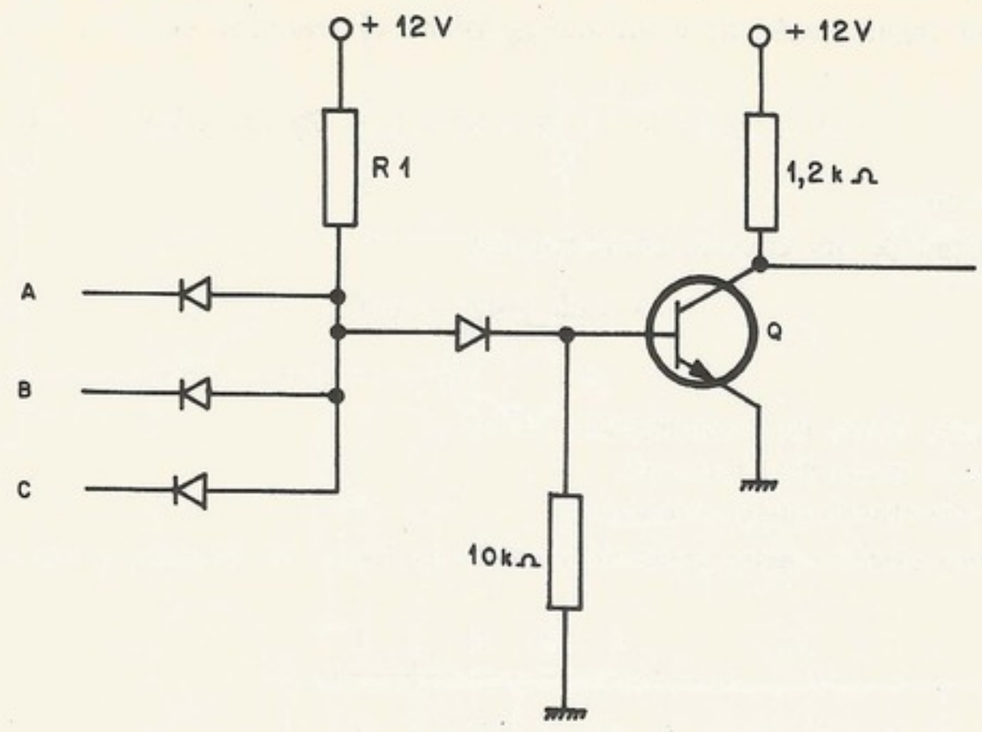


Figura 1-41. Circuito do Exercício 1-8

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, Aplication 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., "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 Sequencial Switching Circuits*, John Wiley Interscience, N.Y., 1969

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 1*K* (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 início do projeto, tentou-se utilizar no máximo 1*K* (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 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 4*K 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-

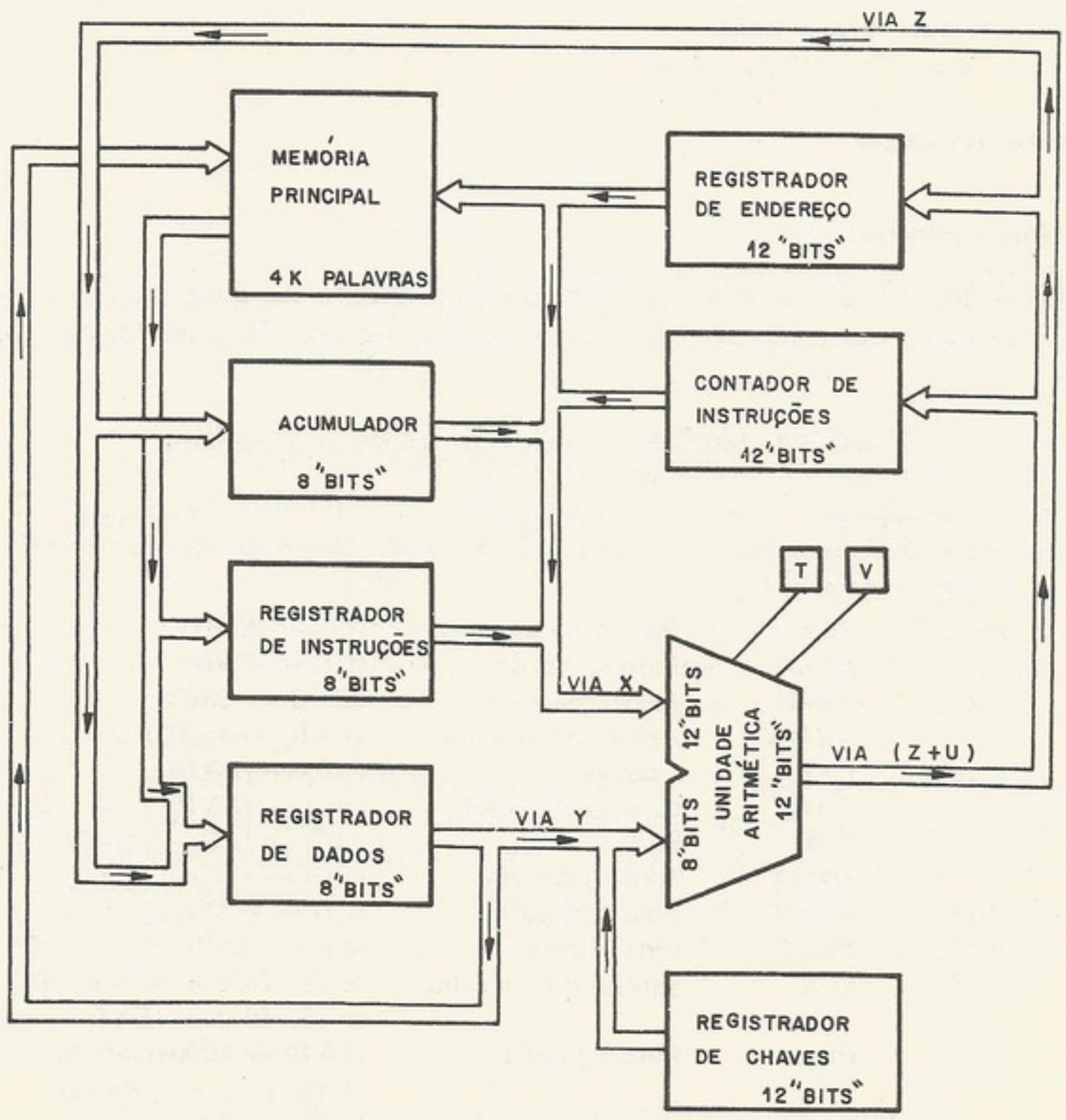


Figura 5-1. Fluxo de dados simplificado

cuções, 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 4*K* 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	$(END) \leftarrow 0000\,(CI(0\text{-}3))$ $(END + 1) \leftarrow (CI(4\text{-}11))$ $(CI) \leftarrow (END) + 2$

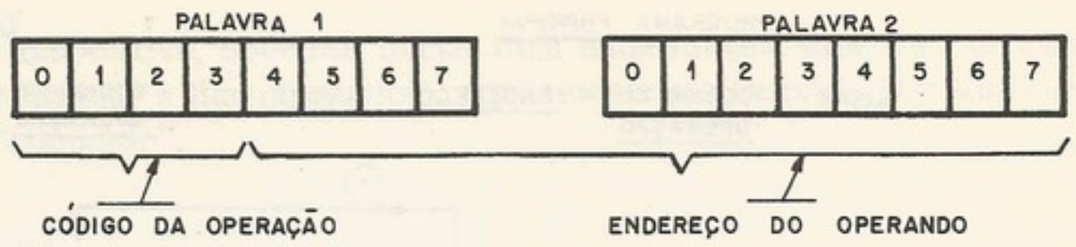


Figura 5-2. Formato das instruções longas

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.

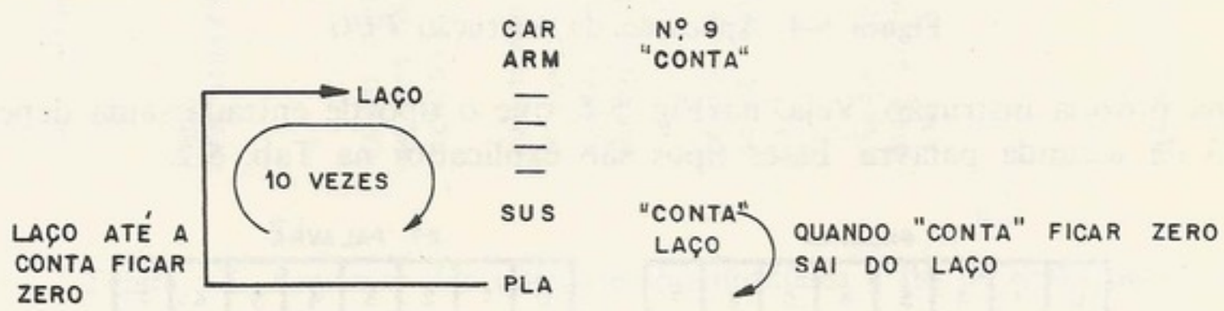


Figura 5-3. Exemplo da aplicação de instrução *SUS*

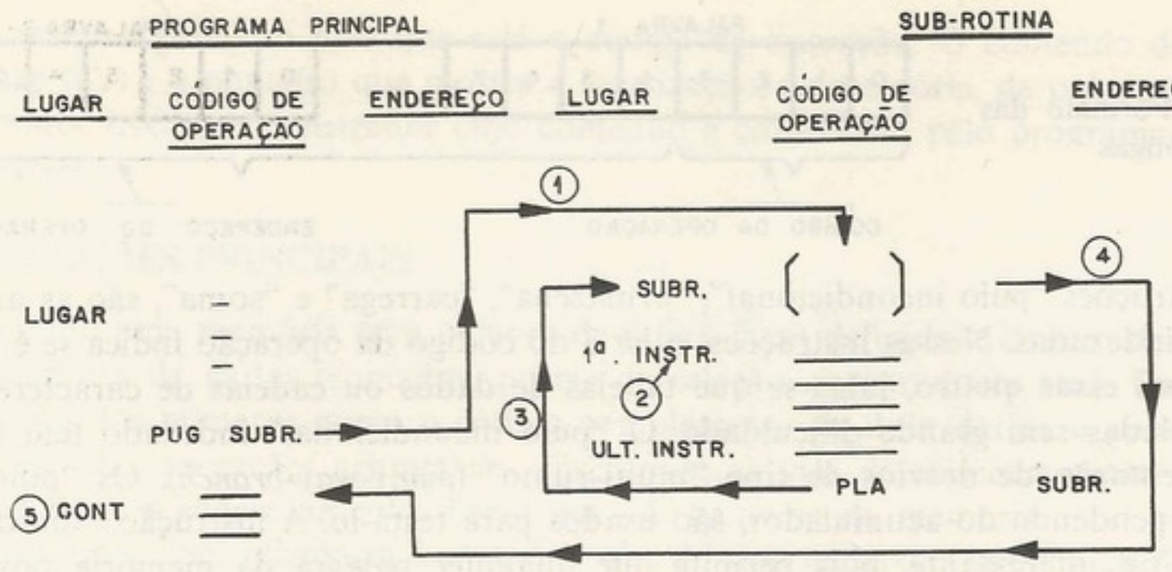
A instrução de "pula e guarda" (*PUG*) é um desvio que permite o uso de técnica de sub-rotinas. Isso porque o *PUG* deixa uma indicação do ponto de saída, para permitir regresso a ele. Vamos dizer algumas palavras sobre essa técnica.

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? Aí 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 dezesseis dispositivos de entrada/saída ende-



1. *PUG*: o *CI* que indica *CONT* está guardado no início 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.

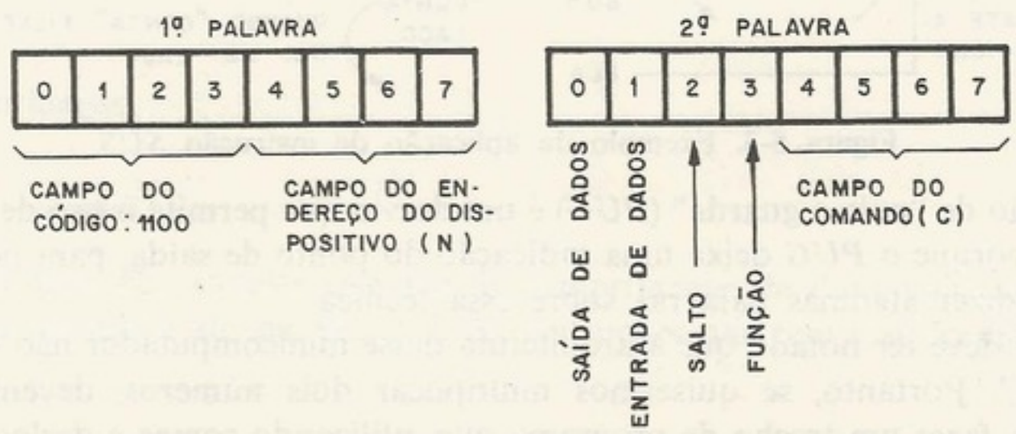


Figura 5-5. Formato das instruções longas do grupo entrada-saída

Tabela 5-2. Instruções do grupo entrada e saída

Mnemônico		Descrição
Inicial	Definitivo	
SAEDS	*SAI, n, c*	Saída de dados, comando *c* para dispositivo *n*
SAEDE	*ENT, n, c*	Entrada de dados, comando *c* do dispositivo *n*
SAESA	*SAL, n, c*	Salto, tipo *c* para o dispositivo *n*
SAEFU	*FNC, n, c*	Função tipo *c* para o dispositivo *n*

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 dispositivos, 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.

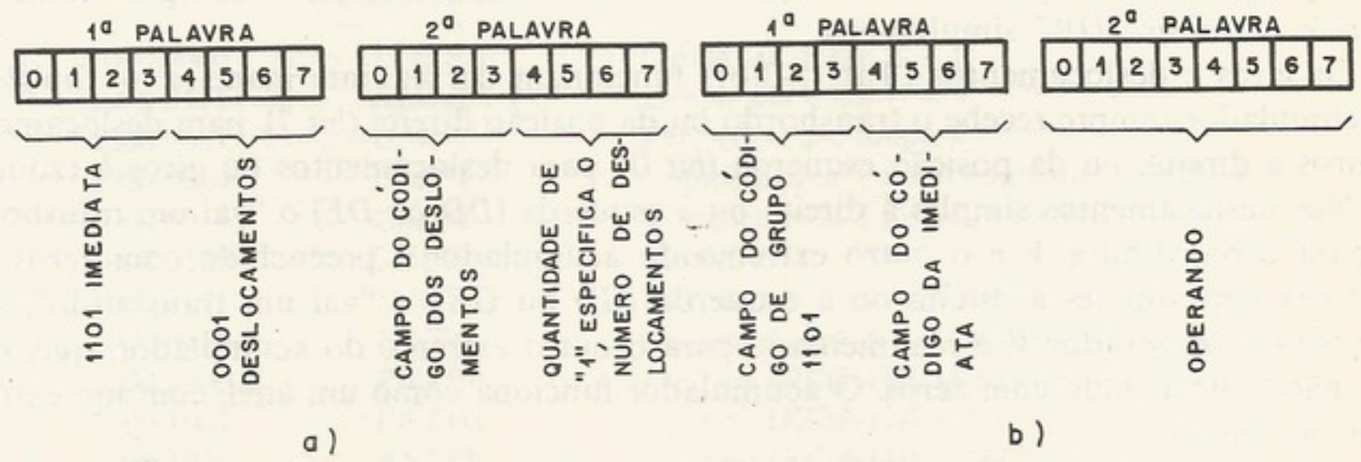


Figura 5-6. Formatos das instruções (a) imediatas e (b) de deslocamentos

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*	*NAND* imediato	$(AC) \leftarrow ((AC) \wedge \text{Operando})'$
0010	*IMEXOR*	*XOR*	*Exclusive-OR* 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 (*MICG*1)

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

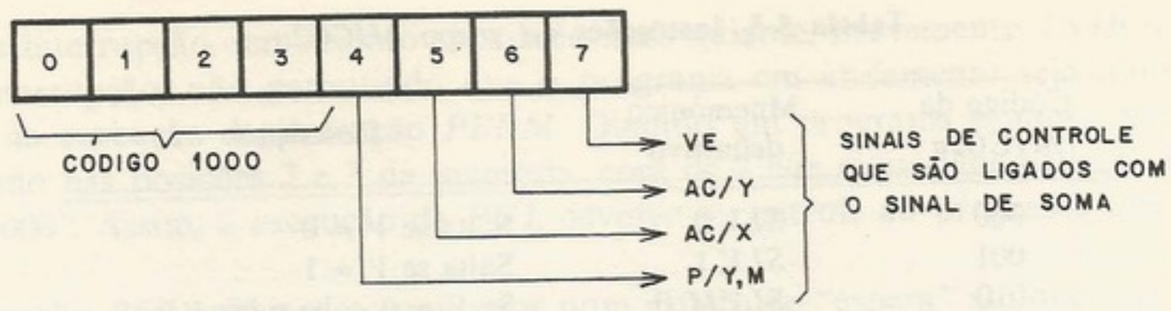


Figura 5-7. Formato da instrução curta, grupo 1

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	*CMP*1	Complementa 1 no acumulador e limpa V
1000 0011	*CMP*2	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)	$(AC) \leftarrow (RC(4\text{-}11)), (V) \leftarrow 0$
1000 1001	*PNL*(1)	$(AC) \leftarrow (RC(4\text{-}11)) + 1$
1000 1010	*PNL*(2)	$(AC) \leftarrow (RC(4\text{-}11)) - (AC) - 1$
1000 1011	*PNL*(3)	$(AC) \leftarrow (RC(4\text{-}11)) - (AC)$
1000 1100	*PNL*(4)	$(AC) \leftarrow (RC(4\text{-}11)) + (AC)$
1000 1101	*PNL*(5)	$(AC) \leftarrow (RC(4\text{-}11)) + (AC) + 1$
1000 1110	*PNL*(6)	$(AC) \leftarrow (RC(4\text{-}11)) - 1$
1000 1111	*PNL*(7)	$(AC) \leftarrow (RC(4\text{-}11)), (V) \leftarrow 1$

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.

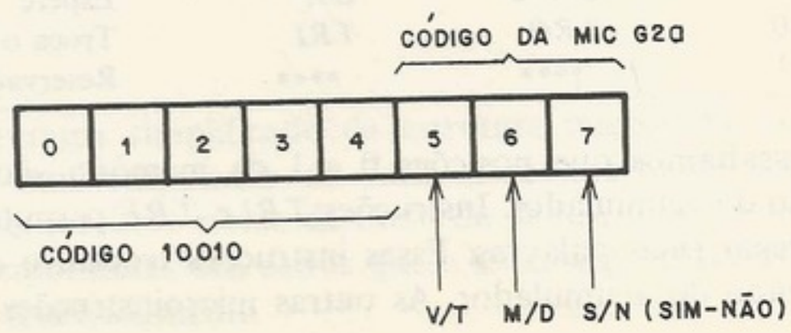


Figura 5-8. O formato das microinstruções, grupo 2

Tabela 5-5. Instruções do grupo *MICG2a*

Código da *MICG2a*	Mnemônico definitivo	Descrição
000	*SLV*, 0	Salta se $V = 0$
001	*SLV*, 1	Salta se $V = 1$
010	*SLVM*, 0	Se $V = 0$, salta e faz $V = 1$
011	*SLVM*, 1	Se $V = 1$, salta e faz $V = 0$
100	*SLT*, 0	Salta se $T = 0$
101	*SLT*, 1	Salta se $T = 1$
110	*SLTM*, 0	Se $T = 0$, salta e faz $T = 1$
111	*SLTM*, 1	Se $T = 1$, salta e faz $T = 0$

As microinstruções do grupo 2*a* 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* 2*b* (*MICG2b*)

O grupo 2*b* consta dos comandos especiais. O formato é visto na Fig. 5-9 e a descrição dos subcódigos é mostrada na Tab. 5-6.

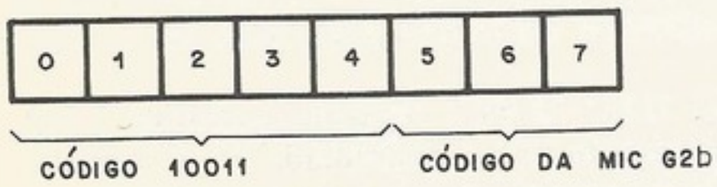


Figura 5-9. Formato das microinstruções grupo 2*b*

Tabela 5-6. Instruções do grupo *MICG2b*

Código da *MICG2b*	Mnemônico		Descrição
	Inicial	Definitivo	
000	*PUL*	*PUL*	Pula para a posição 2 e limpa interrupção
001	*TRE*	*TRE*	Troca o acumulador com extensão
010	*INI*	*INIB*	Inibe interrupção
011	*PER*	*PERM*	Permite interrupção
100	*PAR-D*	*PARE*	Pare
101	*ESP-D*	*ESP*	Espere
110	*TRO*	*TRI*	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 2*b* são vinculadas com o

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
SOM	Soma	*GD*	Giro à direita
SOMX	Soma indexado	*GE*	Giro à esquerda
SOMI	Soma imediato	*GDT*	Giro à direita com *T*
ARM	Armazena	*GET*	Giro à esquerda com *T*
ARMX	Armazena indexado	*SLT*	Salta se *T* igual operando
CAR	Carrega	*SLO*	Salta se *OVF* igual operando
CARX	Carrega indexado	*SLTM*	Salta se *T* = operando e muda *T*
CARI	Carrega imediato	*SLOM*	Salta se *OVF* = operando/muda *OVF*
PUG	Pula e guarda	*LIMP*	Limpa *AC* e faz *T* = operando
SUS	Subtrai um ou salta	*CMP*1	Comp. de 1 o *AC* e limpa *T*
NAND	*NAND*	*CMP*2	Complementa de 2 o *AC*
XOR	*Exclusive-OR*	*INC*	Incrementa *AC*
PLA	Pulo incondicional	*UM*	Faz *AC* = 1 e limpa *T*
PLAX	Pulo indexado	*UNEG*	Faz *AC* = –1 e limpa *T*
PLAN	Pula se negativo	*LIMT*	Limpa *T*
PLAZ	Pula se zero	*PNL*(0)	(*AC*) ← (*RC*(4-11)), (*T*) ← 0
TRE	Troca com extensão	*PNL*(1)	(*AC*) ← (*RC*(4-11)) + 1
TRI	Troca com índice	*PNL*(2)	(*AC*) ← (*RC*(4-11)) – (*AC*) – 1
PUL	Pula e limpa	*PNL*(3)	(*AC*) ← (*RC*(4-11)) – (*AC*)
PARE	Pare	*PNL*(4)	(*AC*) ← (*RC*(4-11)) + (*AC*)
ESP	Espere	*PNL*(5)	(*AC*) ← (*RC*(4-11)) + (*AC*) + 1
INIB	Inibe interrupção	*PNL*(6)	(*AC*) ← (*RC*(4-11)) – 1
PERM	Permite interrupção	*PNL*(7)	(*AC*) ← (*RC*(4-11)), (*T*) ← 1
DDS	Desloca à direita c/dupl. sinal	*SAI*, *n*, *c*	Saída de dados p/dispositivo *n*
DD	Desloca à direita	*ENT*, *n*, *c*	Entr. de dados p/dispositivo *n*
DE	Desloca à esquerda	*SAL*, *n*, *c*	Salto tipo *I* p/dispositivo *n*
DDT	Desloca à direita com *T*	*FNC*, *n*, *c*	Função tipo *I* p/dispositivo *n*
DET	Desloca à esquerda com *T*		

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.

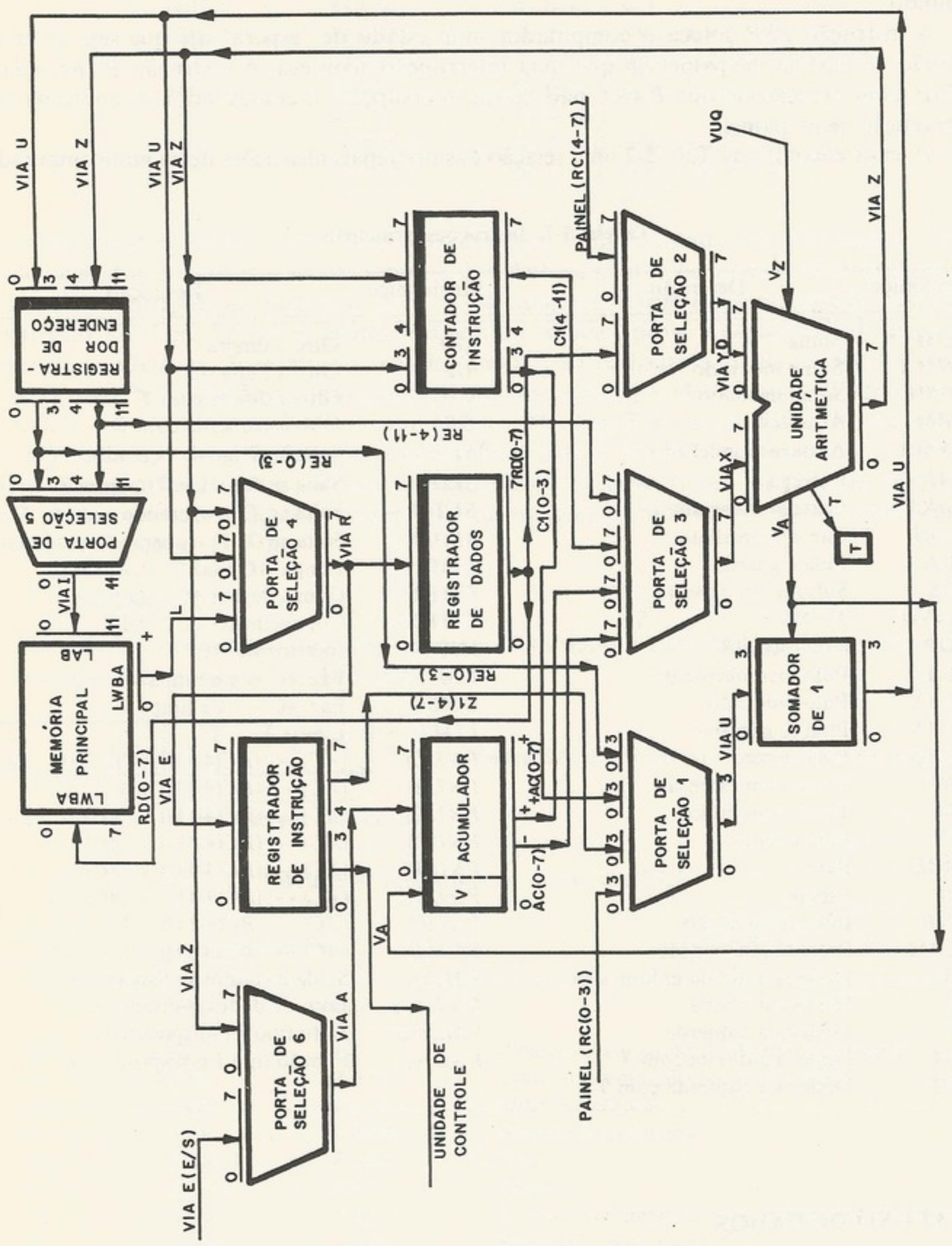


Figura 5-10. Fluxo de dados mais detalhado

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 μs; tempo de acesso, 0,4 μs.

Ciclo rachado

"Ler" (*M*1): duração total, 0,6 μs; tempo de acesso, 0,4 μs.
"Escrever" (*M*3): duração total, 1,0 μs.

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.

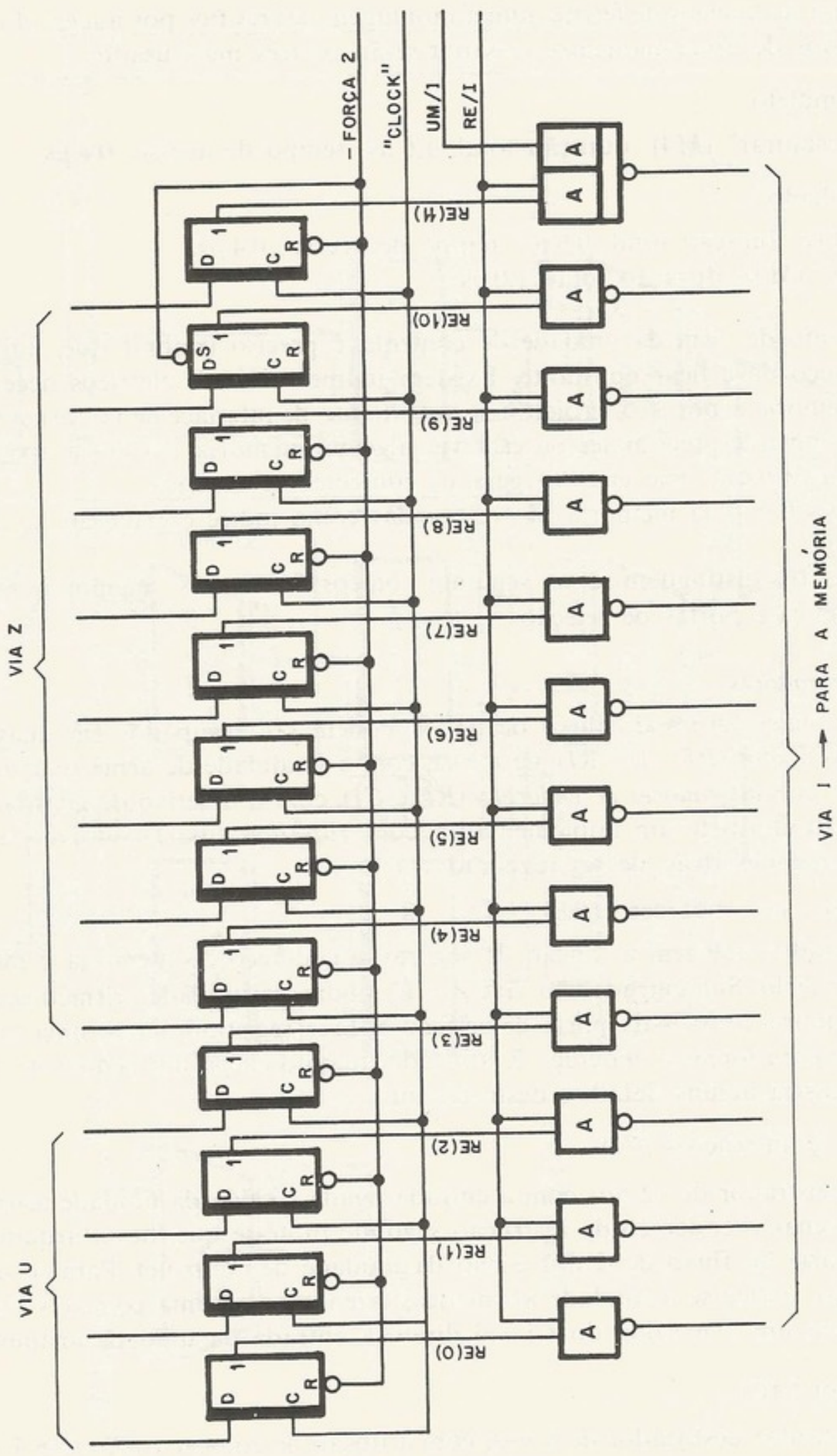


Figura 5-11. Registrador de endereço e porta de seleção 5

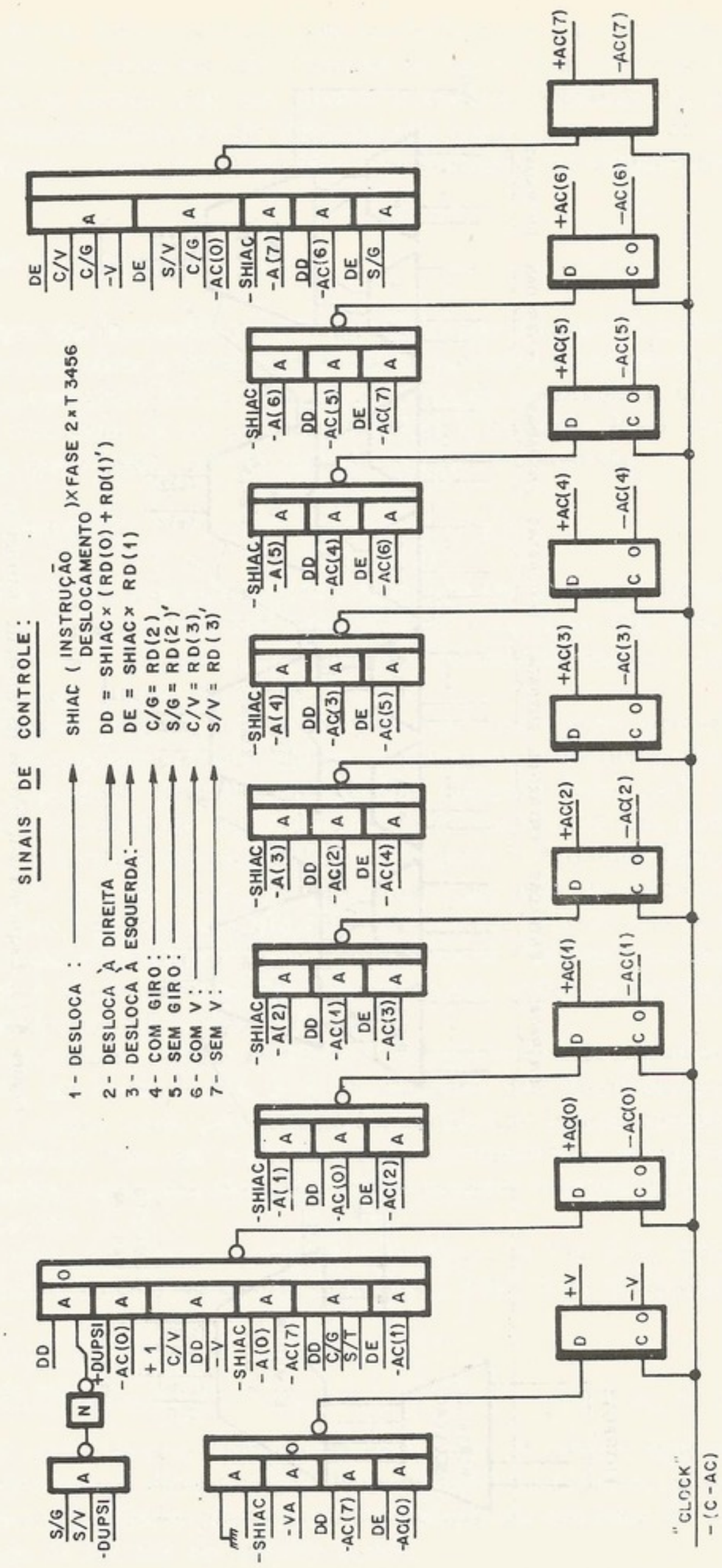


Figura 5-12. Circuito do acumulador

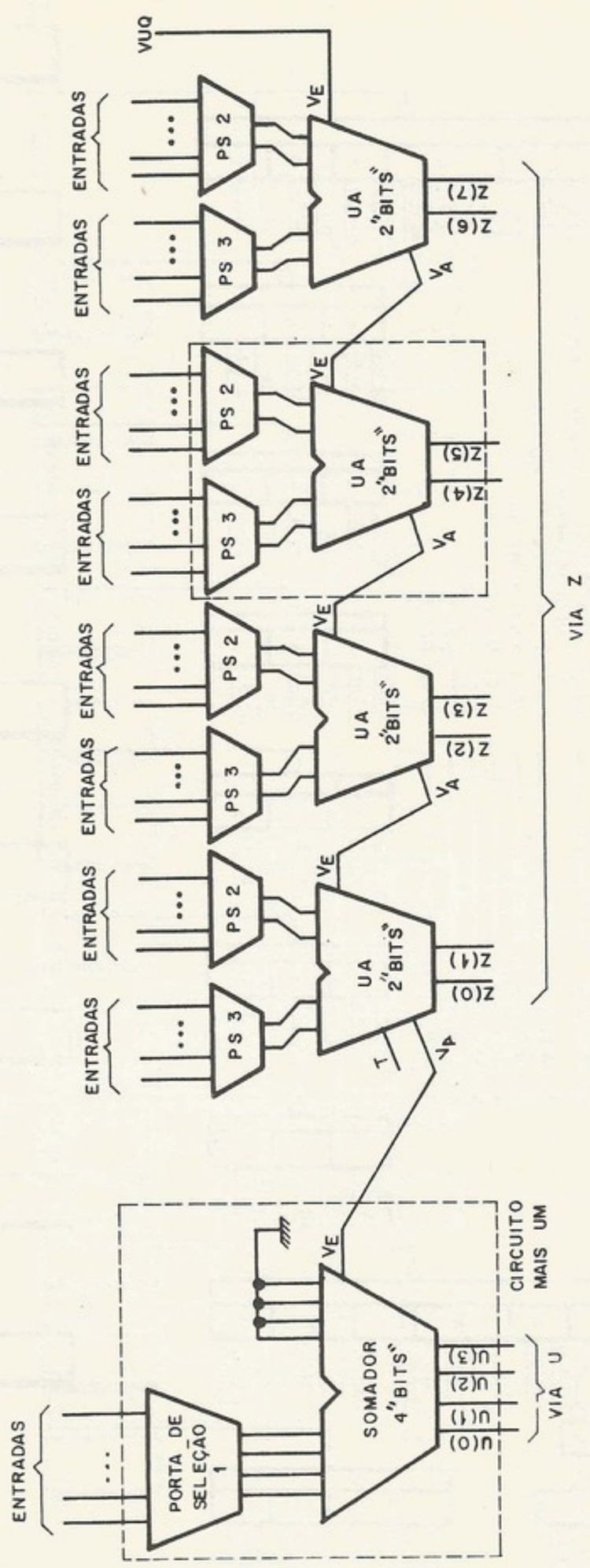


Figura 5-13. Esquema em blocos da unidade aritmética

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. E 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 *E*1/*S* e *E*2/*S* no nível "zero", a saída (*S*) é nula. Quando *E*1/*S* fica igual a "um", a via de entrada *E*1 é colocada na via de saída (*S*). Com *E*2 é análogo. Nessa arquitetura, existem seis portas de seleção.

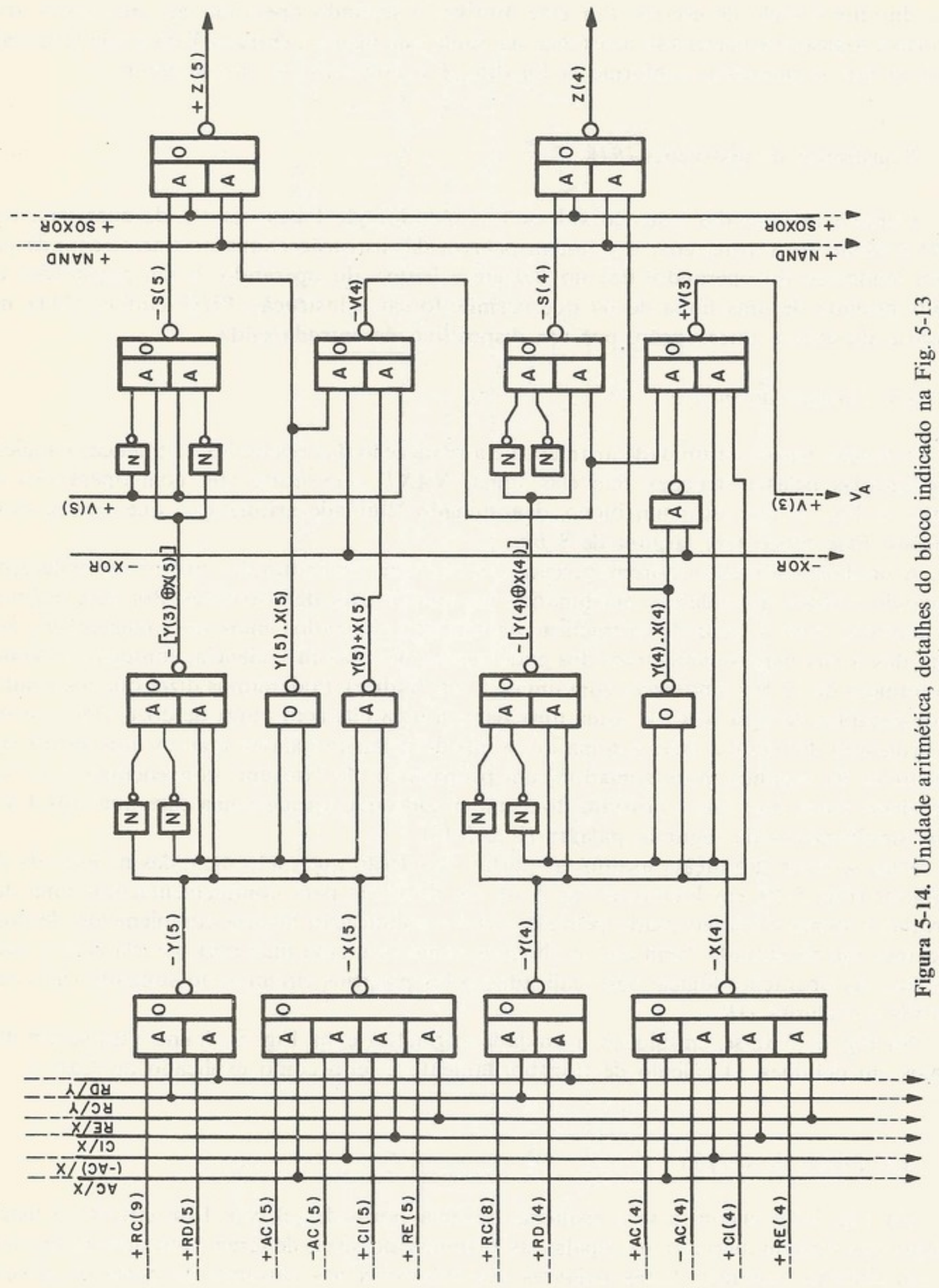


Figura 5-14. Unidade aritmética, detalhes do bloco indicado na Fig. 5-13

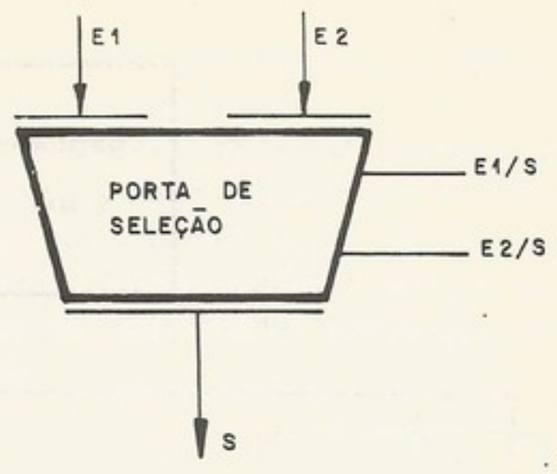


Figura 5-15. Esquema de uma porta de seleção

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;
$\overline{AC}$, 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).

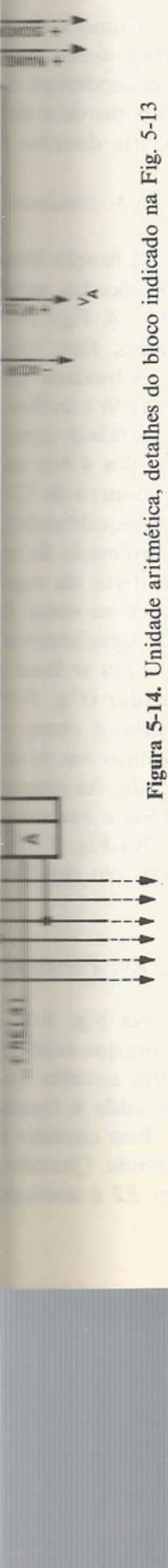


Figura 5-14. Unidade aritmética, detalhes do bloco indicado na Fig. 5-13

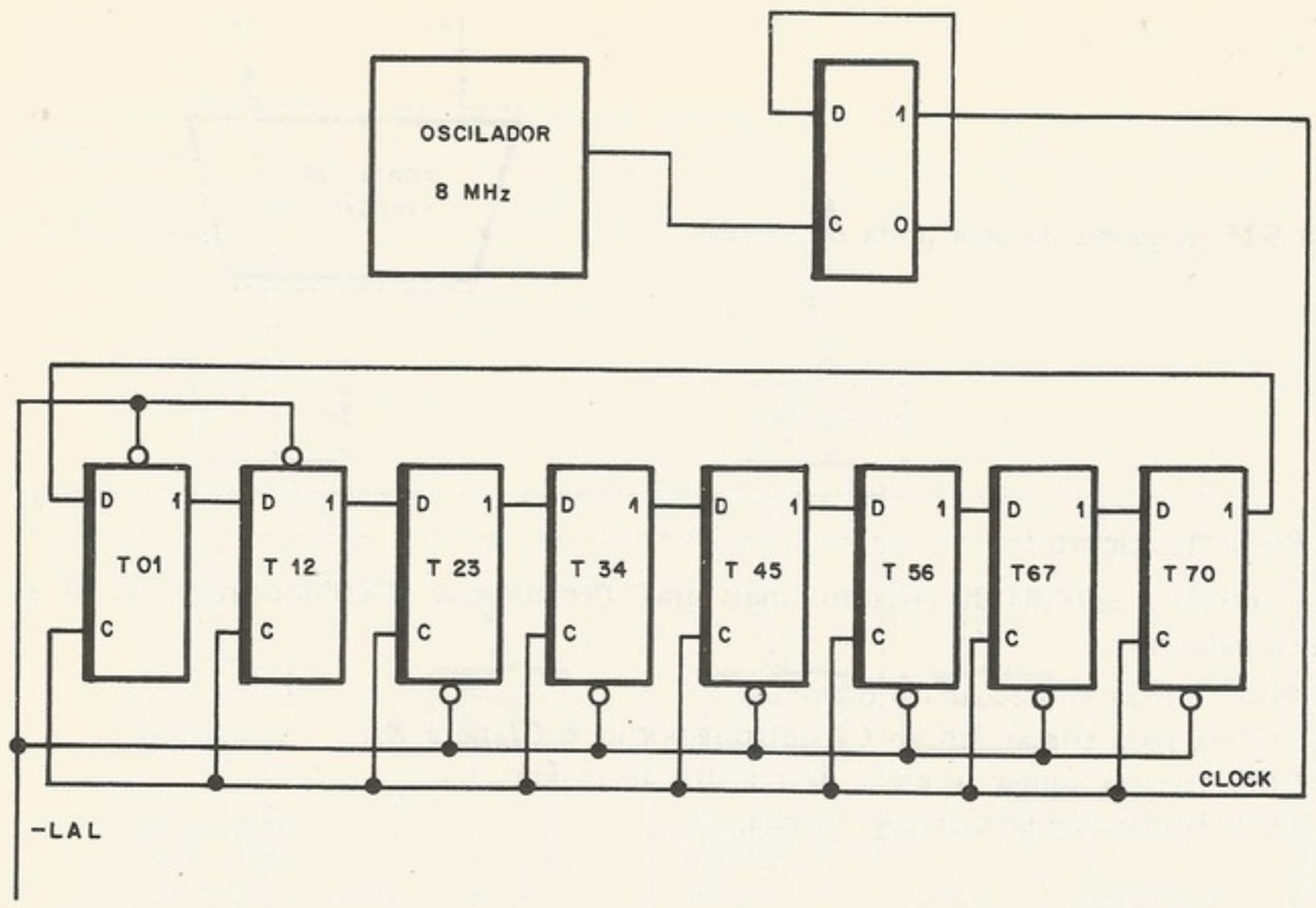


Figura 5-16. Esquema simplificado do relógio central

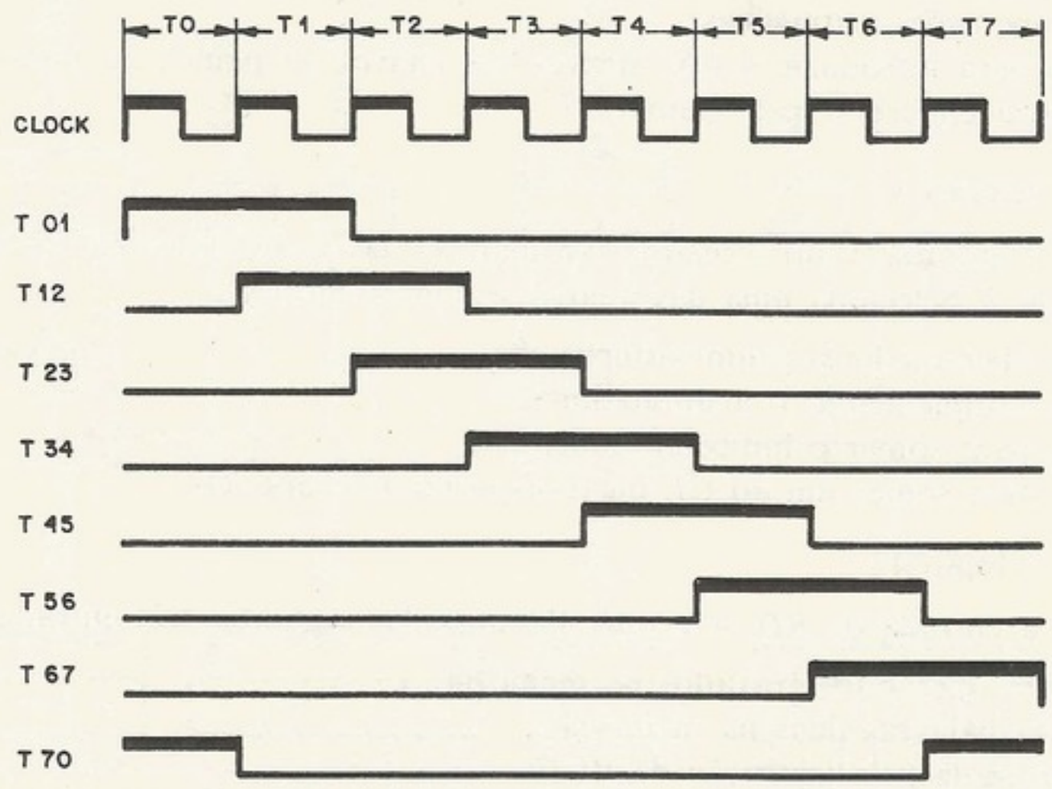


Figura 5-17. Gráfico de *timing* dos sinais do relógio central

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 μs), damos o nome de ciclo da memória. As oito partes do ciclo da memória ($T0$ a $T7$) são as unidades

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:

*FR*1-1, registradores, *bits* pares;
*FR*2-2, registradores, *bits* ímpares;
*FS*1-3, somador, *bits* menos significativos;
*FS*2-4, somador, *bits* mais significativos;
FAC-5, acumulador;
*FM*1-6, "mais um" e porta de seleção 6;
FCM-7, interface com memória;
FIN-8, sinais para cartões de interface.

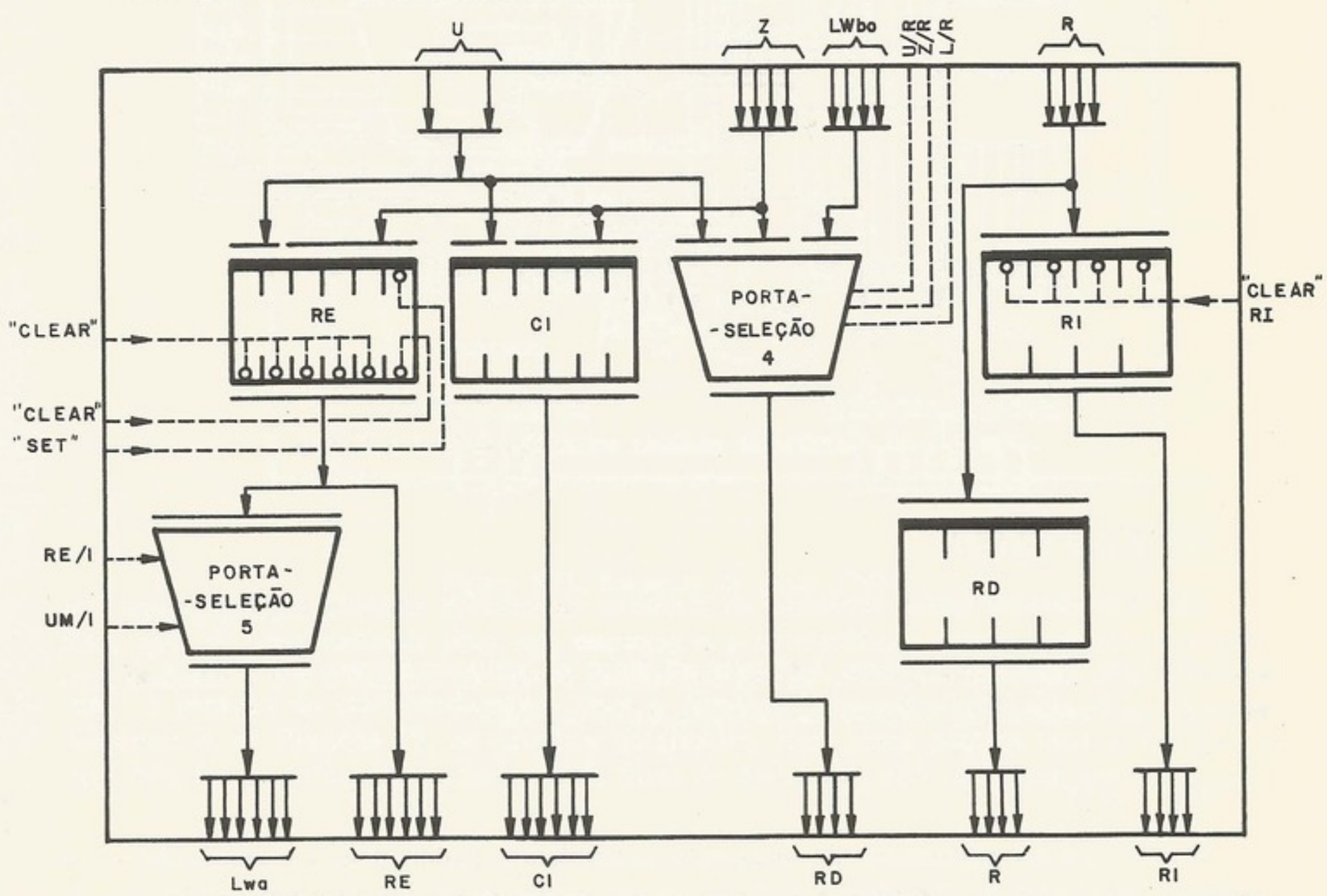


Figura 5-18. Cartão dos registradores (*FR*1-1, *FR*2-2)

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. *FR*1-1 *e* *FR*2-2

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.

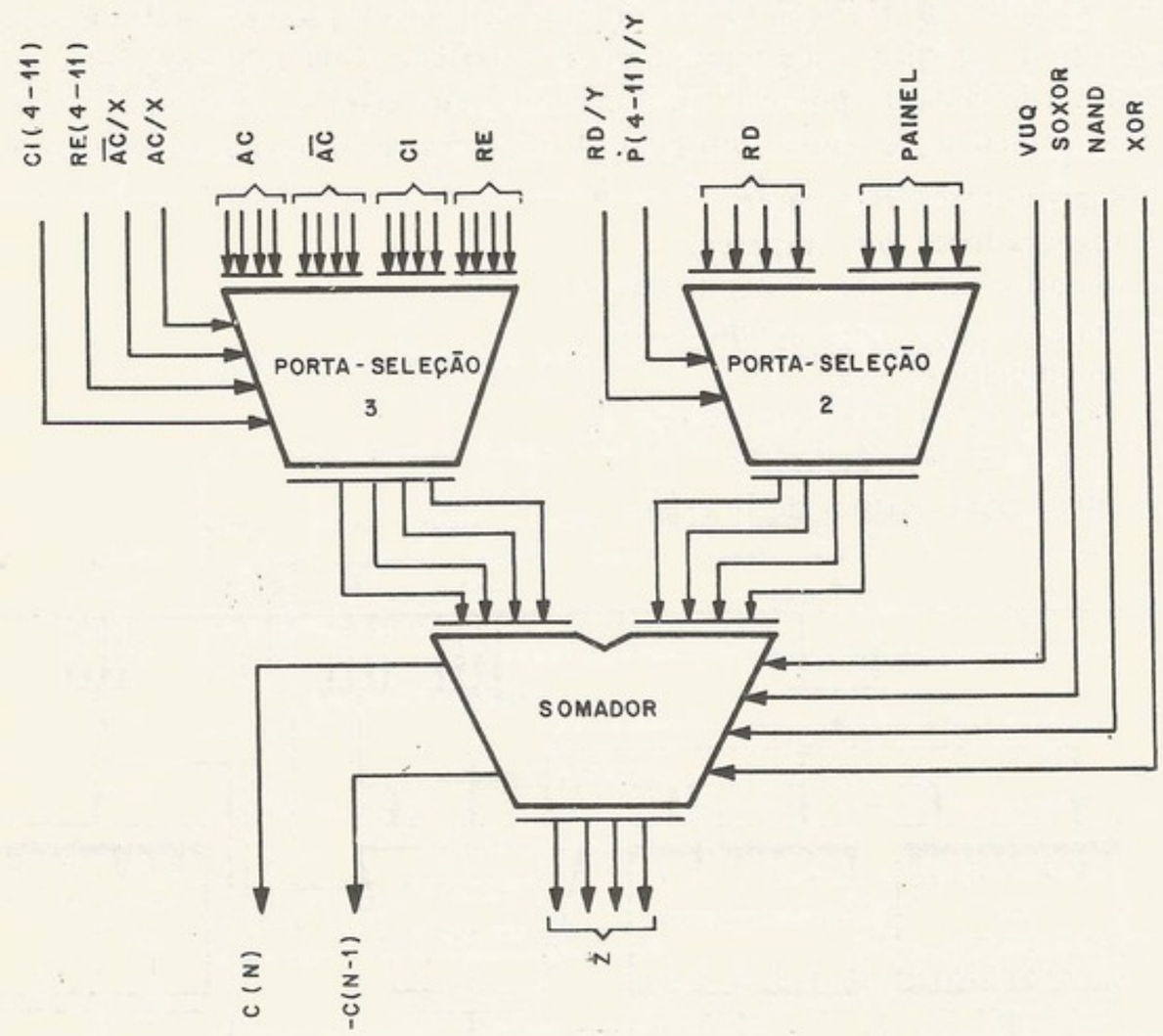


Figura 5-19. Cartões de unidade aritmética (*FS*1-3 e *FS*2-4)

b.2. *FS*1-3 *e* *FS*2-4

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.

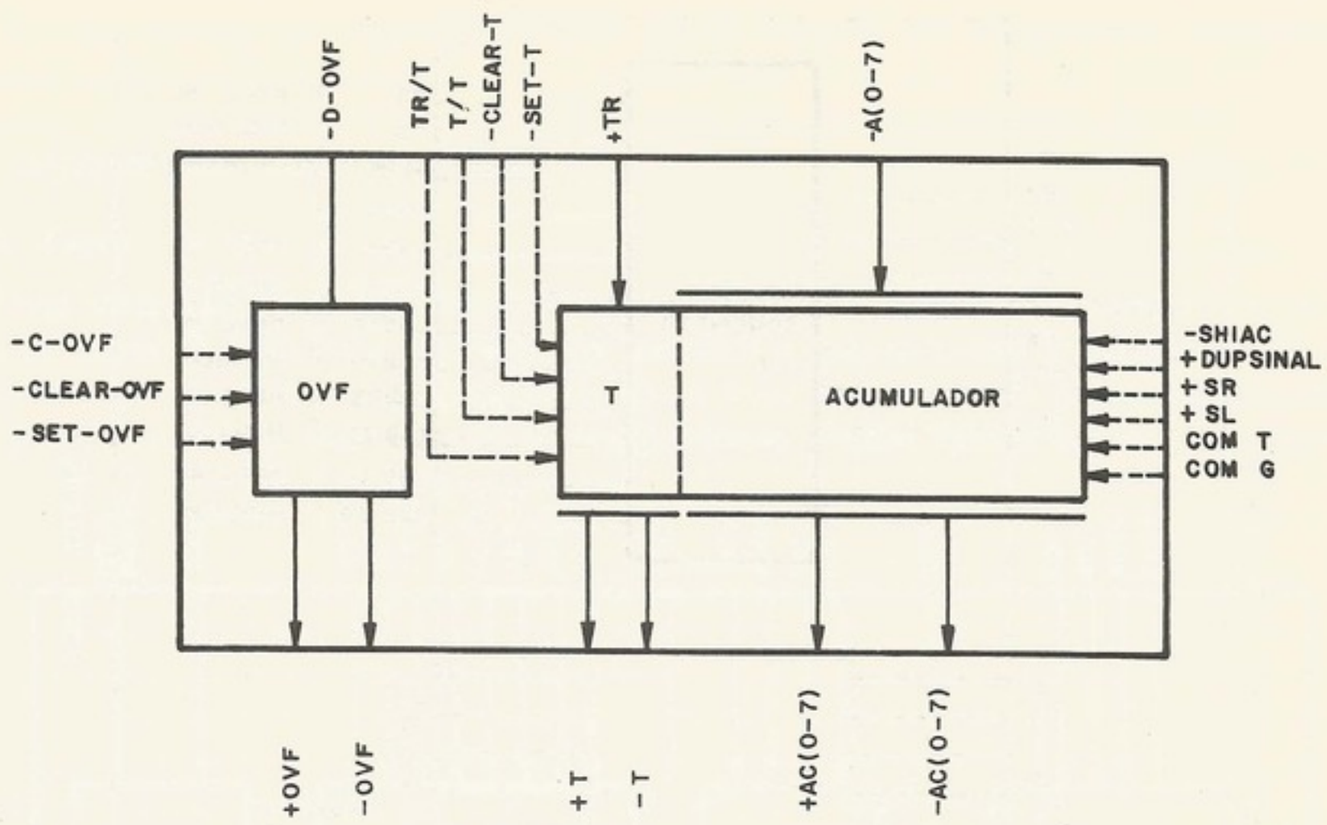


Figura 5-20. Esquema da placa do acumulador (*FAC*-5)

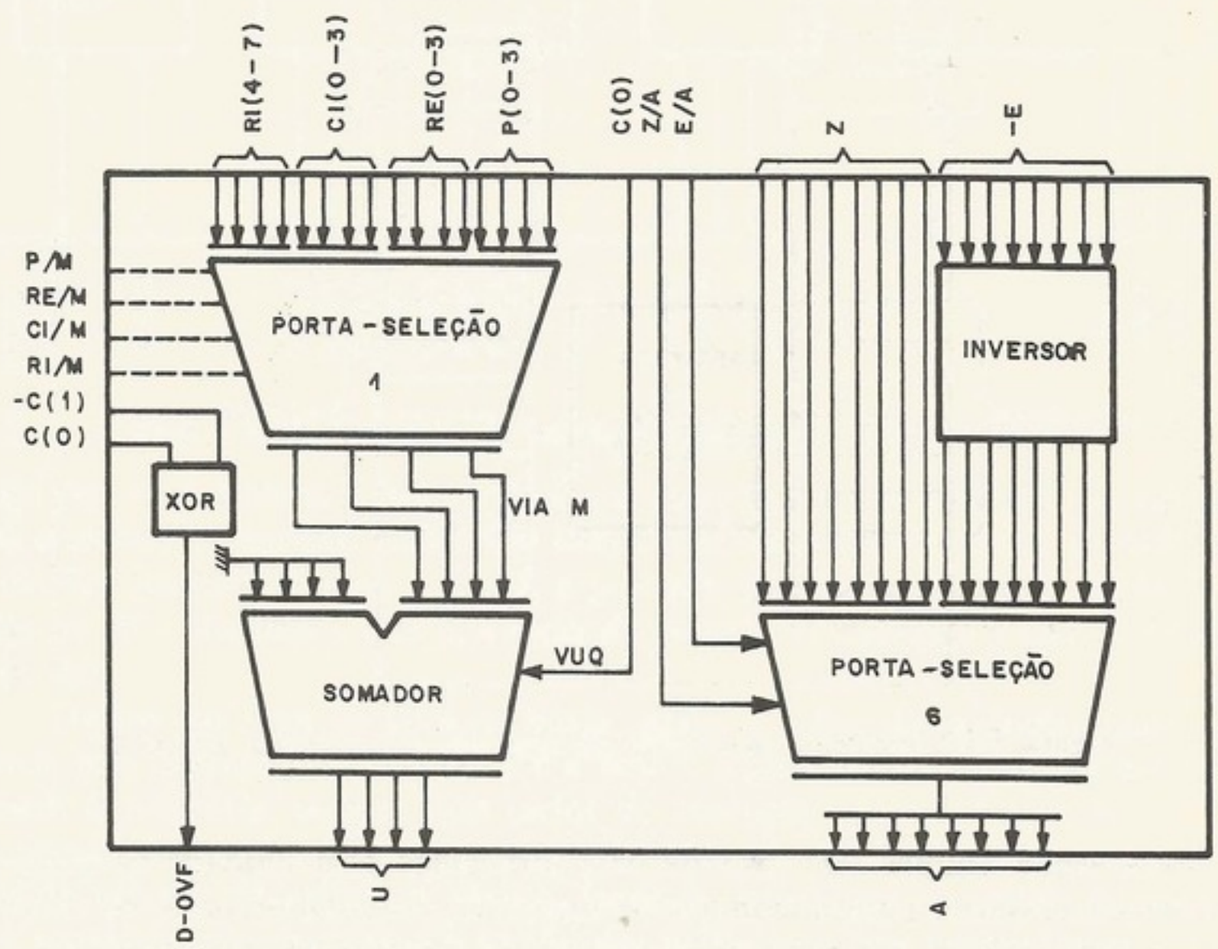


Figura 5-21. Esquema da placa *FM*1-6

b.4. *FM*1-6

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-

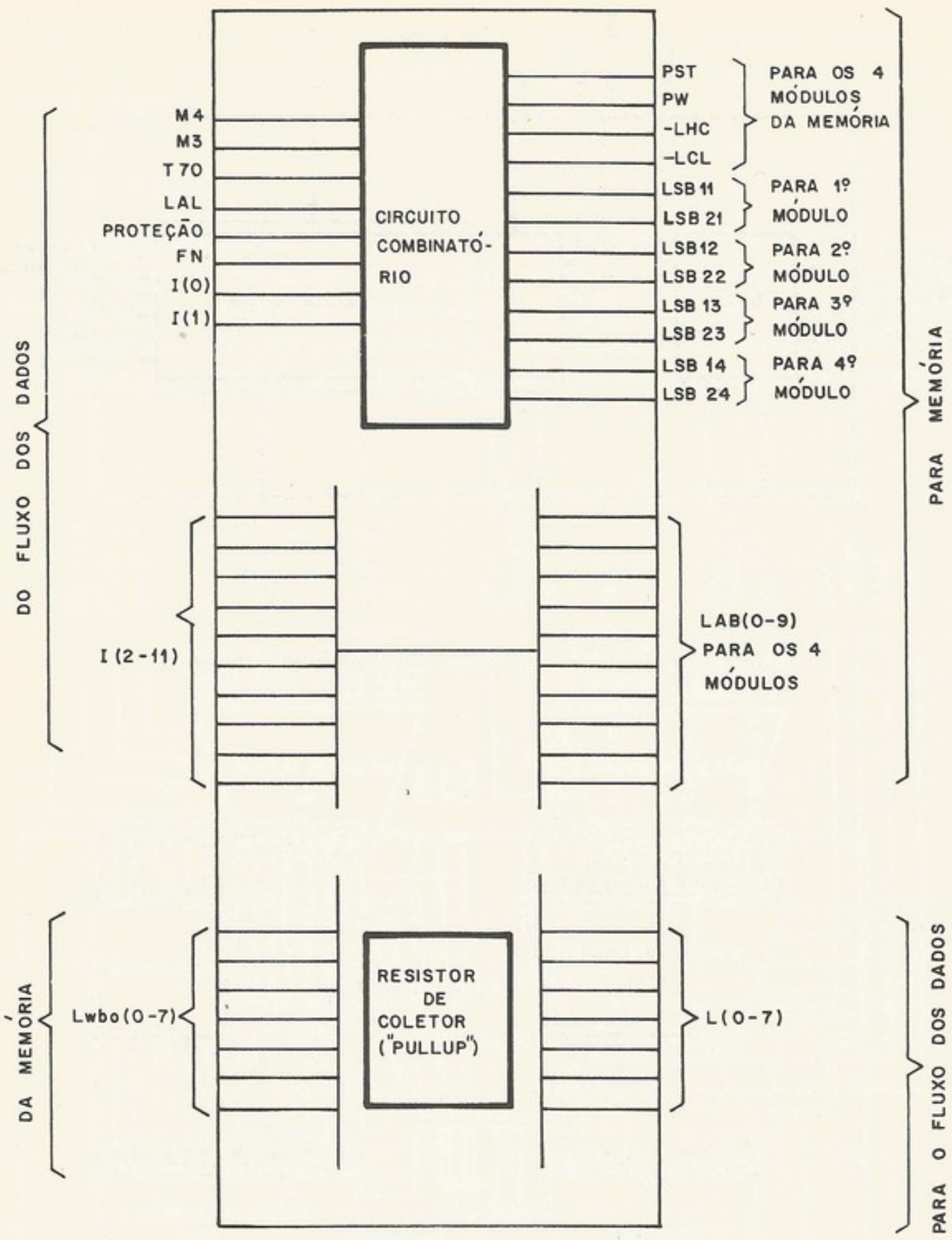


Figura 5-22. Esquema do cartão de controle de memória (*FCM*-7)

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.

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

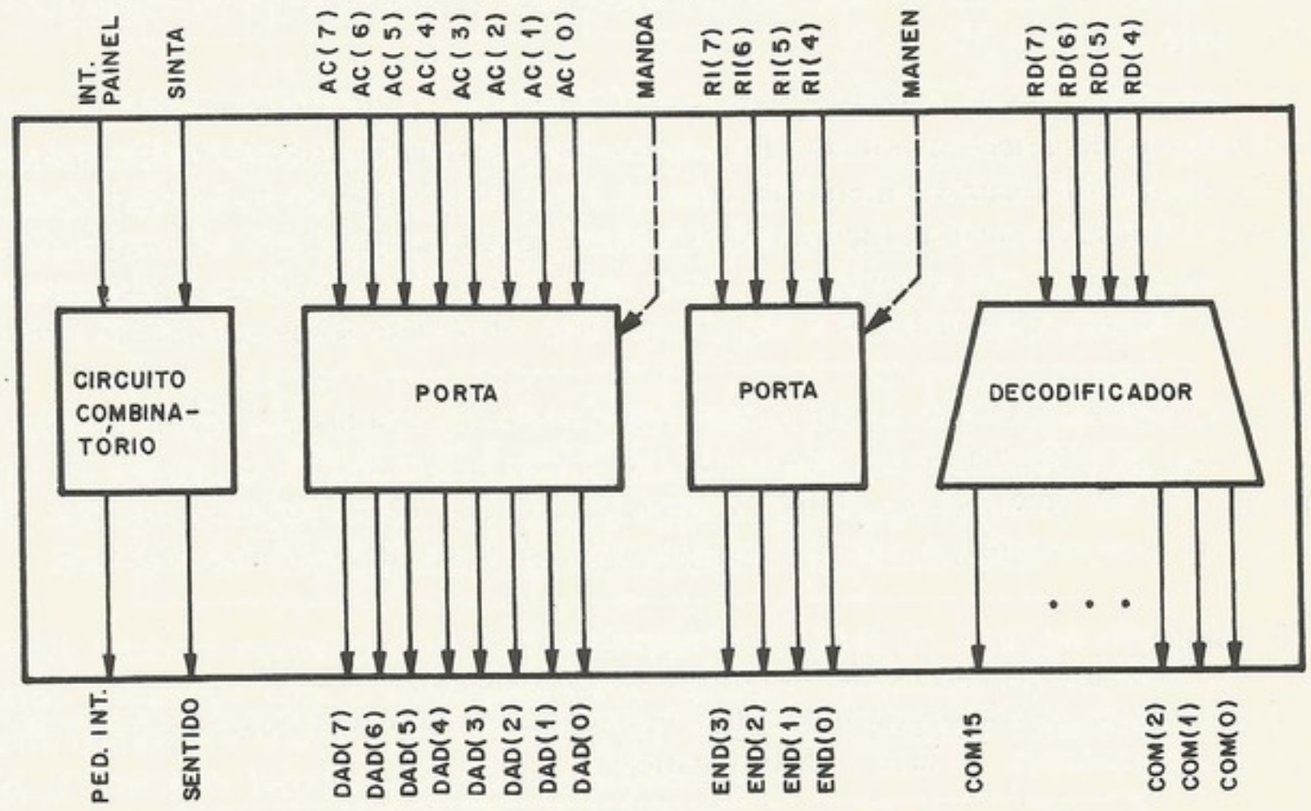


Figura 5-23. Esquema do cartão dos sinais de interface (*FIN*-8)

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, fase 2 } fases de *fetch* ou busca;

fase 3, fase 4 } fases de execução;

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 $CF1$, $CF2$, $CF3$, $CF4$ e $CF5$ (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 μs).

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

$T0$	$T1$	$T2$	$T3$	$T4$	$T5$	$T6$	$T7$
Memória: ler e restaurar							
	Lwbo $RI \leftarrow$				$RE \leftarrow CI$		$F2 \leftarrow 1$
$CI \leftarrow CI + 1$							

Inicia-se o ciclo lendo-se a memória. Nessa fase, a primeira palavra da instrução é lida na memória e, no final do tempo $T1$, é colocada em RI. Enquanto isso, incrementa-se um no contador de instrução. No final desse ciclo ($T5$ e $T6 = T56$), colocamos o conteúdo de CI em RE para endereçar a segunda metade da instrução e, em $T7$, disparamos a fase 2, que será o ciclo de memória seguinte. É importante notar que, até $T2$, 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 $T0$, $T1$ e $T2$.

A fase 2 de $SOMX$ será

$T0$	$T1$	$T2$	$T3$	$T4$	$T5$	$T6$	$T7$
Memória: ler e restaurar							$F5 \leftarrow 1$
	$RD \leftarrow$ *Lwbo*				$RE \leftarrow RI(4\text{-}7)$ e RD		
$CI \leftarrow CI + 1$							

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

T0	*T1*	*T2*	*T3*	*T4*	*T5*	*T6*	*T7*
Memória: ler e restaurar							$F3 \leftarrow 1$
	RD ← *Lwbo*				$RE \leftarrow RE + RD$		
$0 \rightarrow Lwa$							

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 fim 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

T0	*T1*	*T2*	*T3*	*T4*	*T5*	*T6*	*T7*
Memória: ler e restaurar							$F1 \leftarrow 1$
	RD ← *Lwbo*		$AC \leftarrow AC + RD$		$RE \leftarrow CI$		

Aqui, lemos o operando que fora endereçado no final do ciclo anterior. Em *T*34, 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

$$CI \leftarrow CI + 1$$

FASE 1 — INSTRUÇÕES	T0	T1	T2	T3	T4	T5	T6	T7
TODAS MENOS MICRO		MEMÓRIA:	LER E	RESTAURAR				FASE 2←1
		RI ← Lwbo				RE ← CI		
	CI ← CI+1							
MICRO-GRUPO 1		MEMÓRIA:	LER E	RESTAURAR				FASE1←1
		RI ← Lwbo		AÇÃO DAS (AC/X, $\overline{AC}$/X,	MICRO P/X,μ,VUQ)	RE ← CI		
	CI ← CI+1				C−AC			
MICRO-GRUPO 2B: PUL		MEMÓRIA:	LER E	RESTAURAR				FASE1←1
		RI ← Lwbo			"CLEAR" OCUPADO	CI, RE ←	CI+1	
	CI ← CI	+1		CI ← 0 +	0 + VUQ			
MICRO-GRUPO 2A: SALTOS		MEMÓRIA:	LER E	RESTAURAR				FASE 1←1
		RI ← Lwbo		SE SALTO: CI ← CI+1		SE NÃO RE ← CI	SALTO	
						SE SALTO·RE·CI	← CI+1	
	CI ← CI+1						CLEAR/SET V OU T	
MICRO-GRUPO 2B: TRE e TRI		MEMÓRIA:	LER E	RESTAURAR				FASE 3←1
		RI ← Lwbo				RE ← CI		
MICRO-GRUPO 2B: INIB		MEMÓRIA:	LER E	RESTAURAR				FASE1←1
		RI ← Lwbo		SET INIBE		RE ← CI		
	CI ← CI+1							
MICRO-GRUPO 2B: PERM		MEMÓRIA:	LER E	RESTAURAR				FASE1←1
		RI ← Lwbo		"CLEAR" INIBE		RE ← CI		
	CI ← CI+1							
MICRO-GRUPO 2B: ESP		MEMÓRIA:	LER E	RESTAURAR				FASE1←1
		RI ← Lwbo		SET ESP		RE ← CI		
	CI ← CI+1							
MICRO-GRUPO 2B: PARE		MEMÓRIA:	LER E	RESTAURAR				FASE1←1
		RI ← Lwbo		SET PARE		RE ← CI		
	CI ← 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.

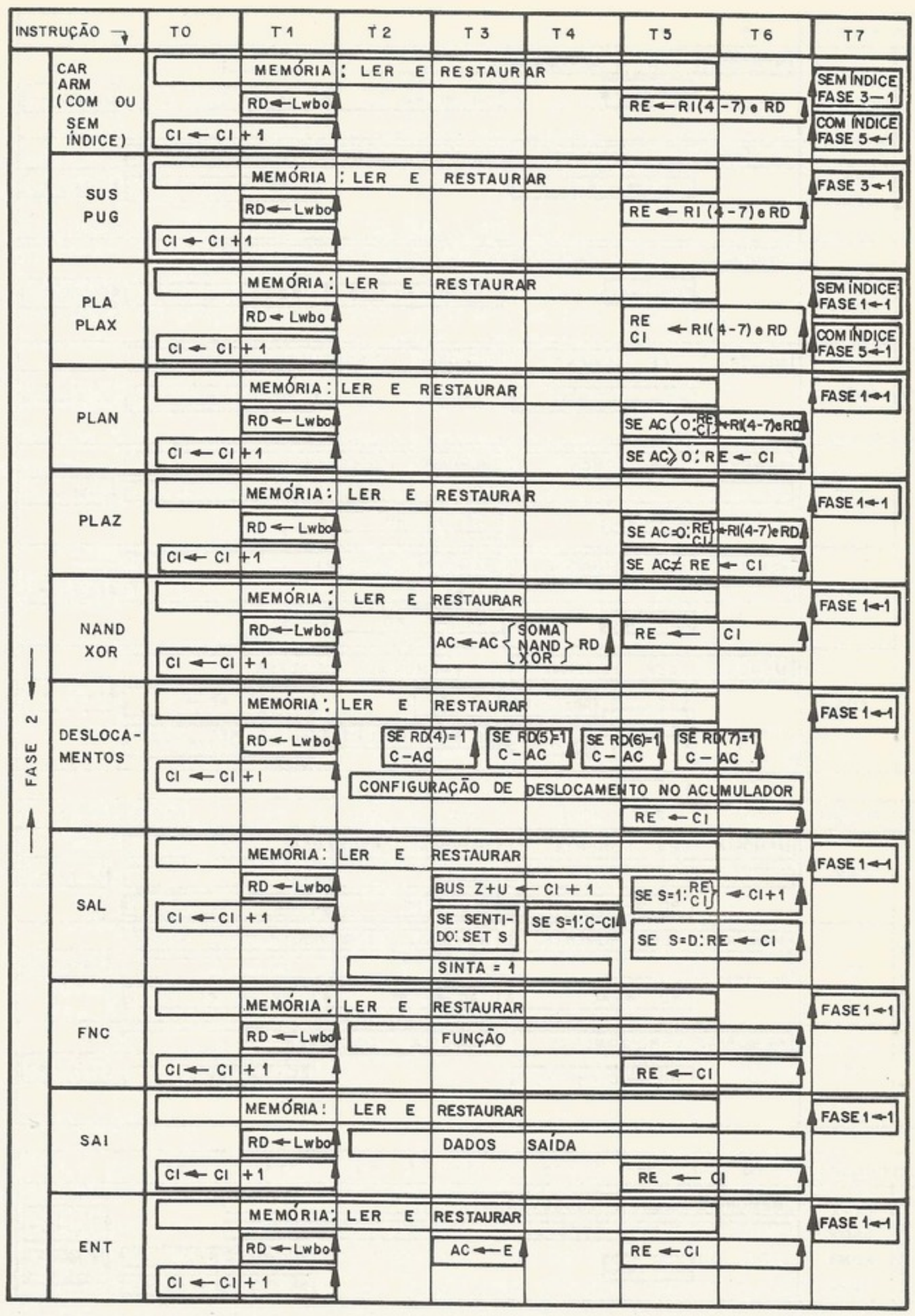


Figura 5-25. Carta de microoperações da fase 2

FASE 3 — INSTRUÇÕES	T0	T1	T2	T3	T4	T5	T6	T7
CAR CARX	MEMÓRIA: LER E RESTAURAR	RD ← Lwbo		AC ← RD		RE ← CI		FASE 1 ← 1
ARM ARMX	MEMÓRIA: LER SOMENTE	RD ← AC		MEMÓRIA: ESCREVER		RE ← CI		FASE 1 ← 1
SOM SOMX	MEMÓRIA: LER E RESTAURAR	RD ← Lwbo		AC ← AC + RD		RE ← CI		FASE 1 ← 1
PUG	MEMÓRIA: LER SOMENTE	RD ← CI(0—3)		MEMÓRIA: ESCREVER		RE ← RE + 1		FASE 4 ← 1
TRE	MEMÓRIA: LER SOMENTE	RD ← Lwbo		MEMÓRIA: ESCREVER 0/Lwa	RE ← RD			FASE 4 ← 1
TRI	MEMÓRIA: LER SOMENTE	RD ← Lwbo		MEMÓRIA: ESCREVER 0/Lwa	RE ← RD			FASE 4 ← 1
SUS	MEMÓRIA: LER SOMENTE	RD ← Lwbo	SE RDZ = 0: RD ← RD − 1 SE RDZ = 1: SET 5	MEMÓRIA: ESCREVER SE S=1: CI ← CI + 1		SE S=0: RE ← CI SE S=1: RE, CI } ← CI + 1		FASE 1 ← 1

FASE 4 — INSTRUÇÕES	T0	T1	T2	T3	T4	T5	T6	T7
TRE	MEMÓRIA: LER SOMENTE	RD ← AC		MEMÓRIA: ESCREVER 1/Lwa AC ← RE		RE ← CI		FASE 1 ← 1
TRI	MEMÓRIA: LER SOMENTE	RD ← AC		MEMÓRIA: ESCREVER 0/Lwa AC ← RE		RE ← CI		FASE 1 ← 1
PUG	MEMÓRIA: LER SOMENTE	RD ← CI (4 − 1)		MEMÓRIA: ESCREVER		RE, CI } ← RE + 1		FASE 1 ← 1

FASE 5 — INSTRUÇÕES	T0	T1	T2	T3	T4	T5	T6	T7
CARX ARMX	MEMÓRIA: LER E ESCREVER 0/Lwa							SE PLI: FASE 1 ← 1
SOMX PLAX		RD ← Lwbo				RE ← RE + RD SE PLI 1 CI ← RE + RD		SE NÃO PLI FASE 4 ← 1

Figura 5-26. Carta de microoperações das fases 3, 4 e 5

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.

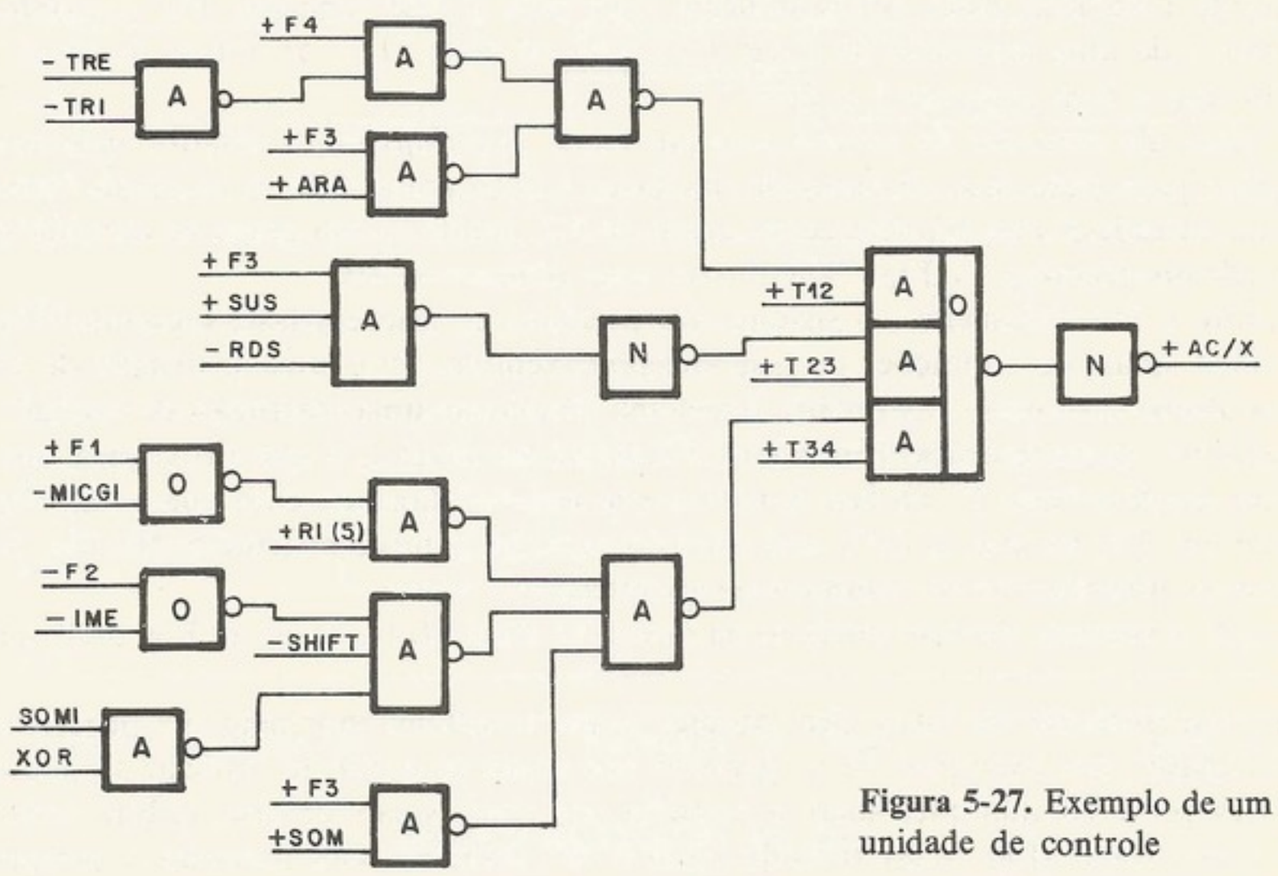


Figura 5-27. Exemplo de um sinal da unidade de controle

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,

*CF*1, condição para fase 1;
*CF*2, condição para fase 2;
*CF*3, condição para fase 3;
*CF*4, condição para fase 4;
*CF*5, 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*, *CF*3 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 início de *T*7, 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 *CF*2 estará no nível 1, o que provocará, em *T*7, o disparo do *flip-flop* "fase 2"; na fase 2, *CF*3 é que será "1", fazendo com que o *flip-flop* "fase 3" se ligue em *T*7; na fase 3, *CF*1 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.

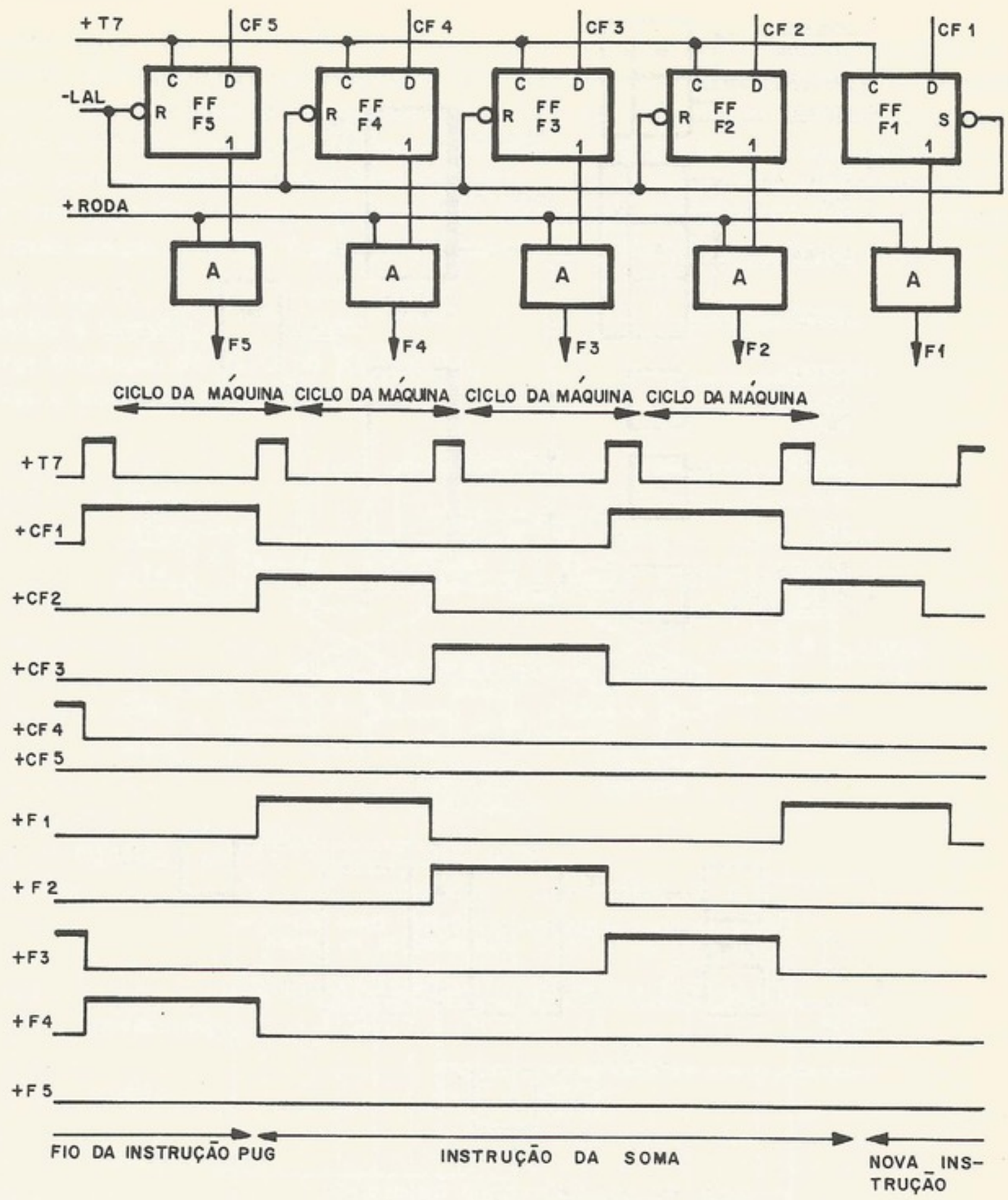


Figura 5-28. Controle das fases

"*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 *T*0 e acabando em *T*7. 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".

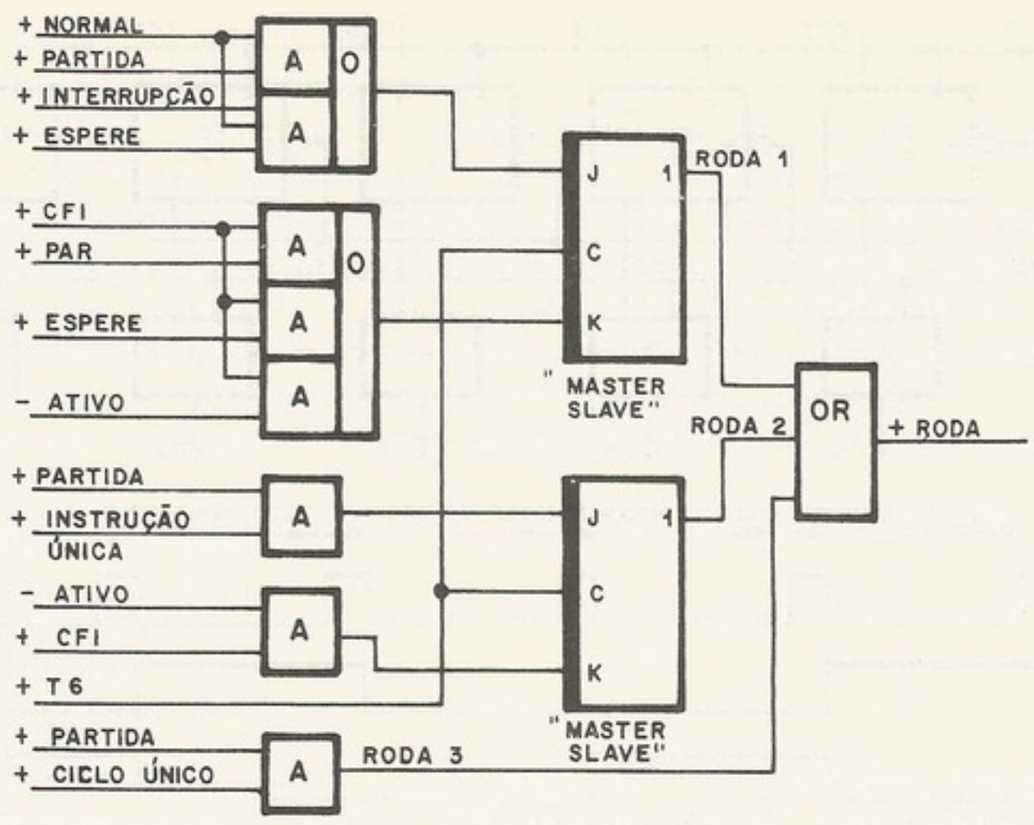


Figura 5-29. Geração do sinal "roda"

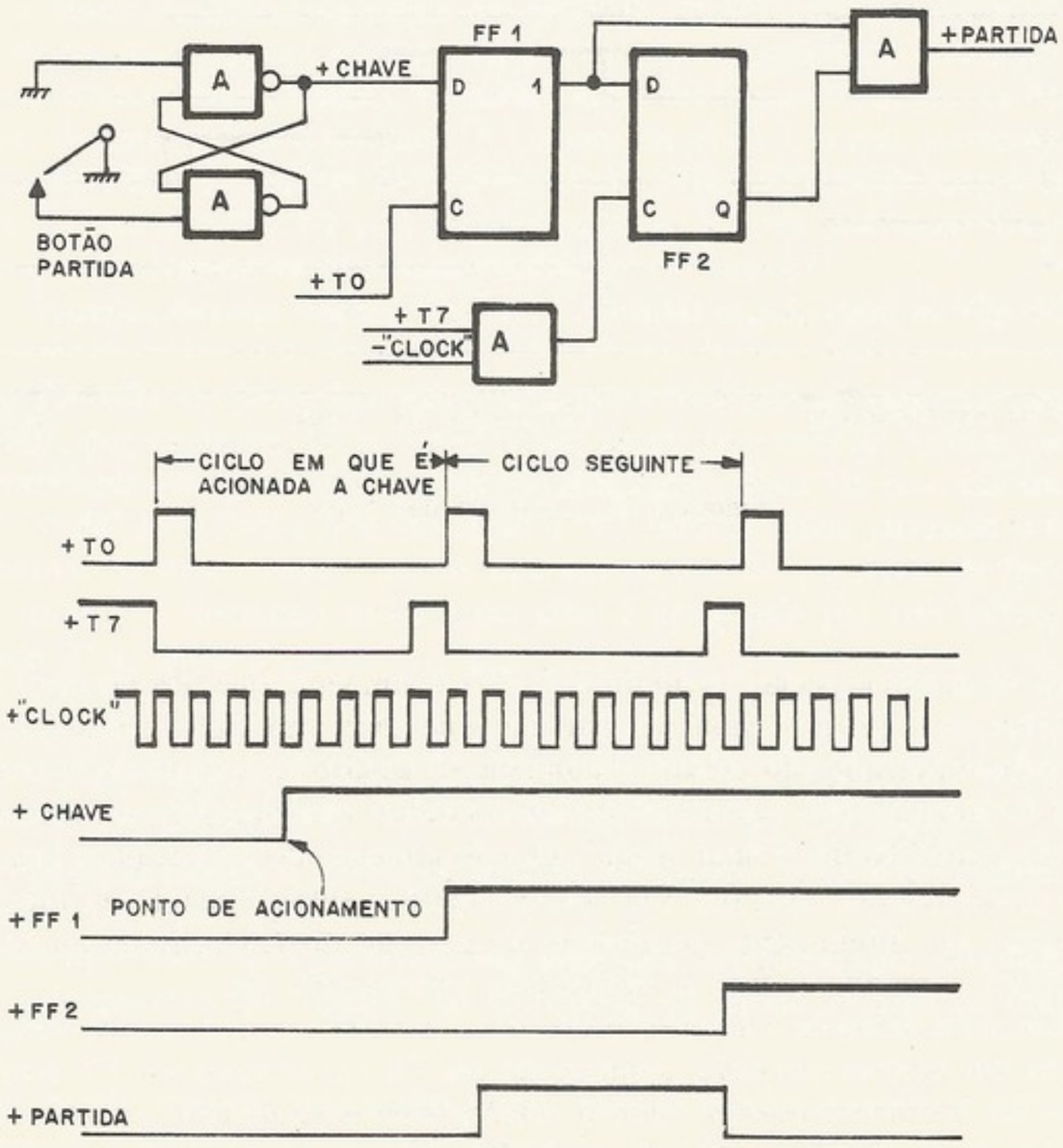


Figura 5-30. Geração do sinal "partida"

Outra função 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*.

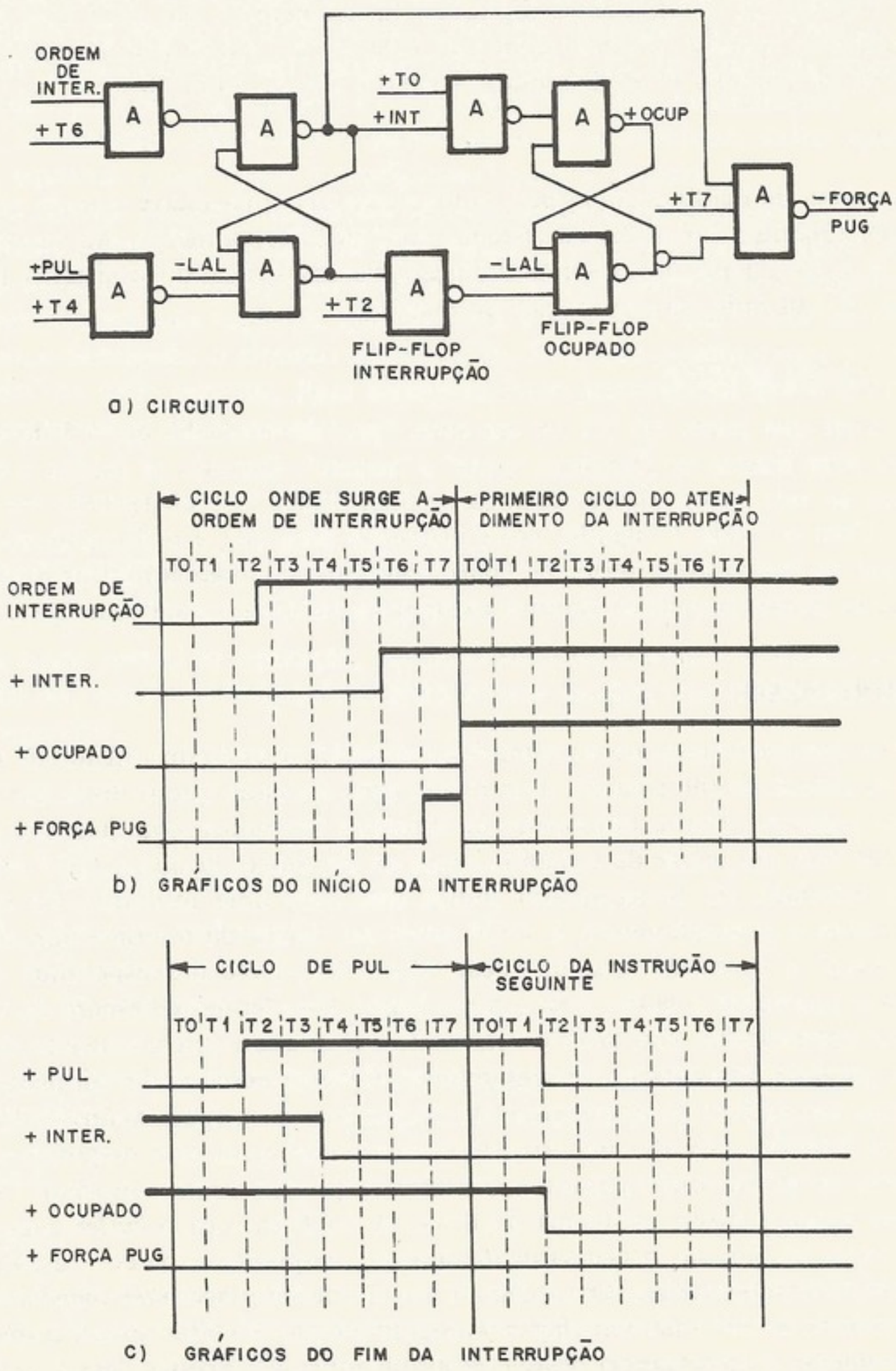


Figura 5-31. Mecanismo de interrupção

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 *T*6 de um sinal "partida" e acaba em *T*5 do ciclo seguinte. Ele é usado para limpar os *flip-flops* "pare" e "espere". Quando se aciona o interruptor de partida para se 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 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 *T*2, após um *T*4, 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 *T7*, 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).

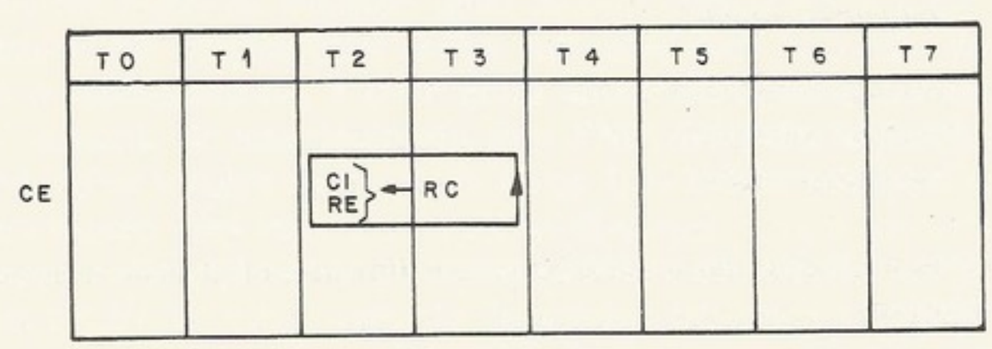


Figura 5-33. Distribuição das microoperações no modo "carrega endereço"

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).

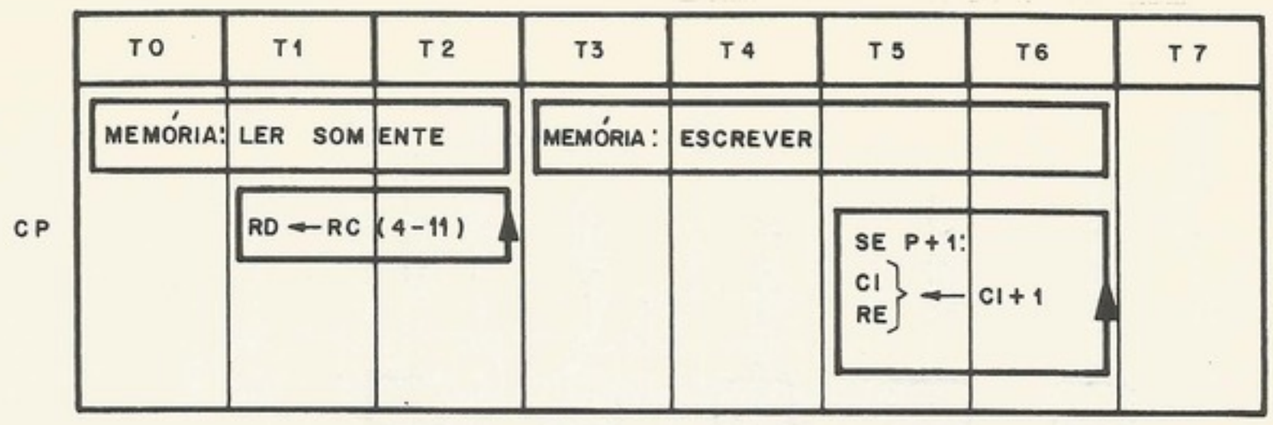


Figura 5-34. Seqüência das microoperações no modo "carrega posição'

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).

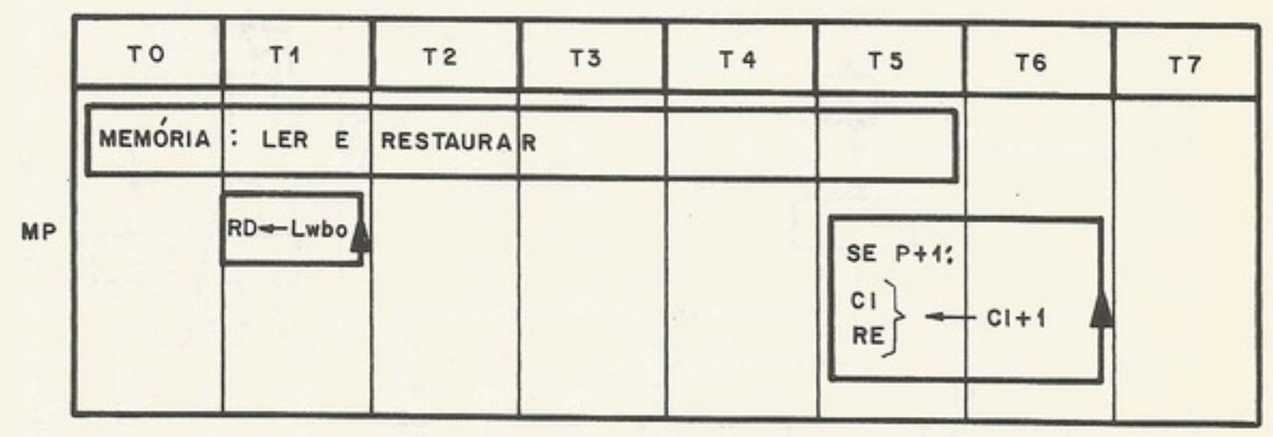


Figura 5-35. Seqüência das microoperações no modo "mostra posição"

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".

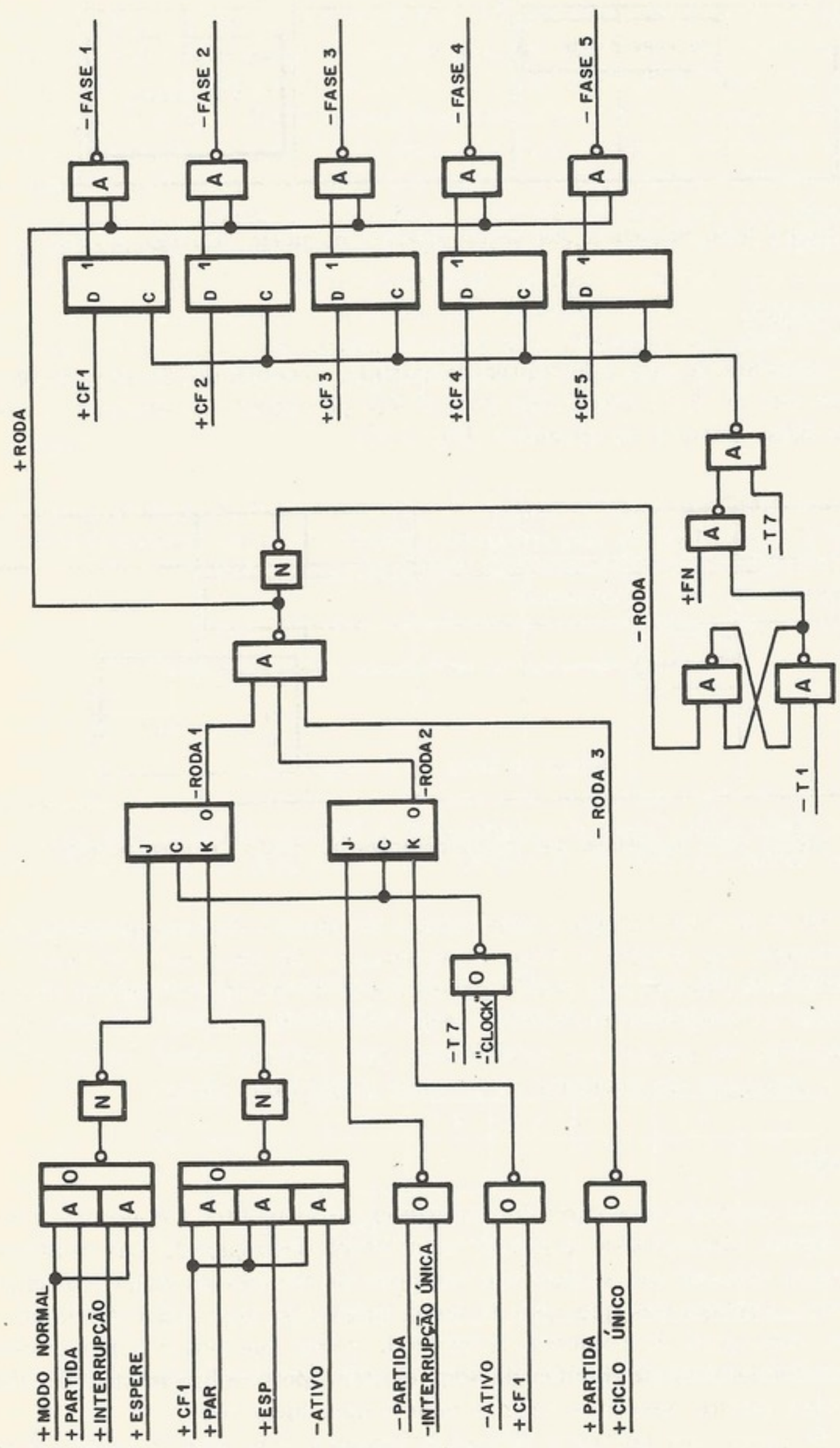
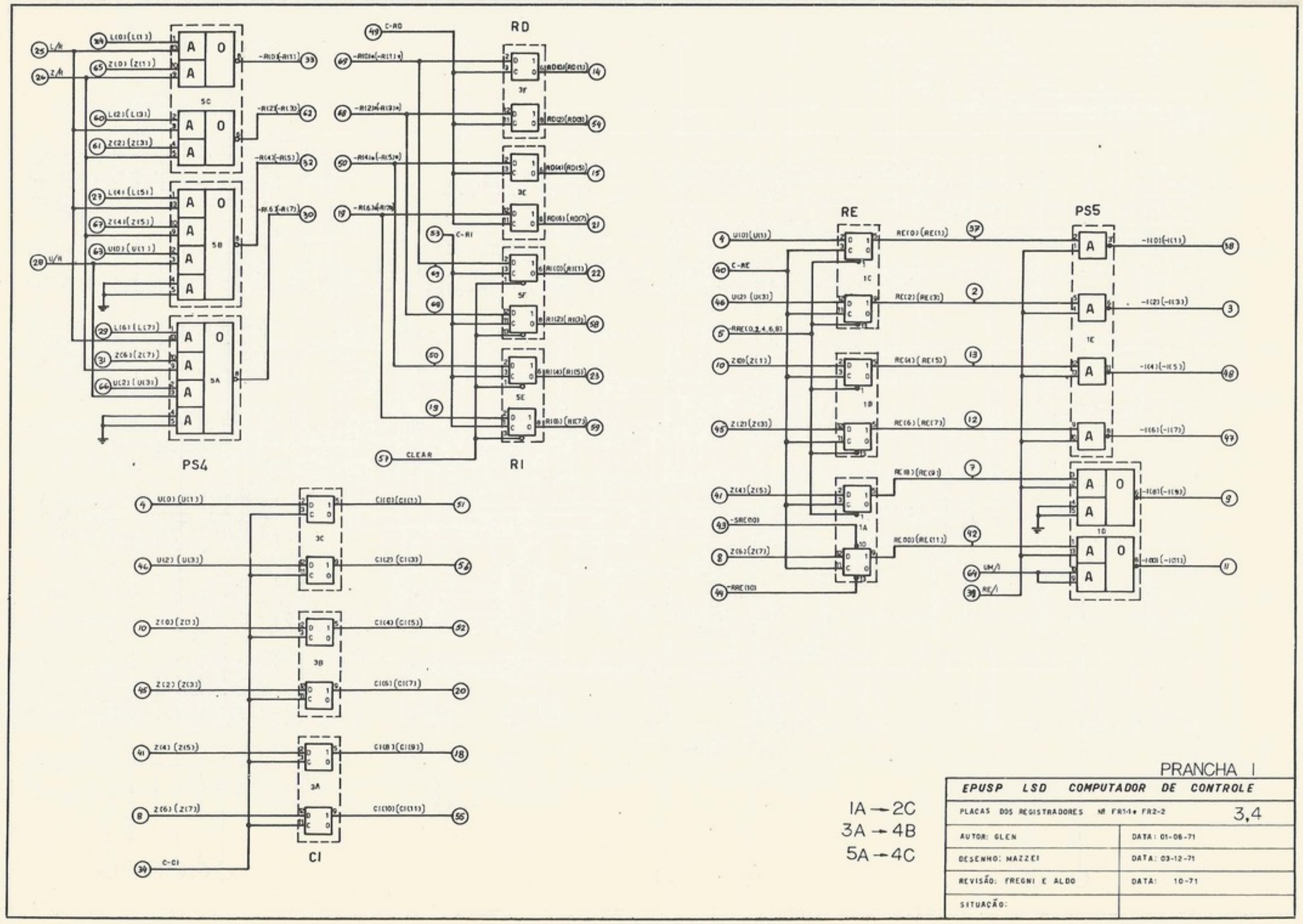
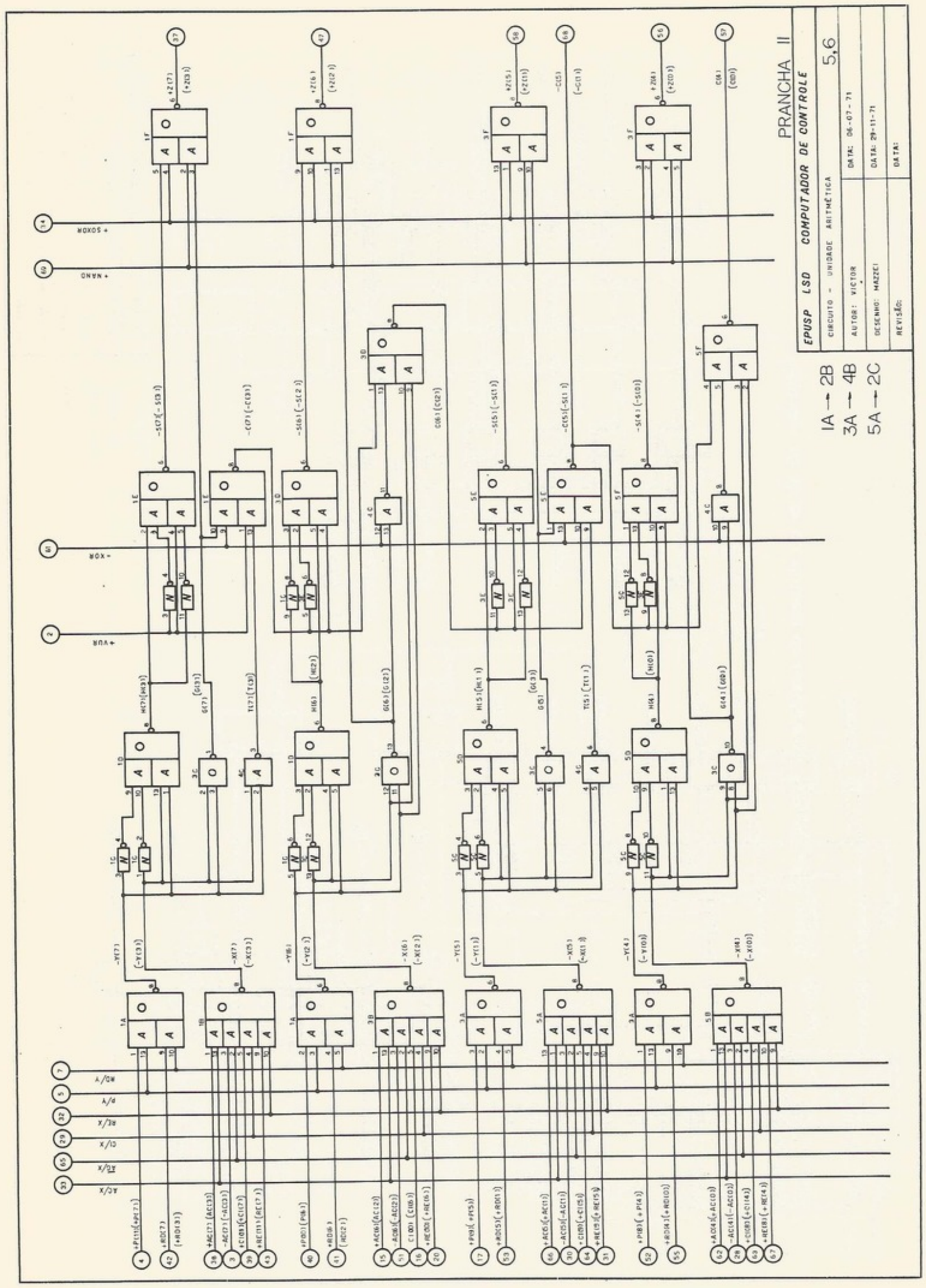


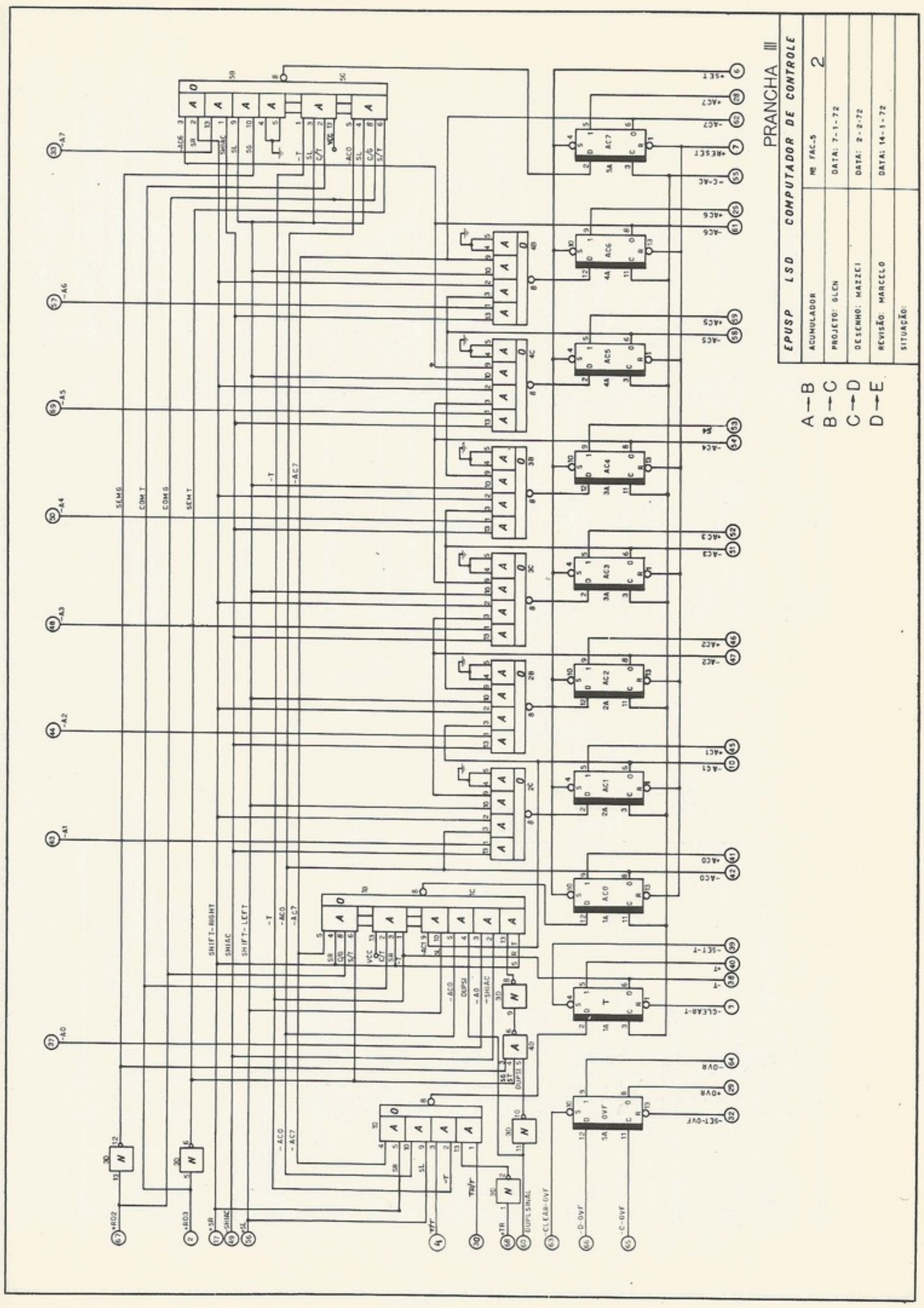
Figura 5-36. Esquema simplificado do controle do modo de operação

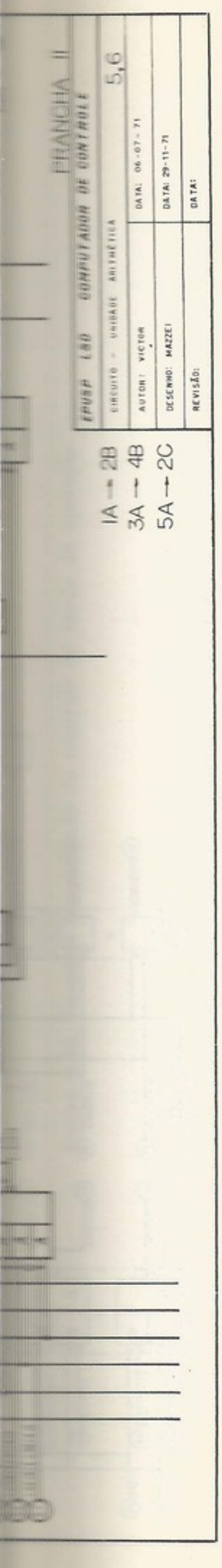
Figura 5-36. Esquema simplificado do controle do modo de operação

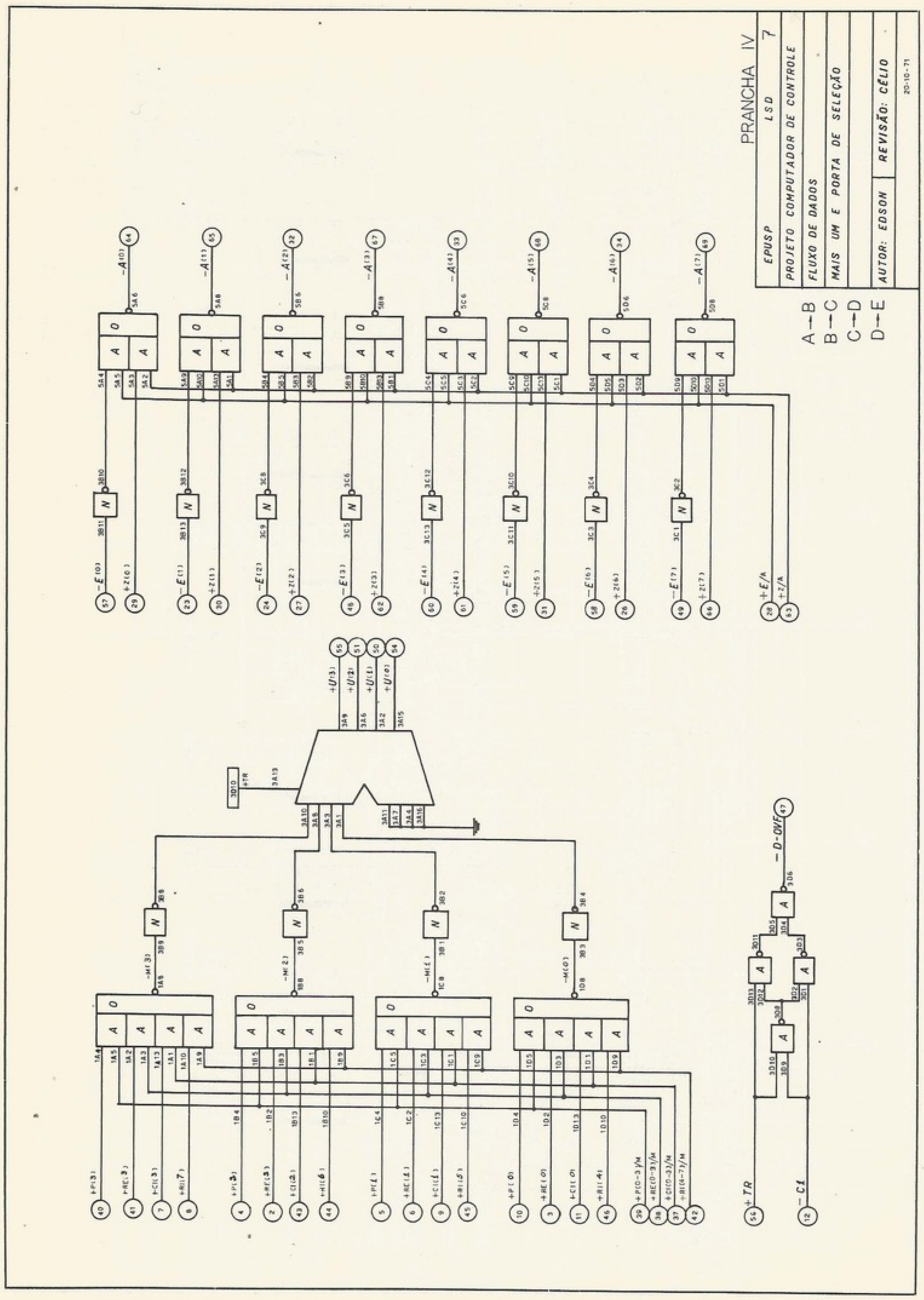
5.10 PRANCHAS

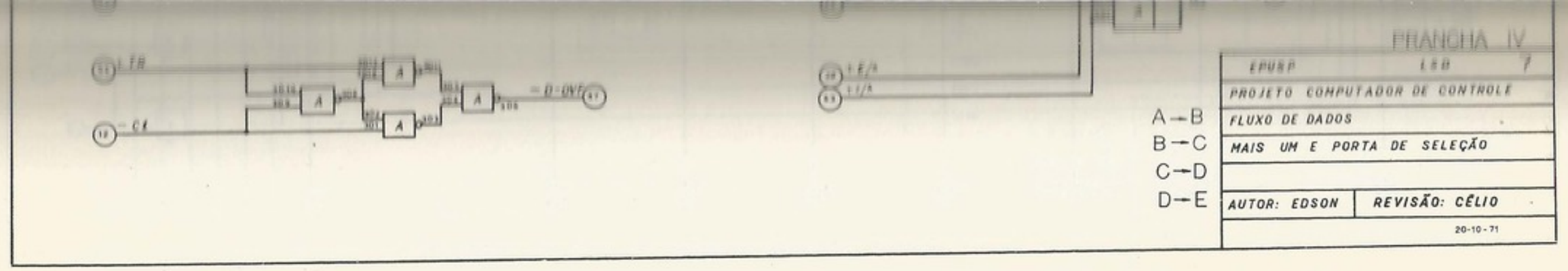


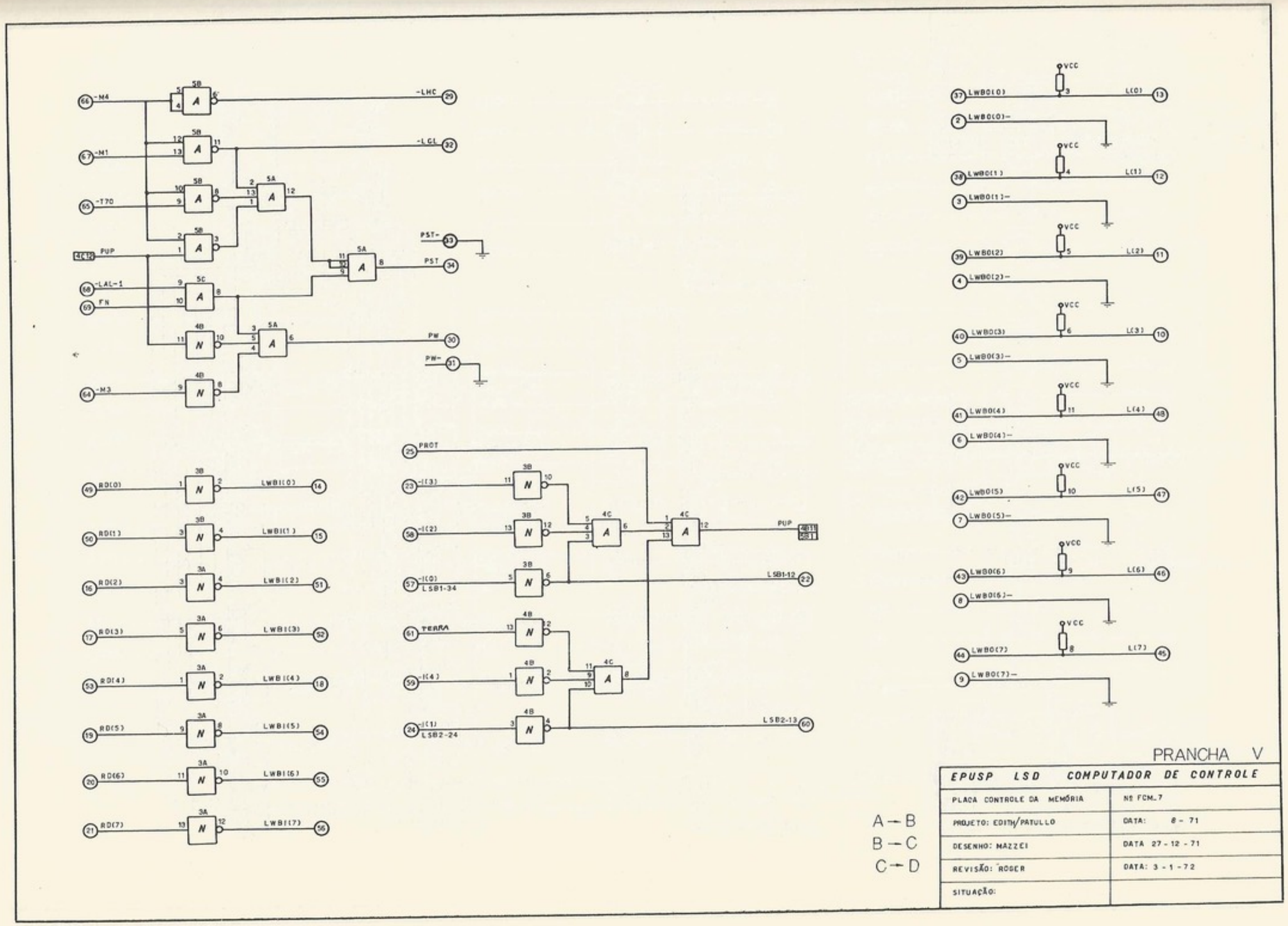

PRANCHA II
EPUSP LSD COMPUTADOR DE CONTROLE
CIRCUITO - UNIDADE ARITMÉTICA
AUTOR: VICTOR
DESENHO: MAZZEI
REVISÃO:
DATA: 06-07-71
DATA: 29-11-71
DATA:
5,6
1A → 2B
3A → 4B
5A → 2C
+XOR
+VUR
+SOXOR
+NAND
RD/Y
P/Y
RE/X
CI/X
RD/X
AC/X

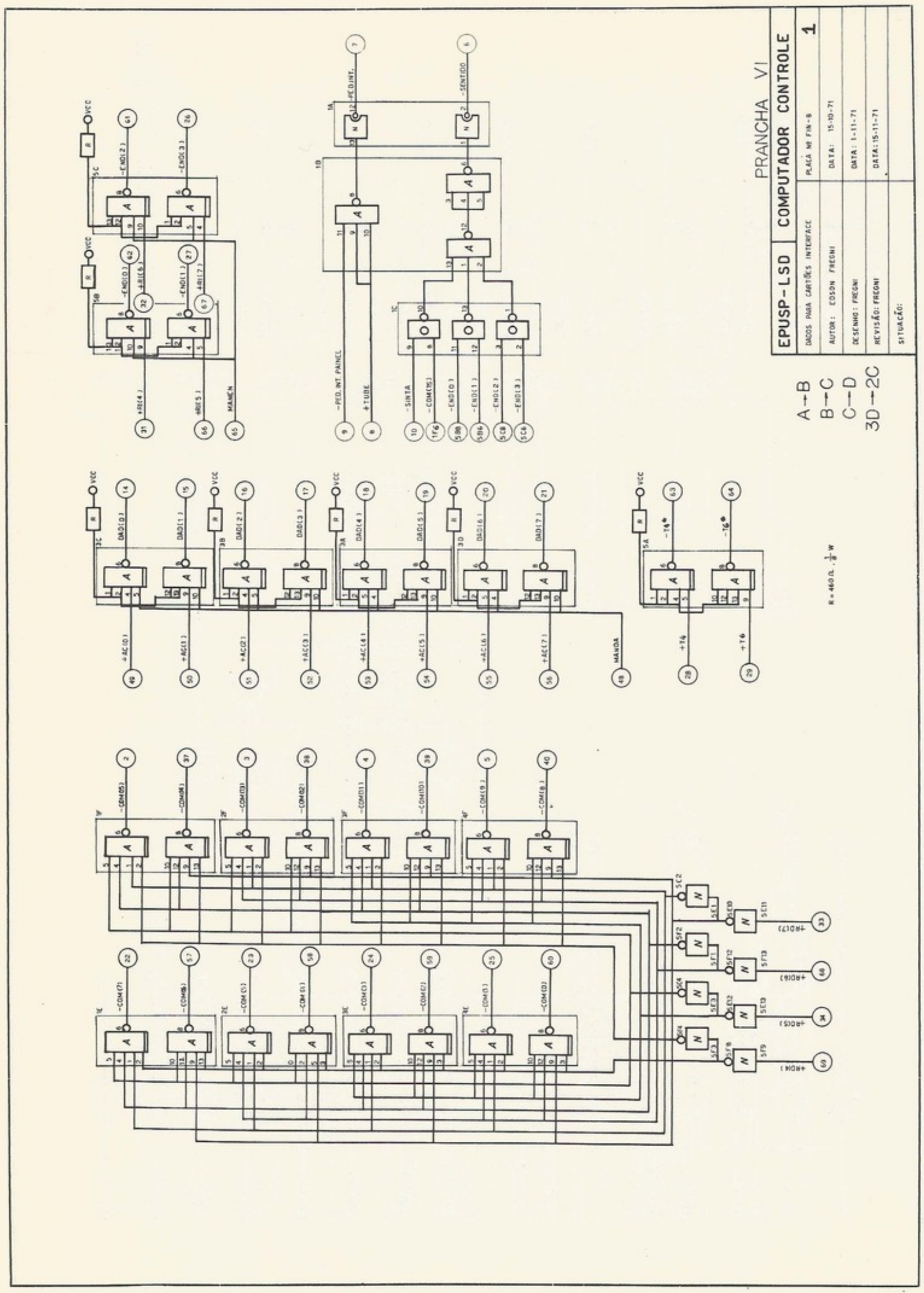

PRANCHA III
EPUSP LSD COMPUTADOR DE CONTROLE
ACUMULADOR
Nº FAC.-5
2
PROJETO: GLEN
DATA: 7-1-72
DESENHO: MAZZEI
DATA: 2-2-72
REVISÃO: MARCELO
DATA: 14-1-72
SITUAÇÃO:
A→B
B→C
C→D
D→E

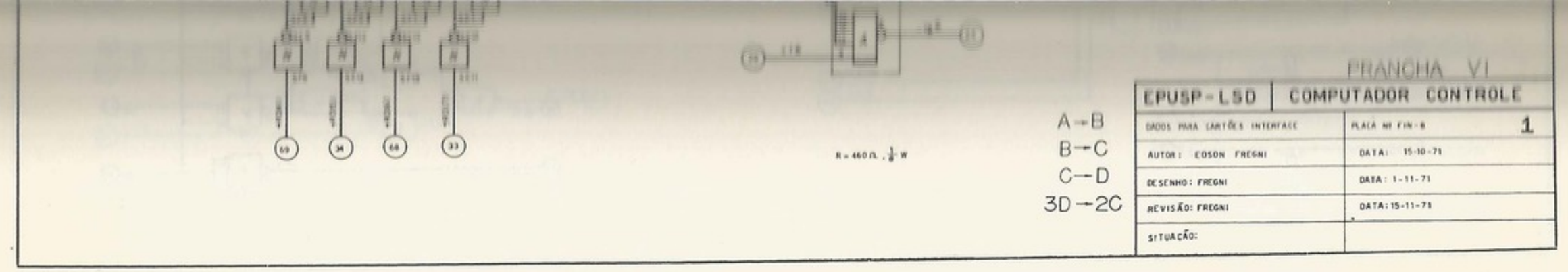

PRANCHA II
EPUSP LSD COMPUTADOR DE CONTROLE
CIRCUITO - UNIDADE ARITMÉTICA
5,6
AUTOR: VICTOR
DATA: 06-07-71
DESENHO: MAZZEI
DATA: 29-11-71
REVISÃO:
DATA:
1A → 2B
3A → 4B
5A → 2C


PRANCHA IV
LSD 7
EPUSP
PROJETO COMPUTADOR DE CONTROLE
FLUXO DE DADOS
MAIS UM E PORTA DE SELEÇÃO
AUTOR: EDSON
REVISÃO: CÉLIO
20-10-71
A→B
B→C
C→D
D→E

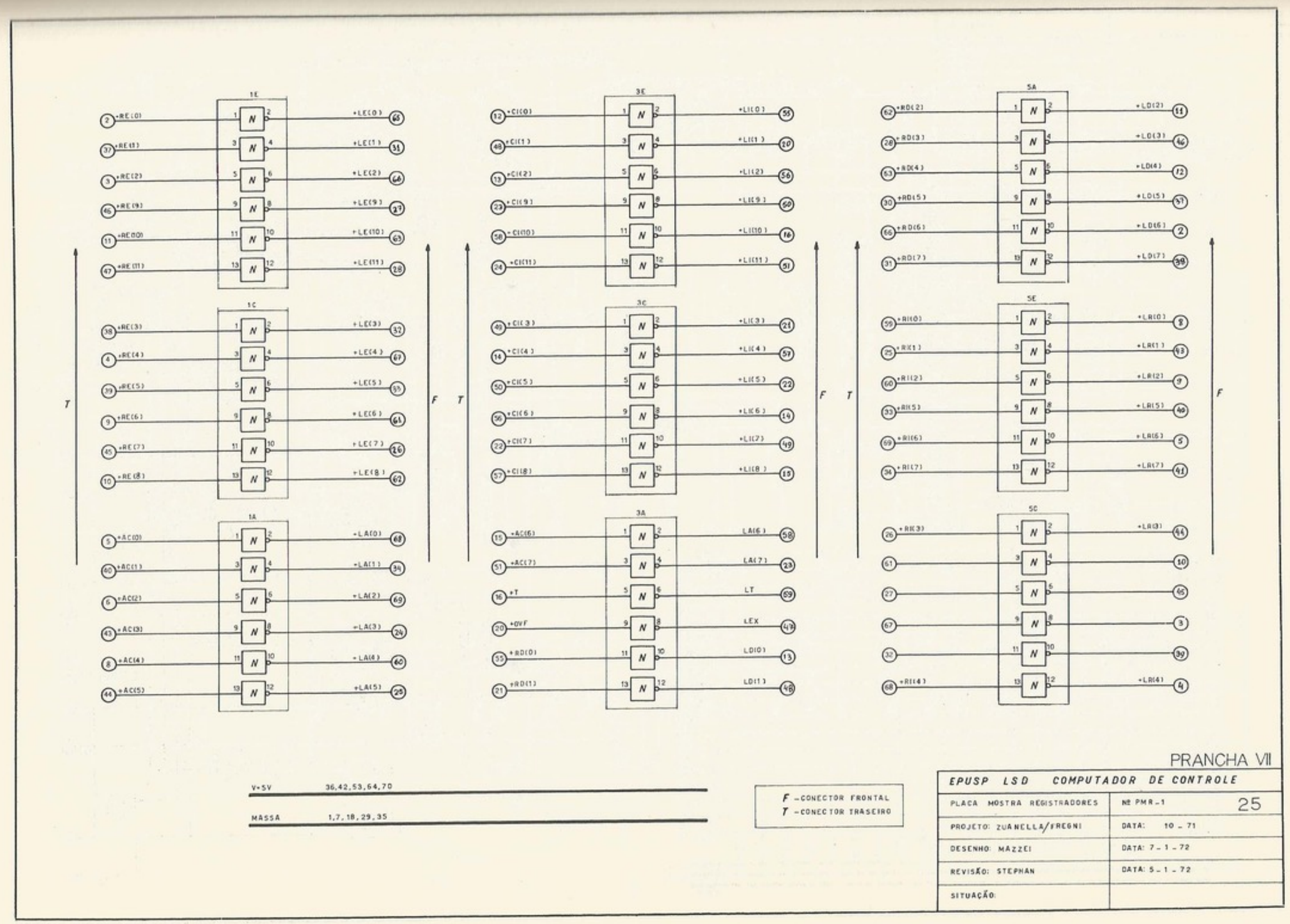

PRANCHA IV
EPUSP LSD
PROJETO COMPUTADOR DE CONTROLE
FLUXO DE DADOS
MAIS UM E PORTA DE SELEÇÃO
AUTOR: EDSON
REVISÃO: CÉLIO
20-10-71
A→B
B→C
C→D
D→E

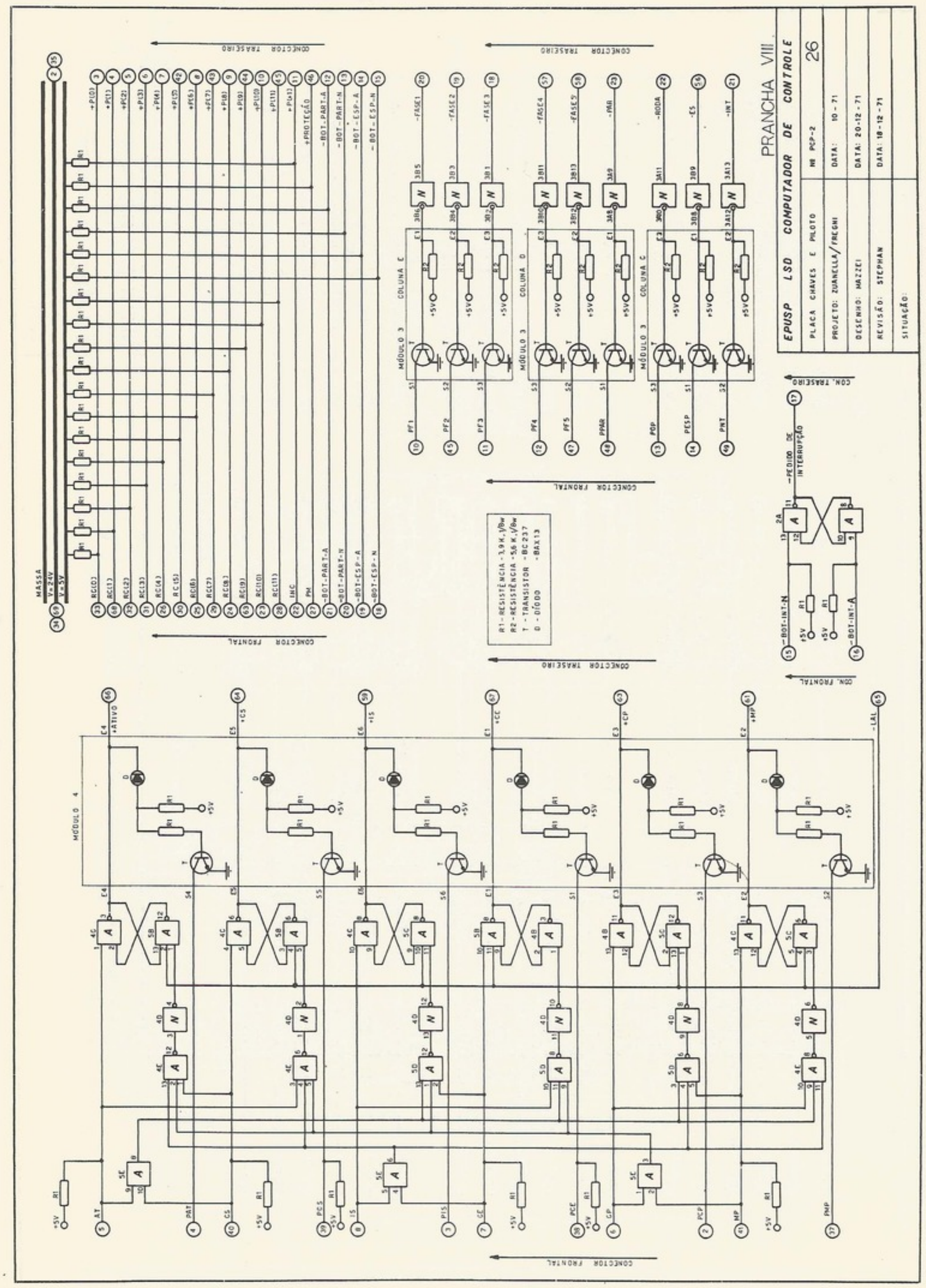

PRANCHA V
EPUSP LSD COMPUTADOR DE CONTROLE
PLACA CONTROLE DA MEMÓRIA
Nº FCM_7
PROJETO: EDITH/PATULLO
DATA: 8 - 71
DESENHO: MAZZEI
DATA 27 - 12 - 71
REVISÃO: ROGER
DATA: 3 - 1 - 72
SITUAÇÃO:
A→B
B→C
C→D

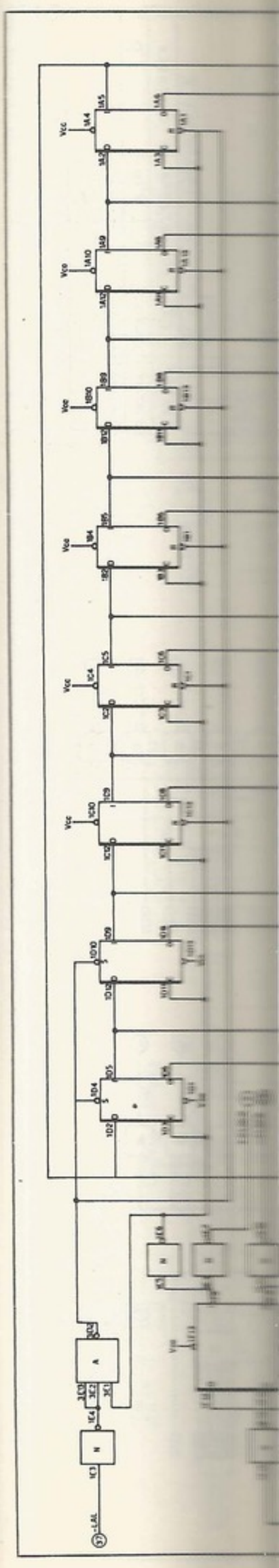

EPUSP-LSD
COMPUTADOR CONTROLE
PRANCHA VI
1
A→B
B→C
C→D
3D→2C

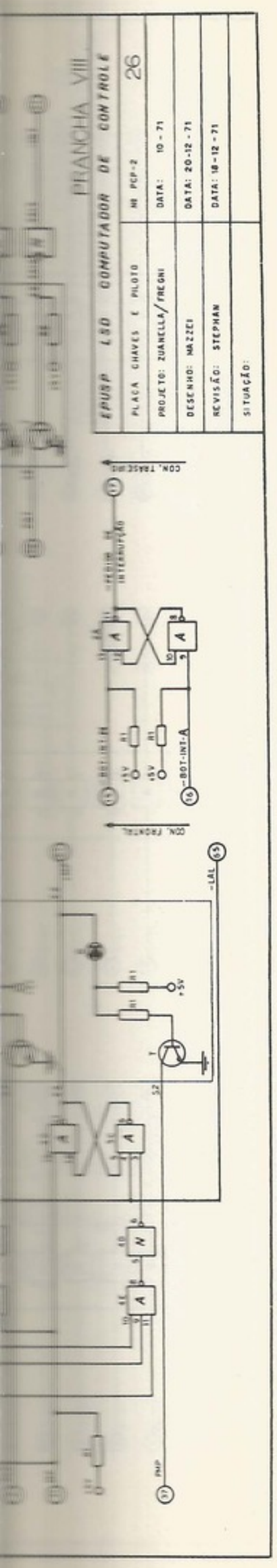


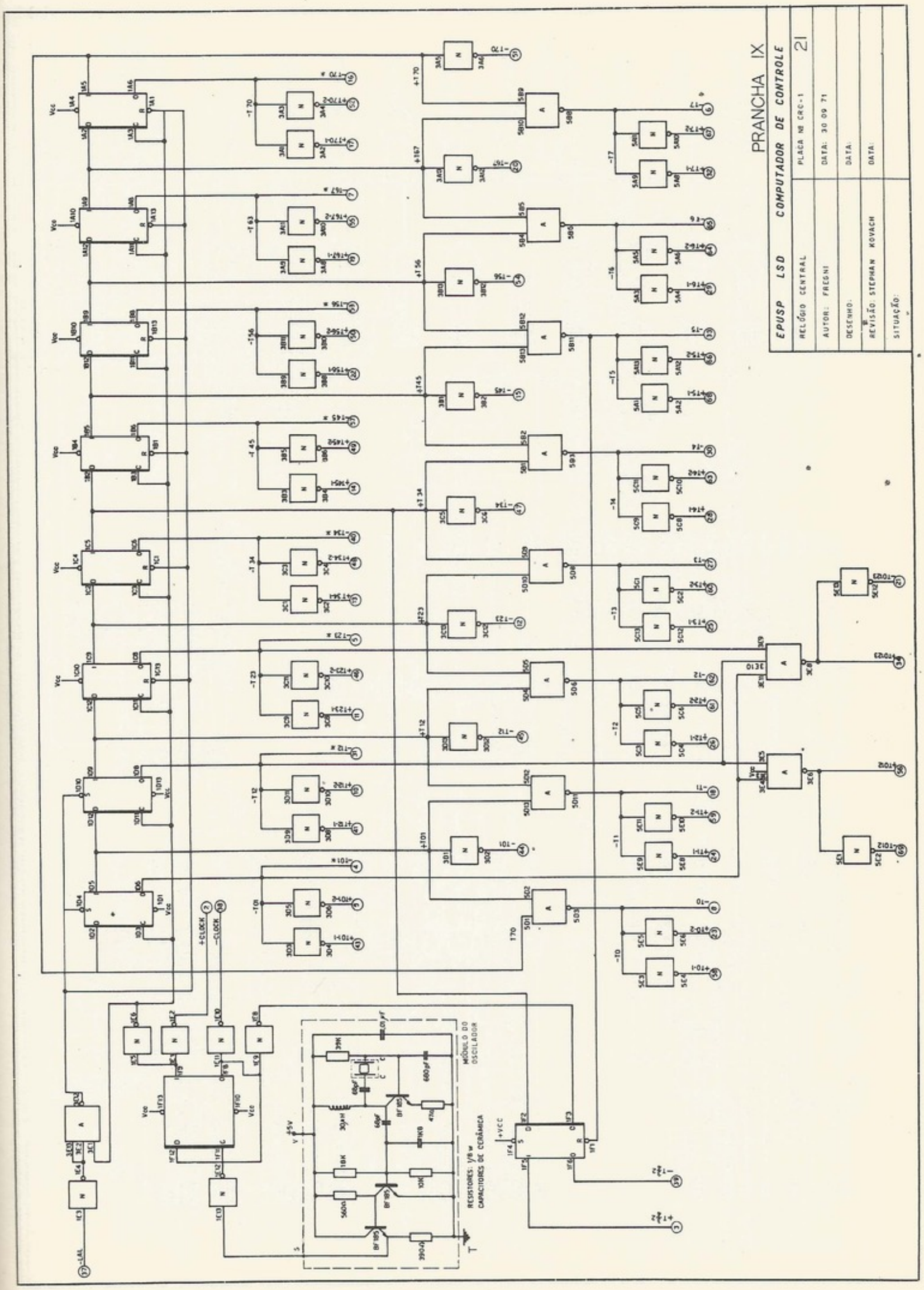
exemplo de um minicomputador

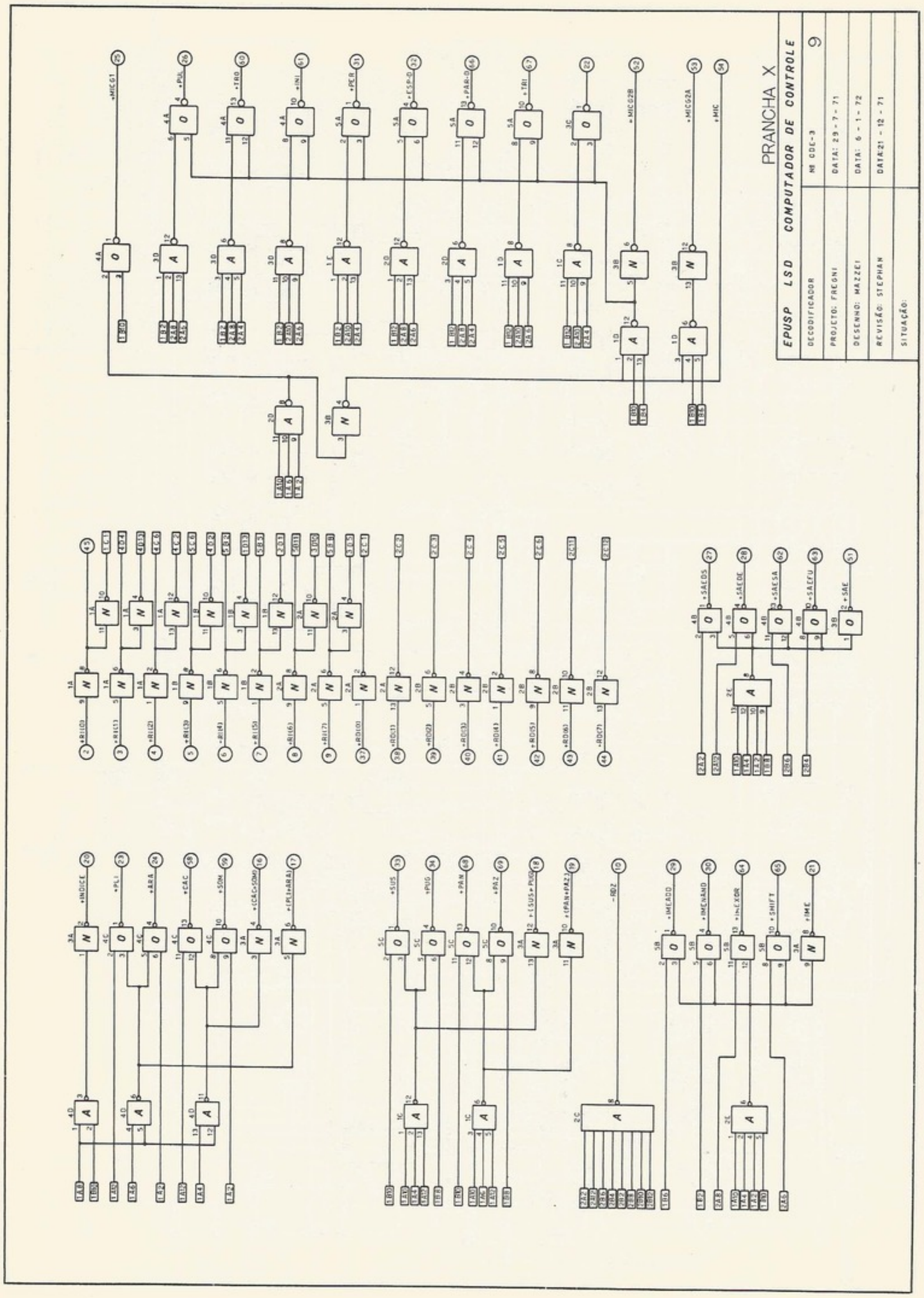


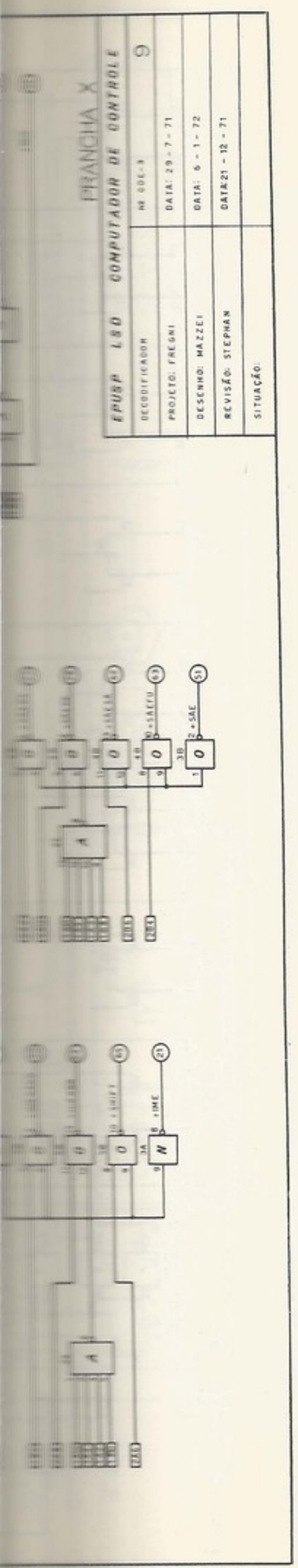

EPUSP LSD COMPUTADOR DE CONTROLE
PRANCHA VIII
PLACA CHAVES E PILOTO
Nº PCP-2
26
PROJETO: ZUANELLA/FREGNI
DATA: 10-71
DESENHO: MAZZEI
DATA: 20-12-71
REVISÃO: STEPHAN
DATA: 18-12-71
SITUAÇÃO:
MÓDULO 3
COLUNA C
COLUNA D
MÓDULO 4
MASSA
V=24V
V=5V
R1 - RESISTÊNCIA - 3,9 K.Ω/8w
R2 - RESISTÊNCIA - 5,6 K.Ω/8w
T - TRANSISTOR - BC 237
D - DÍODO - BAX13
CONECTOR FRONTAL
CONECTOR TRASEIRO
CON. FRONTAL
CON. TRASEIRO

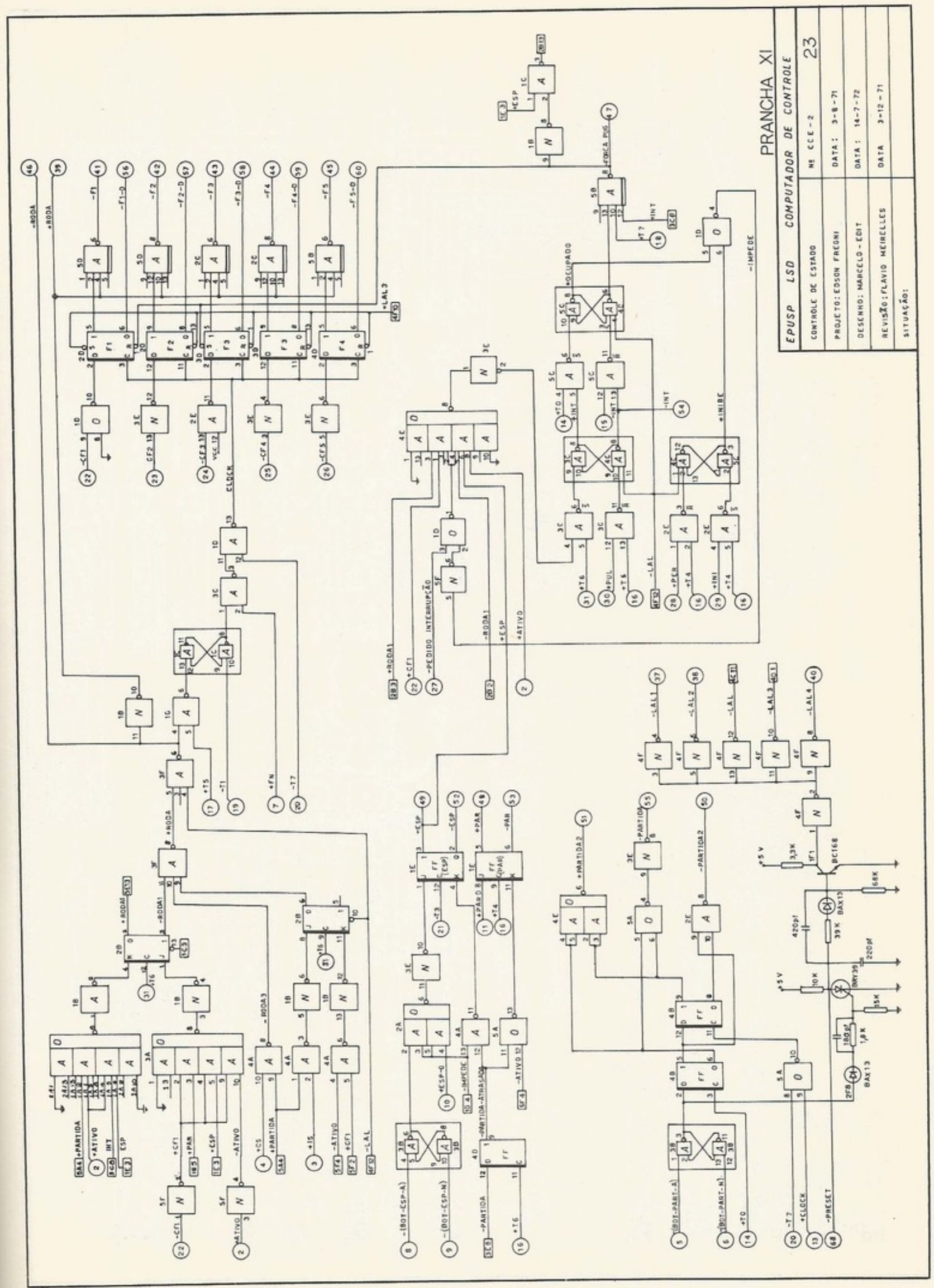


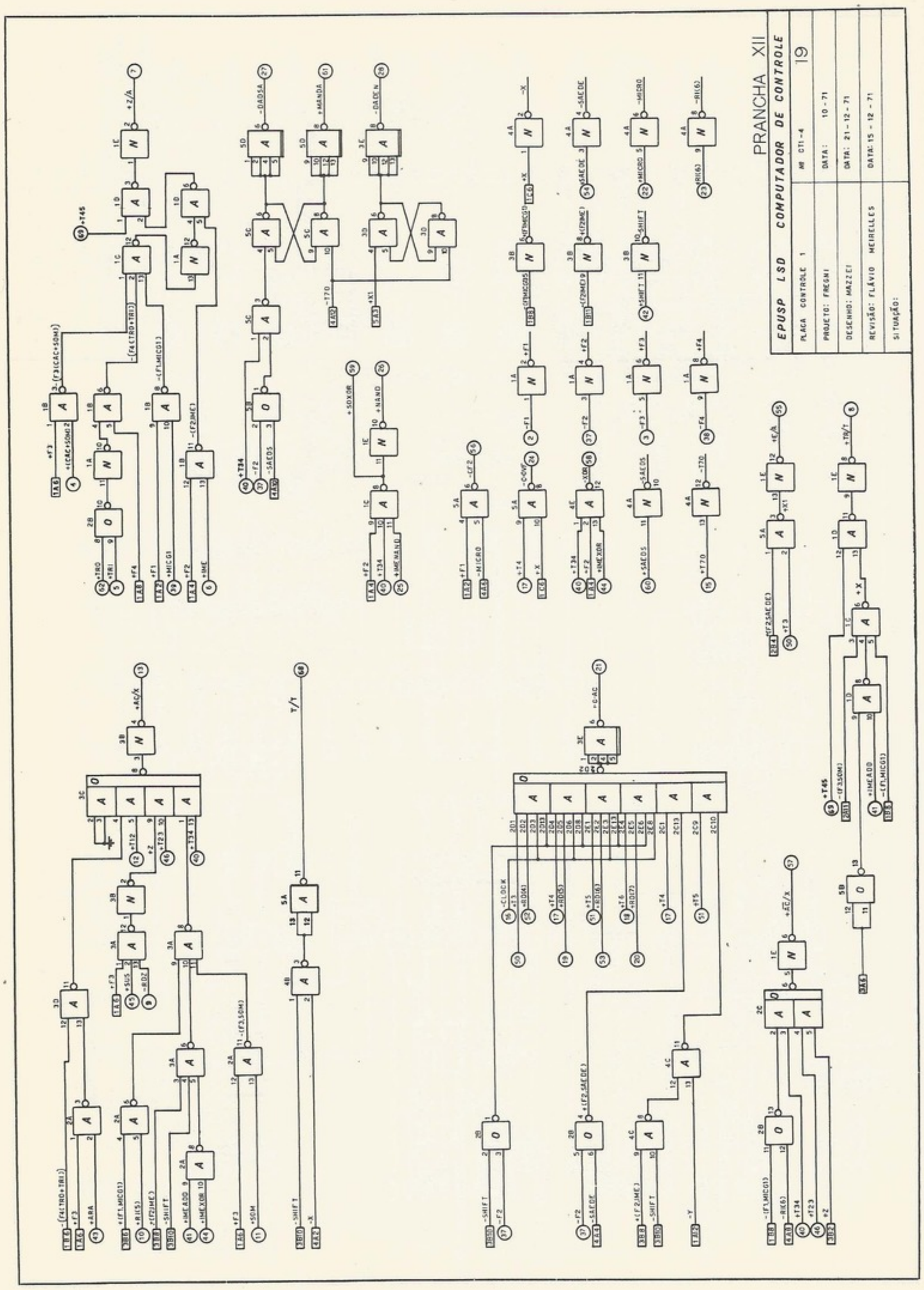

PRANCHA VIII
EPUSP LSD COMPUTADOR DE CONTROLE
PLACA CHAVES E PILOTO
NR PCP-2
26
PROJETO: ZUANELLA/FREGNI
DATA: 10 - 71
DESENHO: MAZZEI
DATA: 20-12-71
REVISÃO: STEPHAN
DATA: 18-12-71
SITUAÇÃO:
COM. TRANSEUNTE
COM. FRONTAL
PNP

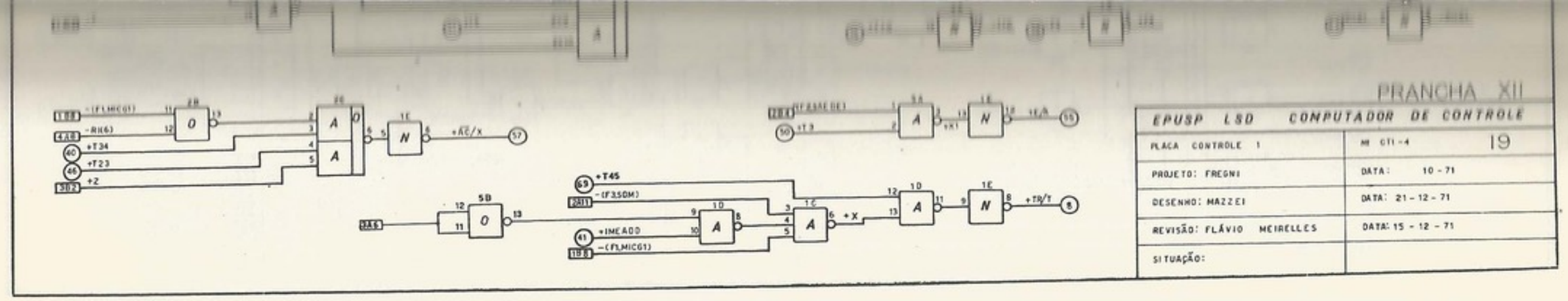

PRANCHA IX
EPUSP LSD COMPUTADOR DE CONTROLE
RELÓGIO CENTRAL
PLACA NR CRC-1
21
AUTOR: FREGNI
DATA: 30 09 71
DESENHO:
DATA:
REVISÃO: STEPHAN KOVACH
DATA:
SITUAÇÃO:
+CLOCK
-CLOCK
MÓDULO DO OSCILADOR
RESISTORES: 1/8 w
CAPACITORES DE CERÂMICA
+VCC

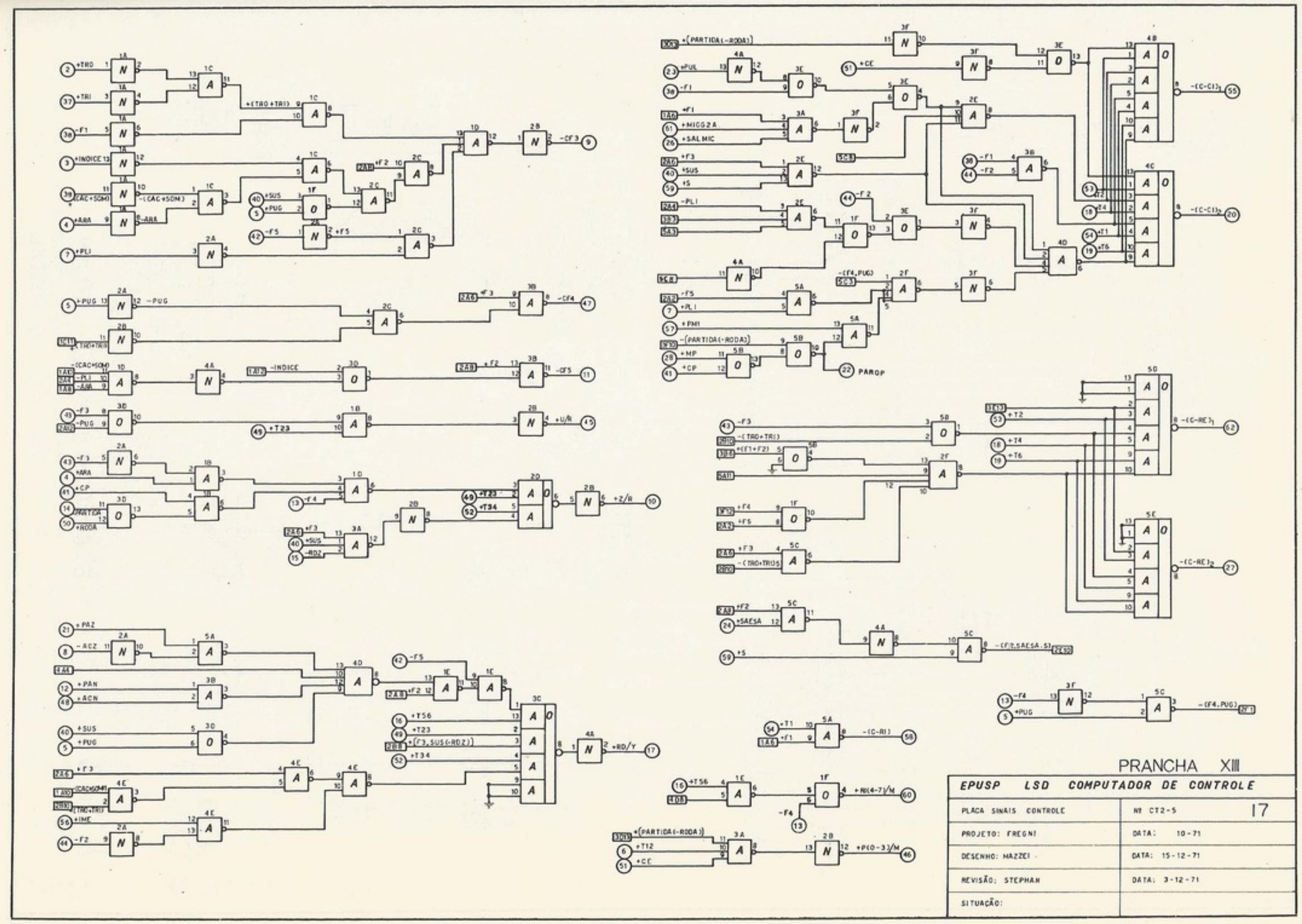

PRANCHA X
EPUSP LSD COMPUTADOR DE CONTROLE
DECODIFICADOR
Nº CDC-3
9
PROJETO: FRECNI
DATA: 29-7-71
DESENHO: MAZZEI
DATA: 6-1-72
REVISÃO: STEPHAN
DATA: 21-12-71
SITUAÇÃO:
6

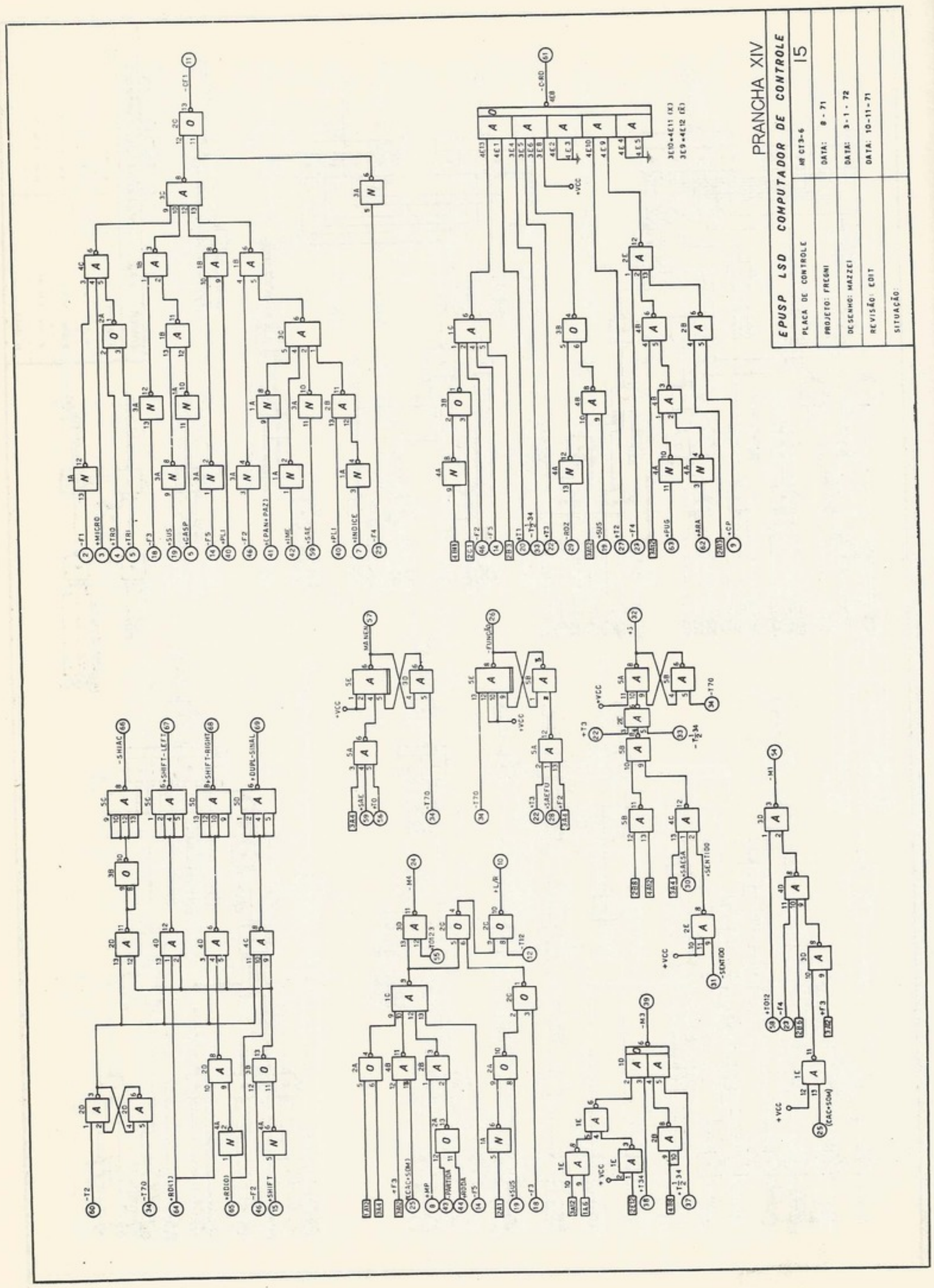

PRANCHA X
EPUSP LSD COMPUTADOR DE CONTROLE
DECODIFICADOR
PROJETO: FREGNI
DESENHO: MAZZEI
REVISÃO: STEPHAN
SITUAÇÃO:
Nº CCE-9
DATA: 29-7-71
DATA: 6-1-72
DATA: 21-12-71
9

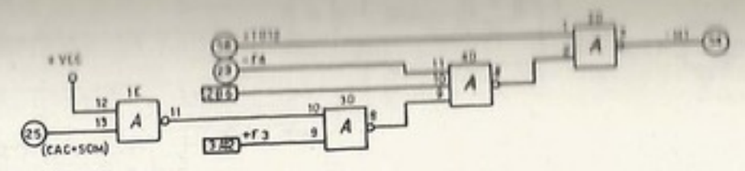

PRANCHA XI
EPUSP LSD COMPUTADOR DE CONTROLE
CONTROLE DE ESTADO
PROJETO: EDSON FREGNI
DESENHO: MARCELO-EDIT
REVISÃO: FLÁVIO MEIRELLES
SITUAÇÃO:
Nº CCE-2
DATA: 3-8-71
DATA: 14-7-72
DATA 3-12-71
23


PRANCHA XII
EPUSP LSD COMPUTADOR DE CONTROLE
19
PLACA CONTROLE 1
Nº CTI-4
PROJETO: FREGNI
DATA: 10-71
DESENHO: MAZZEI
DATA: 21-12-71
REVISÃO: FLÁVIO MEIRELLES
DATA: 15-12-71
SITUAÇÃO:


PRANCHA XII
EPUSP LSD COMPUTADOR DE CONTROLE
PLACA CONTROLE 1
Nº CT1-4
19
PROJETO: FREGNI
DATA: 10 - 71
DESENHO: MAZZEI
DATA: 21 - 12 - 71
REVISÃO: FLÁVIO MEIRELLES
DATA: 15 - 12 - 71
SITUAÇÃO:

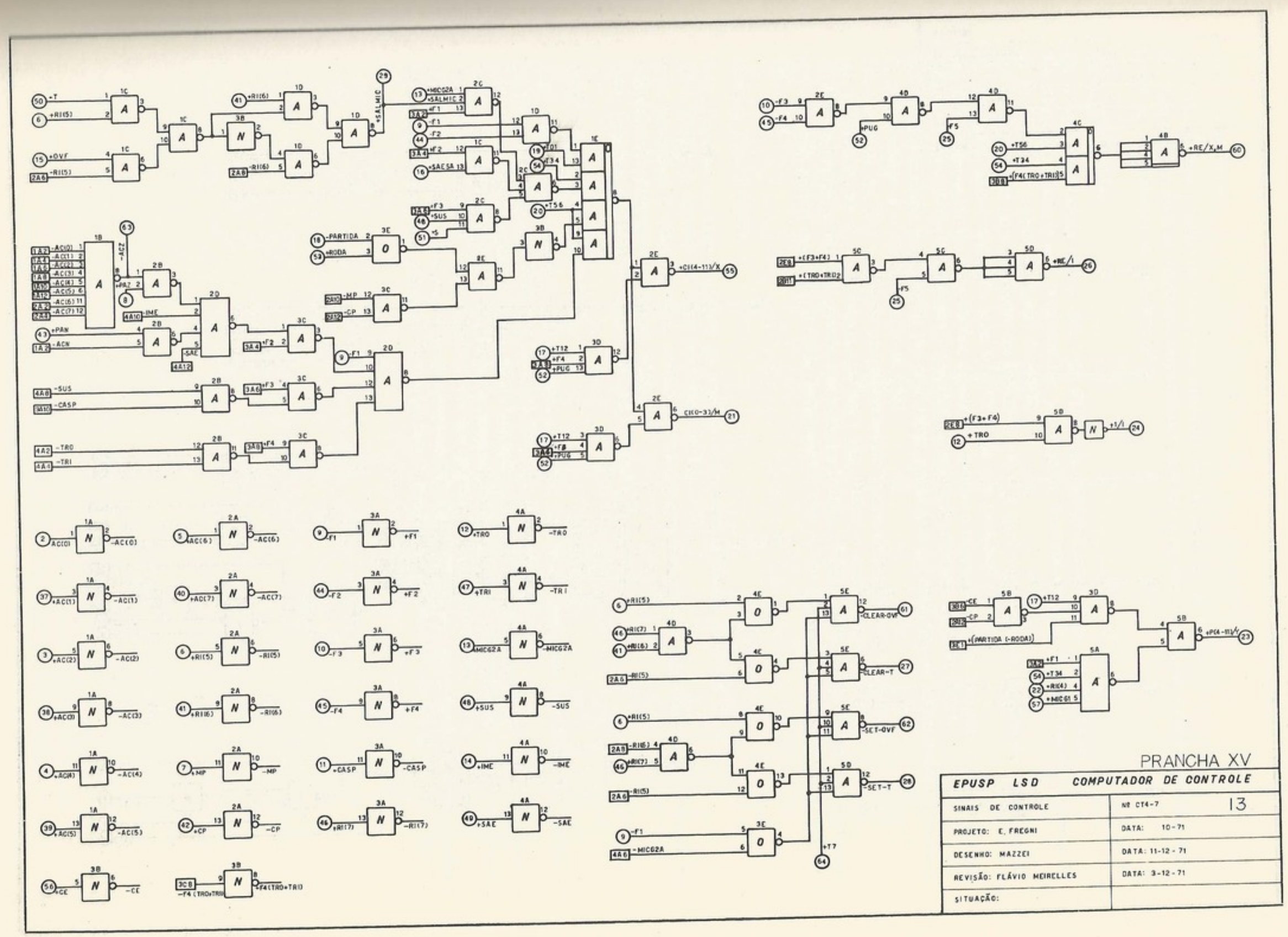

PRANCHA XIII
EPUSP LSD COMPUTADOR DE CONTROLE
PLACA SINAIS CONTROLE
Nº CT2-5
17
PROJETO: FREGNI
DATA: 10-71
DESENHO: MAZZEI
DATA: 15-12-71
REVISÃO: STEPHAN
DATA: 3-12-71
SITUAÇÃO:


PRANCHA XIV
EPUSP LSD COMPUTADOR DE CONTROLE
PLACA DE CONTROLE
Nº CT3-6
15
PROJETO: FREGNI
DATA: 8-71
DESENHO: MAZZEI
DATA: 3-1-72
REVISÃO: COIT
DATA: 10-11-71
SITUAÇÃO:



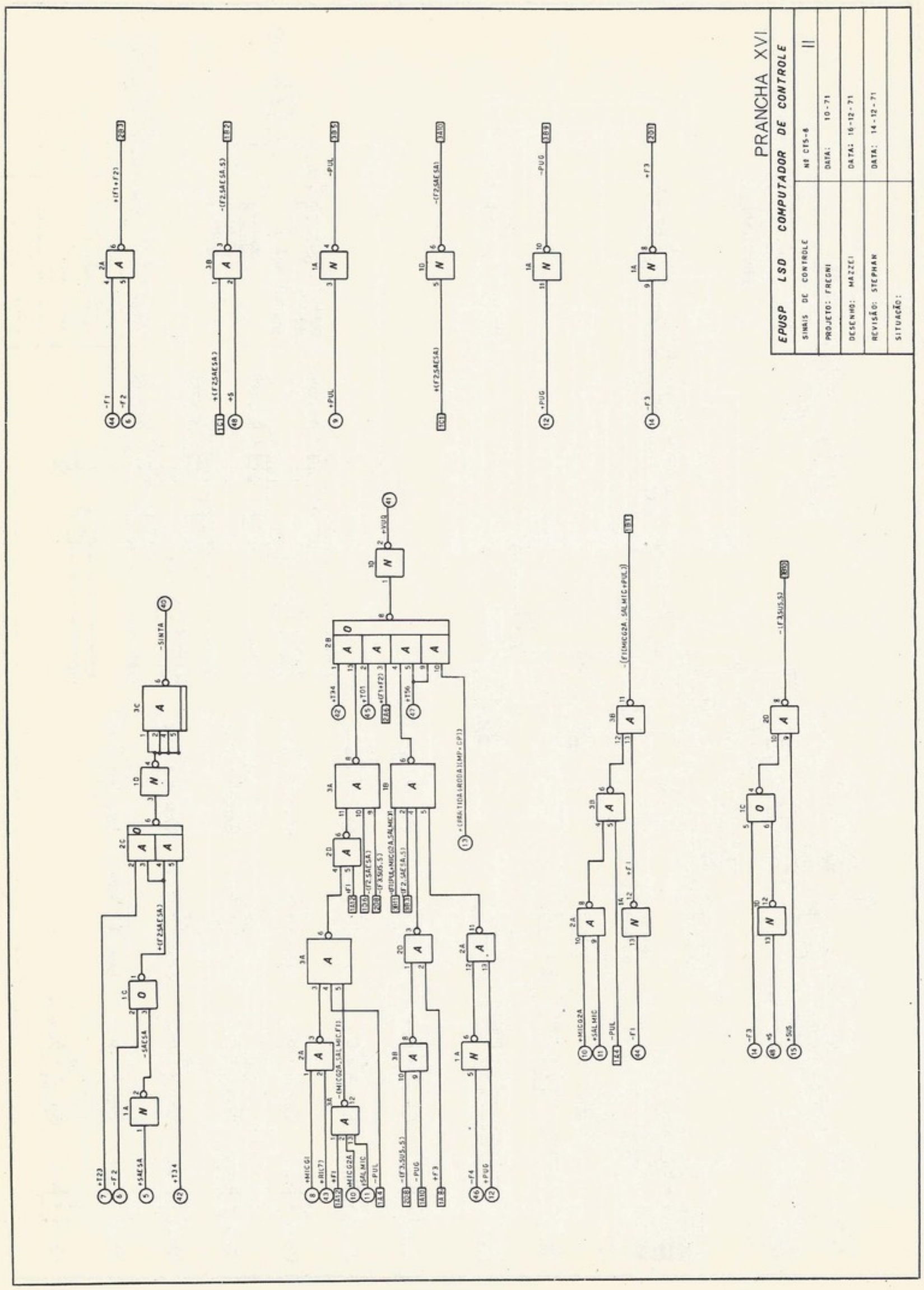
PRANCHA XIV

EPUSP LSD COMPUTADOR DE CONTROLE	
PLACA DE CONTROLE	Nº CT3-6 15
PROJETO: FREGNI	DATA: 8 - 71
DESENHO: MAZZEI	DATA: 3 - 1 - 72
REVISÃO: EDIT	DATA: 10-11-71
SITUAÇÃO:	



PRANCHA XV

EPUSP LSD COMPUTADOR DE CONTROLE	
SINAIS DE CONTROLE	Nº CT4-7 13
PROJETO: E. FREGNI	DATA: 10-71
DESENHO: MAZZEI	DATA: 11-12-71
REVISÃO: FLÁVIO MEIRELLES	DATA: 3-12-71
SITUAÇÃO:	


PRANCHA XVI
EPUSP LSD COMPUTADOR DE CONTROLE
SINAIS DE CONTROLE
Nº CFS-6
II
PROJETO: FREGNI
DATA: 10-71
DESENHO: MAZZEI
DATA: 16-12-71
REVISÃO: STEPHAN
DATA: 14-12-71
SITUAÇÃO:

EXERCÍCIOS

Sobre a arquitetura

5-1. Projete uma arquitetura para uma memória de 1*K* palavras de 8 *bits*.
5-2. O que se perderia se usássemos um somador de 4 *bits* com velocidade igual ao dobro do 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, incluísse 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-se 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*.