CPgEE/CCT-UFPb
COORDENAÇÃO DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA
CENTRO DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE FEDERAL DA PARAÍBA

Parecer final do julgamento da dissertação de mestrado

de PAULO ROBERTO CAMPOS DE ARAÚJO

Título: "PROSAD um Sistema de Aquisição de Dados e Controle de Processos a Microprocessadores"

Conceito obtido: APROVADO

Comissão Examinadora

Prof. Joberto Sérgio Barbosa Martins - M.Sc.
-Presidente-

Prof. Ivan Rocha Neto - Ph.D.

Prof. José Homero Feitosa Cavalcanti - M.Sc.

Prof. Wanderley Lopes de Souza - Dr. Ing.

Campina Grande 24 de janeiro de 1981.

Endereço Postal: Caixa Postal, 518 –58.100 – Campina Grande – Paraíba – Brasil
Telex: 0832211 – Telefone: DDD (083) - 321-0655 – Ramal 133

PROSAD. UM SISTEMA DE AQUISIÇÃO DE DADOS E CONTROLE DE PROCESSOS A MICROPROCESSADOR

POR

PAULO ROBERTO CAMPOS DE ARAÚJO

TESE DE MESTRADO

Apresentada à Coordenação Setorial de Pós-Graduação e Pesquisa da Pró-Reitoria para Assuntos do Interior da Universidade Federal da Paraíba, em cumprimento às exigências para obtenção do grau de Mestre em Ciências.

CAMPINA GRANDE, JANEIRO DE 1981

A663p Araújo, Paulo Roberto Campos de.
PROSAD, um sistema de aquisição de dados e controle de processos a microprocessador / Paulo Roberto Campos de Araújo. - Campina Grande, 1981.
116 f.

Dissertação (Mestrado em Engenharia Elétrica) - Universidade Federal da Paraíba, Centro de Ciências e Tecnologia, 1981.
"Orientação : Prof. Dr. Joberto Sérgio Barbosa Martins".
Referências.

1. Controle de Processos - Engenharia Elétrica. 2. Automação Industrial. 3. Microprocessadores. 4. PROSAD - Sistema de Aquisição de Dados. 5. Engenharia Elétrica - Dissertação. I. Martins, Joberto Sérgio Barbosa. II. Universidade Federal da Paraíba - Campina Grande (PB). III. Título

CDU 621.3:004.451.25(043)

ÍNDICE GERAL

PÁG.

PÁG.

PÁG.

CAPÍTULO IV

DEDICATÓRIA

Dedico este trabalho à
minha filha *Pollyanna*,
minha esposa *Leny*,
minha mãe *Lindomar*
e a todos os que fazem uma Universidade.

RESUMO

O aparecimento dos microprocessadores trouxe mudanças relevantes nas técnicas de controle de processos. Devido ao baixo custo e às reduzidas dimensões dos mesmos, o controle computarizado, antes restrito às aplicações mais caras e sofisticadas, pôde vir a ser utilizado em quase todos os níveis de complexidade.

Este trabalho apresenta o hardware e e software do PROSAD, um sistema de aquisição de dados e controle de processos configurado no modo stand alone. Tal sistema é baseado no microprocessador MC 6800 da Motorola. Seu hardware foi projetado de forma a dar-lhe uma relativa flexibilidade permitindo-se utilizá-lo em vários tipos diferentes de aplicação: Controle de processos industriais de um ou vários loops, aquisição de dados em geral, controle numérico, automação de laboratórios e etc. O sistema possui uma interface de vias que permite o uso de várias interfaces de E/S. O software do sistema consiste de um macro assembler e um núcleo de gerenciamento de operação em tempo real. O macro assembler possui 16 instruções que representam algumas das operações do microprocessador. O núcleo de gerência possui um executivo e um escalonador de tarefas estático.

ABSTRACT

The appearance of microprocessors has brought relevant changes in the process control techniques. Due to their small syze and their low cost, computerized control systems, so for restricted to the more expensive and sofisticated applications, has come to be used in almost all levels of complexity.

This report presents the hardware and software of the PROSAD, a stand alone data acqquisition and process control system based on the Motorola's MC 6800 microprocessor. The system hardware was projected to give it a relative flexibility in order to suit it to many different types of applications: Single and multi loop control of industrial processes, general data acqquisition, numerical control, laboratory automation and e tc. The system has a bus interface that allows the use of various different types of I/O interfaces. The system software consists of a macro assembler and kernell for management of the system real time operation. The macro assembler contains a 16 instructions vocabulary that represent some of the most used operations of the microprocessor. The kernell has an executive and a fixed priority resume scheduller.

AGRADECIMENTOS

Os mais cinceros agradecimentos ao professor e colega *Joberto Martins* pela sua valiosa orientação, bem como ao professor e colega *José Homero Feitosa* pela forma com que esteve disposto à discussão de alguns tópicos deste trabalho.

À minha mãe, que sempre acreditou no meu trabalho.

CAPÍTULO I

OS COMPUTADORES E A AUTOMAÇÃO INDUSTRIAL

1 – INTRODUÇÃO

No fim da década de 40, segundo alguns autores (12), começaram a ser utilizados os primeiros controladores pneumáticos nas indústrias de processos. Se iniciava desta maneira um processo que viria de forma cada vez mais acelerada e profunda, modificar os aspectos tecnológicos desta indústria, através da automação cada vez mais presente nas mesmas. A partir da década de 60 os dispositivos semicondutores começaram a ser introduzidos nestes controladores de processo, determinando dentro deste processo evolutivo o início de uma etapa de mudanças ainda mais profundas, determinadas pelo aparecimento dos circuitos integrados, dos amplificadores operacionais e finalmente dos computadores.

Os computadores foram utilizados pela primeira vez em indústrias de processos no início dos anos 60. Os mesmos trouxeram consigo a possibilidade de intensificar fortemente este processo evolutivo. Possuíam em contrapartida o inconveniente de um alto custo restringindo a automação aos processos mais sofisticados e caros. O surgimento dos microprocessadores veio no entanto resolver este impasse. Através dos mesmos pôde-se então trazer o computador até aos níveis mais baixos de um sistema de controle de processos. Hoje, suas formas de aplicação neste campo são as mais variadas possíveis podendo-se citar, sistemas de controle supervisório para usinas nucleares e outros sistemas de geração de energia, controle numérico de prensas, tornos, robots, motores, sistemas de aquisição de dados, controladores de tráfego, controladores de radares, sistemas de controle de processos químicos e siderúrgicos e etc.

1.1 – Formas de Utilização de um Computador em Controle de Processos

Computadores de processo (dá-se esse nome aos computadores configurados especialmente para atuar em controle de processos), constituem a parte central dos sistemas de controle de processos. Suas formas típicas de aplicação variam desde a execução de tarefas de aquisição de dados com o processamento em "off line" dos parâmetros obtidos até o controle "on line" em tempo real. Sua arquitetura é similar à de um computador convencional e consiste de:

- Unidade central de processamento
- Unidade de memória
- Ports de E/S

Uma de suas principais características é a modularidade, permitindo uma flexibilidade de sua configuração de forma a que o mesmo se ajuste a diversos tipos de aplicação.

Existem três modos típicos de aplicação de um computador como elemento de controle:

- "off line"
- "on line" em loop aberto
- "on line" em loop fechado

No modo off line o computador se encontra fisicamente isolado do processo. Os parâmetros do mesmo são lidos pelo operador introduzidos e processados pelo computador e os resultados são usados pelo operador para exercer manualmente, a ação de controle (Fig. 1.2).

No modo on line em loop aberto o computador adquire os dados diretamente no processo. Depois de processados os dados os resultados são usados pelo operador para controlar o processo manualmente (Fig. 1.3).

No modo on line em loop fechado o computador é diretamente conectado tanto à entrada quanto à saída do processo. Os ciclos de aquisição, processamento e controle das variáveis do mesmo são todos executados pelo próprio computador (Fig. 1.4).

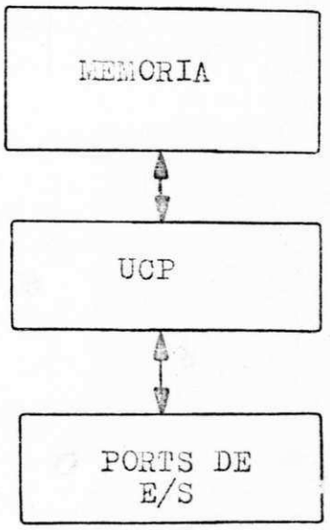


FIGURA 1.1 Arquitetura de um computador de processos

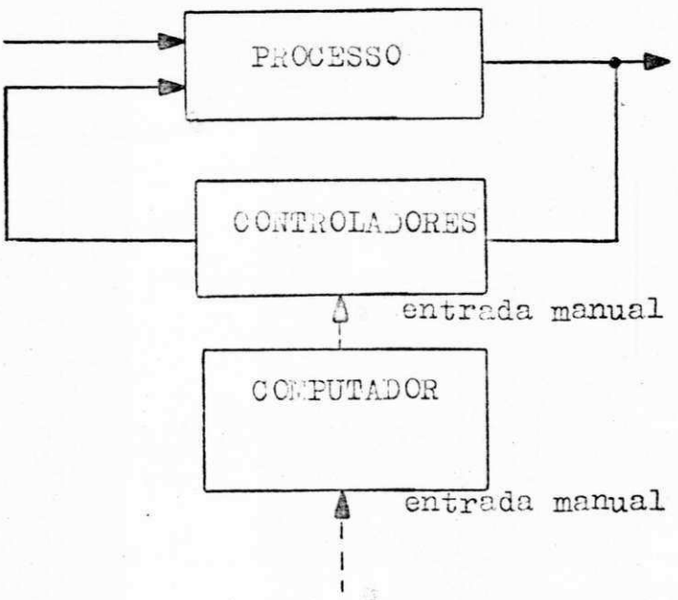


FIGURA 1.2 Controle em"OFF LINE"

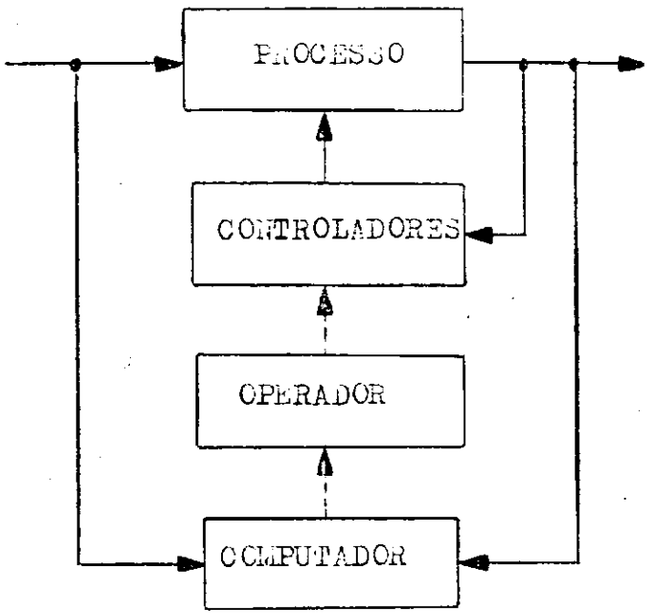


FIGURA 1.3 Modo de controle ON LINE em loop aberto

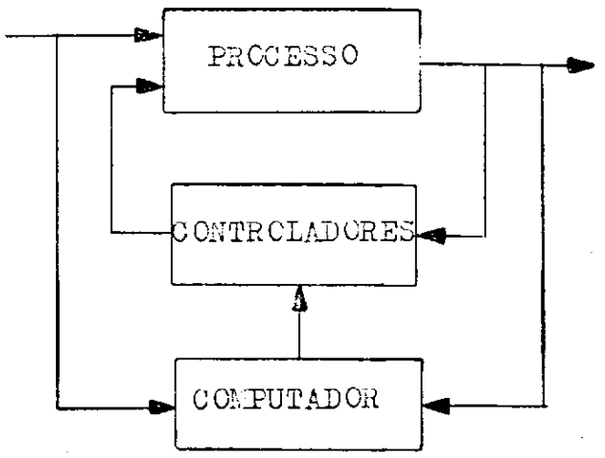


FIGURA 1.4 Modo de controle ON LINE em loop fechado

1.2 – Comunicação Física de um Computador com o Processo

No modo de operação on line em loop fechado viu-se que o computador é ligado fisicamente ao processo tanto através de canais ou linhas de aquisição de dados quanto através de linhas de controle das variáveis deste processo. Desde os anos 60 quando os computadores começaram a ser utilizados em indústrias de processo (12), foram se configurando dois tipos diferentes de ligação física entre as mesmas e o processo, representando duas estratégias diferentes quanto ao seu uso como elemento de controle, são elas:

a) controle digital direto – DDC – (direct digital control)

b) controle supervisório – CSC – (computer supervisory control)

controle digital direto

Um sistema DDC pode ser definido da seguinte maneira: "O computador de controle mede diretamente as variáveis associadas ao processo. Tais medidas são então usadas em conjunto com o algoritmo de controle na geração dos sinais de controle necessários à manutenção dos "set points" de cada loop de controle do processo". Isto implica que o computador executa também o algoritmo de controle das variáveis, atuando o mesmo diretamente sobre os atuadores de controle (válvulas senoidais, aquecedores e etc.) (Fig. 1.5).

Os primeiros sistemas deste tipo se apresentavam acessí

veis apenas para aplicações mais sofisticadas, uma vez que além do computador ser muito caro, havia também o custo do cabeamento entre o mesmo e os atuadores e sensores do processo. Gradualmente, no entanto, tais computadores foram sendo substituídos por mini-computadores mais baratos, menores e mais fáceis de serem instalados. Além desta evolução, podem se citar duas outras:

- O uso de multiplexadores e demultiplexadores analógicos e conversores A/D e D/A diminuiu consideravelmente o custo da ligação física entre o computador e o processo além de minimizar os efeitos do ruído nas linhas de ligação, uma vez que a transmissão passou a ser feita digitalmente;
- A instalação de estações manuais, estações locais ou estações de operadores, melhorou a confiabilidade do sistema pois, na eventualidade de falha do computador, cada loop de controle podia ser monitorado e controlado localmente pelo operador.

controle supervisório (CSC)

O controle supervisório pode ser definido como: "O processo é controlado por controladores individuais localizados junto ao mesmo. A função do computador é meramente supervisória, ou seja, o mesmo faz a medição das variáveis associadas ao processo e gera os set points para os controladores. Os algoritmos de controle das variáveis são executados localmente por cada controlador" (Fig. 1.6). Os primeiros sistemas de controle supervi

sório eram mais caros que os de controle digital direto uma vez que além do computador sensores e atuadores eram necessários ainda um controlador para cada loop do processo.

A evolução sofrida nas técnicas de controle supervisório acompanhou o ritmo sofrido também pelas técnicas de controle digital direto. Os problemas inerentes à ligação física entre o computador e o processo também foram minimizados com o uso de multiplexadores, demultiplexadores e conversores A/D e D/A.

O problema da confiabilidade do sistema com relação a falha do computador é minimizado no modo CSC, uma vez que cada controlador permite o ajuste local do set point. Além disso estes controladores possuem geralmente recursos de monitoração da variável sob controle através de mostradores de set point e saída de controle. O problema da taxa de amostragem das variáveis do processo, existente no modo DDC uma vez que esta amostragem é feita continuamente por cada controlador.

1.3 – Os Microprocessadores em Controle de Processos

O advento dos microprocessadores no início da década de 70 trouxe a possibilidade de levar a computação à quase todos os níveis de um sistema de controle computarizado. O seu baixo custo aliado a sua flexibilidade veio permitir um avanço considerável nas técnicas de implementação destes sistemas de controle. Na técnica de controle digital direto em algumas aplicações pode-se até utilizá-lo em substituição aos minicomputadores. Existe neste caso, no entanto, uma maior limitação imposta

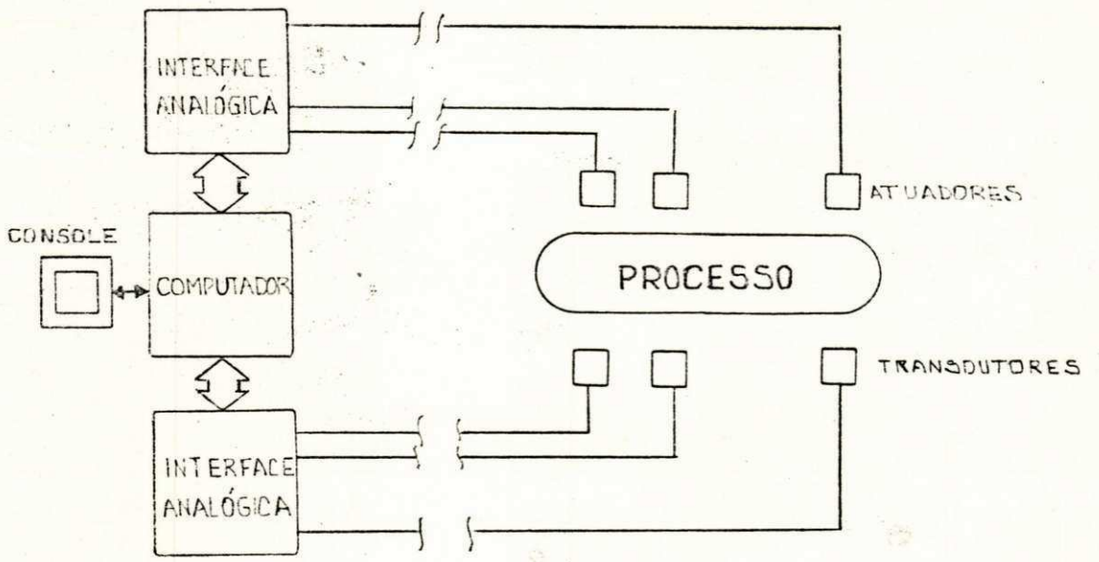


FIGURA 1.5 Sistema de controle digital direto

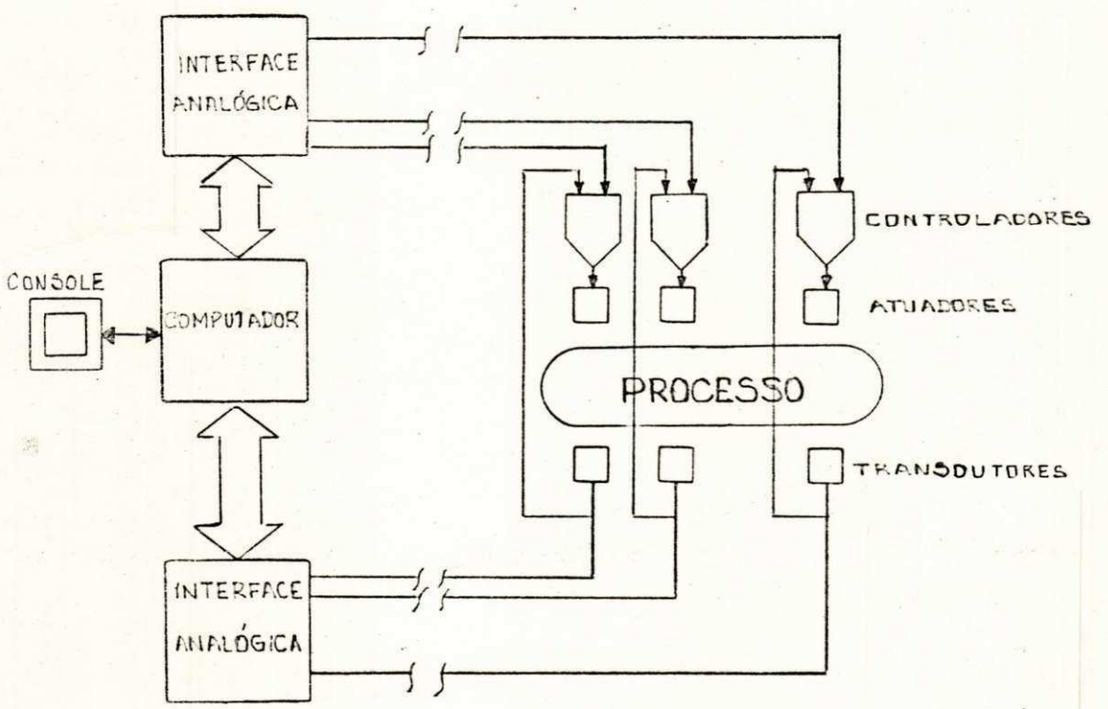


FIGURA 1.6 Sistema de controle supervisório

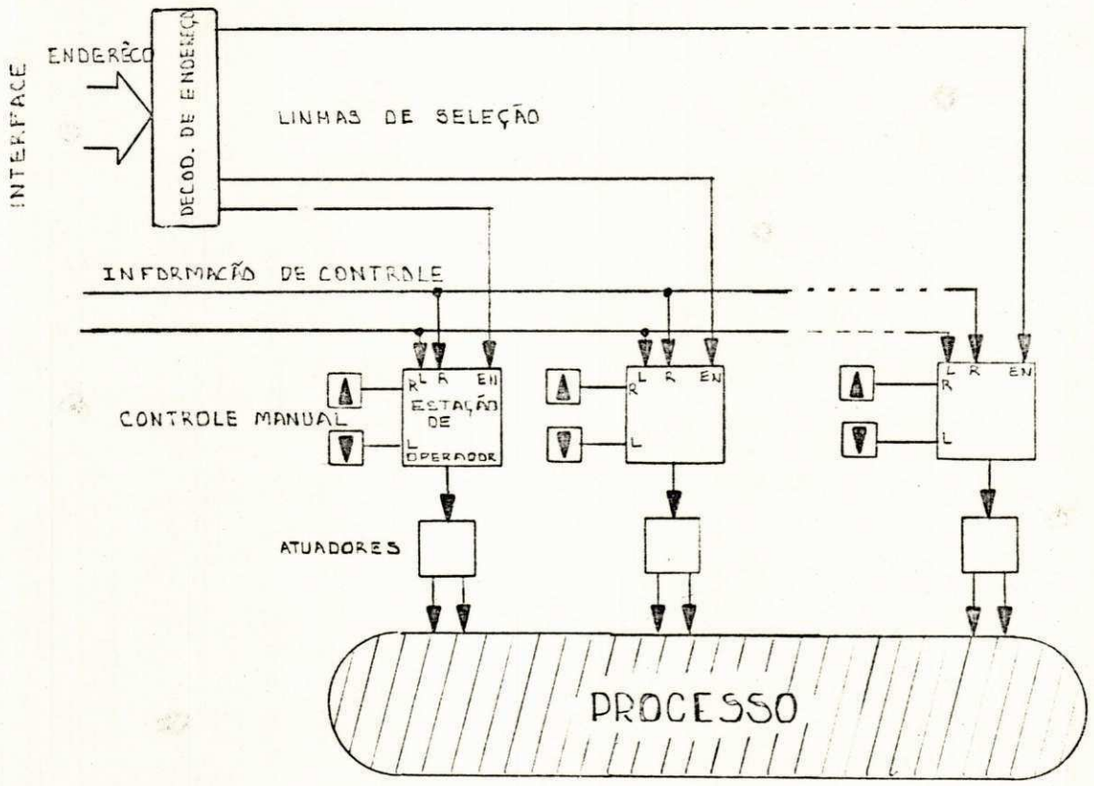


FIGURA 1.7 Saída de um sistema de controle DDC usando estações de operação local

pelo mesmo com relação à taxa de amostragem das variáveis do processo e da execução dos algoritmos de controle. Fabricantes de sistemas de controle do tipo DDC calcularam para os primeiros microprocessadores, baseados na velocidade dos mesmos, um número máximo de 32 loop's de controle em aplicações típicas. Tais sistemas têm sido cada vez mais utilizados, tendo apresentado bons resultados.

Em controle supervisório o quadro é o mesmo. Gradualmente os controladores analógicos convencionais têm sido substituídos por controladores digitais a base de microprocessadores. Além disto os microprocessadores criaram também a possibilidade de se implementar novos tipos de sistema: os sistemas de processamento distribuído. Pode-se definir tais sistemas como malhas de microprocessadores, minicomputadores ou computadores, interligados entre si possuindo cada um, uma ou mais tarefas computacionais específicas hierarquicamente distribuídas.

Neste tipo de sistema o computador central ou master, fica liberado de tarefas como aquisição de dados do processo, e execução de alguns algoritmos específicos de controle, ficando com a tarefa de gerenciar a operação dos diversos microprocessadores, implementar os algoritmos de controle mais gerais e servir de interface entre o operador do sistema e o processo, imprimindo relatórios, e outros tipos de informação sobre o mesmo (Fig. 1.8).

Costuma-se chamar tais sistemas de "sistemas de inteligência distribuída". Nos mesmos os microprocessadores são in

teiramente subordinados ao computador central, limitando-se a executar tarefas solicitadas pelo mesmo, recebendo muitas vezes junto com as interfaces de aquisição de dados e geração dos sinais dos controladores ou atuadores, o nome de "periférico inteligente de aquisição de dados e controle". Um exemplo típico de um sistema deste tipo é o sistema HP2240A da Hewlett Pachard.

O uso destes periféricos tem se diversificado cada vez mais, com a tendência cada vez maior de se utilizar o controle distribuído. Como reflexo desta evolução vêem-se a cada dia mais e mais sistemas deste tipo serem lançados no mercado, podendo-se citar o CAMAC da Kinetic Corp.; ANDS 7000 da Analogic Corp. e o compu DAS da Signal Laboratories. Alguns destes sistemas possuem um sistema operacional que pode permitir a sua operação sem a intervenção de um computador central. Neste caso diz-se que os mesmos operam no modo "stand alone".

CAPÍTULO II

HARDWARE DO SISTEMA

Este Capítulo apresenta o hardware básico do PROSAD, que se constitui de uma unidade central de processamento, circuitos de clock e reset, unidade de memória, interfade de vias e alguns cartões de E/S. O projeto da interface para TTY (RS 232 C e interface em loop de corrente -20A) ou TTY não será apresentado, uma vez que seu projeto é simples já havendo uma boa documentação a este respeito.

2 — INTRODUÇÃO

O PROSAD é um sistema de aquisição de dados e controle de processos controlado a microprocessador, com fins gerais de aplicação. Foi concebido, segundo a filosofia de sistemas co

mo o CAMAC da Kinetik Corp. (USA), o compuDAS da Signal Laboratories e o HP 2240A da Hewllet Pachard (USA). Suas possibilidades de utilização são muitas, podendo serem citadas:

- controle de processos industriais;
- controle numérico;
- aquisição de dados de características gerais como, dados meteorológicos;
- aplicações em engenharia biomédica;
- automação de laboratórios de pesquisa.

Atualmente o mesmo se encontra configurado no modo STAND ALONE, ou seja, opera sem estar sob comando de um outro computador de maior porte. Complementações no seu hardware e seu software podem ser feitas no sentido de que o mesmo opere como processador tipo "FRONT END" em sistemas distribuídos.

Suas principais características são:

- um hardware que permite ao sistema operar com até 256 diferentes circuitos de aquisição de dados e controle, podendo estes circuitos serem de 8 ou 12 bits;
- uma linguagem de programação própria, superior aos assemblers dos microprocessadores.

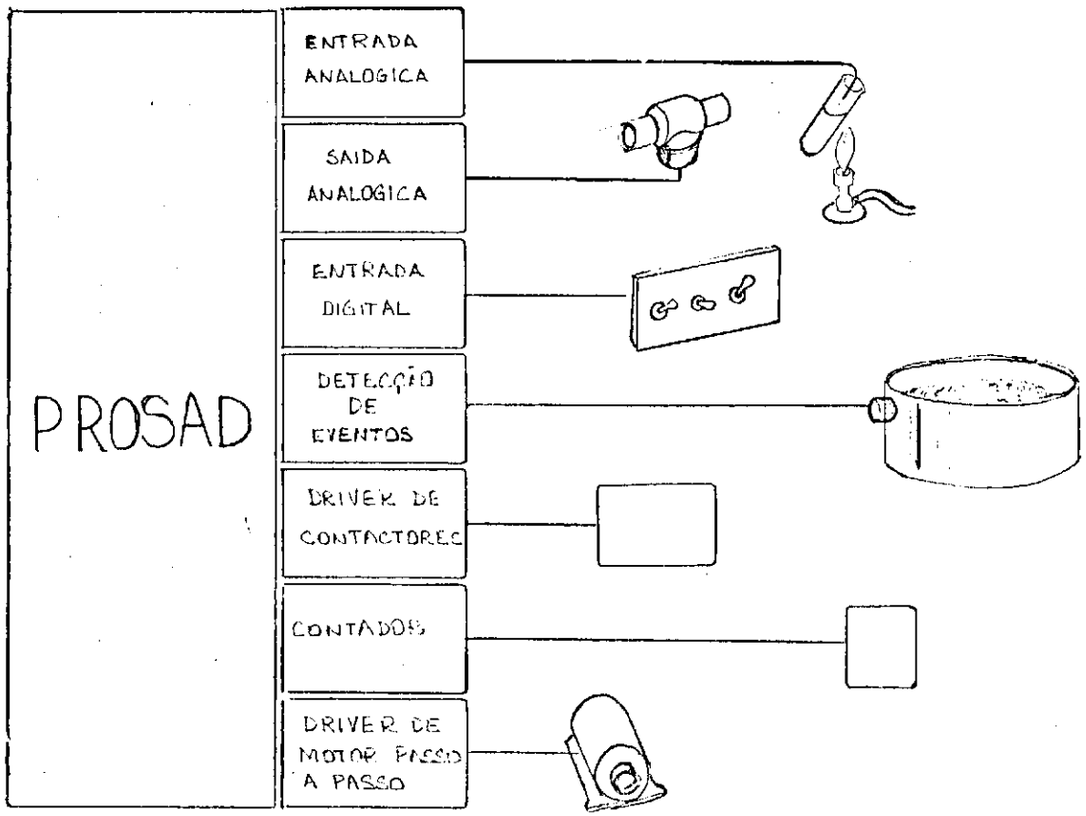


FIGURA 2.1 Exemplos de aplicação do sistema PROSAD

2.1 – Arquitetura do Sistema

O PROSAD é constituído de uma UCP (Unidade Central de Processamento), memória com capacidade inicial de 8 K bytes de RAM e 2 K bytes de EPROM, módulo de interface para teleimpressora e terminal de vídeo, uma interface de vias e de vários circuitos para aquisição de dados e controle (ou cartões de E/S). A figura 2.2 mostra a arquitetura do mesmo.

2.1.1 – Unidade Central de Processamento

A UCP do Prosad é constituída do microprocessador MC 6800 da motorola. o MC 6800 é dentre os micros de 8 bits um dos mais utilizados. Embora seu uso esteja em declínio, devido ao advento de 16 bits, mais rápidos e potentes sendo portanto mais adequados ao uso em controle de processos, O MC 6800 possui um bom conjunto de instruções, uma arquitetura simples e conta com a vantagem de ser juntamente com sua família de circuitos, encontrado com relativa facilidade em nosso país. Suas principais características são:

- tecnologia N MOS dinâmica, compatível com circuitos TTL;
- palavra de 8 bits;
- possui 6 tipos de endereçamento: direto, imediato, inerente, relativo, extendido e indexado;
- possui 6 registros. Dois deles são acumuladores de uso geral contendo 8 bits cada, um contador de pro

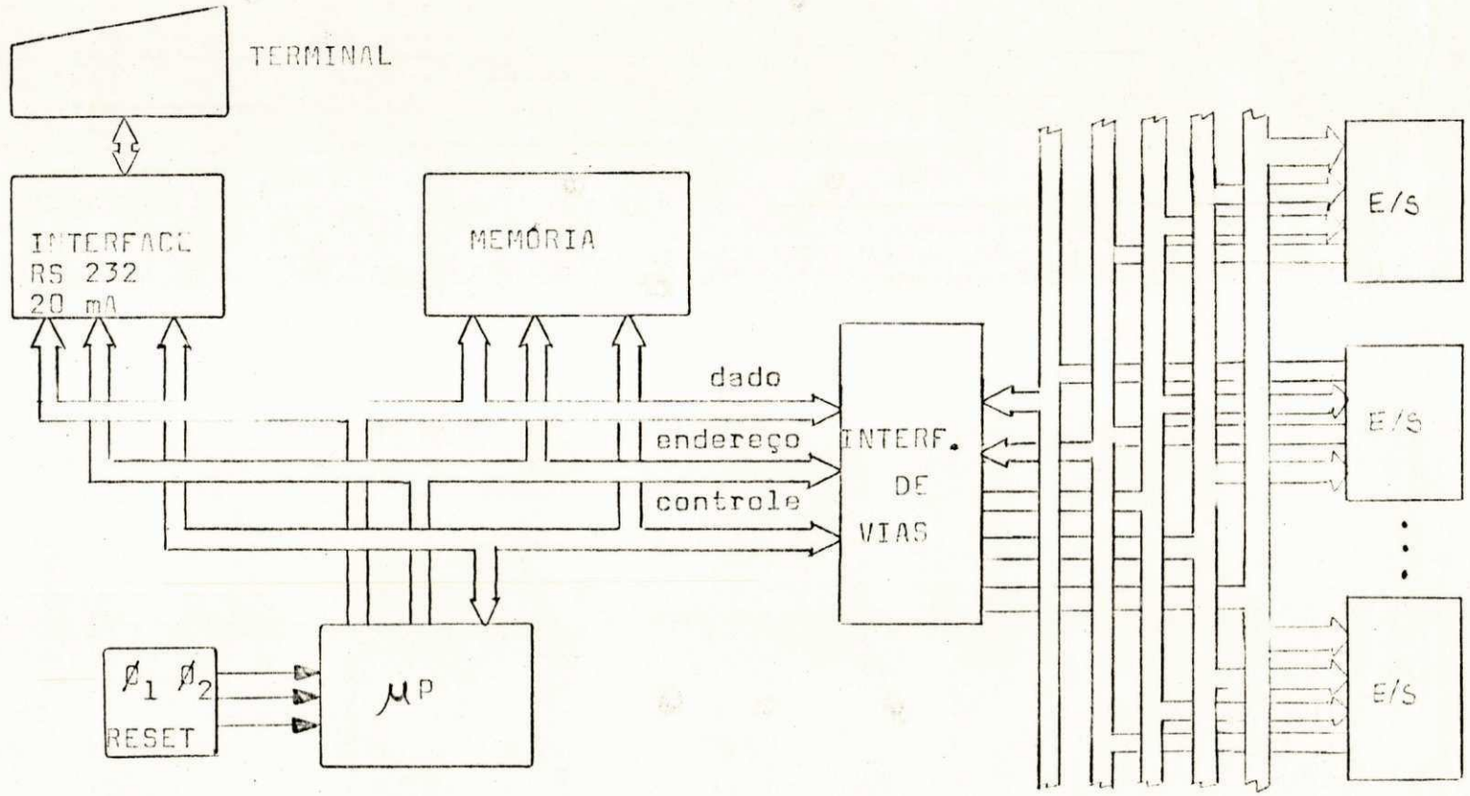


FIGURA 2.2 Arquitetura do PROSAD

grama de 16 bits, um registro de indexação de 16 bits, um registro de indexação de 16 bits, um registro de STATUS de programa de 8 bits, e um apontador de pilha de 16 bits;

- possui um conjunto de 72 instruções;
- atende a dois tipos de interrupção por hardware: interrupção mascarável (IRQ), interrupção não mascarável (NMI), e um tipo de interrupção por software (SWI).

Mais informações sobre este microprocessador podem ser encontradas no Apêndice

2.1.2 – Circuitos de Clock e Reset para o Microprocessador

O MC 6800 necessita de dois clocks externos ϕ_1 e ϕ_2 defasados entre si de 180°. Existem várias opções com relação à implementação do circuito de geração destes clocks, sendo o oscilador a cristal o mais adequado quando se deseja uma estabilidade muito grande, nas freqüências dos mesmos. Uma opção mais barata é se utilizar monoestáveis na geração destes clocks. No caso do PROSAD optou-se pelo uso de monoestáveis. O circuito de RESET do microprocessador e das pias e acias é mostrado juntamente com o gerador de clocks's

2.1.3 – Buffers das Vias de Dados e Endereços do Microprocessador

Segundo dados fornecidos pelos fabricantes do MC 6800 este microprocessador possui, quando utilizado com circuitos de sua família um FAN OUT de 10. No PROSAD o número de circuitos que compartilham as vias de dados e endereços é atualmente 22 podendo aumentar. Torna-se, portanto, necessário introduzir buffers nas mesmas. Os circuitos utilizados como buffers são o 8T26 e o 8T95.

2.1.4 – Unidade de Memória

A unidade de memória do sistema é constituído de 8 memórias RAM's estáticas de 1024 x 4 bits cada, agrupadas de modo a formar um arranjo de memória de 4.096 x 8 bits e duas memórias EPROM's de 1024 x 8 bits cada.

O decodificador de endereços utilizado na seleção dos diversos "chips" de memória é constituído de 2 deMUX de 1 para 8. Isto possibilita a expansão automática da unidade de memória para 8.192 bytes de RAM e 5.120 bytes de EPROM.

As memórias utilizadas neste arranjo são a RAM 2114 e a EPROM 2708. A escolha das mesmas se deu baseada, no caso das RAM's em três fatores: velocidade, capacidade e disponibilidade no mercado e no caso das EPROM's na facilidade de equipamento para a sua programação.

2.1.5 – Memória RAM

A primeira página da memória RAM é utilizada na criação de 256 registros de 1 byte os quais são chamados de registros de E/S do processo. A função destes registros é armazenar os dados resultantes da comunicação processador ⟷ processo. Os mesmos serão tratados com mais detalhes na secção **2.1.12** deste capítulo. A outra área da memória RAM é utilizada para armazenamento do programa do usuário, rascunho e área de trabalho do compilador da qual se falará no capítulo

2.1.6 – Memória EPROM

O software desenvolvido até o momento para o PROSAD consiste de um carregador, um programa gerenciador de tarefas e um macro assembler. Tais programas ocupam aproximadamente 2 K bytes de memória e são armazenados em duas EPROM's 2708.

2.1.7 – A Interface de Vias

Dentre os módulos que compõem o sistema PROSAD, a interface de vias é juntamente com a UCP o mais importante. Isto se deve ao fato de que as características de universalidade do sistema são intrinsecamente dependentes da mesma. Em um sistema de controle de processos os tipos de informação "trocadas" entre o processador e os cartões de aquisição e controle são quatro: dados (entrada e saída), sinais de controle, sinais de estado (STATUS) e sinais de selecionamento dos cartões (endereçamento). Destes 4 tipos, dois são de importância primordial: os sinais de controle e os de estado. Partindo-se do fato de que

tanto mais versátil é um sistema quanto maior for a variedade de cartões de E/S que o mesmo pode suportar, pode-se dizer que tanto mais versátil é o sistema quanto maior for o número de sinais de estado e de controle existentes nas suas vias de comunicação com os cartões de E/S. Este raciocínio norteou o projeto da interface de vias do PROSAD.

2.1.8 – Arquitetura da Interface

A figura **2.3** mostra o diagrama de blocos da interface de vias. A mesma é composta de 6 registros de 8 bits cada. Exceto os pedidos de interrupção solicitados pelos cartões de E/S ao processador, toda informação entre o processador e estes cartões flue através destes registros. A função básica desta interface é servir como multiplicador de vias para o processador.

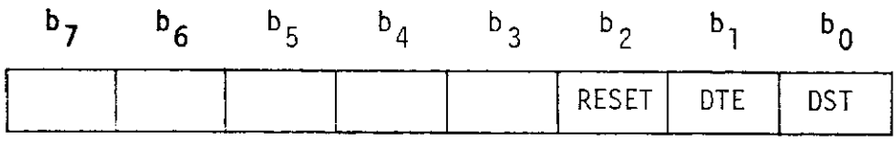
Registros de dados

Através destes registros o processador é capaz de ler ou escrever dados de 8 ou de 12 bits no processo. Na comunicação de 8 bits são utilizados os registros ϕ para leitura e 1 para escrita. Os registros 2 e 3 são de 4 bits cada, sendo utilizados onde se requeira a manipulação de dados de 12 bits.

Registro de controle

Através do registro de controle o processador envia os possíveis sinais de controle para os cartões de E/S. Como este registro possui 8 bits o processador é capaz de gerar através da interface, até 8 sinais diferentes de controle. Dentre es

tes sinais destacam-se o DST (data strobe), o DTE (data transfer enable) e o RESET.



Trataremos em mais detalhes destes sinais na secção 2.1.13.1 deste capítulo.

Registro de estado dos cartões

O registro de estado fornece ao processador informações acerca do estado em que se encontram os cartões de E/S. A exemplo do registro de controle, o de estados pode fornecer ao processador até 8 sinais diferentes.

Registro de endereços

Através do registro de endereços, o processador seleciona o cartão através do qual em determinado instante se comunicará com o processo. O endereçamento é essencial, uma vez que a interligação entre a interface e os cartões é feita através de vias. Qualquer sinal enviado pelo processador a um cartão de E/S chegará virtualmente a todos os cartões, no entanto somente o cartão selecionado será ativado. Como o registro de endereços possui 8 bits, o sistema PROSAD é capaz de operar com até 256 cartões de E/S simultaneamente.

2.1.9 – A Comunicação Processador ⟷ Cartão de E/S

A comunicação entre o processador e um cartão de E/S pode ser feita de dois modos: modo dedicado e não dedicado. A diferença existente entre os dois é que no primeiro, o processador, uma vez iniciada a comunicação ficará a espera do sinal de estado avisando ao mesmo que a transferência do dado foi completada. No segundo caso o processador inicia a operação de leitura ou escrita mas não espera o sinal de estado que indique o fim da operação. O teste deste sinal pode ou não ser feito, ficando esta escolha a critério do usuário. O teste do sinal de estado é importante onde haja necessidade de se efetuar uma operação de "handshake" entre o processador e o cartão. Nos modos de leitura e escrita não dedicada o "handshake" é opcional devendo ser feito apenas em operações repetidas com o mesmo cartão.

2.1.10 – Velocidade de Leitura e Escrita

A partir de um rápido exame da arquitetura da interface de vias pode-se verificar que as operações de aquisição de dados ou saída de variável de controle para o processo (leitura ou escrita) são relativamente lentas. Desprezando o tempo de resposta de um cartão de entrada, uma operação de leitura dedicada o processador deve executar a rotina . . mostrada na pág. 94 . Para uma freqüência de clock de 1 MHz teríamos, segundo dados fornecidos pelo fabricante, um tempo total de 26 µs.

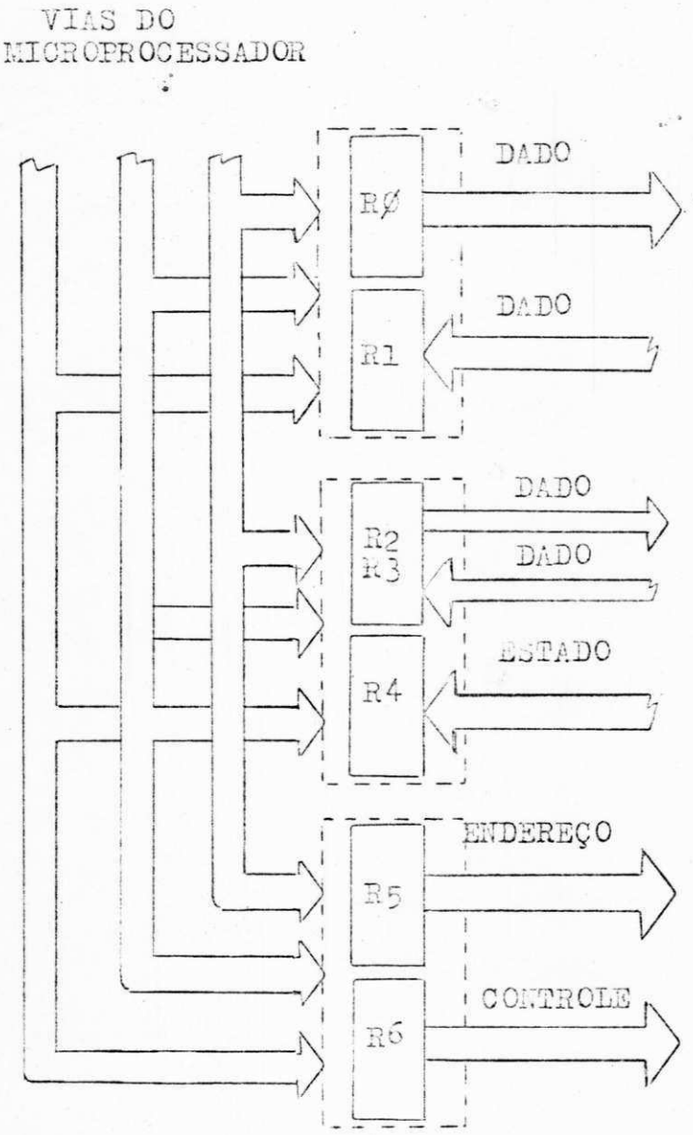


FIGURA 2.3 Diagrama de blocos da interface de vias

2.1.11 – Implementação da Interface

A interface de vias é implementada utilizando três PIA's (peripheral interface adapter) MC 6820 da MOTOROLA. Cada uma destas pia's possui além de seus registros de controle de operação dois registros de dados de 8 bits cada, chamados de: registro de dados A e registro de dados B. Cada um destes registros pode funcionar como registro de entrada (registro de leitura) ou registro de saída (registro de escrita). A escolha de um destes dois modos é feita através da programação do registro de direção de dados da PIA. Na interface de vias cada uma das pias, exceto a PIA 2, é programada de modo a conter um registro de entrada e outro de saída. Na PIA 2 o registro de dados A é programado de modo a operar como dois registros de 4 bits cada, um de entrada e outro de saída. Maiores detalhes sobre estas PIA's podem ser obtidos no Apêndice IV.

2.1.12 – Registros de Dados do Processo

Em uma operação de leitura ou de escrita, a informação trocada entre o microprocessador e o processo através de um cartão qualquer de E/S é sempre armazenada em um byte da memória RAM localizado na primeira página da mesma. Cada cartão de E/S do sistema possui um byte próprio ao qual dá-se o nome de registro de dados do processo. O PROSAD é capaz de operar com até 256 cartões de E/S, simultaneamente, com os endereços destes cartões variando entre 00 e FF (haxadecimal). Como tais registros se encontram na primeira página da memória, os endere

ços dos mesmos correspondem aos endereços dos cartões de E/S que lhe são correspondentes. Por exemplo, o cartão de endereço AF terá um registro cujo endereço é também AF. O objetivo do uso destes registros é "memorizar" o último dado transacionado entre o processador e o processo através dos cartões de E/S. Desta forma pode-se quando necessário ler os mesmos diretamente na memória.

2.1.13 – Cartões de E/S

Através dos cartões de E/S se processam todas as operações de aquisição de dados ou de saída de sinais de controle no PROSAD. A variedade na configuração destes cartões de interface pode ser muito grande e a escolha das mesmas depende basicamente das características do processo a ser controlado. A título de exemplo podem ser citados:

Cartão de entrada analógica – É utilizado na aquisição de dados provenientes de transdutores de pressão, de temperatura, de vazão ou quaisquer outros instrumentos de medição que forneçam informação na forma de um sinal analógico. O PROSAD é compatível com cartões de 8 ou 12 bits;

Cartão de saída analógica – É utilizado na geração de sinais analógicos para atuação de controladores ou atuadores que requeiram sinais analógicos na entrada como por exemplo, controladores convencionais com entrada de set point;

Cartão de entrada digital – É utilizado na leitura de sinais digitais provenientes de chaves de comutação, sensores digitais, voltímetros ou amperímetros digitais ou quaisquer outros dispositivos de medida que tenham saída digital. A exemplo dos cartões analógicos este tipo de cartão pode ter 8 ou 12 bits;

Cartão de saída digital – É utilizado na ativação de chaves de estado sólido, relés de contacto, relés do tipo solenoide, displays digitais, painéis de sinalização digitais, ou controladores e atuadores que requeiram um sinal digital como entrada;

Relógio de tempo real – Usado na geração de uma base de tempo ou na contagem do tempo, indispensáveis na operação em tempo real do sistema.

Neste trabalho foram considerados para efeito de estudo os cartões de entrada digital, de saída digital e o cartão de detecção de eventos, o qual será mostrado em detalhes na seção deste capítulo. Outros tipos de cartão como os de entrada analógica e saída analógica também serão consideradas numa segunda etapa do trabalho.

2.1.13.1 – Cartão de Saída Digital

A figura 2.4 mostra o diagrama de blocos deste cartão. O mesmo é constituído de: bloco de controle, bloco de decodificação de endereço e bloco de armazenamento.

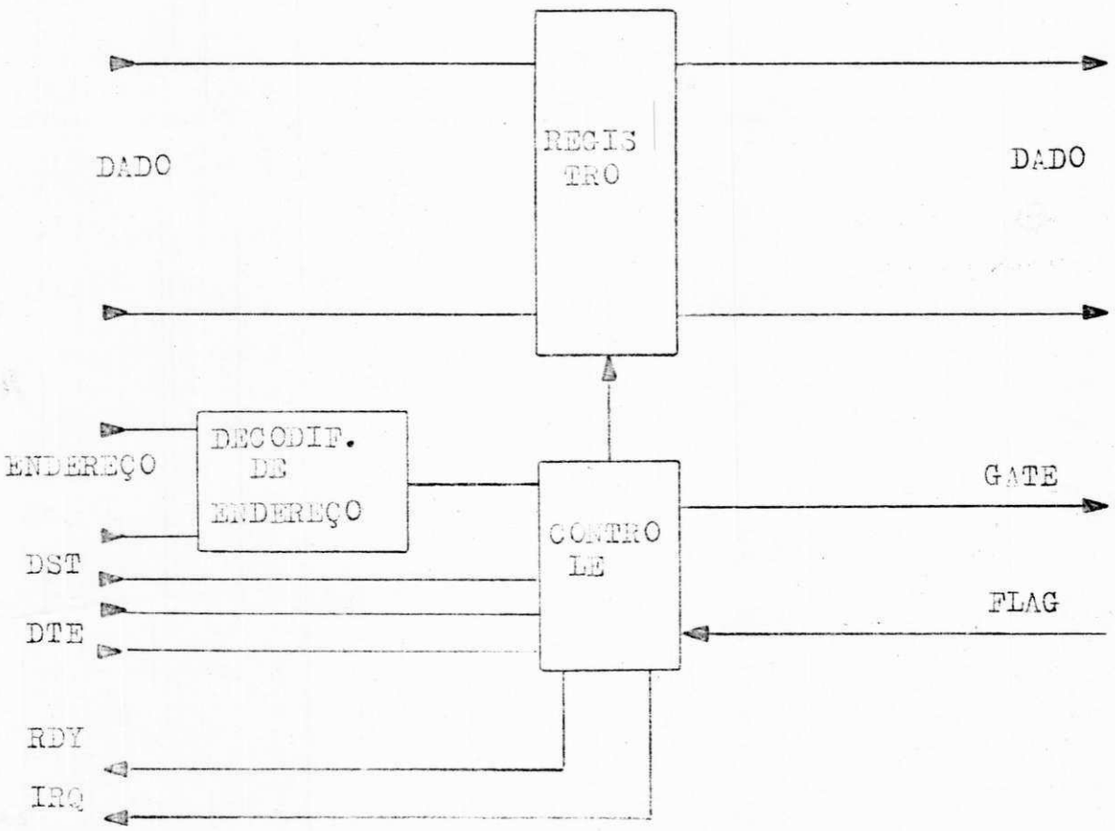


FIGURA 2.4 Diagrama de blocos do cartão de saída digital

bloco de controle

DST (data strobe) – gera o sinal de clock para acionamento do registro. Este sinal comanda o carregamento do dado disponível no registro ϕ da interface de vias no cartão.

DTE (data transfer enable) – o sinal DTE é particularmente importante quando o cartão está ligado a um controlador, atuador ou outro dispositivo de saída que necessita um sinal indicando a validação de um dado. Ao receber o sinal DTE o bloco de controle gera sinal "GATE" indicando que um novo dado "válido" se encontra no cartão.

FLAG – Indica ao bloco de controle que o dado foi lido pelo atuador ou controlador.

READY – Indica ao processador que a operação de saída do dado foi concluída e um novo dado já pode ser enviado ao cartão.

IRQ – O sinal de IRQ ao contrário do DIE, DST e READY é aplicado diretamente ao processador através do pino de IRQ do mesmo. O uso deste sinal é opcional. O usuário poderá utilizá-lo no modo de escrita não dedicada quando desejar que o processador retorne ao cartão através de um pedido de interrupção.

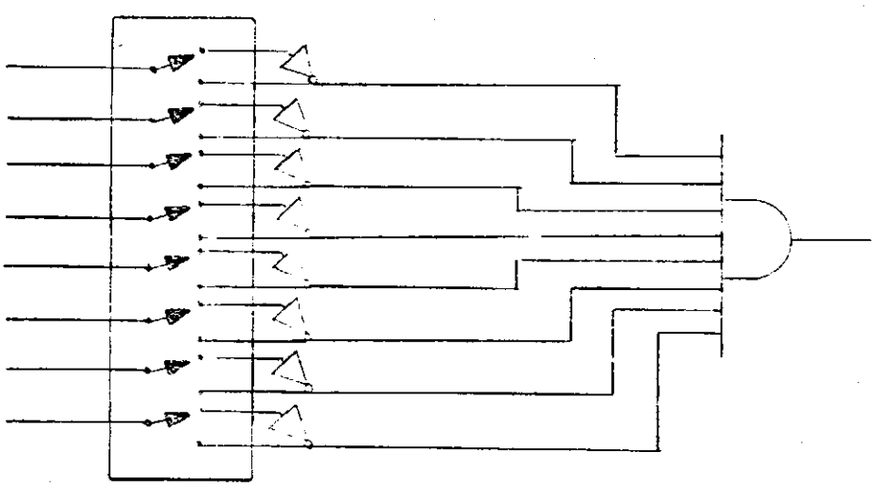
Registro – É usado no armazenamento do dado no cartão. Seu tamanho pode ser de 8 ou 12 bits.

bloco de decodificação de endereço

Conforme foi dito anteriormente o endereçamento dos cartões de E/S é indispensável, uma vez que os mesmos se comuni

cam com o processador através de vias.

O decodificador de endereços consiste do seguinte circuito:



Este decodificador de endereços é utilizado em todos os cartões do sistema.

Ativação de vários cartões simultaneamente

Como foi dito anteriormente o PROSAD é compatível com cartões de 8 ou 12 bits. Em determinadas aplicações, pode no entanto surgir a necessidade de se transmitir para o processo, simultaneamente mais de 12 bits, ou seja, se deseja escrever simultaneamente no processo, dados de um conjunto qualquer de car

tões. No Prosad este tipo de transferência também pode ocorrer. Neste caso o sinal DTE só será gerado pelo processador depois de todos os dados já haverem sido transferidos para seus respectivos cartões. O fluxograma da página 33 ilustra esta operação.

2.1.13.2 – Cartão de Entrada Digital

O cartão de entrada digital é na sua essência semelhante ao de saída digital. Nele, porém, o sinal DST não é utilizado, uma vez que a geração do sinal de carregamento do dado no latch é gerado pelo FLAG. A exemplo do cartão de saída digital, este também pode ter um comprimento de palavra de 8 ou 12 bits.

A exemplo das possibilidades de utilização do cartão de saída digital, o PROSAD pode fazer a aquisição simultânea dos dados de vários cartões de entrada. O fluxograma da página 34 ilustra esta operação.

IRQ – Este sinal deve ser utilizado quando no modo de leitura não dedicada o retorno do processador ao cartão se fizer através de uma interrupção.

2.1.13.3 – Cartão de Detecção de Eventos

Este cartão faz a monitoração de um dado digital qualquer do processo. Esta monitoração consiste na comparação de um dado de referência armazenado no cartão e o dado do processo. Se o resultado da comparação for falso o cartão informa ao processador. A detecção de uma ocorrência deste tipo, a qual dá-se

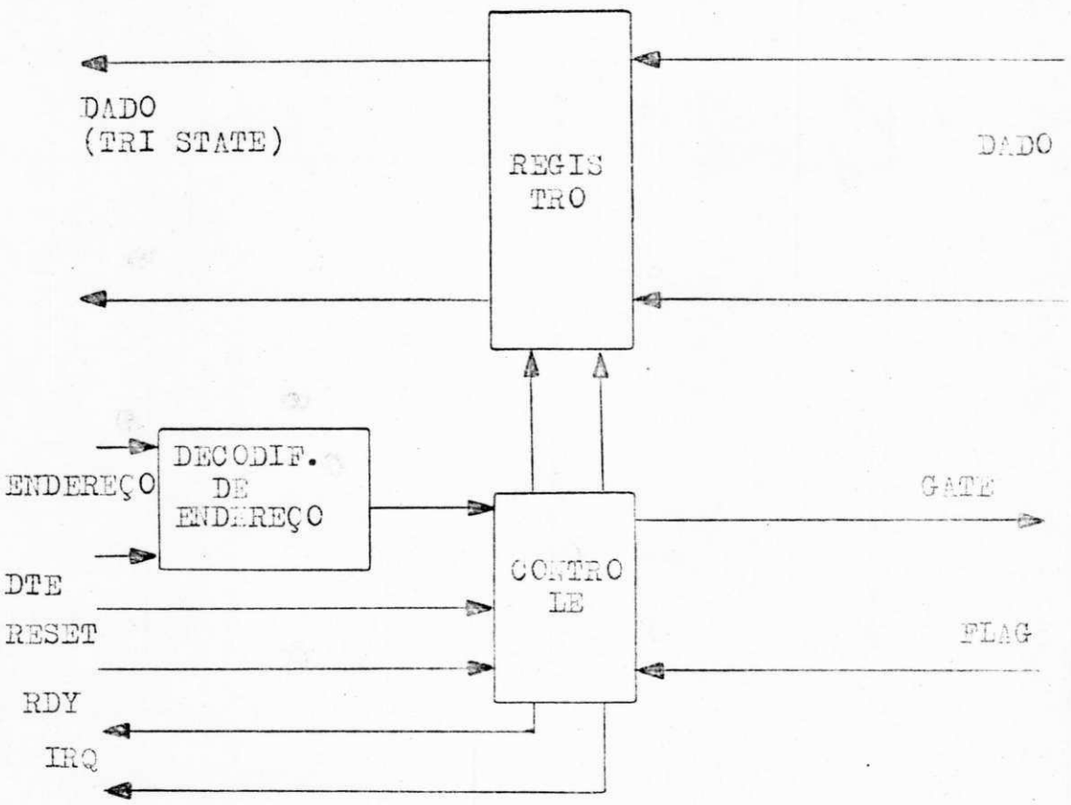
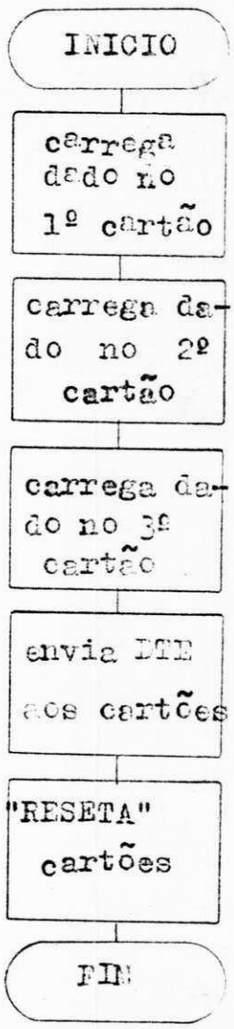
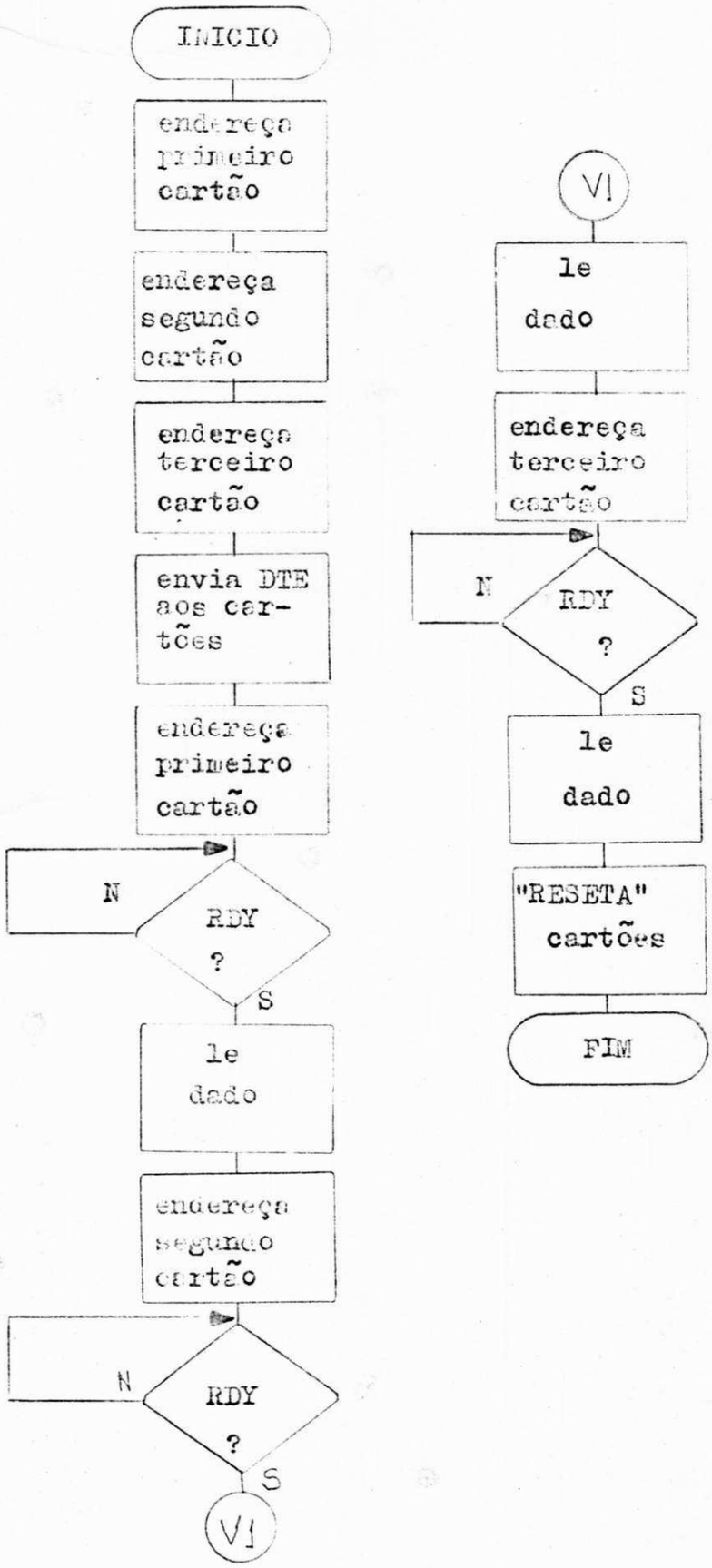


FIGURA 2.5 Diagrama de blocos do cartão de entrada digital

OPERAÇÃO DE ESCRITA DE UM DADO ATRAVES DE TRES CARTÕES SIMULTANEAMENTE.



OPERAÇÃO DE LEITURA DE UM DADO DO PROCESSO
ATRAVÉS DE TRES CARTÕES SIMULTANEAMENTE



aqui o nome de evento é muito importante quando se quer ter uma resposta rápida do sistema a uma mudança crítica de uma variável do processo. Tome-se o seguinte exemplo: Em um determinado processo existem 4 relés R_1, R_2, R_3 e R_4 os quais em regime de estado devem estar:

R_1 = ligado = 1

R_2 = desligado = 0

R_3 = desligado = 0

R_4 = ligado = 1

Através do cartão de detecção de eventos pode-se informar rapidamente ao processador qualquer mudança no estado de um ou mais relés. Uma monitoração deste tipo pode também ser feita através do cartão de entrada digital. Neste caso o processador se encarregaria da mesma, no entanto quanto a mudança da variável monitorada requerer uma resposta muito rápida do sistema o uso deste cartão pode-se fazer necessário. Três tipos de comparação podem ser feitos com este cartão:

dado do processo > dado de referência?

dado do processo < dado de referência?

dado do processo = dado de referência?

A figura 2.6 mostra o diagrama de blocos do cartão de detecção de eventos. O mesmo é constituído de 2 registros de dados de 8 bits cada, um comparador de magnitudes de 8 bits, 2 decodificadores de endereço, um bloco de controle e um seletor de tipo de comparação.

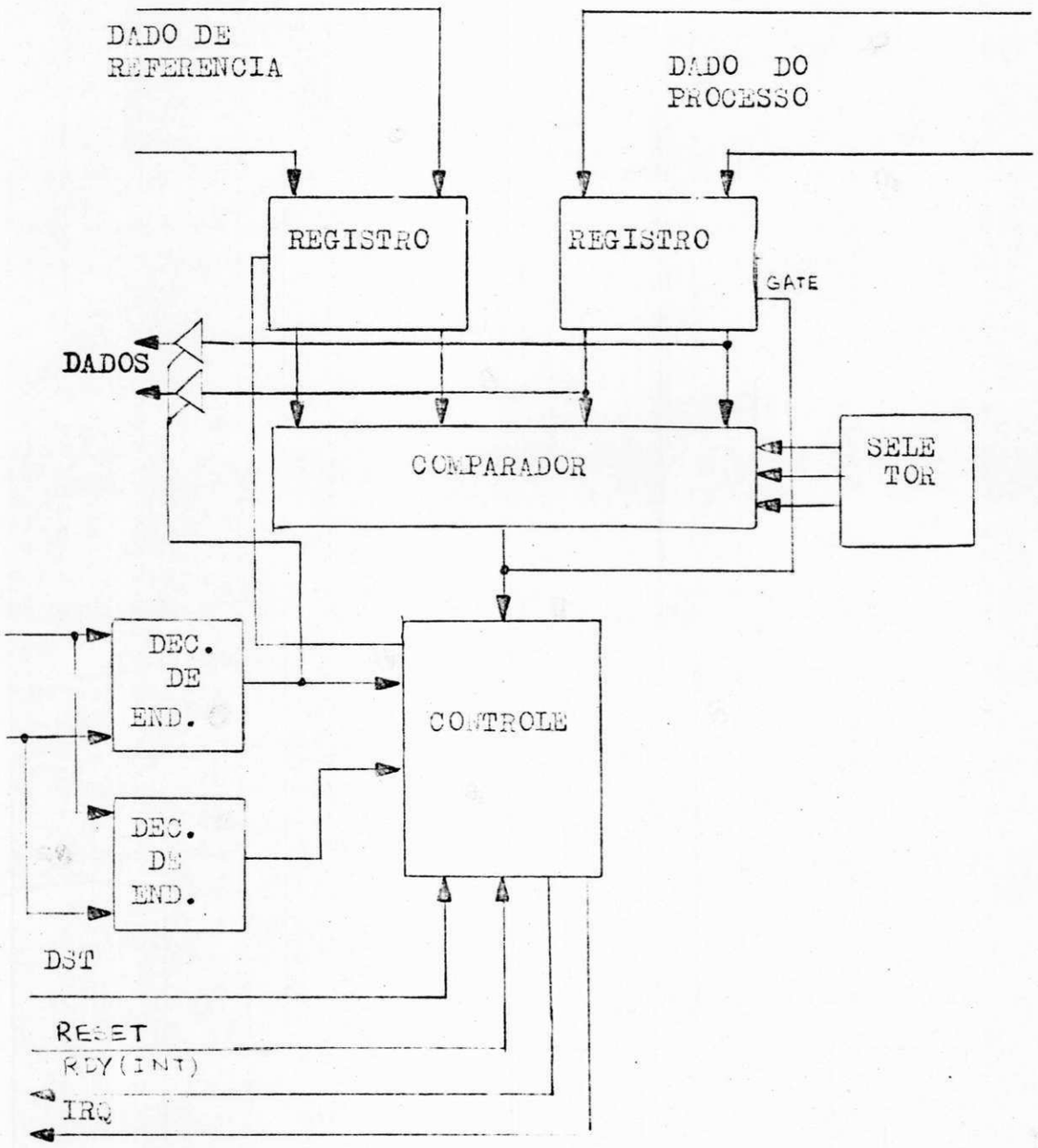


FIGURA 2.6 Diagrama de blocos do cartão de detecção de eventos

Comparador – O comparador de magnitude é constituído de 2 comparadores de 4 bits (7485) ligados em cascata de modo a formar um comparador de 8 bits. A seleção do tipo de comparação a ser feita pelo mesmo é feita através do seletor de comparação. Este seletor pode ser implementado de duas formas: com uma chave miniatura do tipo "dip switch" ou com um conjunto de Flip Flop's tipo D, como mostram as figuras 2.7 e 2.8 .

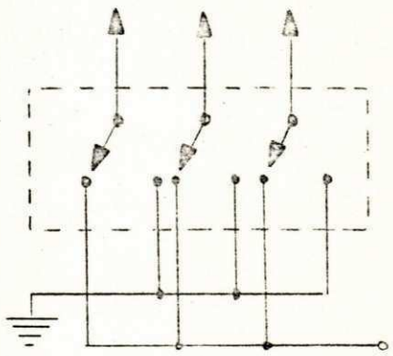


FIGURA 2.7 Seletor de comparação usando **"dip switch"**

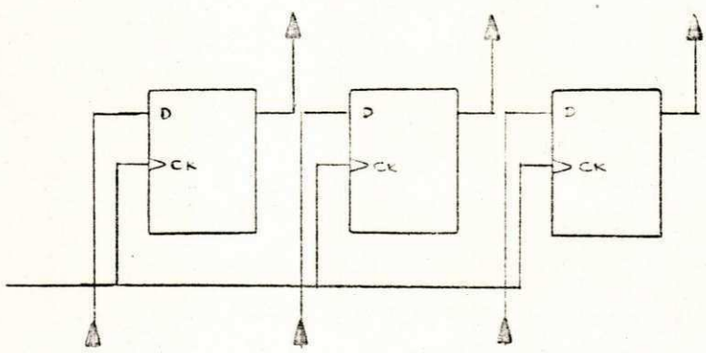


FIGURA 2.8 Seletor de magnitude usando **"Flip Flop's"**

A diferença proporcionada pelos dois tipos de seletores é que a programação do primeiro é feita manualmente no próprio cartão, ao passo que a do segundo pode ser feita por software. A vantagem da programação por software é que o tipo de comparação pode ser mudado pelo próprio processador. Neste caso o cartão é programado através de uma palavra de 11 bits onde os 3 bits mais significativos representam o tipo de comparação e os 8 bits restantes o dado de referência.

Tipo de comparação	b_{10} b_9 b_8	b_7 b_6 b_5 b_4...
Dado do processo > Dado de REFER?	0 1 1	Dado
Dado do processo < Dado de REF?	1 0 1	Dado
Dado do processo = Dado de REF?	1 1 0	Dado

2.2 – Considerações Adicionais Sobre o Projeto dos Cartões de E/S

Os cartões de entrada digital e saída digital projetados são capazes de operar nos modos dedicado ou não dedicado, podem ou não interromper o processador e podem ser ativados em conjunto, permitindo transferências de dados de comprimento variável. O usuário dispõe, portanto, de várias opções quanto à operação dos mesmos. Em aplicações onde se deseja minimizar o custo dos cartões de E/S o hardware dos mesmos pode ser simplificado.

Os cartões de entrada analógica e saída analógica bem como outros cartões mais complexos, como por exemplo, cartões de

temporização podem ser projetados a partir dos cartões de entrada e saída digital. Outras facilidades podem ser obtidas dos cartões digitais. Por exemplo: cartões de alimentação de relés de contacto ou solenoides podem ser obtidos através da adição de drivers apropriados nas saídas do cartão de saída digital.

2.3 – Mapa de Memória do Sistema

O mapeamento de memória do PROSAD foi feito observando-se as seguintes condições:

- a memória RAM devia ocupar a primeira página da memória;
- devia-se prever uma expansão automática das memórias RAM's e EPROM's;
- a memória EPROM deveria ocupar o tôpo da memória do sistema.

2.4 – Instalação do PROSAD

Visando facilitar a expansão do sistema bem como a sua manutenção, o PROSAD deverá ser montado em um armário no qual serão encaixados os vários módulos que compõem o sistema: fonte de alimentação, UCP, clock, reset e interface de vias, interface para TTV e TTY, unidade de memória e cartões de E/S. Este armário terá capacidade de acomodar até 10 cartões de E/S. A expansão do número de cartões de E/S demandará um ou mais armários que deverão acomodar além dos cartões um extensor de vias.

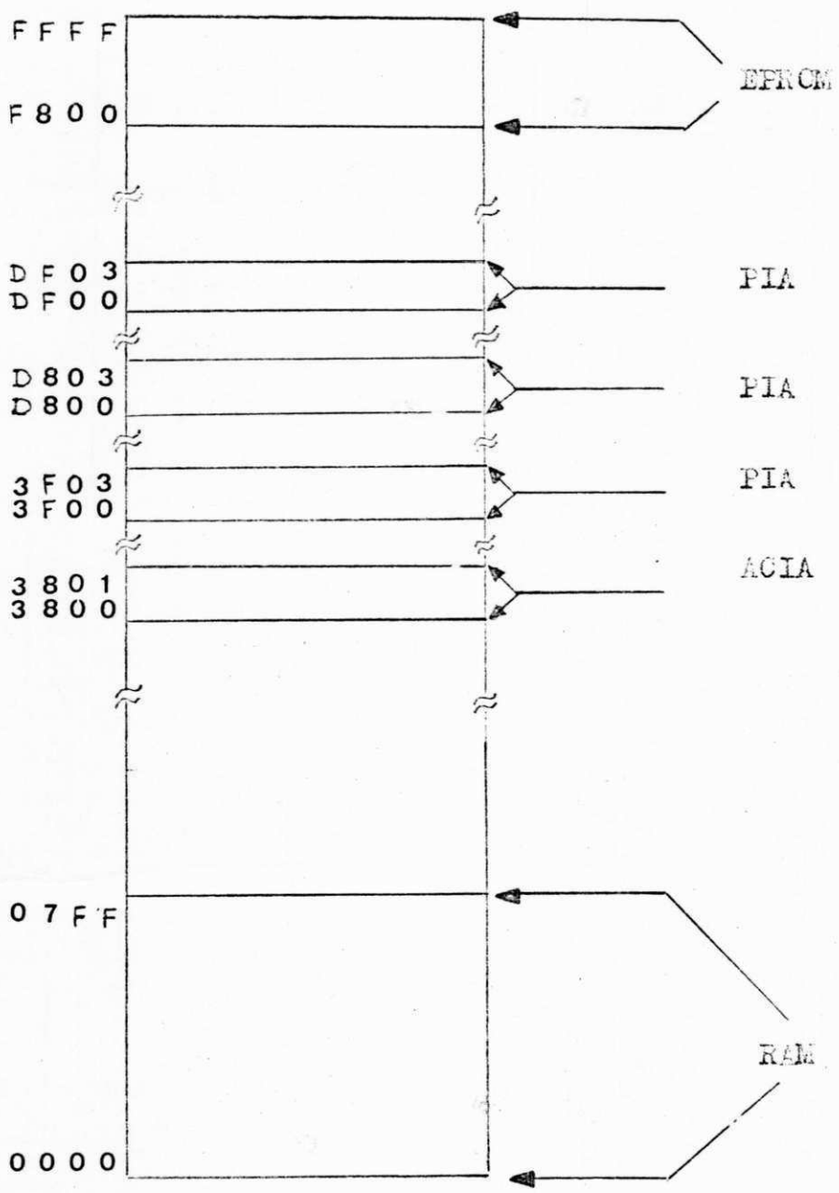


FIGURA 2.9 Mapa de memória do sistema

CAPÍTULO III

SOFTWARE DO SISTEMA

Este Capítulo apresenta o software do PROSAD. São apresentados no mesmo, o programa executivo e o programa escalonador de tarefas que constituem o núcleo de gerência de operação do sistema, o programa carregador e o macro assembler do sistema. A apresentação dos algoritmos será feita através de fluxogramas. A listagem em linguagem Assembly destes programas pode ser vista no Apêndice I.

3.1 – Software do Sistema

A elaboração do software de um sistema computacional em tempo real envolve basicamente a elaboração de três tipos de programa: programas de aplicação, programas supervisórios e programas de suporte.

Programas de aplicação

São os programas que representam as diversas tarefas que devem ser executadas pelo processador. São, por exemplo, os programas de aquisição de dados, programas de processamento, impressão de relatórios e etc.

Programas supervisórios

Tais programas são responsáveis pela coordenação da operação do sistema, se encarregando do atendimento aos pedidos de interrupção solicitados ao processador, disciplinando o acesso de cada uma das tarefas ao processador, bem como executando outras tarefas,como por exemplo a atualização de hora, gerenciamento de arquivo e etc.

Programas de suporte

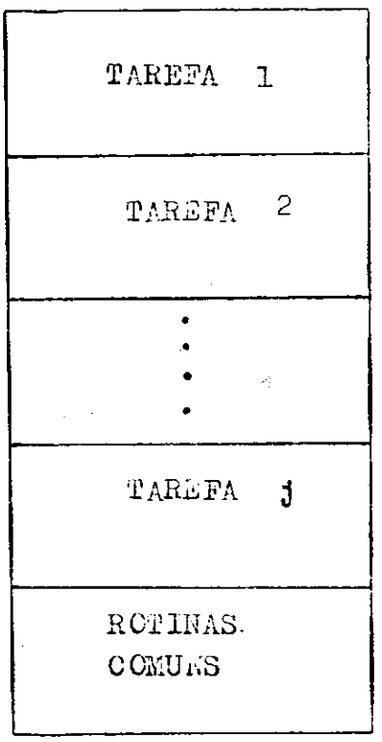
Os programas de suporte não interferem diretamente na operação em tempo real do sistema mas são também de grande importância, pois servem para facilitar a programação e manutenção do mesmo. São os programas de diagnóstico de erro, teste de hardware, assemblers, compiladores e etc.

3.1.1 – Programas de Aplicação

A gama de programas de aplicação é relativamente grande e deve variar de um sistema para outro. No PROSAD existem basicamente três tipos de programa: programas de aquisição de dados e de controle do processo, programas de processamento de dados e programas de comunicação com o console do sistema. Com a

provável expansão do sistema, outros programas deverão ser acrescentados, como por exemplo, um programa de controle de unidade de fita cassete. Como o PROSAD é um sistema de uso geral e de pequeno porte não possuindo ainda nenhum programa de aplicação em ROM, praticamente todos os programas de aplicação são aqueles escritos pelo usuário. Estes programas devem descrever de forma clara para o sistema, as diversas tarefas a serem executadas pelo processador. São tarefas de aquisição de dados analógicos, dados digitais, atuação de válvulas, controle de motores, impressão de relatórios, processamento de dados e etc.

PROGRAMA DE APLICAÇÃO



3.1.2 – Programas Supervisórios

Na secção anterior pôde-se ter uma idéia dos tipos de tarefas a serem executadas pelo PROSAD. Evidentemente sistemas mais complexos possuem tarefas bem mais complexas e em maior número a serem executadas, por exemplo: tarefas de transmissão de dados, leitura de arquivos, controle de unidades de disco e etc. Vê-se que,em função da complexidade do sistema e das características das diversas tarefas a serem executadas, varia também a complexidade no estabelecimento das normas que regem a utilização do processador por estas diversas tarefas.

Geralmente um sistema operacional em tempo real deve executar 3 funções: controlar a execução das tarefas do sistema, executar tarefas de alta prioridade como respostas a condições de alarme do processo sob controle e executar tarefas de baixa prioridade não críticas no tempo. Estes dois últimos tipos de tarefas são evidentemente dependentes do tipo de aplicação para o sistema. Os programas de controle ou programas supervisórios são gerais, merecendo aqui um estudo mais detalhado. Tais programas constituem a parte mais importante deste sistema operacional. A figura 3.1 mostra a estrutura do software de um pequeno sistema operacional. Os programas supervisórios são geralmente divididos em 3 categorias: gerenciador de tarefas, gerenciador de jobs e gerenciador de arquivo. O gerenciador de tarefas são responsáveis pelo atendimento aos pedidos de interrupção e pelo controle do acesso das diversas tarefas ao processador.

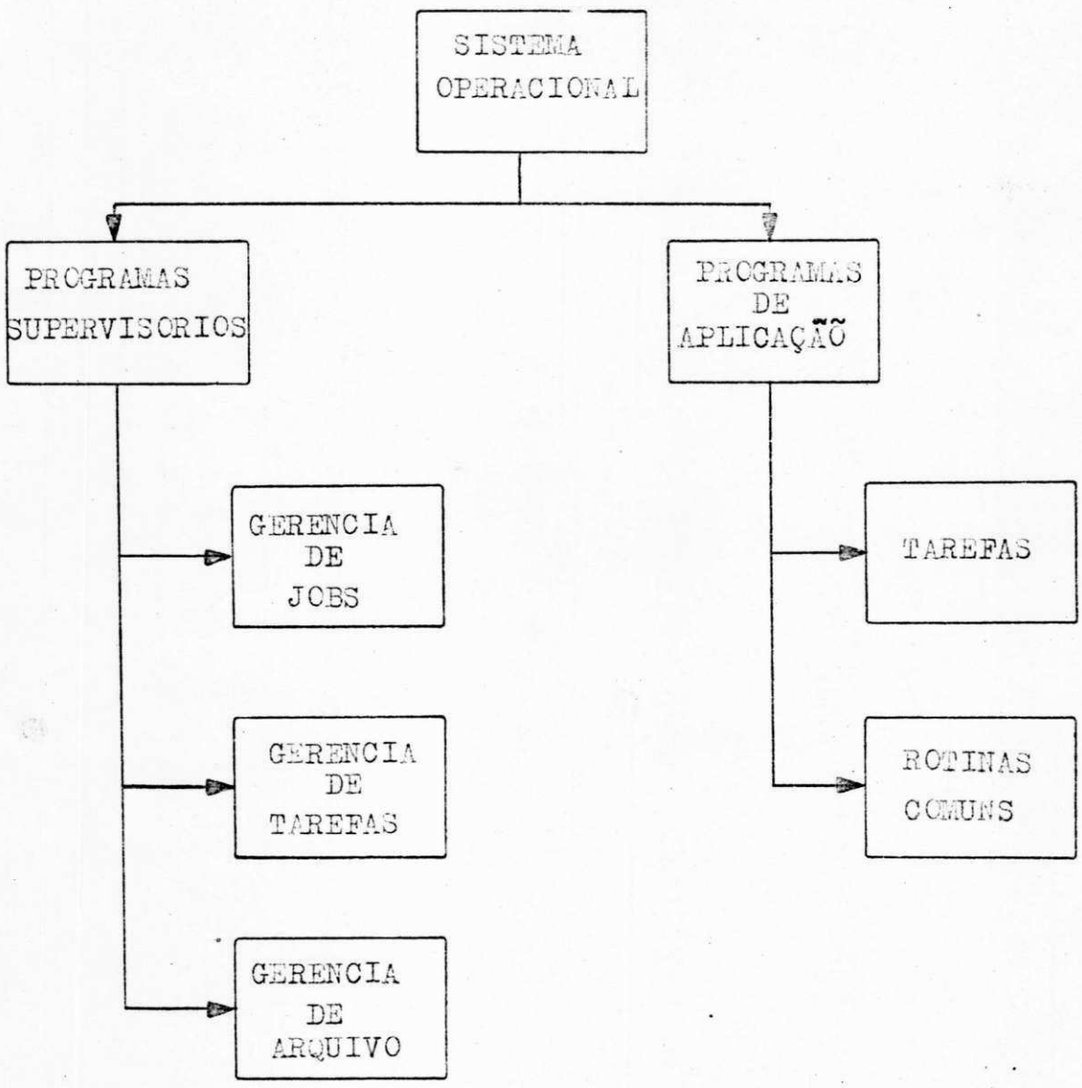


FIGURA 3.1 Estrutura de um sistema operacional em tempo real

Dentre estes três tipos de programas supervisórios, inicialmente o PROSAD deverá ter apenas o gerenciador de tarefas (tasks), uma vez que o sistema não dispõe de arquivos, não havendo, portanto, necessidade de gerenciamento deste tipo. Além disso, apesar do PROSAD ser um sistema de uso geral, trabalhará de forma dedicada, ou seja, um só programa (job) ocupará o processador durante uma determinada aplicação, não havendo portanto, também, a necessidade de se gerenciar jobs no sistema.

3.1.2.1 – Gerenciamento de Tarefas

Conforme foi mostrado na secção 3.1.1 um programa de aplicação qualquer pode ser dividido em várias tarefas que deverão ser executadas pelo processador. Tal programa apesar de ser visto pelo programador como um único todo, é visto de forma diferente pelo processador, pois cada uma das tarefas é vista pelo mesmo completamente isolada das outras. Tudo se passa como se vários usuários tivessem acesso ao processador em intervalos de tempo próprios (cada um destes usuários recebe também o nome de processo).

O gerenciador de tarefas deve então se encarregar de efetivar todo o disciplinamento do uso do processador por estas diversas tarefas ou processos. No PROSAD o pedido de ativação de uma tarefa é feito através de um pedido de interrupção ao microprocessador, conseqüentemente é trabalho também deste gerenciador efetuar o atendimento e a devida identificação destes pedidos de interrupção. Este gerenciador de tarefas é constituído

de dois programas: o executivo e o escalonador de tarefas (scheduler). Ao executivo cabe atender e identificar os pedidos de interrupção, efetuar o acerto de hora e data do sistema e atualizar o status das diversas tarefas. Ao escalonador de tarefas cabe verificar, a partir do executivo, quais são as tarefas que desejam ter acesso ao processador em um determinado instante, e mediante critérios de prioridade, entregar o processador a uma delas.

3.1.2.2 – O Programa Executivo

A figura 3.2 mostra o fluxograma do programa executivo do PROSAD. Este programa assume o controle do processador através de dois tipos de interrupções externas: requisição de interrupção não mascarável (NMI) e requisição de interrupção mascarável (IRQ). A interrupção NMI deverá ser utilizada para o RTC (real time clock) do sistema. As interrupções de ativação de outras tarefas serão feitas através de IRQ. Ao receber um pedido de interrupção do tipo IRQ o microprocessador executa um pooling verificando quais dos cartões, cujos endereços se encontram na tabela de tarefas, está solicitando esta interrupção. Este pooling é baseado na tabela de tarefas de modo a evitar que um cartão de entrada e saída de dados seja consultado. Uma vez identificado o cartão autor deste pedido, o microprocessador verifica na tabela de tarefas quais são as tarefas associadas a este cartão atualizando o status das mesmas. Este status é de fundamental importância para o programa escalonador de tarefas, conforme se verá na secção 3.1.2.3. Feita esta atualização da tabela

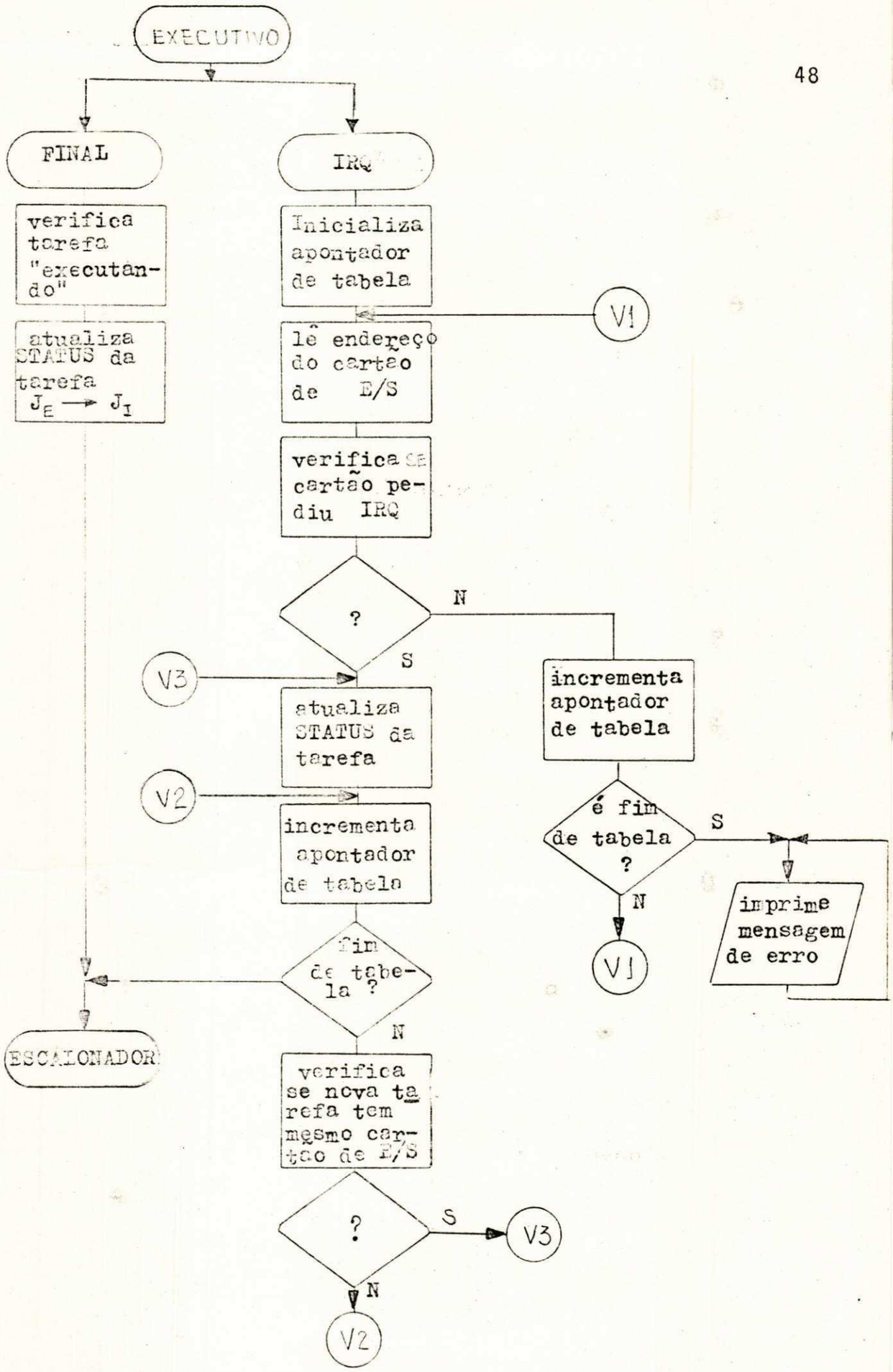

EXECUTIVO
FINAL
IRQ
verifica tarefa "executando"
atualiza STATUS da tarefa $J_E \rightarrow J_I$
Inicializa apontador de tabela
V1
lê endereço do cartão de E/S
verifica se cartão pediu IRQ
?
N
S
V3
atualiza STATUS da tarefa
V2
incrementa apontador de tabela
fim de tabela ?
N
S
ESCALONADOR
incrementa apontador de tabela
é fim de tabela ?
S
N
V1
imprime mensagem de erro
verifica se nova tarefa tem mesmo cartão de E/S
?
S
V3
N
V2

FIGURA 3.2

de tarefas o executivo entrega o microprocessador ao programa escalonador.

3.1.2.3 – Programa Escalonador de Tarefas

O escalonador de tarefas, conforme foi dito, faz sob demanda do executivo a verificação na tabela de tarefas do sistema, quais as tarefas que estão solicitando acesso ao processador naquele instante e entrega o mesmo à de prioridade mais alta. A complexidade do algoritmo deste escalonador deve variar em função dos tipos de tarefas as quais o sistema deve executar e a forma como serão auferidas as prioridades a cada uma delas.

Dentre os vários algoritmos propostos para escalonadores, o mais simples é sem dúvida o algoritmo de Huang e Parrish (1). Sua descrição é feita a seguir.

Uma tarefa qualquer a ser executada pelo processador pode durante a operação do sistema assumir os quatro seguinte estados:

1) Inativo ou dormindo
2) Acordado (ready)
3) Executando
4) Suepenso.

O estado "inativo" é associado a uma tarefa quando sua execução não está sendo solicitada ao executivo.

O estado "acordado" é associado a uma tarefa quando sua execução foi solicitada e a mesma está à espera do processador.

O estado "executando" é associado a uma tarefa quando a mesma está utilizando o processador.

O estado "suspenso" é associado a uma tarefa quando a mesma teve sua execução suspensa e outra tarefa de maior prioridade ocupou o processador.

Em um instante qualquer da operação do sistema todas as tarefas possuem um destes estados, no entanto, somente uma pode estar no estado executando. Uma vez assumido o controle do processador uma tarefa é executada até que ocorra um pedido de interrupção no caso de a mesma ser preemptível (possa ser ou até que a mesma termine, caso suspensa). Se uma tarefa preemptível está sendo executada e uma interrupção ocorre, duas situações podem surgir: se a interrupção solicita a ativação de uma outra tarefa de mais alta prioridade, seu estado muda de executando para suspensa e a outra tarefa assume o controle do processador. Se caso contrário, a prioridade da outra tarefa for mais baixa o processador retorna para a mesma continuando sua execução do ponto em que a mesma havia sido interrompida.

O algoritmo de Huang e Parrish (1) é estático pois a prioridade de cada tarefa é fixa durante todo o tempo de operação do sistema.

Definições:

J_E - tarefa executada

J_A - tarefa acordada

J_{Si} - a J-ésima tarefa no estado suspenso

J - stack pointer da pilha de tarefas suspensas

P_E - prioridade da tarefa em execução

P_A - prioridade da tarefa acordada

P_{SJ} - prioridade da J-ésima tarefa suspensa.

Uma vez que existem vários níveis de prioridade, é fácil se entender que em um determinado instante da operação do sistema pode haver um número qualquer de tarefas suspensas. Além disto todas tarefas terão prioridade mais baixa daquela em execução. Quando a tarefa em execução for terminada e o processador retornar ao executivo, o mesmo desativará esta tarefa e pedirá ao escalonador para reativar a tarefa suspensa de maior prioridade, ou seja, a última delas. A área reservada para o armazenamento das informações referentes a cada tarefa suspensa (acumuladores, registro de índice, registro de código de condição e contador de programa) é uma pilha do tipo LIFO. O apontador de pilha do microprocessador é usado na geração da mesma.

Algoritmo:

Inicialmente o apontador de pilha é feito igual a zero. Cada vez que uma tarefa é suspensa J é incrementado. Se uma tarefa é terminada e o executivo não recebe pedido de interrup

ção J_{sj} é reativada. $J_{sj} \rightarrow J_E$. Se uma interrupção ocorre, solicitando a ativação de uma tarefa o escalonador fará os seguintes testes:

1) Se não existe J_{sj} nem J_E a tarefa acordada passa a ser executada $J_A \rightarrow J_E$;

2) Se existe J_E o escalonador verifica se $P_E \geq P_R$. Se o teste for verdadeiro o processador continua executando J_E caso contrário, $J_E \rightarrow J_{sj}$ e $J_R \rightarrow J_E$.

A figura 3.3 mostra o fluxograma do escalonador.

Implementação do Algoritmo:

Os parâmetros essenciais, monitorados pelo escalonador são os estados e prioridades das tarefas. A partir destas duas informações é feita a decisão quanto a qual tarefa utilizará o processador em um determinado instante. Tal informação é armazenada em uma área de memória chamada de tabela de tarefas do sistema. Nesta tabela (Fig. 3.4), cada tarefa é representada pela sua prioridade, seu estado atual, o endereço do cartão que deve solicitar a ativação da mesma e o vetor de inicialização da mesma.

A prioridade e o estado da tarefa são representados pelo primeiro byte do bloco de contexto da mesma, sendo os dois bits mais significativos reservados para o estado e os 6 restantes para a prioridade.

ESCALONADOR DE TAREFAS

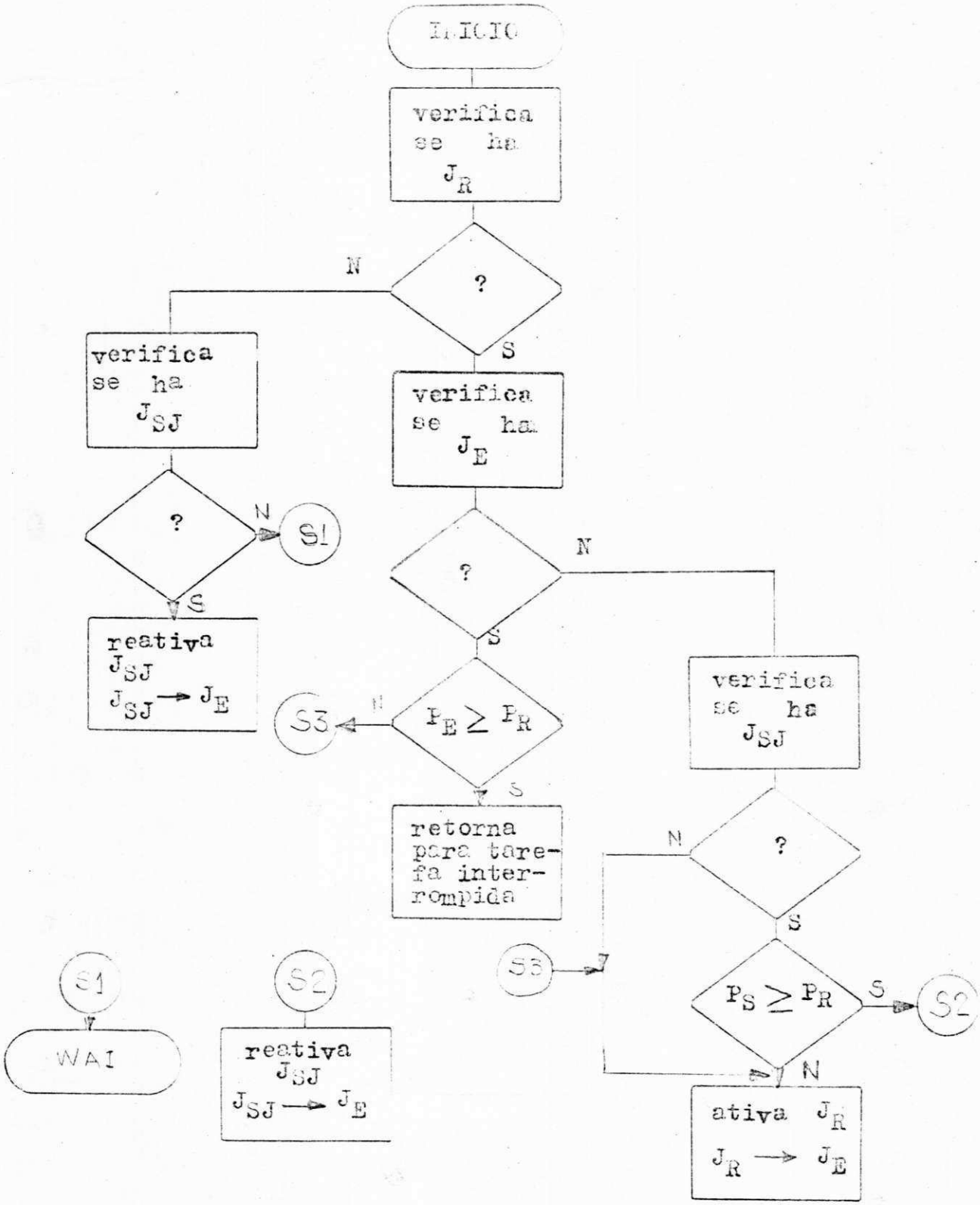


FIGURA 3.3

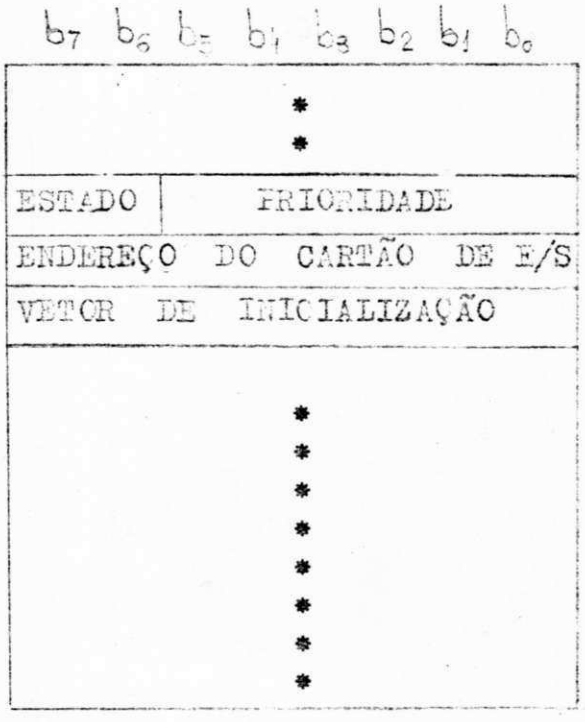


FIGURA 3.4 tabela de tarefas do sistema

Estados da Tarefa:

Inativo - 00

Acordado - 01

Executando - 10

Suspenso - 11

Como 6 bits são reservados para a prioridade, o sistema pode ter 2^6 = 64 níveis de prioridade diferentes. O endereço do cartão de E/S que ativa esta tarefa faz parte do bloco de contexto da mesma, pois todos os pedidos de ativação de tarefas são feitos através de interrupções externas ao microprocessador. Este endereço será usado pelo microprocessador para efetuar o pooling nestes cartões e identificar na tabela qual(is) tarefa(s) deve(m) ser acordadas.

A implementação deste algoritmo neste trabalho difere um pouco da proposta por Huang e Parrish (1), uma vez que os mesmos a fizeram no micro processador 8085 da Intel.

3.2 — Programa Carregador

O programa carregador armazena o programa fonte em uma área da memória chamada de buffer do programa fonte. Tal programa é lido pelo carregador através de um terminal de vídeo ou TTY. O procedimento para chamada deste programa ainda não foi definida uma vez que o programa de atendimento ao operador ainda não foi elaborado.

O fluxograma da página 58 mostra o programa carregador. Uma de suas características é o eco de todos os caracteres batidos no terminal. Este eco é particularmente importante em terminais que não possuem cópia local como terminais de vídeo simples. No caso de se utilizar uma TTY como terminal console este eco deve ser eliminado. Uma outra característica deste programa é o line feed automático que é enviado ao terminal em resposta a um "carriage return". Visando ainda referenciar um início de linha do programa fonte um caracter é também enviado ao terminal após o line feed. O fim do carregamento de um programa qualquer ocorre quando o carregador detectar a instrução FIM que indica o fim do programa fonte. Finalizado o carregamento, o monitor deve reassumir o controle do processador.

3.3 – A Linguagem Fonte do Sistema

Os programas fontes podem ser constituídos de dois tipos de declarações. Declarações cuja instrução é escrita em linguagem de máquina e declarações escritas em linguagem assembly. O uso destas duas linguagens decorre do fato de que o macro assembler elaborado não abrange todas as instruções capazes de ser executadas pelo microprocessador.

Uma declaração deste tipo pode conter qualquer operação em linguagem de máquina do microprocessador. Desta forma o programador pode em função de sua experiência utilizar todos os recursos do microprocessador. Visando minimizar os problemas de documentação acarretados pela programação em linguagem de máqui

na, reservou-se neste tipo de declaração um campo para comentário. Tal comentário pode se constituir de qualquer caracter no código ASCII. Um "carriage return" marca o fim deste statement.

Formato da Declaração:

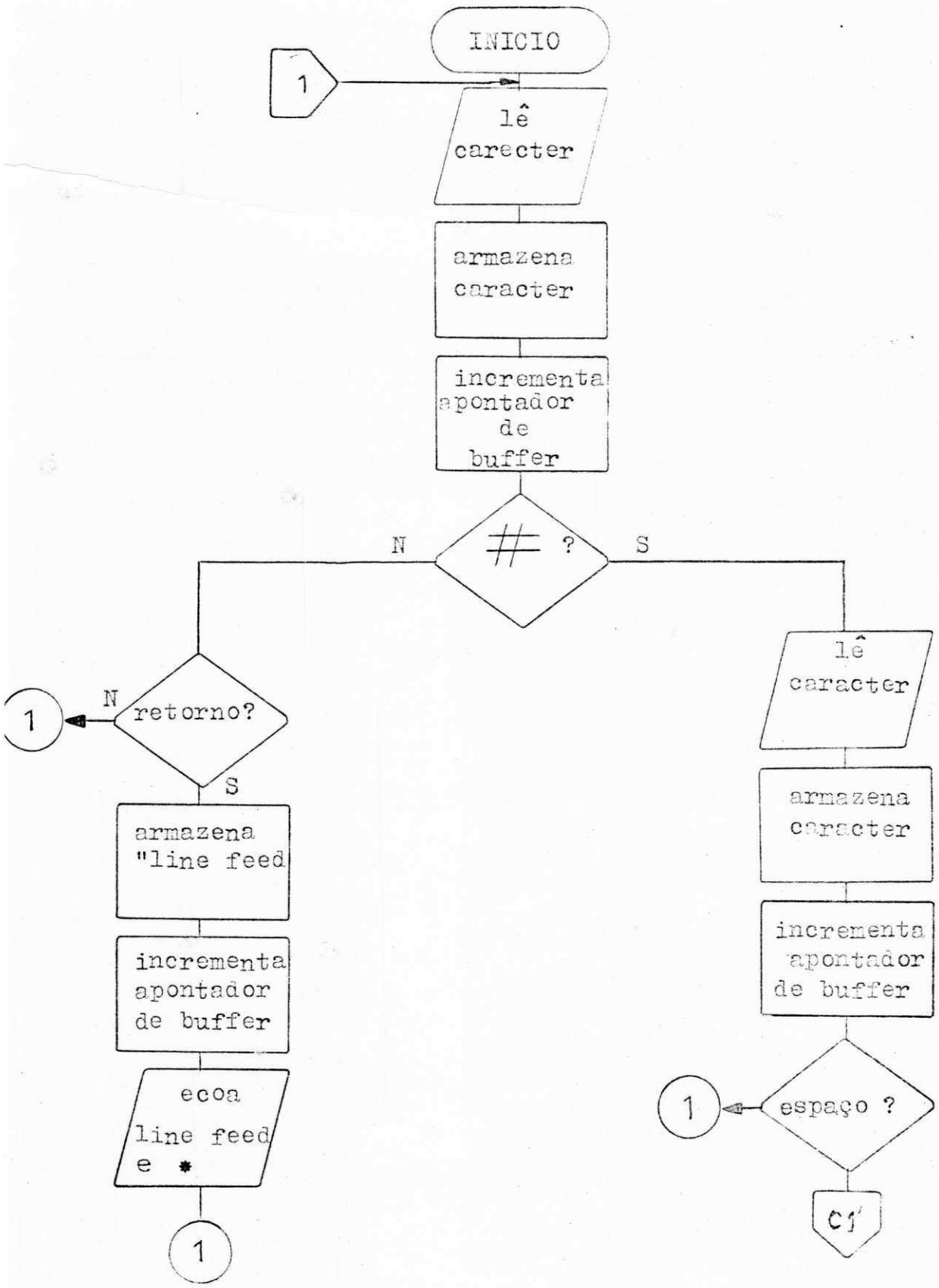
*————————Espaço————————Espaço $————————————————CRLF

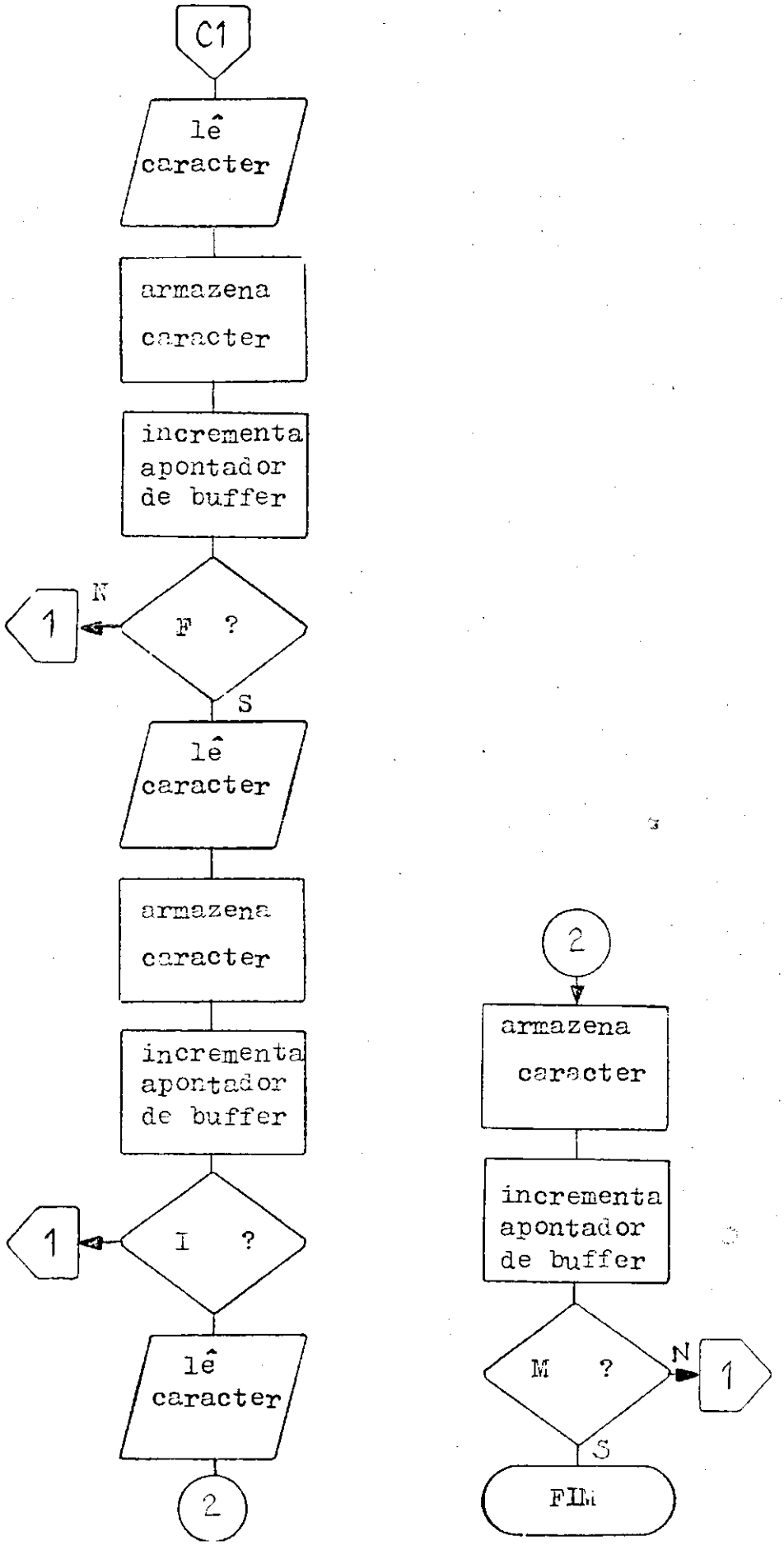
O primeiro campo desta declaração é reservado para o código da operação a ser executada. Qualquer uma das 72 diferentes operações do microprocessador podem preencher este campo. O campo do operando contém o operando, em linguagem de máquina, a ser manipulado. Este operando pode ter dois ou quatro caracteres. Isto significa que um operando que ocupa 1 byte não precisará ser precedido por dois zeros. Por exemplo:

Operando	Formato
F	ϕF
FA	FA
FOFB	FϕFB
A	ϕA
FCϕ	ϕFCϕ

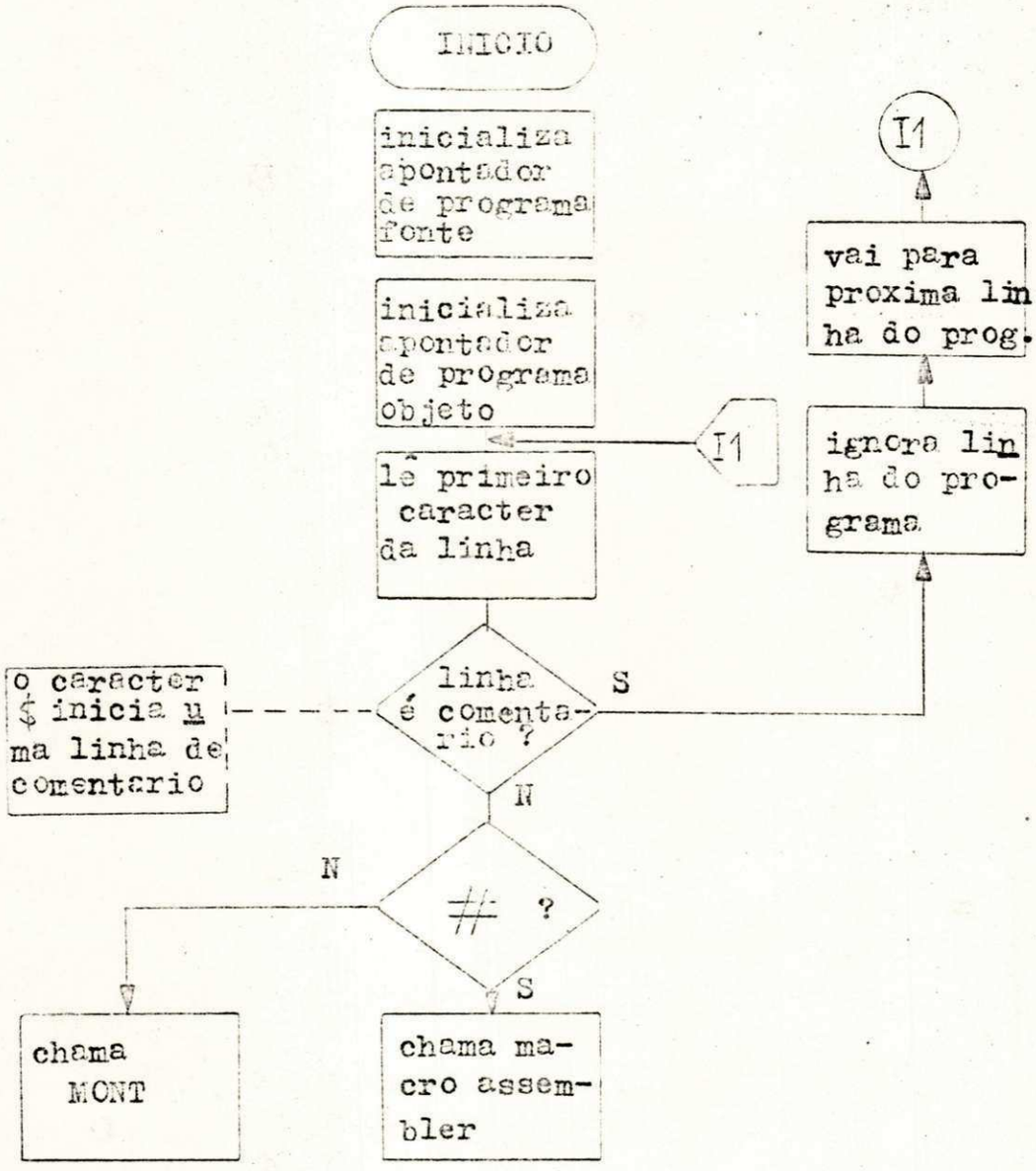
O campo de comentário deve ser iniciado pelo caracter "$". Este campo, conforme foi dito, pode conter qualquer conjunto de caracteres. A importância de um campo de comentário num "statement" deste tipo é clara, uma vez que o programador pode exprimir no mesmo as informações acerca do código de operação usado.

CARREGADOR

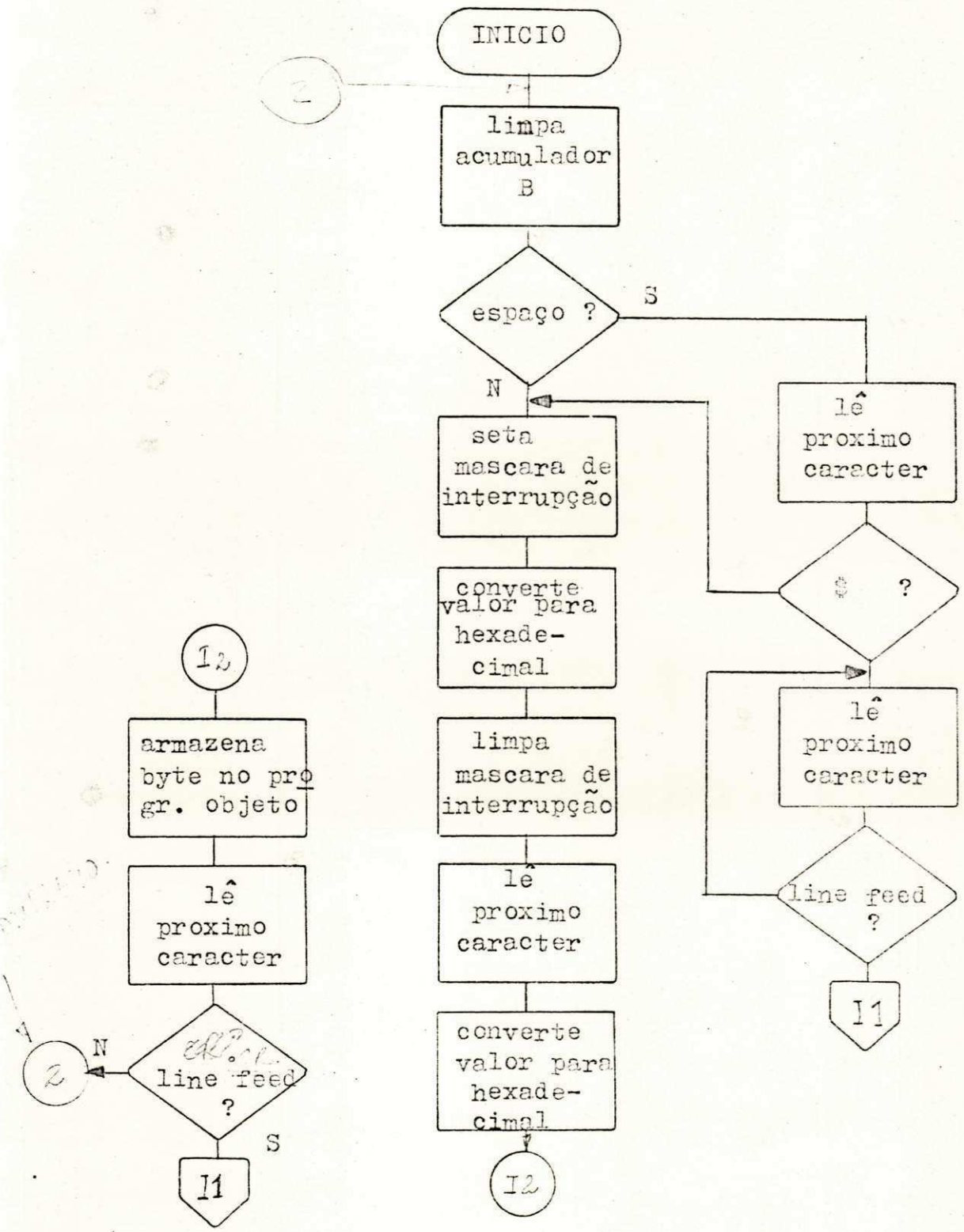

INICIO
1
lê caracter
armazena caracter
incrementa apontador de buffer
?
N
S
retorno?
N
1
S
armazena "line feed
incrementa apontador de buffer
ecoa line feed e *
1
lê caracter
armazena caracter
incrementa apontador de buffer
espaço ?
1
C1'


C1
lê caracter
armazena caracter
incrementa apontador de buffer
F ?
N
1
S
lê caracter
armazena caracter
incrementa apontador de buffer
I ?
1
lê caracter
2
2
armazena caracter
incrementa apontador de buffer
M ?
N
1
S
FIM

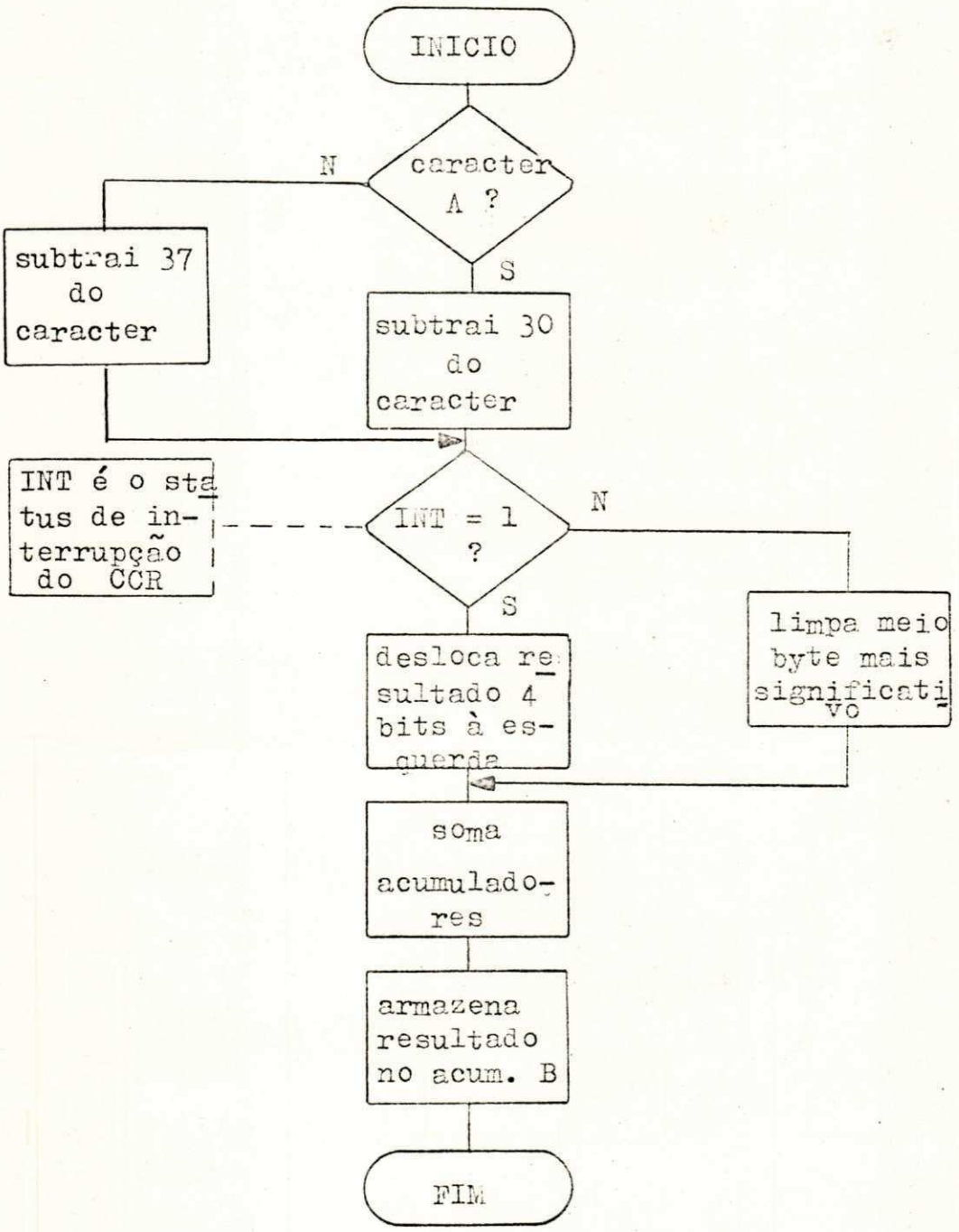
IDENTIFICAÇÃO DO TIPO DE DECLARAÇÃO



ROTINA "MONT"


INICIO
limpa acumulador B
espaço ?
S
N
seta mascara de interrupção
converte valor para hexade-cimal
limpa mascara de interrupção
lê proximo caracter
converte valor para hexade-cimal
I2
lê proximo caracter
$?
lê proximo caracter
line feed ?
I1
armazena byte no prog. objeto
lê proximo caracter
line feed ?
N
S
I1
I2

SUBROTINA DE MONTAGEM DE UM BYTE HEXADECIMAL



3.4 – Macro Assembler

3.4.1 – Formato das Declarações

Uma declaração em assembler do programa fonte é dividida em quatro campos: o primeiro campo contém o label da instrução; o segundo contém a mnemônica; o terceiro contém o operando e o quarto campo contém comentários.

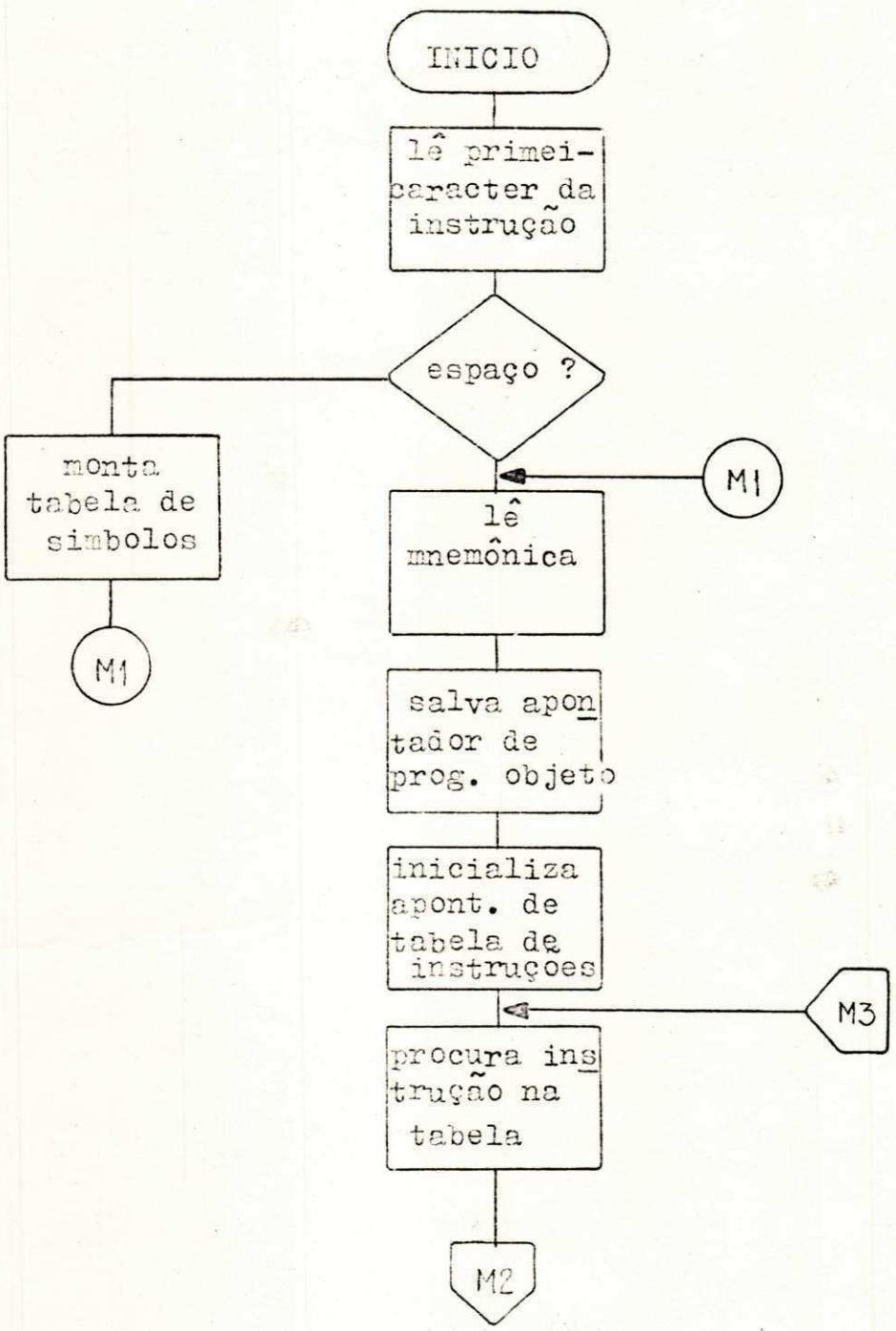
Label – O label pode ser constituído de quaisquer caracteres em AS-CII, exceto espaço. Seu comprimento é ilimitado, ou seja, o mesmo pode conter um número ilimitado de caracteres. O caracter espaço determina o fim do label.

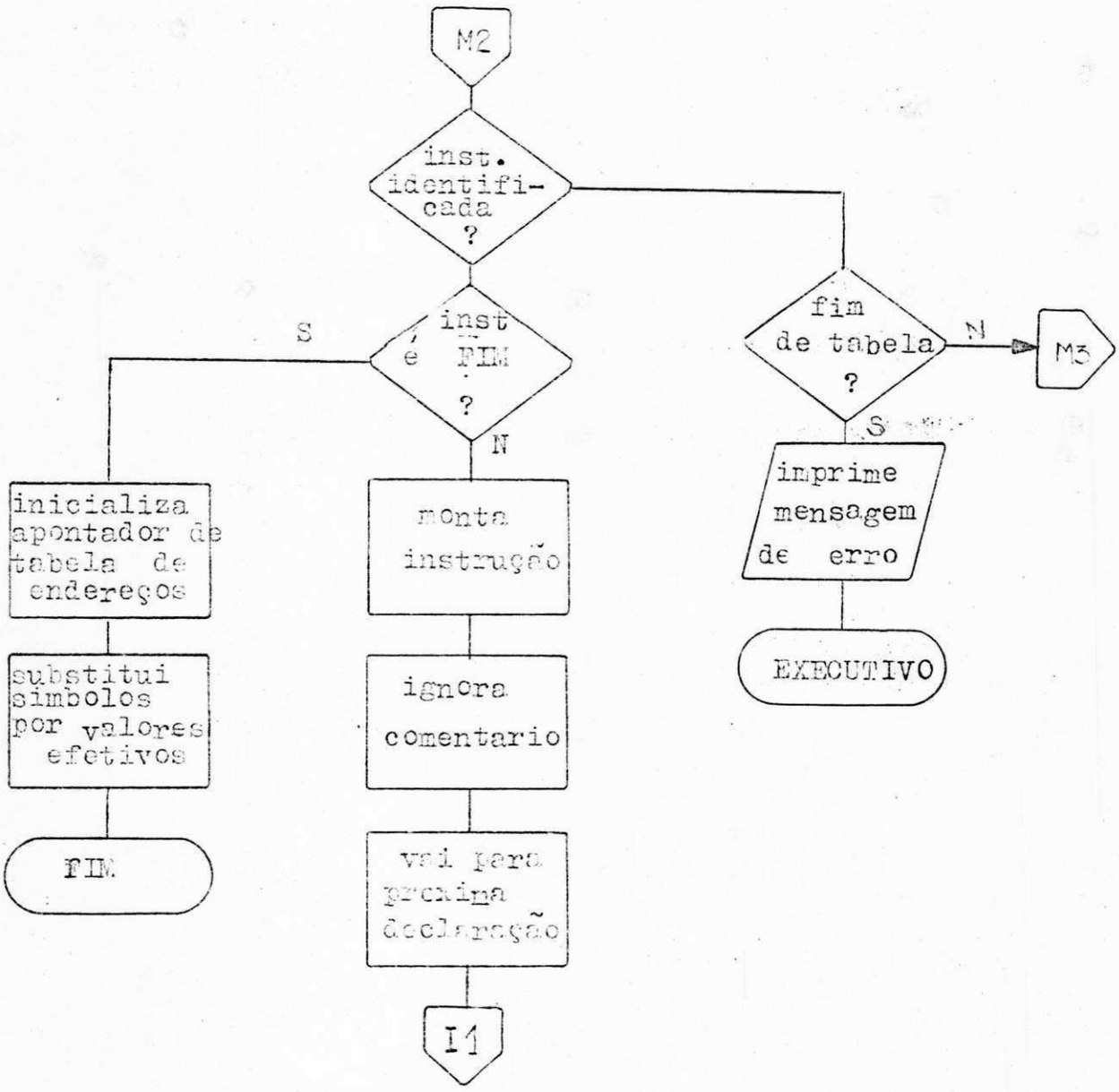
Mnemônica – A mnemônica representa a operação a ser executada pelo processador.

Operando – Conforme ver-se-á na secção 3.4.4.4 deste capítulo, existem macro instruções com mais de um operando. Quando isto ocorrer os operandos são separados por um espaço. Tais operandos podem ser formados de quaisquer caracteres em ASCII exceto o caracter espaço que determina o fim das mesmas. Tais operandos a exemplo dos labels tem comprimento ilimitado. Uma característica deste assembler é que qualquer símbolo contido neste campo é interpretado como um símbolo não resolvido, isto é, o programador não pode definir um número que representa o valor real do operando como sendo o próprio operando.

Comentários – O campo de comentários é iniciado depois do espaço que define o fim do campo dos operandos. Tal cam

MACRO ASSEMBLER




M2
inst. identifi-cada ?
S
inst e FIM ?
N
fim de tabela ?
N
M3
S
inicializa apontador de tabela de endereços
monta instrução
imprime mensagem de erro
substitui simbolos por valores efetivos
ignora comentario
EXECUTIVO
FIM
vai para proxima declaração
I1

po pode conter quaisquer caracteres em ASCII exceto "LINE FEED" e "RETURN" que indicam o fim do statement.

Label Espaço Mnemônica Espaço Operando(s) Espaço Coment LFCR

O uso do label é opcional. Se o statement possui-lo, este deverá ser escrito imediatamente após o caracter ## , caso contrário um espaço deverá separar a mnemônica deste caracter.

3.4.2 — Tabelas do Macro Assembler

Três tabelas são utilizadas na montagem do programa fonte. São elas: a tabela de símbolos, a tabela de endereços e a tabela de instruções. A seguir serão descritas as organizações destas tabelas.

3.4.2.1 — Tabela de Símbolos

Em sistemas onde a montagem de programas fonte é feita ON LINE, por exemplo, em sistemas de processamento do tipo BATCH o tempo de montagem destes programas é importante no cálculo do Throughput do mesmo. No sistema PROSAD a montagem do programa fonte é feita OFF-LINE e tal fator deixa de ser relevante. Ao se implementar o algoritmo para a construção e uso desta tabela, pensou-se, sobretudo, na simplicidade do mesmo.

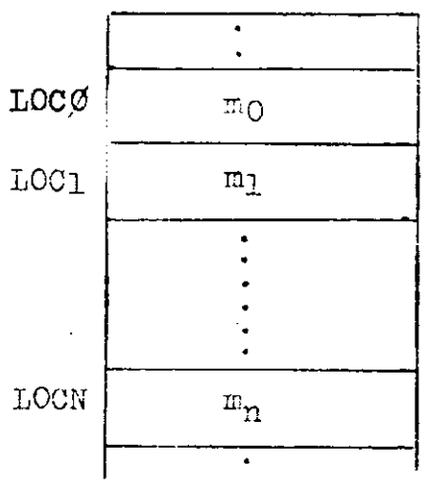
Geralmente uma tabela de símbolo consiste dos vários símbolos utilizados no programa fonte e seus valores numéricos efetivos, valores estes que são definidos pelo programador no

caso de o símbolo ser um operando e pelo próprio assembler, no caso do símbolo ser um label. Cada símbolo ocupa nesta tabela um número predefinido de bytes. Para cada caracter do símbolo um byte é utilizado. No assembler do microprocessador MC 6800 (sistema motorola), por exemplo, os símbolos do programa fonte podem conter até 6 caracteres. Isto implica que cada símbolo ocupará na tabela 8 bytes: 6 bytes para armazenamento do símbolo e 2 para armazenamento do seu valor efetivo.

No PROSAD, durante a operação de montagem do programa fonte, deve estar armazenado em uma área da memória denominada Buffer do programa fonte. Tal programa é armazenado neste buffer através do programa carregador. Com isto o programa fonte que contém todos os símbolos pode ser utilizado como parte da tabela de símbolos minimizando assim o espaço ocupado pela mesma e permitindo o uso de símbolos de comprimento ilimitado.

Montagem da Tabela:

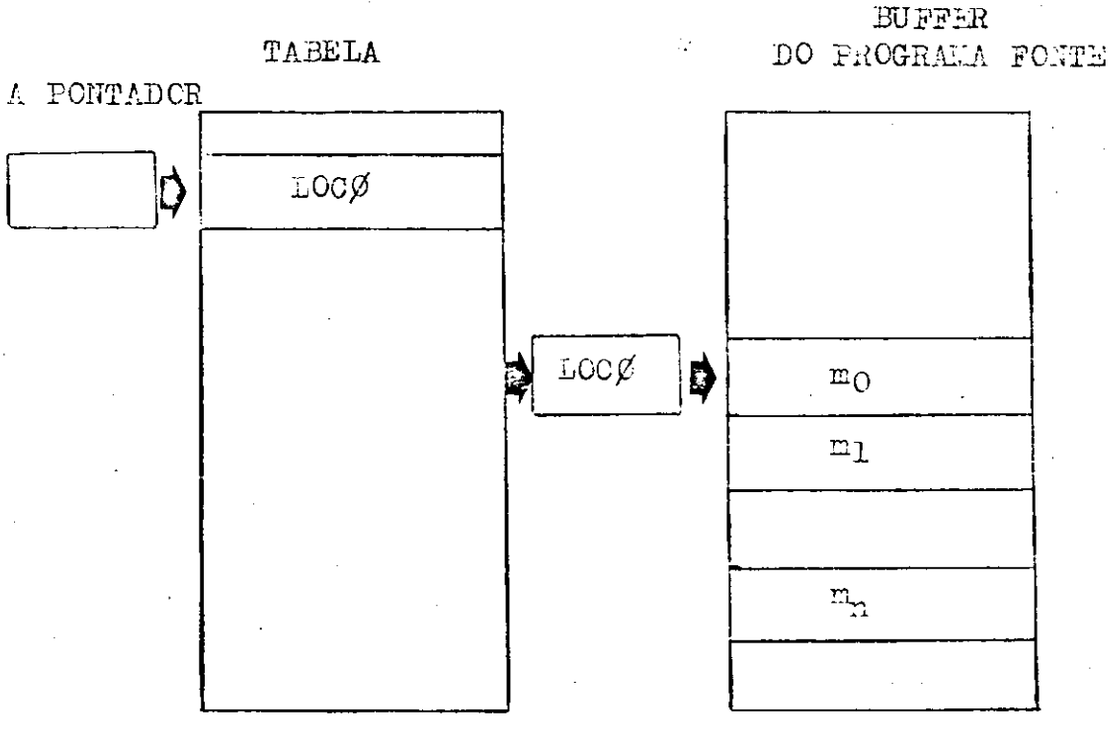
Suponha-se um certo símbolo M representado pelos caracteres m_0 m_1 m_2 m_3 ..... m_n. Tal símbolo ocupa no buffer de programa fonte n locações de memória (LOC ϕ, LOC 1, ... LOC N)



O vetor de posição do símbolo é simplesmente o vetor de LOCϕ que é o valor do apontador de programa fonte no instante em que o Assembler inicia a leitura do símbolo. Com isto obtém-se uma diminuição considerável no tamanho da tabela.

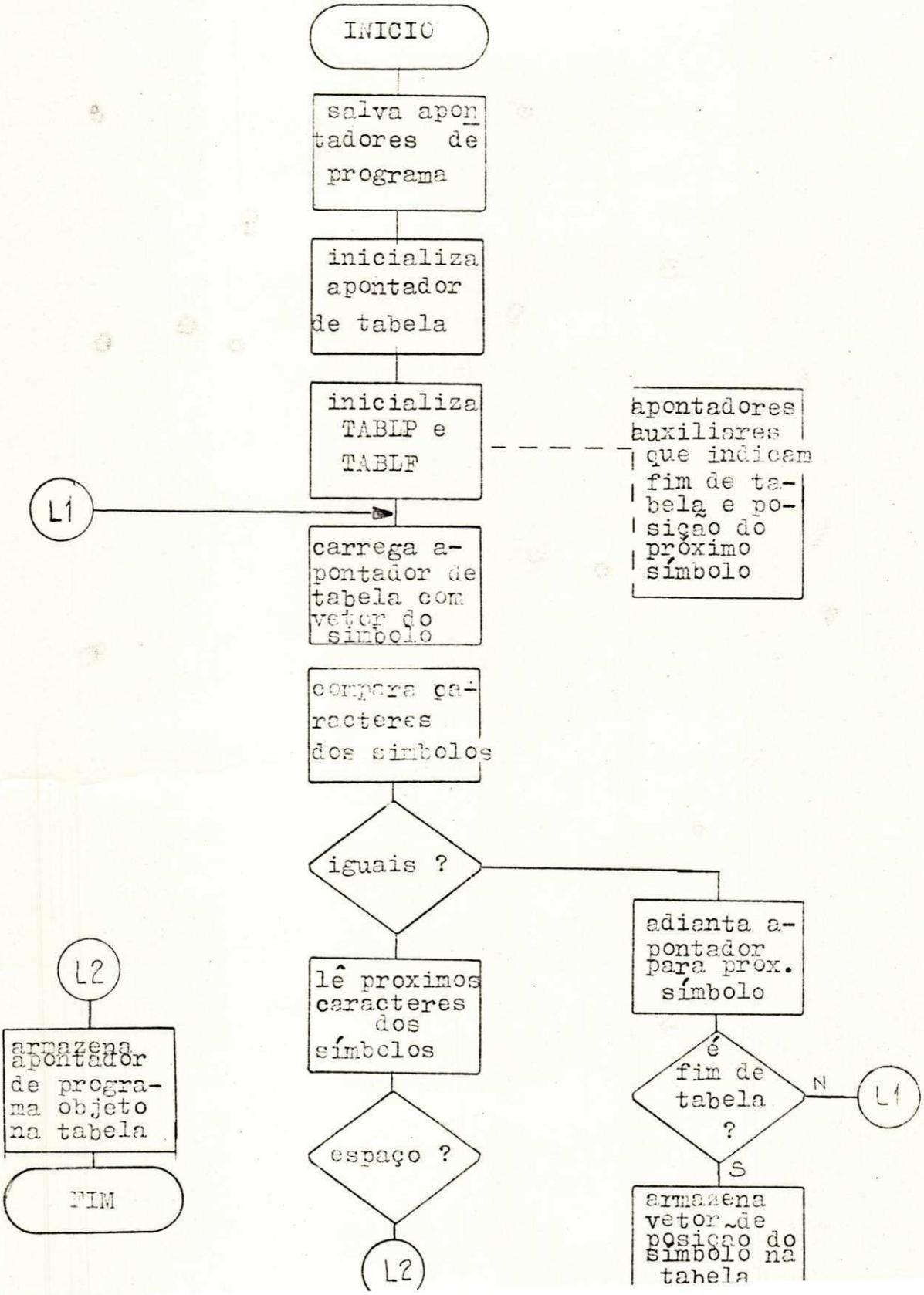
Procura de um símbolo

A operação de procura de um símbolo nesta tabela é feita de modo muito simples: um apontador de tabela chamado apontador de tabela de símbolos deverá varrer a tabela lendo os símbolos na ordem em que os mesmos foram postos na tabela. Para o Assembler a tabela se comporta como se o próprio símbolo estivesse armazenado na mesma. O que ocorre, no entanto, é que a cada leitura de um novo símbolo o apontador será carregado com o vetor de posição deste símbolo deslocando-se desta forma para o buffer do programa fonte, onde o símbolo se encontra armazenado.



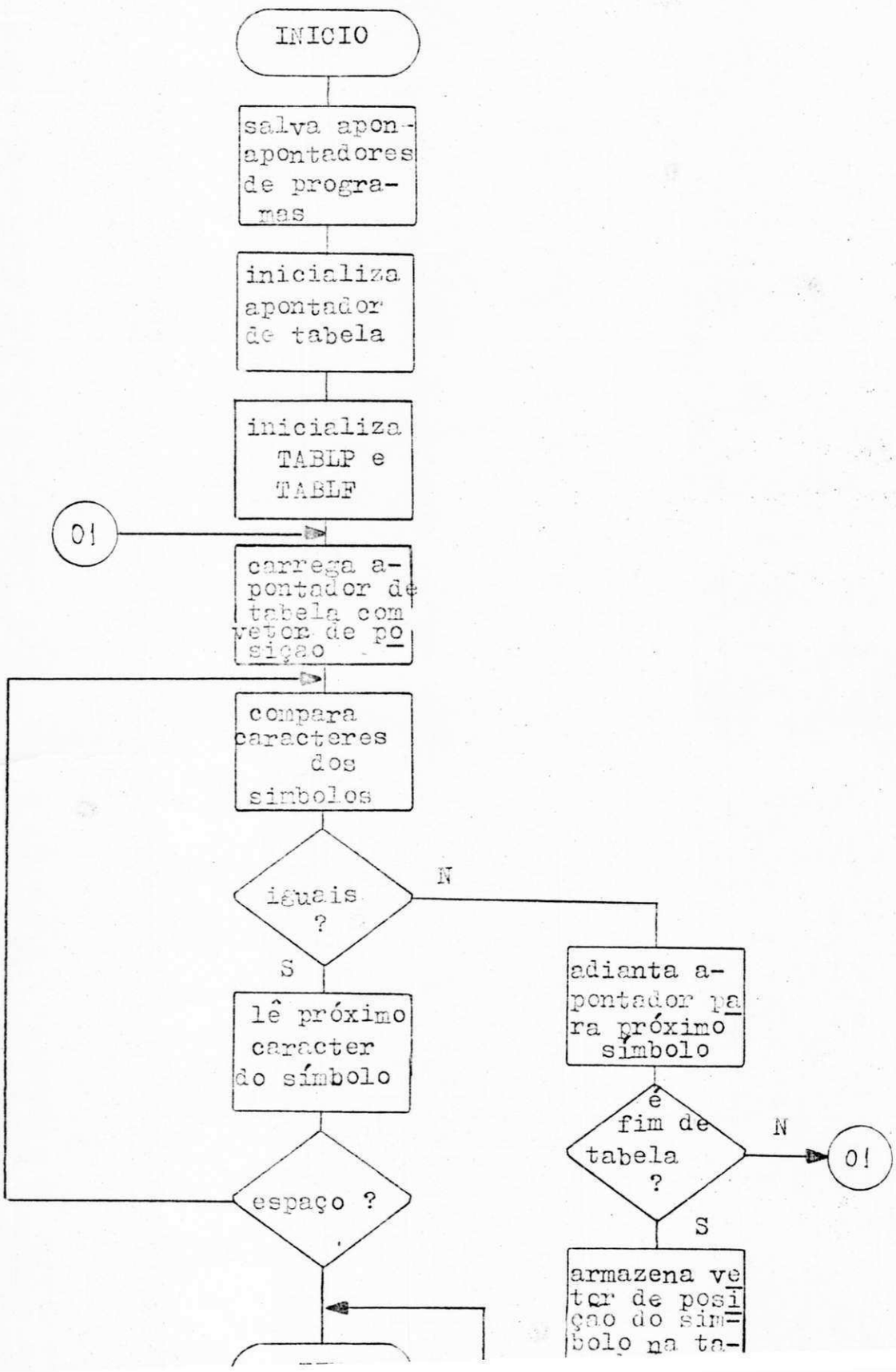
MONTAGEM DA TABELA DE SÍMBOLOS

a) SÍMBOLO É UM " LABEL "



MONTAGEM DA TABELA DE SÍMBOLOS

b) SÌMBOLO É UM OPERANDO



3.4.2.2 – Tabela de Instruções

A tabela de instruções armazena dois tipos de informação: a informação a respeito da mnemônica e um vetor que indica onde inicia a rotina que o Assembler deve executar para montar a instrução correspondente a esta mnemônica.

Visando condensar tal tabela, uma mnemônica é representada numa forma condensada. Ao invés de se representar a mesma através de todos os seus caracteres, um byte hexadecimal que é o resultado da operação de "OU exclusivo" entre todos os seus caracteres é utilizado. Desta forma a mnemônica é comprimida.

Seja uma certa mnemônica constituída de n caracteres:

$$N = n_1\ n_2\ n_3\ n_4\ \ldots\ n_n$$

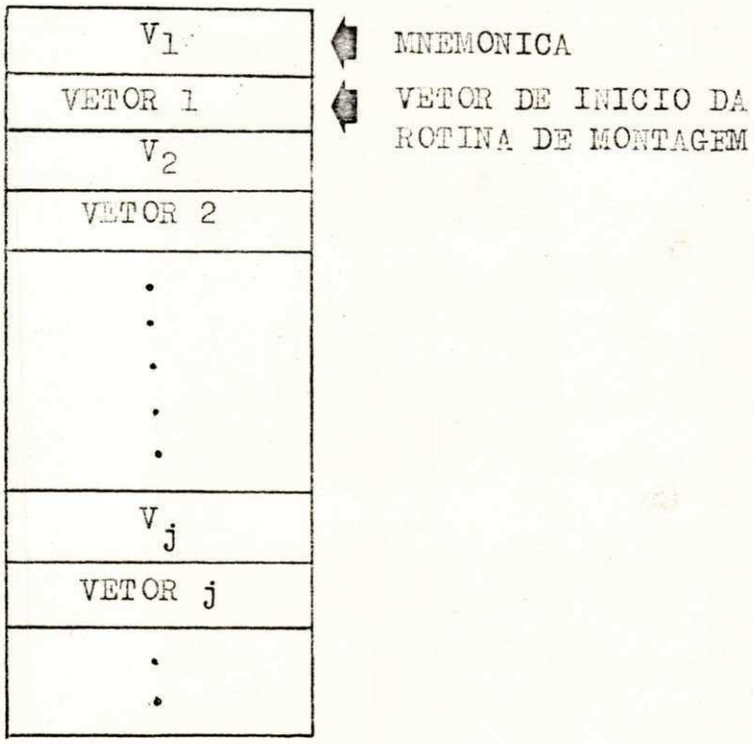
$$V = n_1 + n_2 + n_3 + n_4 + \ldots + n_n$$

onde V é o valor que representa a mnemônica N na tabela.

Esta forma de representação é bastante limitada, uma vez que existe a possibilidade de duas mnemônicas quaisquer distintas entre si possuírem o mesmo valor V. No caso do macro assembler em questão tal método foi utilizado depois de se verificar que todas as mnemônicas existentes na linguagem possuem valores diferentes de V. A grande vantagem desta forma de representação é a diminuição do espaço de memória ocupado pela tabela de instruções.

3.4.3 – Quadro de Instruções – Pseudo Instruções e Macro Instruções da Linguagem Assembly

INSTRUÇÃO	MNEMÔNICA	VALOR DE V
Nenhuma operação	NOP	51
Chama subrotina	CHAME	82
Saia da subrotina	RETOR	5E
Pseudo instrução	MNEMÔNICA	
Resolve um símbolo	IGUAL	56
Define um texto	TEXTO	52
Determina fim de programa fonte	FIM	42
Inicia uma task	EXECUTE	5F
Macro instrução		
Limpa cartão de E/S	RESET	55
Lê cartão de entrada de 8 bits na forma dedicada	LEIA	01
12 bits	LEIA(E)	45
Escreve dado de 8 bits em cartão de saída (dedicado)	ESCR	07
12 bits	ESCR(E)	43
Imprime relatório	IMPR	06
Determina fim de task e retorna p/ EXECUT.	FINAL	94
Desvio incondicional	PULE	0C
Desvio condicional	DESVIE	08

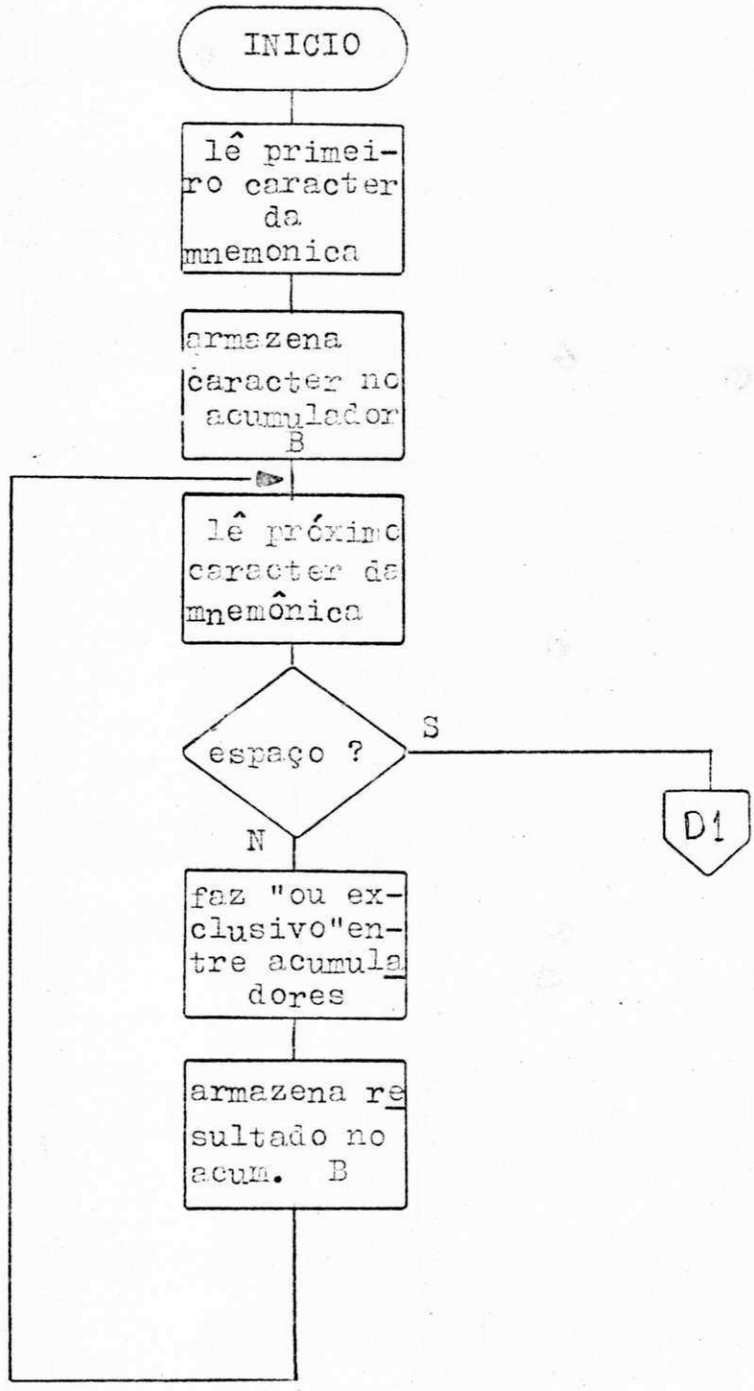


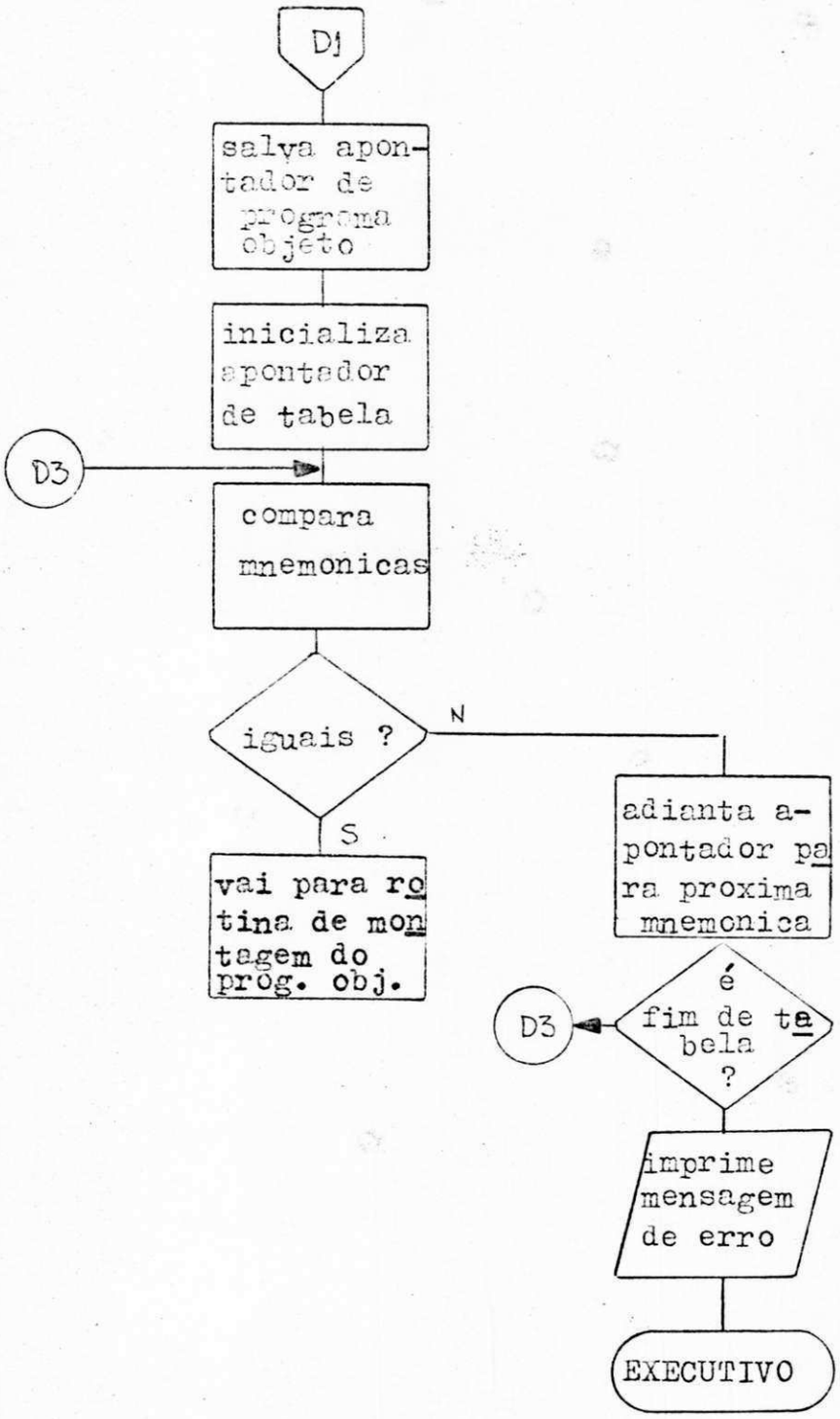
3.4.4 — Montagem das Instruções e Macro Instruções

A montagem destes dois tipos de instrução consiste basicamente na transferência, para o programa objeto, do correspondente em linguagem de máquina da instrução na linguagem assembler. Este correspondente pode ser uma só instrução ou uma rotina no caso da instrução ser uma macro, conforme foi dito anteriormente. Uma vez identificada a mnemônica o assembler executará a rotina de montagem desta instrução.

A operação de montagem de instruções e macro instruções é basicamente a mesma, havendo pequenas diferenças de instrução em função do número de seus operandos.

OPERAÇÃO DE LEITURA E IDENTIFICAÇÃO DE UMA INSTRUÇÃO DO PROGRAMA FONTE




DJ
salva apontador de programa objeto
inicializa apontador de tabela
D3
compara mnemonicas
iguais ?
N
S
vai para rotina de montagem do prog. obj.
adianta apontador para proxima mnemonica
é fim de tabela ?
D3
imprime mensagem de erro
EXECUTIVO

3.4.4.1 – Montagem de instruções sem operando:

O apontador de objeto serve para identificar na área da memória onde se encontram as rotinas e instruções que irão ser colocadas no programa objeto. A estas rotinas e instruções foi dado o nome de "objeto". Cada instrução possui o seu objeto correspondente armazenado em área própria da memória.

Para fins de melhor compreensão da operação de montagem de uma instrução deste tipo, tal operação é ilustrada a seguir:

Suponha-se que a instrução NOP deverá ser montada: Depois de identificada a instrução o STACK POINTER do microprocessador é carregado com o endereço do objeto da instrução NOP. Uma outra rotina chamada "MONTE" transfere este objeto para o programa objeto.

3.4.4.2 – Montagem das Instruções e Macro Instruções de um Operando

A diferença básica entre a montagem de instruções de um operando e a instrução NOP é que neste caso um operando deverá ser introduzido no objeto da instrução.

A transferência do objeto para o programa objeto neste caso é um pouco mais complexa que na instrução NOP.

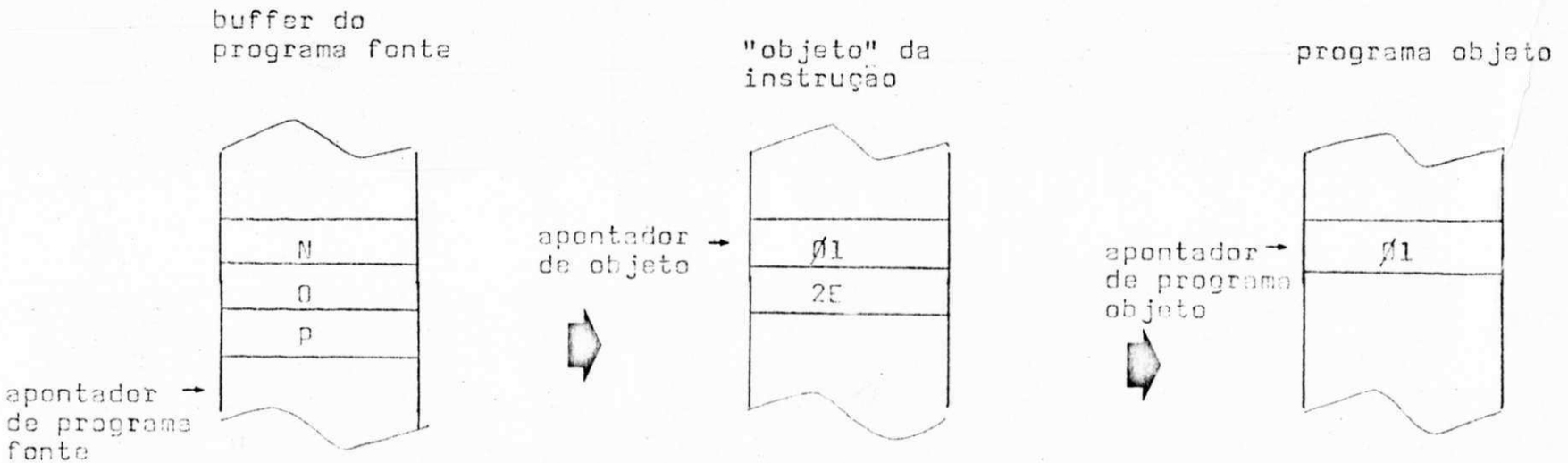
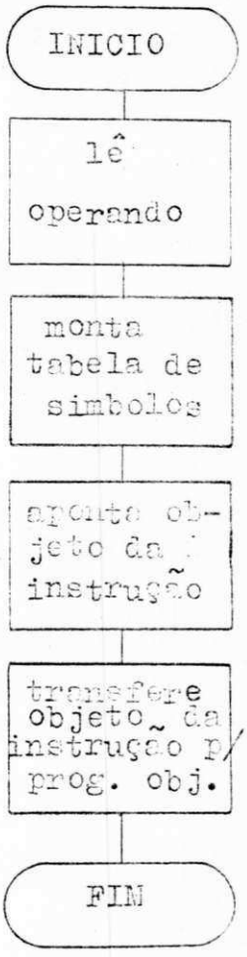
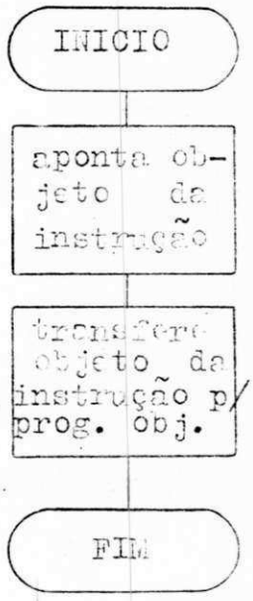


FIGURA 3.5 montagem de uma instrução

MONTAGEM DE INSTRUÇÃO COM UM OPERANDO



MONTAGEM DE INSTRUÇÃO SEM OPERANDO



A rotina que faz esta montagem é a mesma, no entanto, trata-se da rotina MONTE.

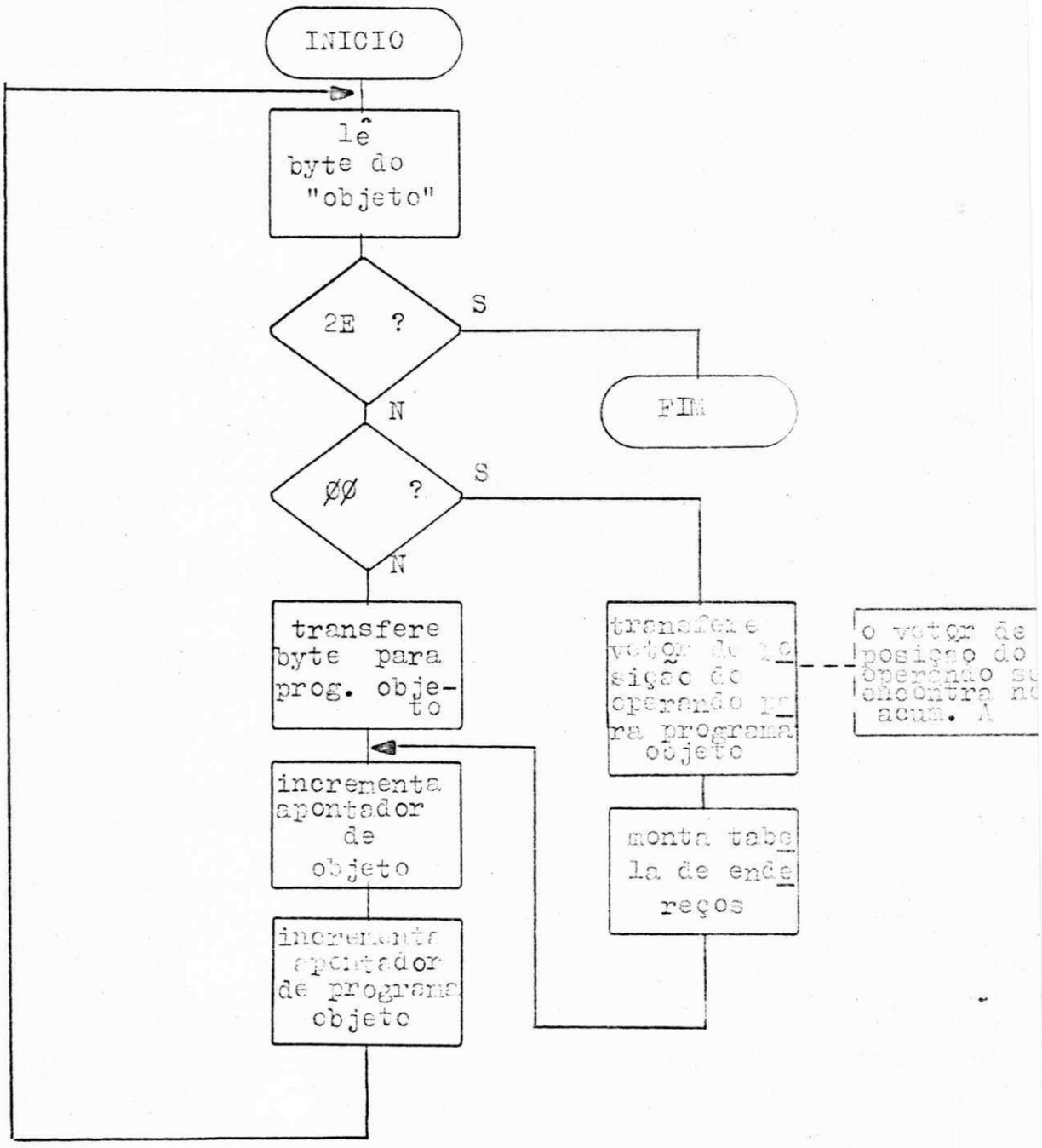
A função básica desta rotina é ler um a um os bytes do objeto da instrução e transferi-los para o programa objeto. Este processo deve continuar até que um byte indique o fim do objeto.

Sempre que a rotina encontrar um byte igual a zero, este byte não deverá ser transferido. No seu lugar deverá ser colocado o operando da instrução. Como neste estágio de montagem existe a possibilidade do operando ainda não ter sido resolvido será colocado no programa objeto o seu vetor de posição LOCϕ (veja tabela de símbolos). Um byte igual a 2E ("ponto" no código ASCII) indicará a rotina MONTE que o objeto terminou.

3.4.4.3 – A Tabela de Endereços

Depois de terminada a primeira fase da montagem do programa fonte o Assembler deverá substituir todos os vetores de posição dos operandos pelos seus valores efetivos. O aspecto do programa objeto antes desta segunda fase é mostrado na figura

SUBROTINA " MONTE "



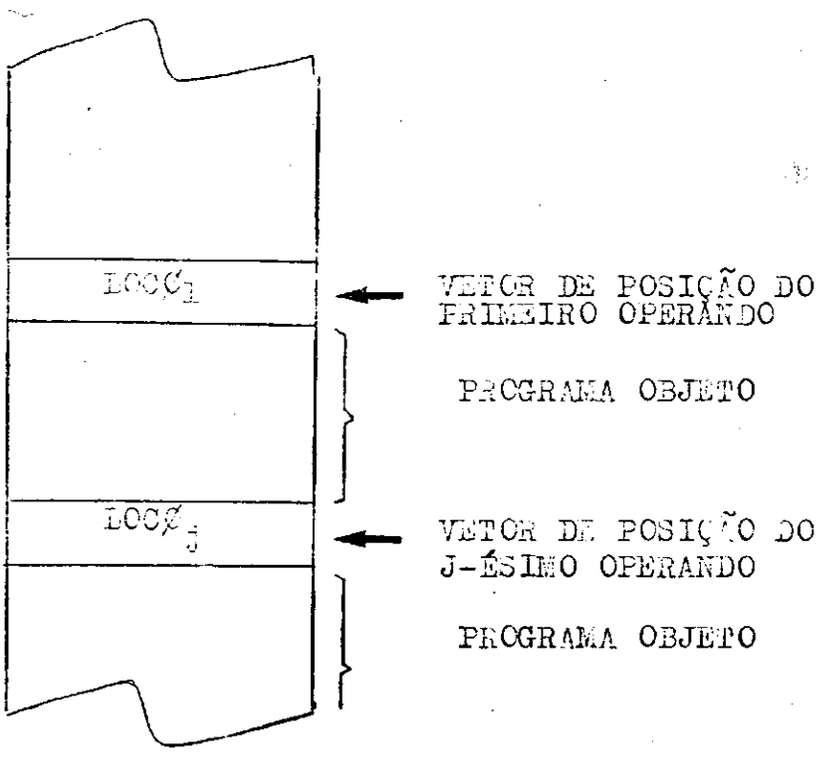


FIGURA 3.6 Programa objeto antes da operação de substituição dos operandos por seus valores efetivos.

Ao efetuar o processo de substituição o assembler deverá saber onde se encontram, no programa objeto, os operandos que deverão ser substituídos. A informação sobre estas posições é obtida pelo assembler na tabela de endereços. Esta tabela consiste dos valores das posições na memória destes operandos. Sempre que um operando é colocado no programa objeto, o valor do apontador de programa objeto naquele instante é colocado na tabela de endereços.

3.4.4.4 – Montagem de Macro Instruções de mais de um Operando

A montagem de uma instrução deste tipo consiste basicamente na execução, repetidas vezes, da operação de montagem das instruções de um operando. Neste caso, o objeto de instrução é dividido em várias partes separadas pelo caracter ponto. O número de divisões deste objeto deve ser igual ao número de operandos da instrução, sendo que cada uma destas divisões deve conter um dos operandos.

3.4.5 – Instruções do Macro Assembler

NOP

A instrução NOP provoca um incremento no valor do contador de programa do microprocessador. A implementação desta instrução parece estranhar a princípio, uma vez que das 72 instruções do Assembly do microprocessador só a mesma foi implementada. Isto se deveu ao fato de que os statements da linguagem fonte que possuem o código de operação em linguagem de máquina não podem conter label's. Desta forma as mesmas não podem originalmente serem destinos de instruções de desvio. Como a instrução NOP pode conter label pode-se lançar mão do recurso de utilizar a mesma antes de um destes statements quando o mesmo for um destino. O desvio seria feito não para este statement, mas para o statement anterior que contém um NOP.

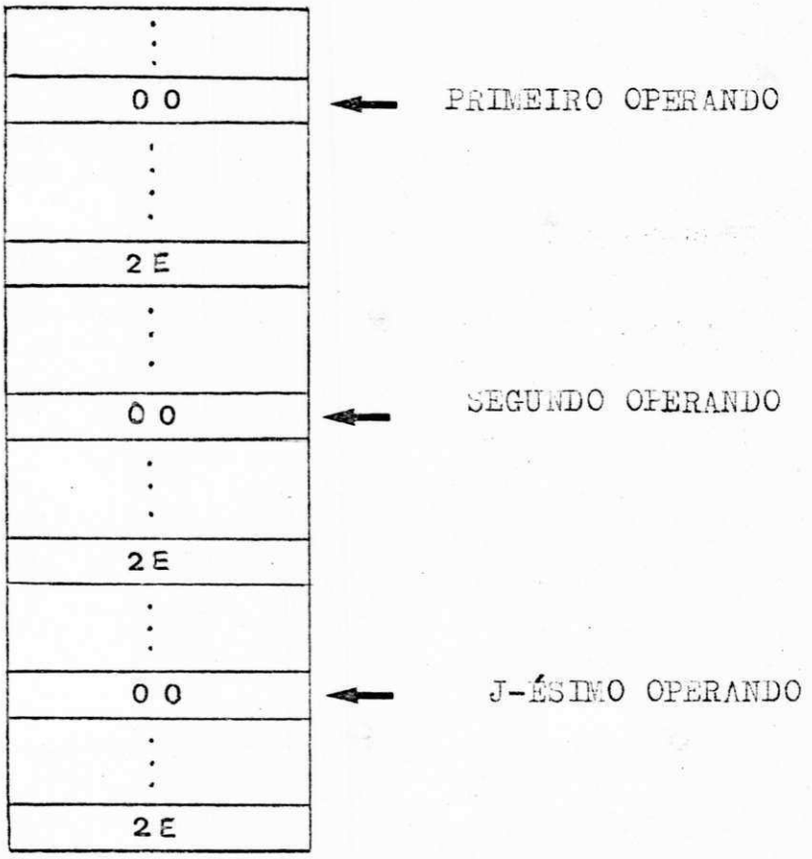
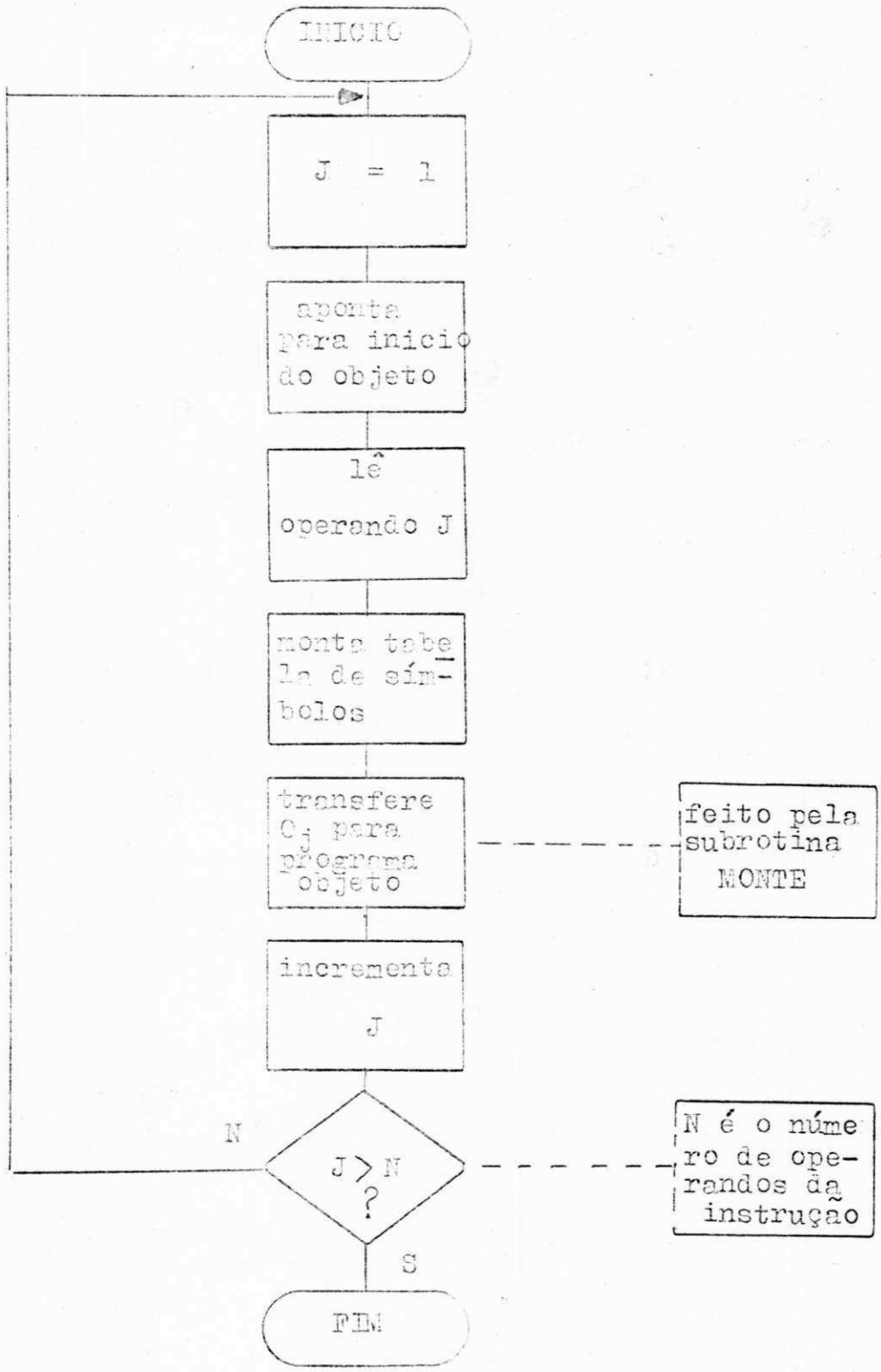


FIGURA 3. Objeto de uma instrução de mais de um operando.

MONTAGEM DE INSTRUÇÕES DE MAIS DE UM OPERANDO



CHAME

A instrução CHAME faz o microprocessador desviar para a subrotina cujo nome é o operando da mesma.

RETOR

A instrução RETOR indica o fim de uma subrotina, fazendo com que o microprocessador retorne ao programa principal. Esta instrução não contém operando.

LEIA

Esta macro instrução faz o microprocessador ler de forma dedicada o dado do cartão representado no seu campo do operando.

Esta macro instrução utiliza dois acumuladores do microprocessador. Qualquer dado que esteja armazenado num destes acumuladores antes da execução da mesma será portanto destruído. Esta macro possui um operando.

ESCR

A macro instrução ESCR instrui o microprocessador a escrever no processo através de um cartão de saída digital, o dado anteriormente armazenado no acumulador A. A operação é feita no modo dedicado.

A exemplo da macro instrução LEIA, ESCR também utiliza os dois acumuladores do microprocessador. Esta macro possui um só operando que é o cartão a ser utilizado. O programador deve antes de colocar esta instrução no programa carregar o acumu

lador A do microprocessador com o dado a ser transferido para o processo.

LEIA(E)

A macro instrução LEIA(E) instrui o microprocessador a ler um dado do processo através de um cartão de entrada digital de 12 bits. Nesta leitura o dado resultante fica disponível nos acumuladores A e B do microprocessador, os 8 bits menos significativos no acumulador A e os 4 bits mais significativos no acumulador B.

Neste tipo de leitura, o dado é também armazenado no registro reservado ao cartão na primeira página da memória. É importante notar aqui que o dado ocupará dois bytes de memória. Desta forma é necessário se deixar um endereço acima do endereço do cartão, desocupado.

ESCR(E)

Esta macro instrução instrui o microprocessador a escrever no processo, através de um cartão digital de 12 bits, um dado qualquer. O dado a ser transferido deve se encontrar inicialmente nos acumuladores A e B, sendo os 8 bits menos significativos no acumulador A e os quatro mais significativos no acumulador B.

Esta macro instrução utiliza o registro de índice do microprocessador. Devido a isto o conteúdo do mesmo antes da execução da mesma será modificado.

FINAL

A instrução FINAL indica o término de uma tarefa qualquer do programa de aplicação e instrui o microprocessador a retornar ao programa executivo. Esta instrução não possui operando.

PULE

A instrução PULE representa um desvio incondicional na seqüência de execução do programa. O desvio será feito para a declaração cujo label for o operando desta instrução.

DESVIE

A macro instrução DESVIE instrui o microprocessador a efetuar o desvio condicional na seqüência de execução do programa. Esta macro possui três operandos que devem ser separados entre si por um único espaço. O primeiro operando deve indicar o destino do desvio caso o resultado da última operação tenha sido negativo, o segundo operando é o destino caso o resultado tenha sido nulo, e o terceiro é o destino, caso o resultado tenha sido positivo.

O resultado do teste é feito nos bits Z (zero) e N (negativo) do registro de código de condição do microprocessador.

Ao se observar o conjunto de instruções do MC 6800 (ver apêndice V) se pode constatar que o microprocessador pode executar desvios a partir de testes de outros bits de status, além do bit zero e negativo. A instrução DESVIE não cobre todos estes tipos de teste de maneira que diante da necessidade de pro

gramar um desvio baseado, por exemplo, em um teste de overflow, o programador deverá executar este desvio em linguagem de máquina. Poderá também ser utilizada a instrução PULE em conjunto com o teste em linguagem de máquina.

RESET

Esta macro instrução instrui o microprocessador a "resetar" qualquer um dos cartões de E/S do sistema.

IMPR

A macro instrução IMPR instrui o microprocessador a imprimir no terminal um texto qualquer definido pela diretiva TEXTO. O nome do texto a ser impresso é o operando da instrução. Como foi visto na diretiva TEXTO, um texto pode ser composto de qualquer caracter ASCII exceto * (asterisco). A principal característica desta instrução é que, além de ser impresso um texto em ASCII, podem ser também impressos, dados armazenados na memória do sistema.

Suponha-se, por exemplo, que o resultado da medição de uma temperatura através do cartão de E/S cujo endereço é FA deve ser periodicamente impresso no terminal da seguinte forma:

Temperatura do Ar = ——————— Graus Celcius
(Valor da Temp.)

O programador define então este texto no seu programa, da seguinte forma:

* ## Relatório Texto (CR) (LF)

Temperatura do Ar = (FA) Graus Celcius*

Ao executar a macro instrução ## IMPR RELATÓRIO o microprocessador imprimirá este texto no terminal. Ao encontrar o caracter (parêntesis) o mesmo entenderá que os próximos caracteres até o fechamento do parêntesis representam um endereço de memória que contêm um dado que deve ser impresso. Este dado será então lido, convertido para decimal, formatado em ASCII e impresso. Depois disto o micro continuará imprimindo o texto até encontrar o caracter * (asterisco) que indica o fim do texto. Suponha-se, por exemplo, que o conteúdo do endereço FA é 1B. O seguinte texto será então impresso:

Temperatura do Ar = 27 Graus Celcius.

3.4.3 – Diretivas do Macro Assembler

IGUAL

A diretiva IGUAL atribui ao seu LABEL o valor em hexadecimal contido no seu operando. Uma vez que o valor máximo deste label é FF o campo do operando desta diretiva só deve conter dois caracteres.

Cada símbolo ocupa na tabela de símbolos 3 bytes.

Como o comprimento de cada símbolo na tabela é fixo e existem símbolos de valores restritos a 1 byte e outros a 2 bytes, sempre que a diretiva IGUAL resolve um símbolo na tabela, o byte menos significativo do valor deste símbolo é feito igual a ϕ1. Este artifício é utilizado porque no programa objeto todos estes símbolos ocupam 2 bytes e muitas vezes o código de operação que precede estes símbolos requer operandos de um byte. O número ϕ1 é interpretado pelo microprocessador como a instrução NOP. Desta forma o microprocessador simplesmente ignorará o segundo byte do operando.

TEXTO

A diretiva TEXTO define um texto formado de quaisquer caracteres em ASCII exceto o caracter * que determina para o Assembler o fim do texto e os caracteres "("e")" (parêntesis) que definem o endereço de um dado a ser incorporado ao texto. Basicamente o que esta diretiva faz é definir um nome para um texto qualquer. Este texto pode ter várias linhas, uma vez que LF e CR são caracteres válidos para o mesmo.

Exemplo de um statement válido com a diretiva TEXTO:

RELATÓRIO1 TEXTO (CRLF) PRESSÃO DA VÁLVULA 3 É IGUAL A (Aϕ) PSI *

↑
LABEL

Uma vez definido que o relatório 1 é o texto contido no operando do statement sempre que se quiser imprimir este texto basta instruir o microprocessador a executar IMPR RELATÓRIOS.

Ao imprimir o texto o microprocessador substituirá (Aϕ) pelo conteúdo em hexadecimal do endereço Aϕ da memória.

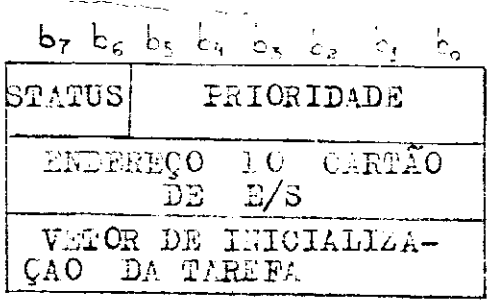
FIM

A diretiva FIM indica para o assembler o fim do programa fonte. Esta instrução não possui label nem comentários.

EXECUTE

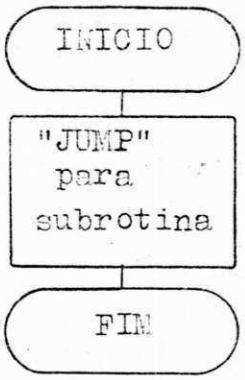
A diretiva EXECUTE deve iniciar todas as tarefas a serem executadas pelo microprocessador em tempo real. Sua importância na elaboração dos programas de aplicação é primordial, uma vez que a tabela de tarefas é montada a partir da mesma. Esta diretiva não deve possuir label. Seu operando deve ser um número hexadecimal de 2 caracteres representando o endereço do cartão de E/S que solicita a ativação da tarefa que segue tal diretiva. Diz-se que uma tarefa é ativada quando o controle do microprocessador é entregue à mesma. A compreensão clara desta diretiva só é possível uma vez compreendida também os princípios da operação do sistema. Sabe-se que no PROSAD dois programas são responsáveis pela supervisão da operação em tempo real do sistema. O programa executivo e o programa escalonador de tarefas (tasks). A tabela de tarefas de (tasks) deve conter a informação essencial acerca das tarefas a serem executadas pelo microprocessador. Estas informações são: prioridade da tarefa, cartão de E/S que ativa a tarefa e posição da tarefa na memória do sistema. A diretiva EXECUTE põe todas estas informações na tabela de tarefas. A tabela de tarefas reserva 3 bytes para o armazenamento dos dados acerca de cada tarefa, sendo estes da

dos armazenados na ordem mostrada na figura



A determinação da prioridade de uma tarefa é feita automaticamente a partir da ordem da posição da mesma no programa. Isto quer dizer que a primeira tarefa a ser descrita no programa terá a maior prioridade e a última a menor prioridade. A função da diretiva EXECUTE é colocar nesta tabela a prioridade da task, seu número ou cartão de E/S e o valor do contador de programa que inicia esta task. O status da tarefa durante esta operação deve ser $\phi\phi$ indicando que a tarefa está inativa.

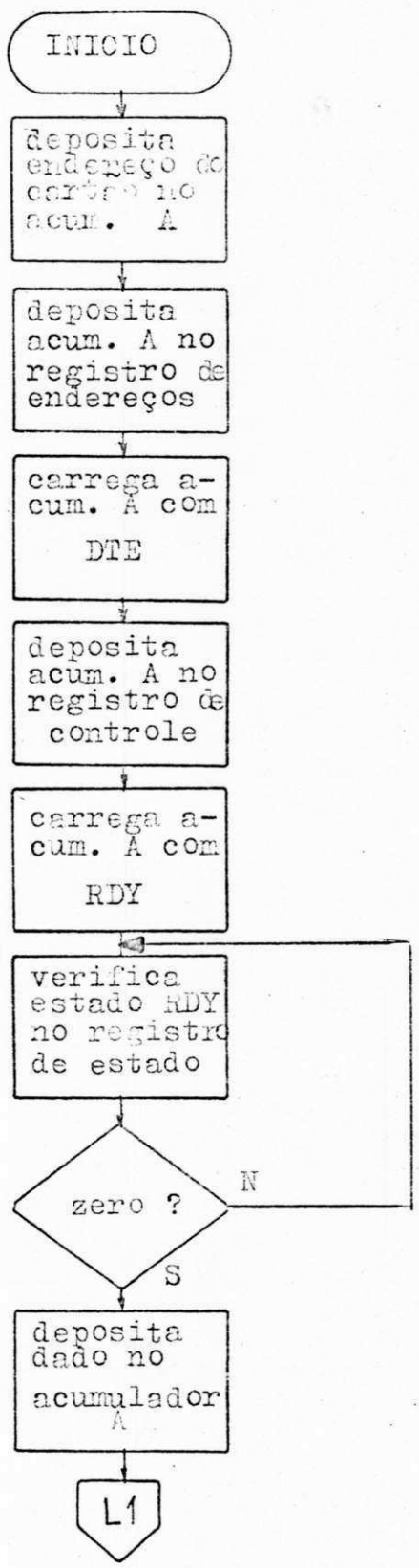
CHAME

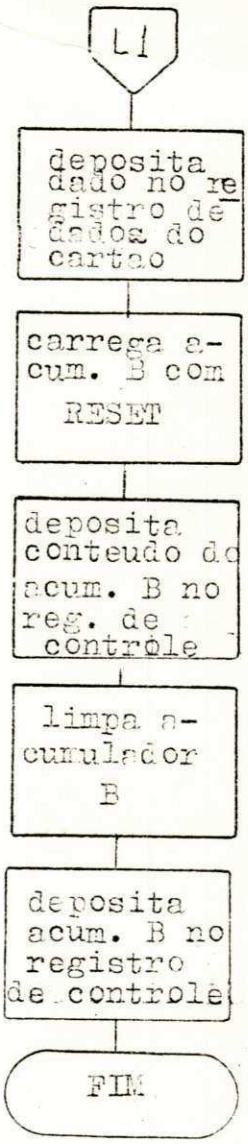

INICIO
"JUMP"
para
subrotina
FIM

RETOR

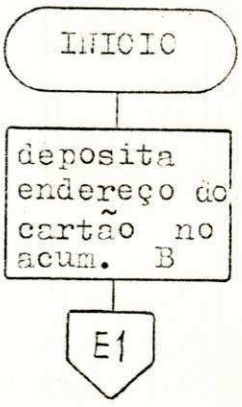

INICIO
retorna
de uma
subrotina
FIM

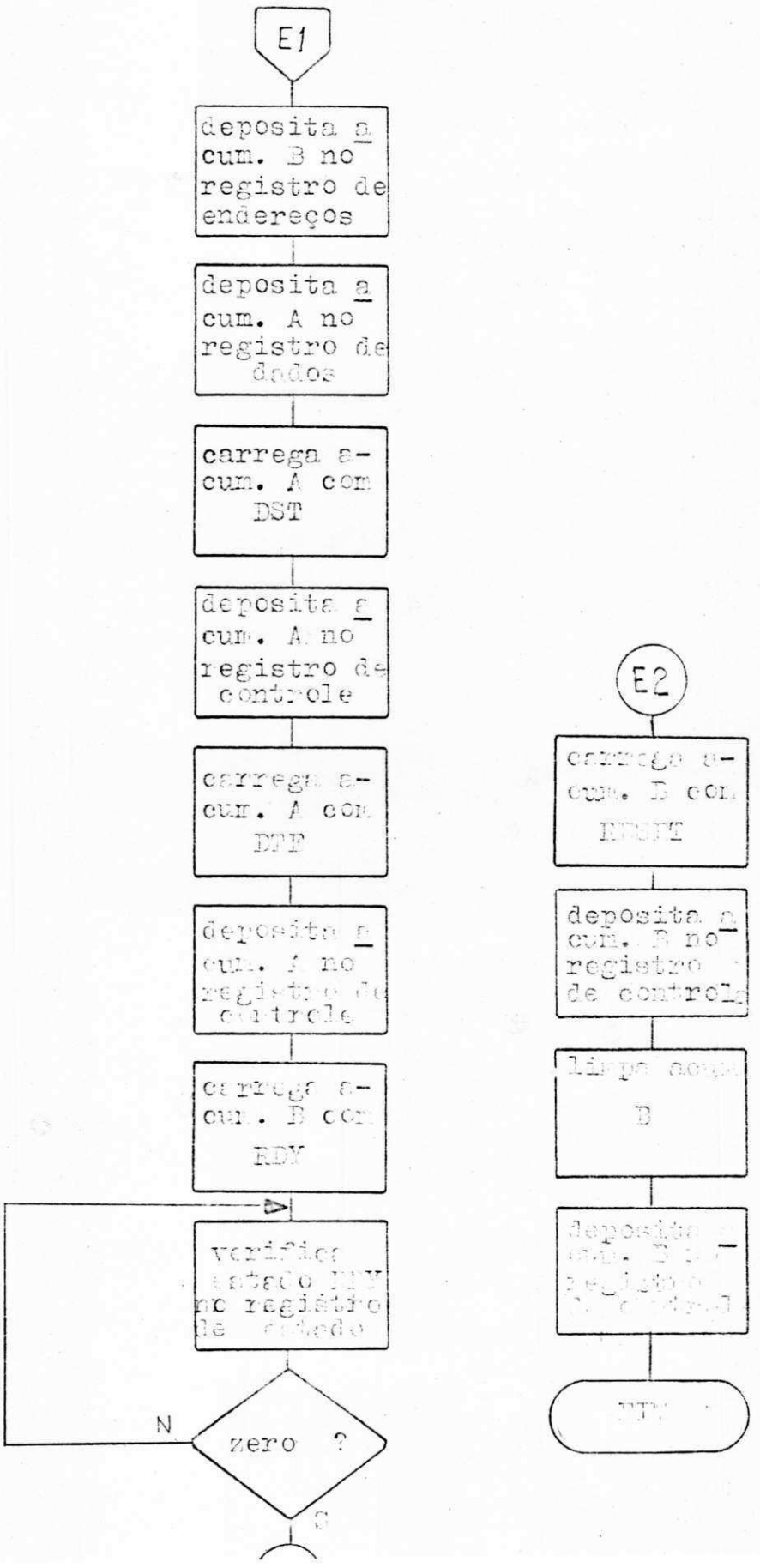
LEIA


INICIO
deposita endereço do cartão no acum. A
deposita acum. A no registro de endereços
carrega acum. A com DTE
deposita acum. A no registro de controle
carrega acum. A com RDY
verifica estado RDY no registro de estado
zero ?
N
S
deposita dado no acumulador A
L1

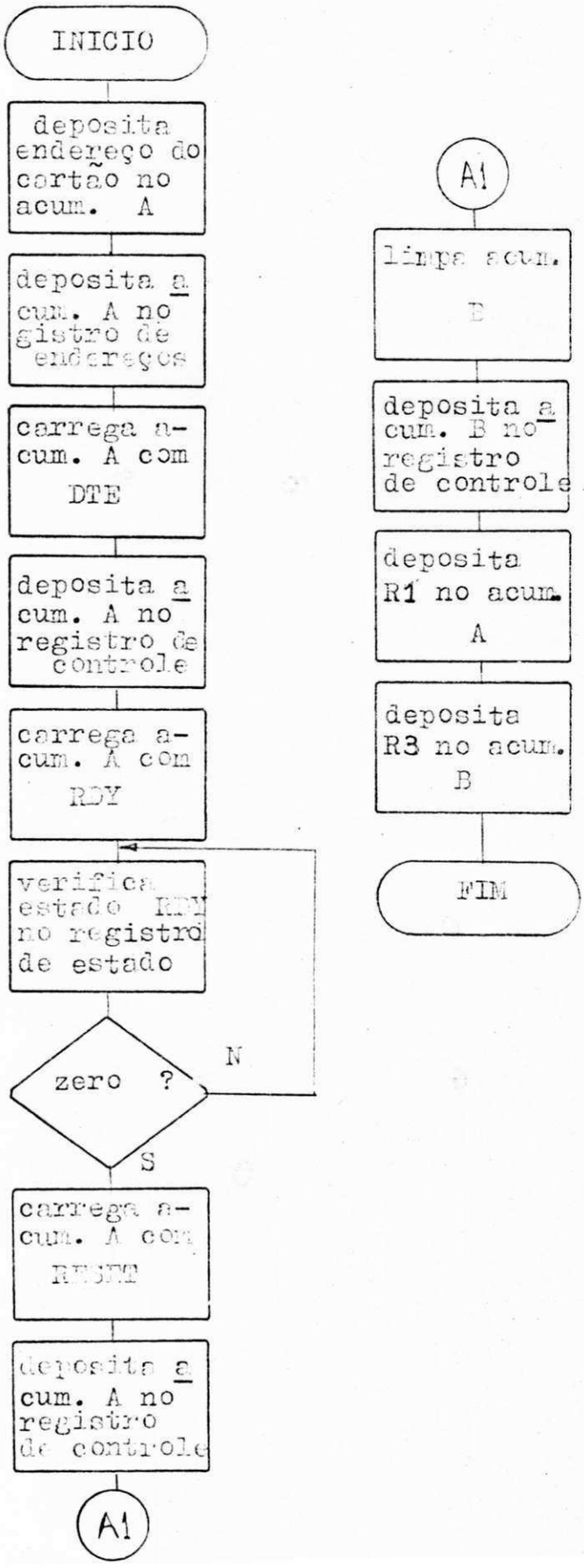

L1
deposita dado no registro de dados do cartao
carrega acum. B com RESET
deposita conteudo do acum. B no reg. de controle
limpa acumulador B
deposita acum. B no registro de controle
FIM

ESCR

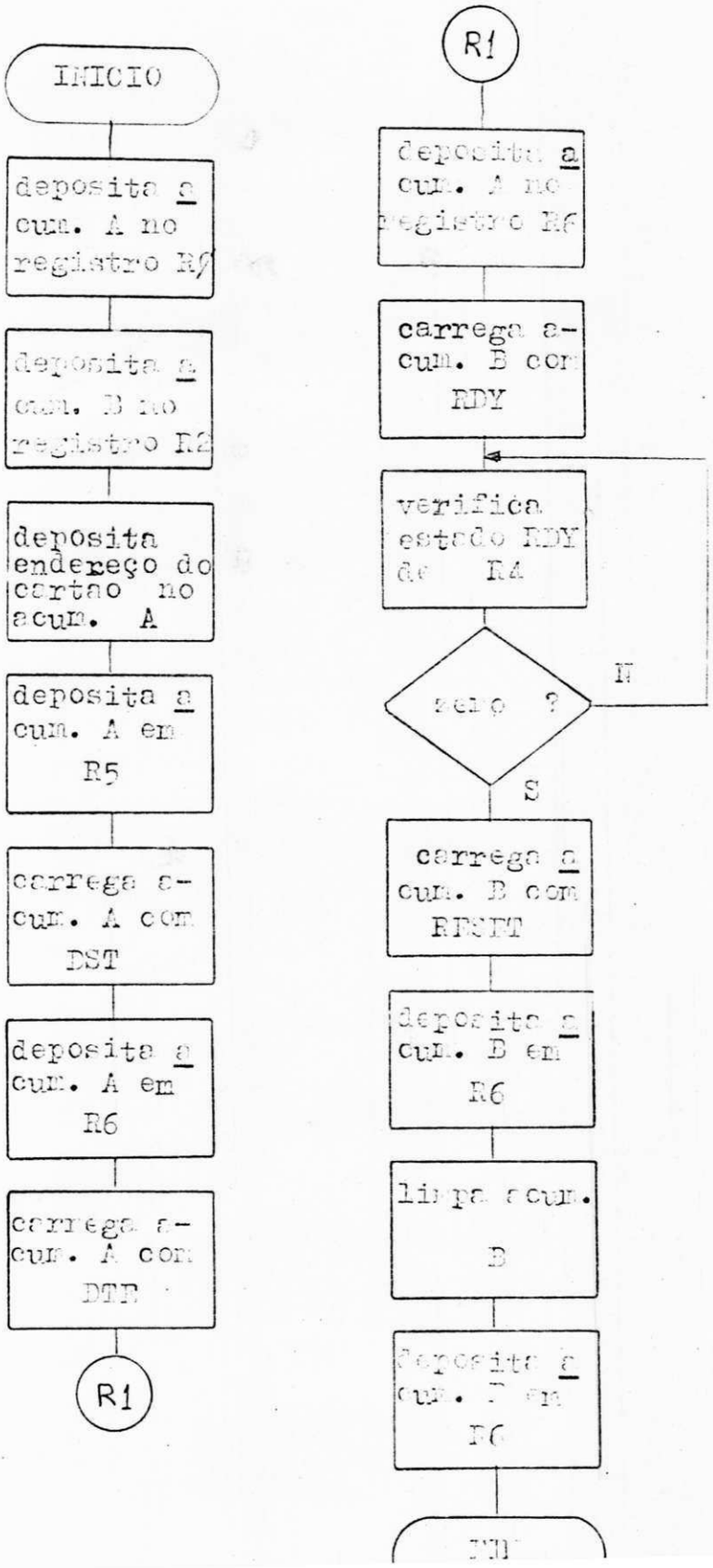

INICIO
deposita endereço do cartão no acum. B
E1


E1
deposita acum. B no registro de endereços
deposita acum. A no registro de dados
carrega acum. A com DST
deposita acum. A no registro de controle
carrega acum. A com DTF
deposita acum. A no registro de controle
carrega acum. B com RDY
verifica estado RDY no registro de estado
N
zero ?
S
E2
carrega acum. B com DRSRT
deposita acum. B no registro de controle
limpa acum. B
deposita acum. B no registro de controle

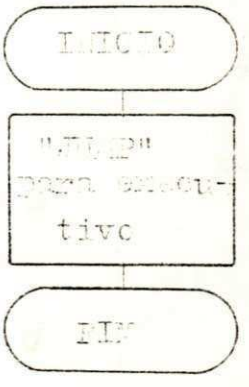
LEIA(E)


INICIO
deposita endereço do cartão no acum. A
deposita acum. A no registro de endereços
carrega acum. A com DTE
deposita acum. A no registro de controle
carrega acum. A com RDY
verifica estado RDY no registro de estado
zero ?
N
S
carrega acum. A com
deposita acum. A no registro de controle
A1
A1
limpa acum. B
deposita acum. B no registro de controle
deposita R1 no acum. A
deposita R3 no acum. B
FIM

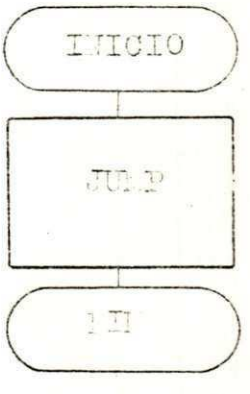
ESCR(E)


INICIO
deposita acum. A no registro R0
deposita acum. B no registro R2
deposita endereço do cartao no acum. A
deposita acum. A em R5
carrega acum. A com DST
deposita acum. A em R6
carrega acum. A com DTR
R1
R1
deposita acum. A no registro R6
carrega acum. B com RDY
verifica estado RDY de R4
N
sinal ?
S
carrega acum. B com RDSTT
deposita acum. B em R6
limpa acum. B
deposita acum. B em R6

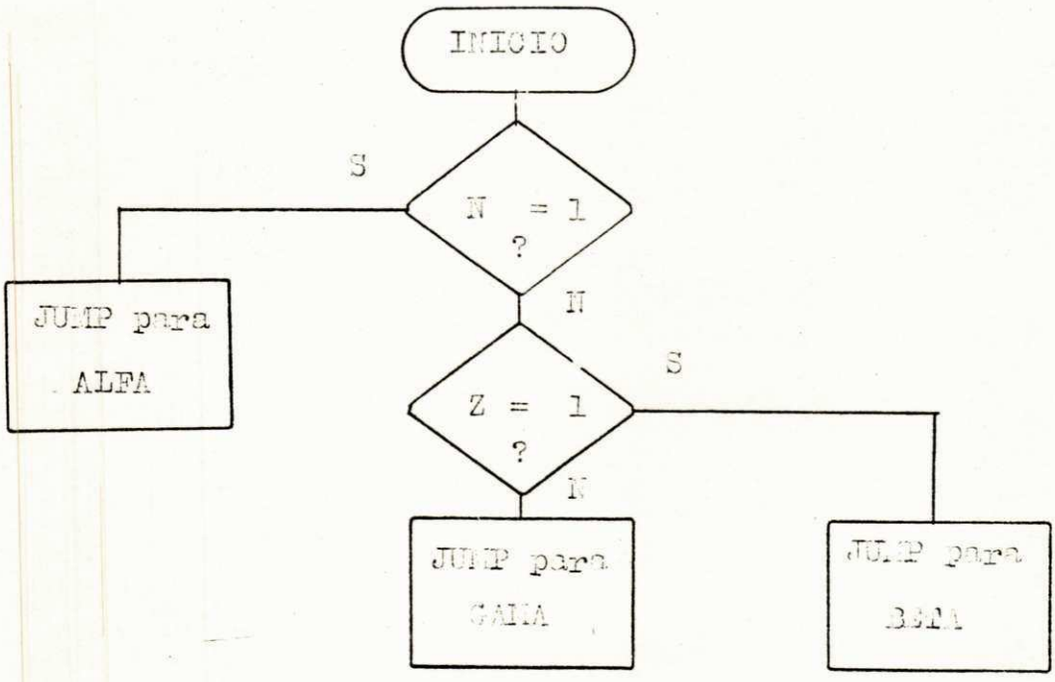
FINAL



PULC



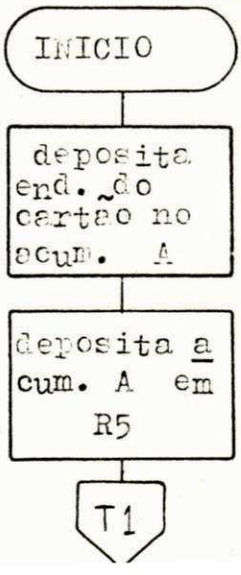
DESVIE

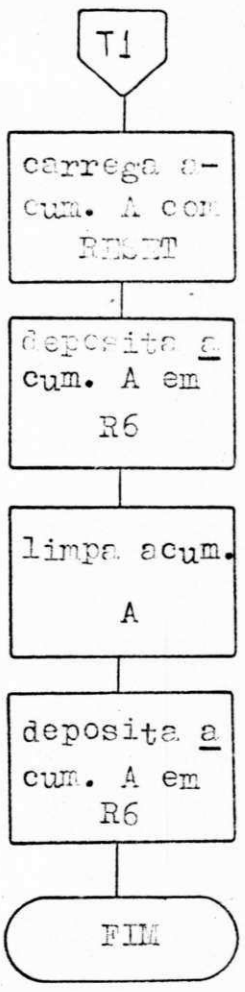


FIM

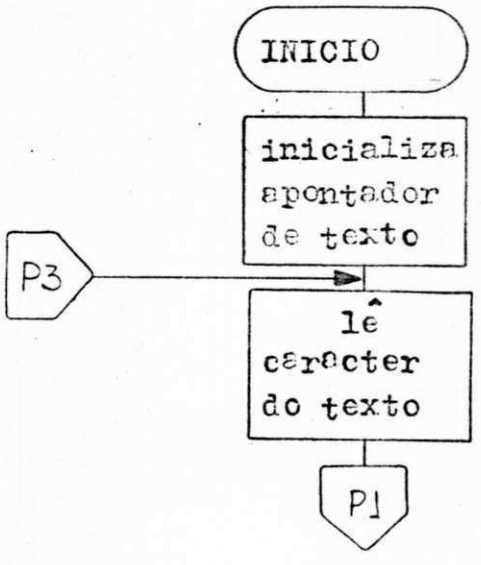
ALFA 1º OPERANDO
BETA 2º OPERANDO
GAMA 3º OPERANDO

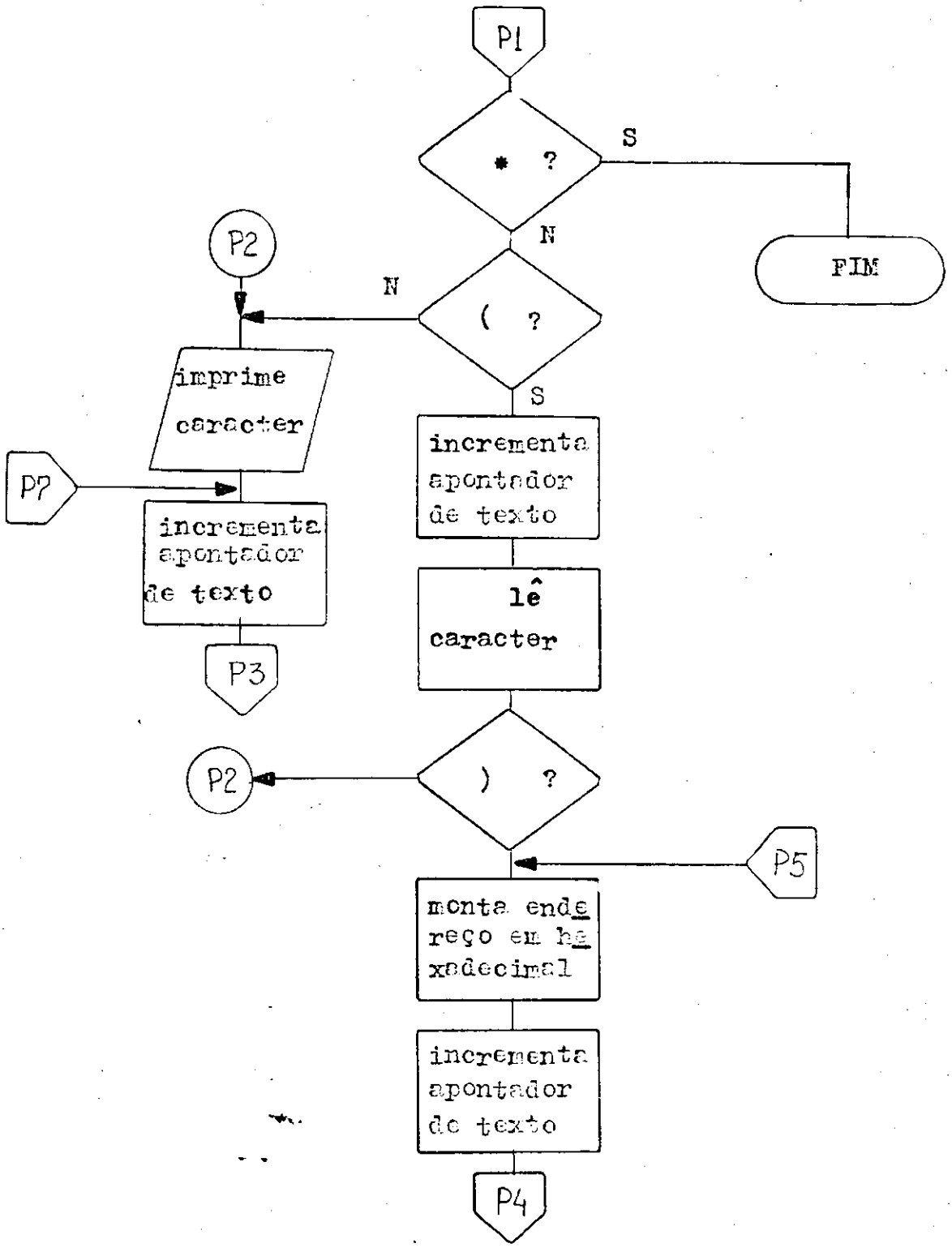
RESET

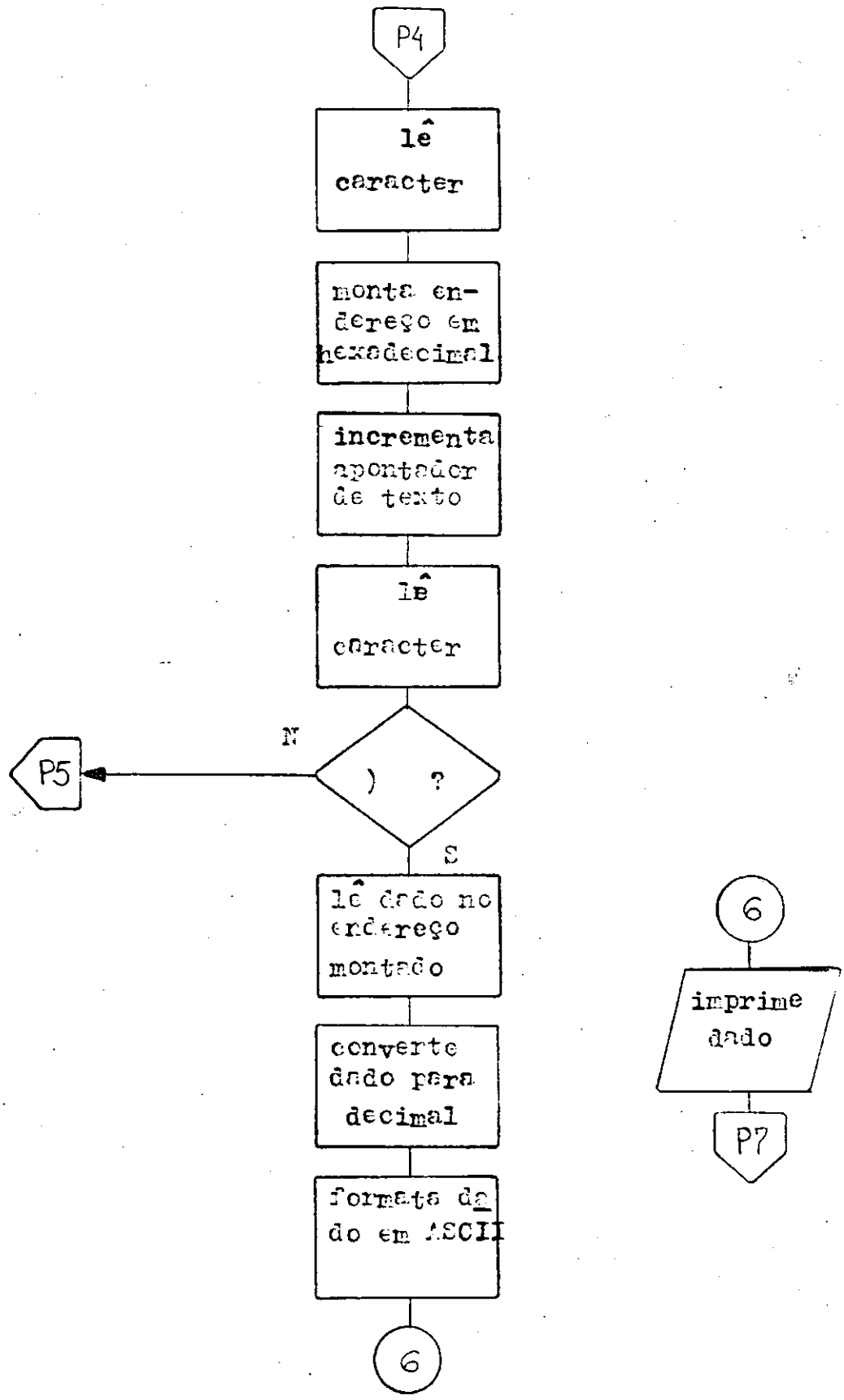



T1
carrega acum. A com RTEMT
deposita acum. A em R6
limpa acum. A
deposita acum. A em R6
FIM

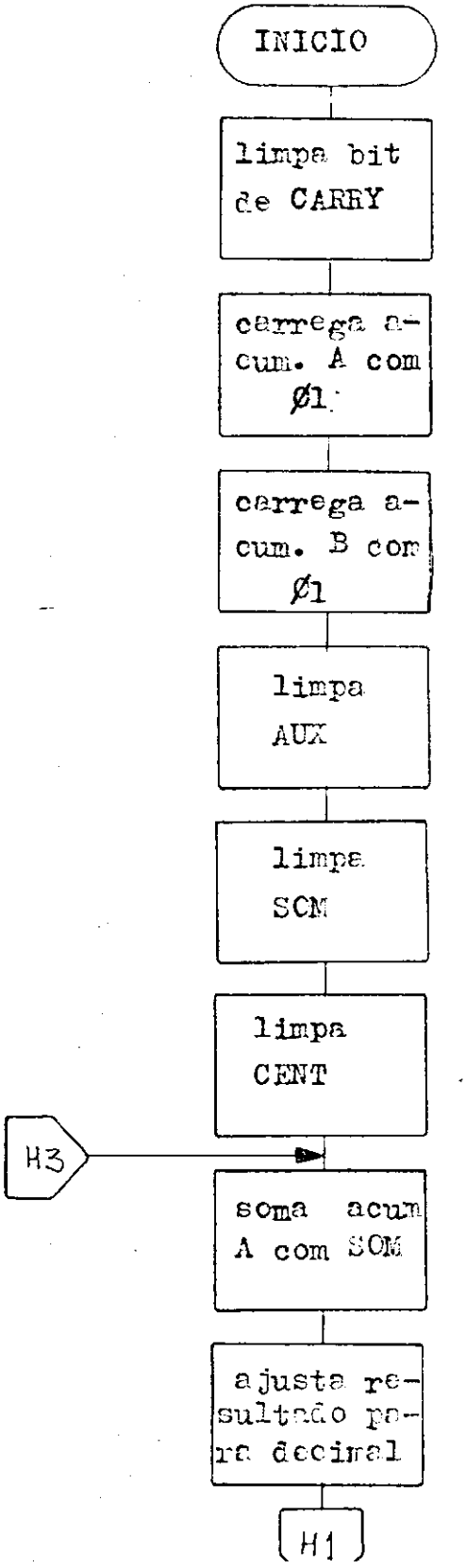
IMPR

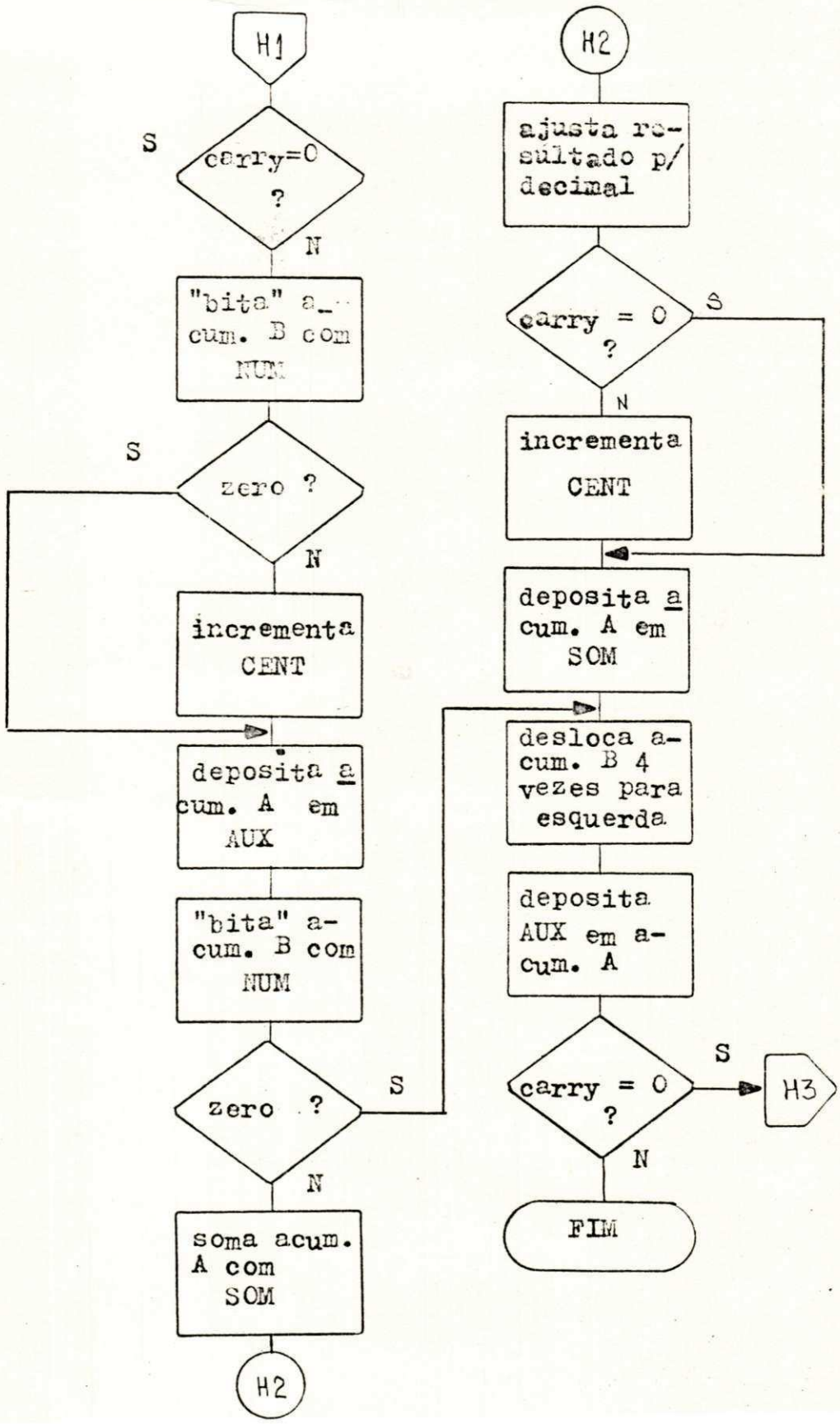

INICIO
inicializa apontador de texto
P3
lê caracter do texto
P1


P1
* ?
S
FIM
N
P2
N
(?
imprime caracter
S
incrementa apontador de texto
P?
incrementa apontador de texto
lê caracter
P3
P2
) ?
P5
monta endereço em hexadecimal
incrementa apontador de texto
P4

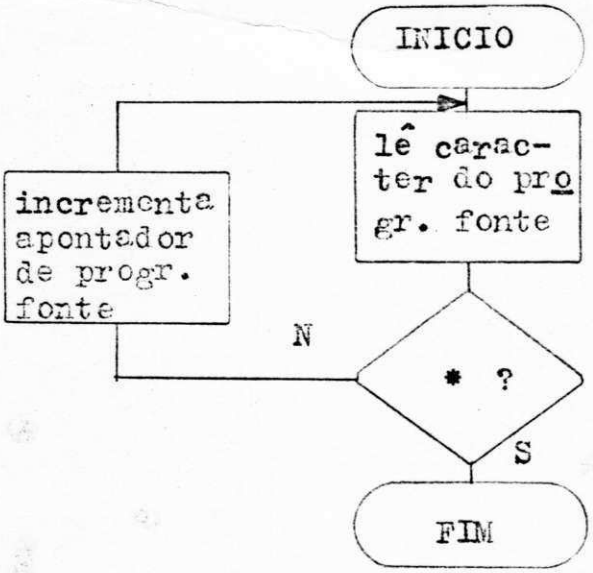

P4
lê caracter
monta endereço em hexadecimal
incrementa apontador de texto
lê caracter
N
P5
) ?
S
lê dado no endereço montado
converte dado para decimal
formata dado em ASCII
6
6
imprime dado
P7

SUBROTINA DE CONVERSÃO DE UM NÚMERO HEXADECIMAL PARA A BASE DECIMAL " HEXAD"

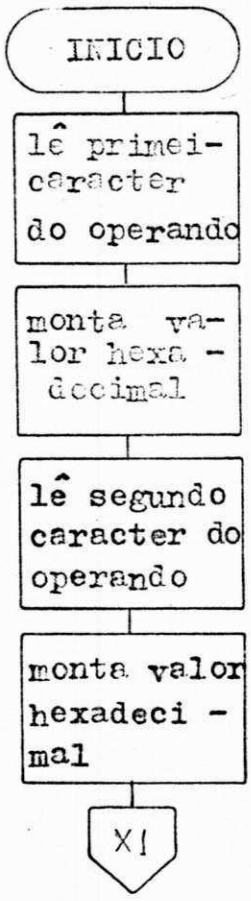


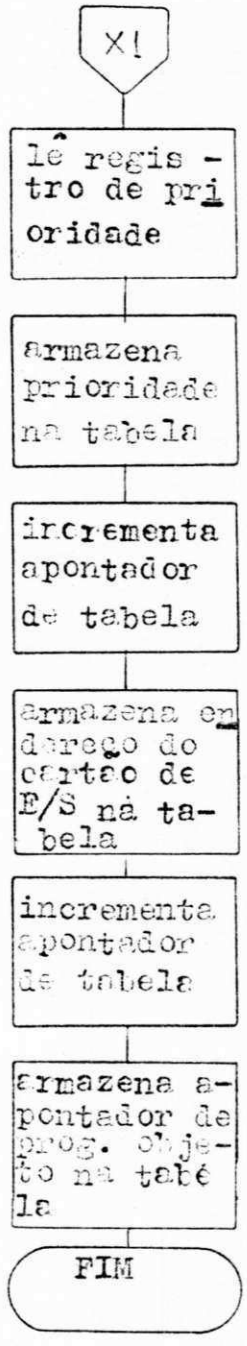

H1
carry=0 ?
S
N
"bita" acum. B com NUM
zero ?
S
N
incrementa CENT
deposita acum. A em AUX
"bita" acum. B com NUM
zero ?
S
N
soma acum. A com SOM
H2
H2
ajusta resultado p/ decimal
carry = 0 ?
S
N
incrementa CENT
deposita acum. A em SOM
desloca acum. B 4 vezes para esquerda
deposita AUX em acum. A
carry = 0 ?
S
N
H3
FIM

TEXTO



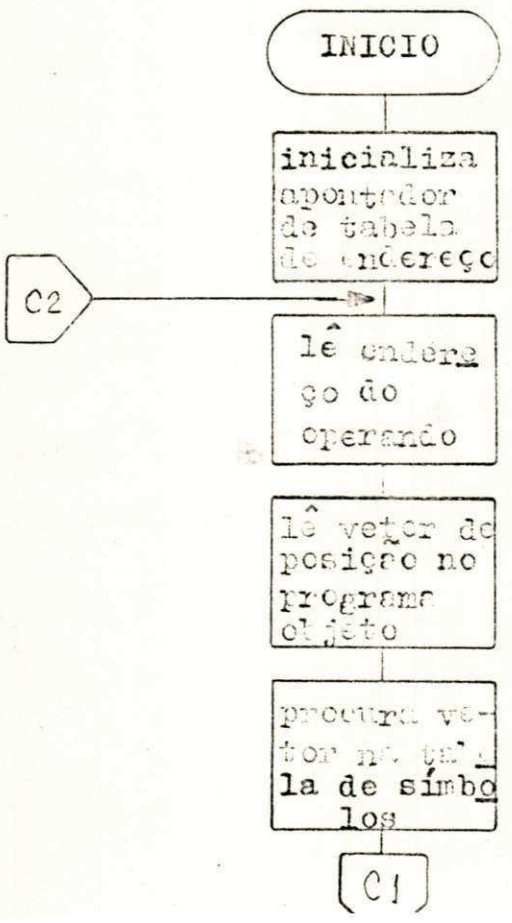
EXECUTE

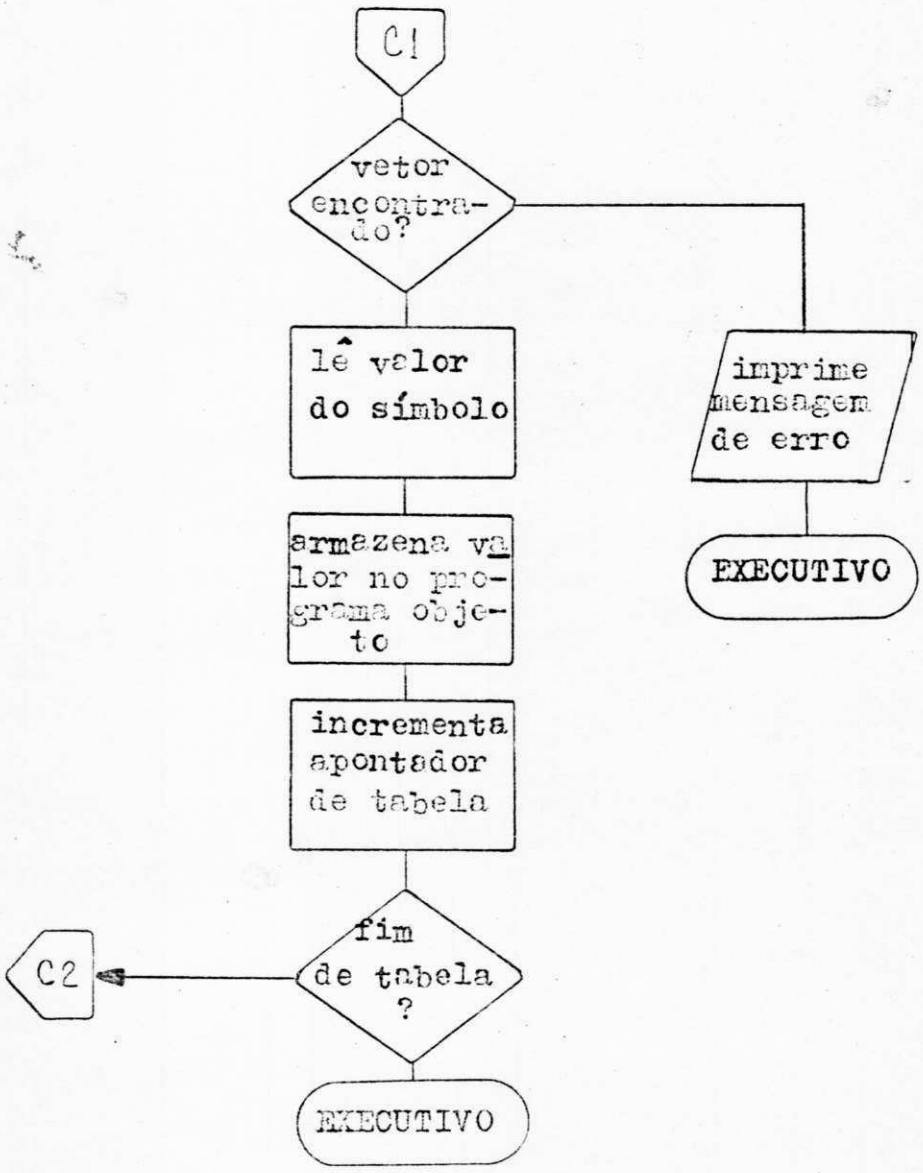



X!
lê regis - tro de pri oridade
armazena prioridade na tabela
incrementa apontador de tabela
armazena en dereço do cartao de E/S na ta- bela
incrementa apontador de tabela
armazena a- pontador de prog. Objeto na tabé la
FIM

3.4.7 – Colocação dos Valores Efetivos dos Símbolos no Programa Objeto

Como foi mostrado na secção 3.4.4.2 deste capítulo, ao ser montada uma instrução ou macroinstrução é colocado no objeto da instrução não o valor efetivo do seu operando mas o seu vetor de posição. Feito isto a posição que este operando ocupa no programa objeto é armazenada na tabela de endereços. Concluída a montagem dos objetos o macro Assembler deve então substituir todos estes vetores pelos valores efetivos dos operandos que devem se encontrar na tabela de símbolos. Esta operação é feita de acordo com o fluxograma que se segue:




C1
vetor encontrado?
lê valor do símbolo
armazena valor no programa objeto
incrementa apontador de tabela
fim de tabela ?
C2
EXECUTIVO
imprime mensagem de erro
EXECUTIVO

CAPÍTULO IV

CONCLUSÃO

Após a leitura deste relatório, pode-se perceber que o trabalho apresentado é abrangente. Poder-se-ia até sugerir que o mesmo fosse não um só, mas vários trabalhos, evidentemente sem querer dizer que o volume do mesmo é demasiado, mas pura e simplesmente porque várias das questões abordadas poderiam ser melhor estudadas em trabalhos distintos. Entende-se que este desmembramento não só pode mas também deve ser feito. A abrangência do trabalho foi, no entanto, um dos principais objetivos pretendidos quando da definição e encaminhamento do mesmo. As razões que motivaram tal pretensão serão apresentadas a seguir.

A importância dos microprocessadores nas mais diversas áreas do controle automático tem crescido consideravelmente. Há pouco tempo atrás ciente desta realidade se resolveu estudar de forma mais sistemática, as questões relativas aos mesmos e suas aplicações neste campo. A melhor forma de se fazer isto

era, entendia-se, se desenvolver um sistema de aquisição de dados e controle de processos que pudesse ser utilizado em várias situações diferentes possuindo um hardware e um software mínimos necessários para tal. Buscava-se, desta forma, não apresentar um protótipo acabado e de alta qualidade, mas antes disso conseguir um certo discernimento com relação às questões envolvidas no desenvolvimento deste hardware e deste software. Desta forma poder-se-ia definir com mais clareza novos trabalhos nesta área.

O hardware do PROSAD é, exceto os cartões de E/S, baseado na família do microprocessador MC 6800. Seu projeto não difere consideravelmente de projetos de microcomputadores baseados neste microprocessador. Sua única característica nova é a interface de vias. Graças a esta interface é possível com um mínimo de software e hardware acrescentar vários tipos de cartões de E/S ao sistema. Em detrimento desta flexibilidade, a velocidade da mesma é baixa. O sistema, na sua configuração atual, não conta com memória auxiliar, e tem a desvantagem de ter como terminal console uma TTY ou TTV que são equipamentos caríssimos, destoando da filosofia básica do sistema que deve ser barato. Propõe-se aqui como trabalhos de expansão deste hardware, a construção de uma interface para gravador de áudio do tipo cassete. Tal dispositivo poderia funcionar como um meio barato de armazenamento auxiliar para o sistema. A TTY ou TTV deverão ser substituídos por uma máquina de escrever elétrica adaptada como terminal de E/S ou opcionalmente poder-se-ia construir um conjunto teclado/display alfanumérico. Com relação aos cartões de E/S,

propõe-se inicialmente o projeto de cartões de entrada analógica de 8 e 12 bits, podendo o nível de complexidade dos mesmos, variar desde os mais simples até os dotados de multiplexadores e com memória RAM própria.

O software do sistema o qual é composto de um gerenciador de tarefas (executivo e scheduler) e de um macro Assembler, mereceu durante o desenvolvimento do trabalho uma atenção maior que o hardware. Isto se deveu ao fato de que não se tinha praticamente nenhum conhecimento nesta área. Ao se iniciar o trabalho não se tinha uma visão clara do que seria o software para um sistema como o PROSAD, ou seja, que tipos de programa deveria ter o mesmo. A esse respeito deseja-se aqui salientar, como impressão pessoal, um dos aspectos mais positivos do trabalho: o relativo discernimento conseguido com relação a este software, principalmente no que diz respeito à gerência de operação do sistema em tempo real. Pode-se também adquirir uma visão mais adequada da importância de um estudo mais sistemático deste software.

Fica aqui a sugestão para que o Grupo de Sistemas Digitais procure criar competência também neste campo, ou de outra forma que se intensifique o trabalho milticiplinar com membros do GSD e do Departamento de Sistemas e Computação.

Dentre os vários algoritmos de escalonadores de tarefas (schedulers) praticáveis em microprocessadores, o proposto por Parrish e Hunaug é sem dúvida a mais simples, estando por isto sujeito a muitas falhas. A ausência de um mecanismo de dis

ciplinamento do tempo de uso do processador por cada tarefa (time slicing) cria a possibilidade de uma tarefa de baixa prioridade não ter acesso ao mesmo, atento do limite de tempo previsto para atendimento à mesma. Este algoritmo não leva em conta também a possibilidade de ocorrência de regiões críticas no programa, constituindo uma outra desvantagem do mesmo. Tais inconvenientes podem ser minimizados mediante uma elaboração cuidadosa das diversas tarefas a serem executadas pelo processador, no entanto, a possibilidade de falha do mesmo aumentará consideravelmente com o aumento do número de tarefas. Com relação às regiões críticas, o programador pode lançar mão da instrução SEI (set interrupt mask) evitando que, uma vez na mesma, uma tarefa possa ser interrompida.

Esta, no entanto, é uma solução precária e deverá ser substituída por mecanismos mais efetivos na garantia da mútua exclusão de tarefas nestes casos. O macro assembler desenvolvido não é completo. Tanto o seu conjunto de instruções quanto os recursos oferecidos pelo mesmo podem e devem ser ampliados. Acredita-se que a despeito das falhas existentes com relação ao mesmo, muitas vantagens podem ser obtidas através do seu uso, podendo-se destacar: o processo de montagem da tabela de tarefas do sistema é transparente ao programador; combinando um branch condicional qualquer com a instrução PULE pode-se obter JUMP's condicionais poupando o programador da tarefa dispendida que é o cálculo dos OFF SET's dos branches de um programa; o programador pode também melhorar sensivelmente a documentação do programa através de comentários; a descrição de uma tarefa para o exe

cutivo é feita de forma simples através da diretiva EXECUTE e da instrução FINAL. Além disso vale-se salientar o pequeno custo deste macro Assembler que ocupa aproximadamente 1.5 k bytes de memória (exceto as tabelas do mesmo).

Todo o software apresentado foi depurado no sistema de desenvolvimento EXORCISER II da motorola (ver Apêndice V). Sugere-se aqui que novos trabalhos visando complementar e melhorar este software sejam desenvolvidos. Por exemplo: um núcleo de gerência mais elaborado contendo um escalonador mais eficiente. Um estudo teórico deveria definir o melhor algoritmo para o mesmo. No que se refere aos programas de apoio pode-se desenvolver um editor de textos para o macro assembler e programas de "self test" do sistema. Ainda com relação ao sistema operacional pode-se elaborar rotinas de atendimento ao console do operador visando dotar o sistema de características interativas permitindo-se desta forma uma conversação do mesmo com o operador. Finalizando, sugere-se que baseando-se neste trabalho é incorporando-se as sugestões aqui apresentadas se monte um protótipo capaz de atender às exigências do mercado nacional.

BIBLIOGRAFIA

1) Parrish Jr. Edward A. e Huang Victor K. L. - A scheduler for real time task control in microcomputers. IEEE Transaction on IECI, Vol. IECI-25, Nº 1.

2) Hewlett Pachard Journal - An inteligent peripheral for measurement and control. Julho, 1978.

3) Kinetic Systems, CAMAC Catalog. 1978-1979.

4) Motorola, MC 6800 Microprocessor Application Manual.

5) CAVALCANTI, José H. F. e DEEP, Gurdip Singh - Um Kernel para controle de processos industriais em tempo real usando microcomputadores. 1º SEMISH, 1980.

6) SERLIM, Omvi - Scheduling of time critical processes. Spring Joint computer Conference, 1972.

7) LACERDA, J. R. Costa - As remotas inteligentes e o processo industrial. 1º Simpósio de automação de processos industriais por computador, 1979.

8) Texas Instruments - The TTL Data Book for design engineers, 1973.

9) GUIMARÃES, Célio C. - Princípios de Sistemas Operacionais. Escola de Computação. Instituto de Matemática e Estatística da USP, 1979.

10) Philips/Signetics. Signetics integrated circuits, 1978.

11) Stuckenberg, Hans J. CAMAC for Newcomers, 1975.

12) Kendler M. B. e Lutte N. P. Microprocessors applied to Industrial control systems. Electronic Engineering, 1979.

13) MARAZAS, Gerald - Microprocessors heed the call to supervise I/O. Electronics. 2 de Fevereiro de 1978.

14) Euwe, W. J. G. - System Software. Philips International Institute, 1978.

15) Motorola. MC 6800 - Microprocessor Programming Manual, 1975.

16) Motorola. MC 6800 - Systems reference and data sheets, 1975.

17) GRABBE, Ramo e WOOLDRIDGE - Handbook of Automatic Computation and Control. John Wiley e Sons, Inc., 1961.

18) COURY, Fred F. - A practical guide to minicomputer applications. IEEE PRESS, 1972.

19) Campbell-Kelly, M. - An Introduction to Macros. Mac Donald e Co. Ltd., 1973.

APÊNDICE I

PROGRAMA EM ASSEMBLY DO MACRO ASSEMBLER, EXECUTIVO E ESCALONADOR DE TAREFAS

APÊNDICE II

CONJUNTO DE INSTRUÇÕES
DO MICROPROCESSADOR MC 6800

TABLE 3 – ACCUMULATOR AND MEMORY INSTRUCTIONS

OPERATIONS	MNEMONIC	IMMED OP	IMMED ~	IMMED =	DIRECT OP	DIRECT ~	DIRECT =	INDEX OP	INDEX ~	INDEX =	EXTND OP	EXTND ~	EXTND =	IMPLIED OP	IMPLIED ~	IMPLIED =	BOOLEAN/ARITHMETIC OPERATION (All register labels refer to contents)	5 H	4 I	3 N	2 Z	1 V	0 C
Add	ADDA	8B	2	2	9B	3	2	AB	5	2	BB	4	3				A + M → A	↕	•	↕	↕	↕	↕
	ADDB	CB	2	2	DB	3	2	EB	5	2	FB	4	3				B + M → B	↕	•	↕	↕	↕	↕
Add Acmltrs	ABA													1B	2	1	A + B → A	↕	•	↕	↕	↕	↕
Add with Carry	ADCA	89	2	2	99	3	2	A9	5	2	B9	4	3				A + M + C → A	↕	•	↕	↕	↕	↕
	ADCB	C9	2	2	D9	3	2	E9	5	2	F9	4	3				B + M + C → B	↕	•	↕	↕	↕	↕
And	ANDA	84	2	2	94	3	2	A4	5	2	B4	4	3				A · M → A	•	•	↕	↕	R	•
	ANDB	C4	2	2	D4	3	2	E4	5	2	F4	4	3				B · M → B	•	•	↕	↕	R	•
Bit Test	BITA	85	2	2	95	3	2	A5	5	2	B5	4	3				A · M	•	•	↕	↕	R	•
	BITB	C5	2	2	D5	3	2	E5	5	2	F5	4	3				B · M	•	•	↕	↕	R	•
Clear	CLR							6F	7	2	7F	6	3				00 → M	•	•	R	S	R	R
	CLRA													4F	2	1	00 → A	•	•	R	S	R	R
	CLRB													5F	2	1	00 → B	•	•	R	S	R	R
Compare	CMPA	81	2	2	91	3	2	A1	5	2	B1	4	3				A − M	•	•	↕	↕	↕	↕
	CMPB	C1	2	2	D1	3	2	E1	5	2	F1	4	3				B − M						
Compare Acmltrs	CBA													11	2	1	A − B	•	•	↕	↕	↕	↕
Complement, 1's	COM							63	7	2	73	6	3				M → M	•	•	↕	↕	R	S
	COMA													43	2	1	A → A	•	•	↕	↕	R	S
	COMB													53	2	1	B → B	•	•	↕	↕	R	S
Complement, 2's (Negate)	NEG							60	7	2	70	6	3				00 − M → M	•	•	↕	↕	①	②
	NEGA													40	2	1	00 − A → A	•	•	↕	↕	①	②
	NEGB													50	2	1	00 − B → B	•	•	↕	↕	①	②
Decimal Adjust, A	DAA													19	2	1	Converts Binary Add. of BCD Characters into BCD Format	•	•	↕	↕	↕	③
Decrement	DEC							6A	7	2	7A	6	3				M − 1 → M	•	•	↕	↕	4	•
	DECA													4A	2	1	A − 1 → A	•	•	↕	↕	4	•
	DECB													5A	2	1	B − 1 → B	•	•	↕	↕	4	•
Exclusive OR	EORA	88	2	2	98	3	2	A8	5	2	B8	4	3				A⊕M → A	•	•	↕	↕	R	•
	EORB	C8	2	2	D8	3	2	E8	5	2	F8	4	3				B⊕M → B	•	•	↕	↕	R	•
Increment	INC							6C	7	2	7C	6	3				M + 1 → M	•	•	↕	↕	⑤	•
	INCA													4C	2	1	A + 1 → A	•	•	↕	↕	⑤	•
	INCB													5C	2	1	B + 1 → B	•	•	↕	↕	⑤	•
Load Acmltr	LDAA	86	2	2	96	3	2	A6	5	2	B6	4	3				M → A	•	•	↕	↕	R	•
	LDAB	C6	2	2	D6	3	2	E6	5	2	F6	4	3				M → B	•	•	↕	↕	R	•
Or, Inclusive	ORAA	8A	2	2	9A	3	2	AA	5	2	BA	4	3				A + M → A	•	•	↕	↕	R	•
	ORAB	CA	2	2	DA	3	2	EA	5	2	FA	4	3				B + M → B	•	•	↕	↕	R	•
Push Data	PSHA													36	4	1	A → M_{SP}, SP − 1 → SP	•	•	•	•	•	•
	PSHB													37	4	1	B → M_{SP}, SP − 1 → SP	•	•	•	•	•	•
Pull Data	PULA													32	4	1	SP + 1 → SP, M_{SP} → A	•	•	•	•	•	•
	PULB													33	4	1	SP + 1 → SP, M_{SP} → B	•	•	•	•	•	•
Rotate Left	ROL							69	7	2	79	6	3				M	•	•	↕	↕	⑥	↕
	ROLA													49	2	1	A	•	•	↕	↕	⑥	↕
	ROLB													59	2	1	B	•	•	↕	↕	⑥	↕
Rotate Right	ROR							66	7	2	76	6	3				M	•	•	↕	↕	⑥	↕
	RORA													46	2	1	A	•	•	↕	↕	⑥	↕
	RORB													56	2	1	B	•	•	↕	↕	⑥	↕
Shift Left, Arithmetic	ASL							68	7	2	78	6	3				M	•	•	↕	↕	⑥	↕
	ASLA													48	2	1	A	•	•	↕	↕	⑥	↕
	ASLB													58	2	1	B	•	•	↕	↕	⑥	↕
Shift Right, Arithmetic	ASR							67	7	2	77	6	3				M	•	•	↕	↕	⑥	↕
	ASRA													47	2	1	A	•	•	↕	↕	⑥	↕
	ASRB													57	2	1	B	•	•	↕	↕	⑥	↕
Shift Right, Logic	LSR							64	7	2	74	6	3				M	•	•	R	↕	⑥	↕
	LSRA													44	2	1	A	•	•	R	↕	⑥	↕
	LSRB													54	2	1	B	•	•	R	↕	⑥	↕
Store Acmltr.	STAA				97	4	2	A7	6	2	B7	5	3				A → M	•	•	↕	↕	R	•
	STAB				D7	4	2	E7	6	2	F7	5	3				B → M	•	•	↕	↕	R	•
Subtract	SUBA	80	2	2	90	3	2	A0	5	2	B0	4	3				A − M → A	•	•	↕	↕	↕	↕
	SUBB	C0	2	2	D0	3	2	E0	5	2	F0	4	3				B − M → B	•	•	↕	↕	↕	↕
Subtract Acmltrs.	SBA													10	2	1	A − B → A	•	•	↕	↕	↕	↕
Subtr. with Carry	SBCA	82	2	2	92	3	2	A2	5	2	B2	4	3				A − M − C → A	•	•	↕	↕	↕	↕
	SBCB	C2	2	2	D2	3	2	E2	5	2	F2	4	3				B − M − C → B	•	•	↕	↕	↕	↕
Transfer Acmltrs	TAB													16	2	1	A → B	•	•	↕	↕	R	•
	TBA													17	2	1	B → A	•	•	↕	↕	R	•
Test, Zero or Minus	TST							6D	7	2	7D	6	3				M − 00	•	•	↕	↕	R	R
	TSTA													4D	2	1	A − 00	•	•	↕	↕	R	R
	TSTB													5D	2	1	B − 00	•	•	↕	↕	R	R
																		H	I	N	Z	V	C

LEGEND:

- OP — Operation Code (Hexadecimal).
- ~ — Number of MPU Cycles.
- = — Number of Program Bytes.
- + — Arithmetic Plus.
- − — Arithmetic Minus.
- · — Boolean AND.
- M_{SP} — Contents of memory location pointed to be Stack Pointer.
- + — Boolean Inclusive OR.
- ⊕ — Boolean Exclusive OR.
- M — Complement of M.
- → — Transfer Into.
- 0 — Bit = Zero.
- 00 — Byte = Zero.

Note – Accumulator addressing mode instructions are included in the column for IMPLIED addressing.

CONDITION CODE SYMBOLS

- H — Half-carry from bit 3.
- I — Interrupt mask.
- N — Negative (sign bit).
- Z — Zero (byte).
- V — Overflow, 2's complement.
- C — Carry from bit 7.
- R — Reset Always.
- S — Set Always.
- ↕ — Test and set if true, cleared otherwise.
- • — Not Affected.

TABLE 4 – INDEX REGISTER AND STACK MANIPULATION INSTRUCTIONS

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

TABLE 5 – JUMP AND BRANCH INSTRUCTIONS

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

TABLE 6 – CONDITION CODE REGISTER MANIPULATION INSTRUCTIONS

		IMPLIED				COND. CODE REG.					
						5	4	3	2	1	0
OPERATIONS	MNEMONIC	OP	~	=	BOOLEAN OPERATION	H	I	N	Z	V	C
Clear Carry	CLC	0C	2	1	0 → C	•	•	•	•	•	R
Clear Interrupt Mask	CLI	0E	2	1	0 → I	•	R	•	•	•	•
Clear Overflow	CLV	0A	2	1	0 → V	•	•	•	•	R	•
Set Carry	SEC	0D	2	1	1 → C	•	•	•	•	•	S
Set Interrupt Mask	SEI	0F	2	1	1 → I	•	S	•	•	•	•
Set Overflow	SEV	0B	2	1	1 → V	•	•	•	•	S	•
Acmltr A → CCR	TAP	06	2	1	A → CCR	———		(12)			———
CCR → Acmltr A	TPA	07	2	1	CCR → A	•	•	•	•	•	•

CONDITION CODE REGISTER NOTES:

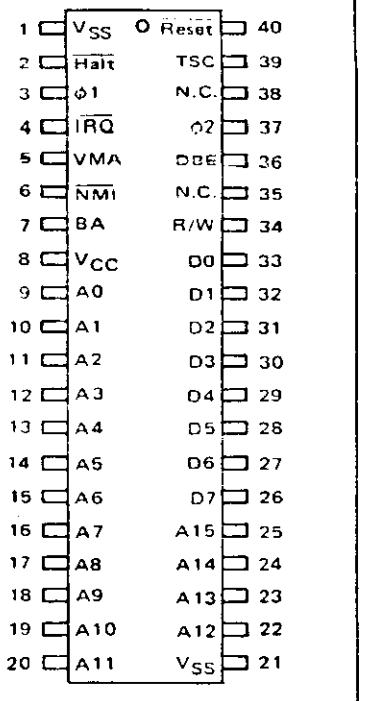
(Bit set if test is true and cleared otherwise)

1 (Bit V) Test: Result = 10000000?
2 (Bit C) Test: Result = 00000000?
3 (Bit C) Test: Decimal value of most significant BCD Character greater than nine? (Not cleared if previously set.)
4 (Bit V) Test: Operand = 10000000 prior to execution?
5 (Bit V) Test: Operand = 01111111 prior to execution?
6 (Bit V) Test: Set equal to result of N⊕C after shift has occurred.
7 (Bit N) Test: Sign bit of most significant (MS) byte = 1?
8 (Bit V) Test: 2's complement overflow from subtraction of MS bytes?
9 (Bit N) Test: Result less than zero? (Bit 15 = 1)
10 (All) Load Condition Code Register from Stack. (See Special Operations)
11 (Bit I) Set when interrupt occurs. If previously set, a Non-Maskable Interrupt is required to exit the wait state.
12 (All) Set according to the contents of Accumulator A.

APÊNDICE III

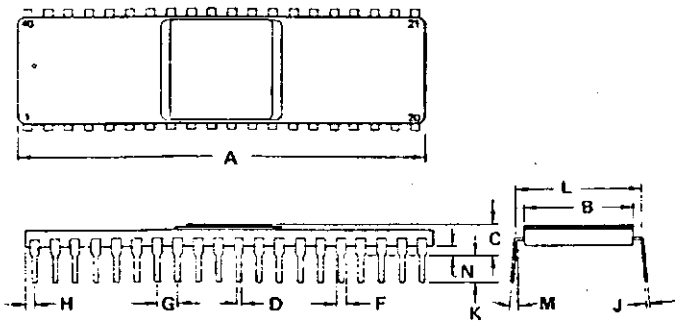
CARACTERÍSTICAS ELÉTRICAS DO MICROPROCESSADOR MC 6800

PIN ASSIGNMENT



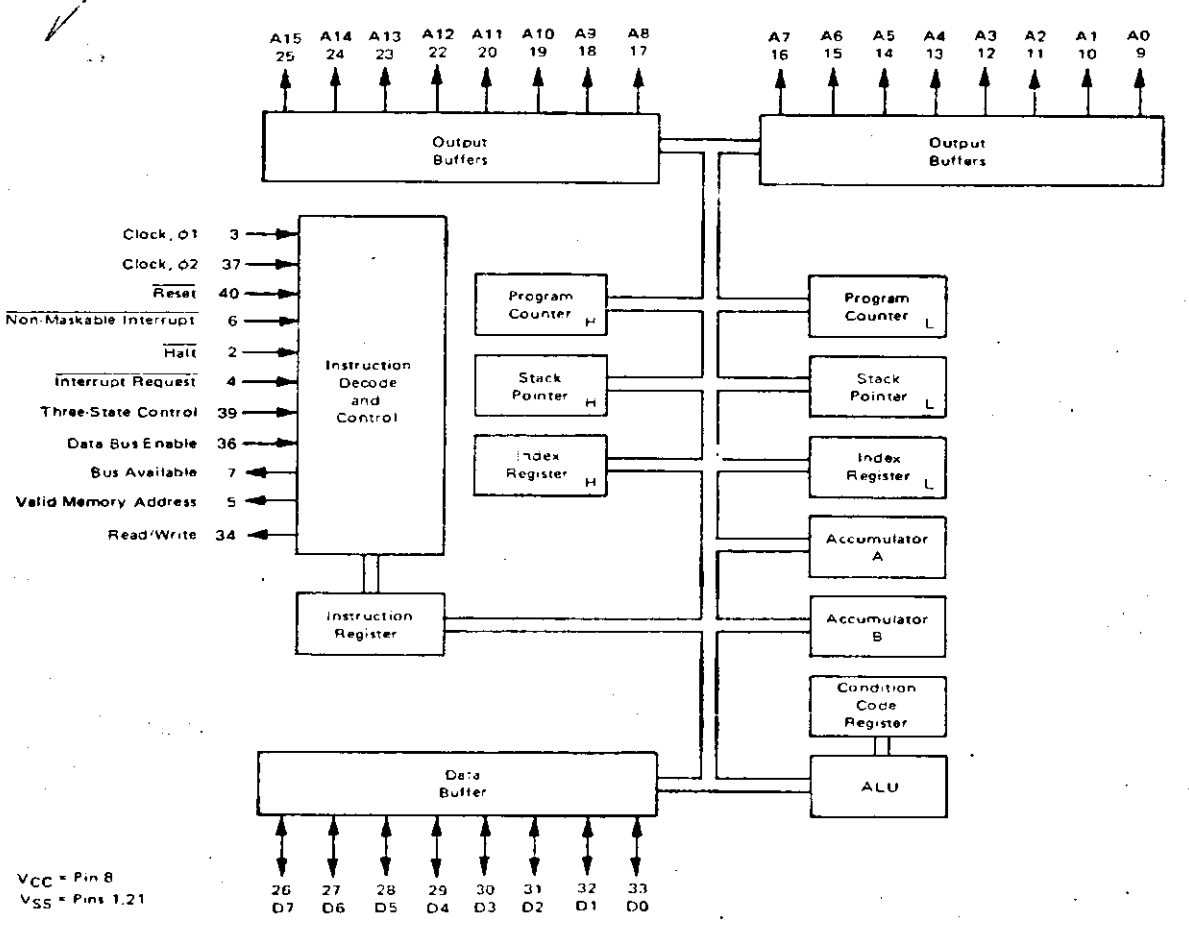
PACKAGE DIMENSIONS

CASE 699-04

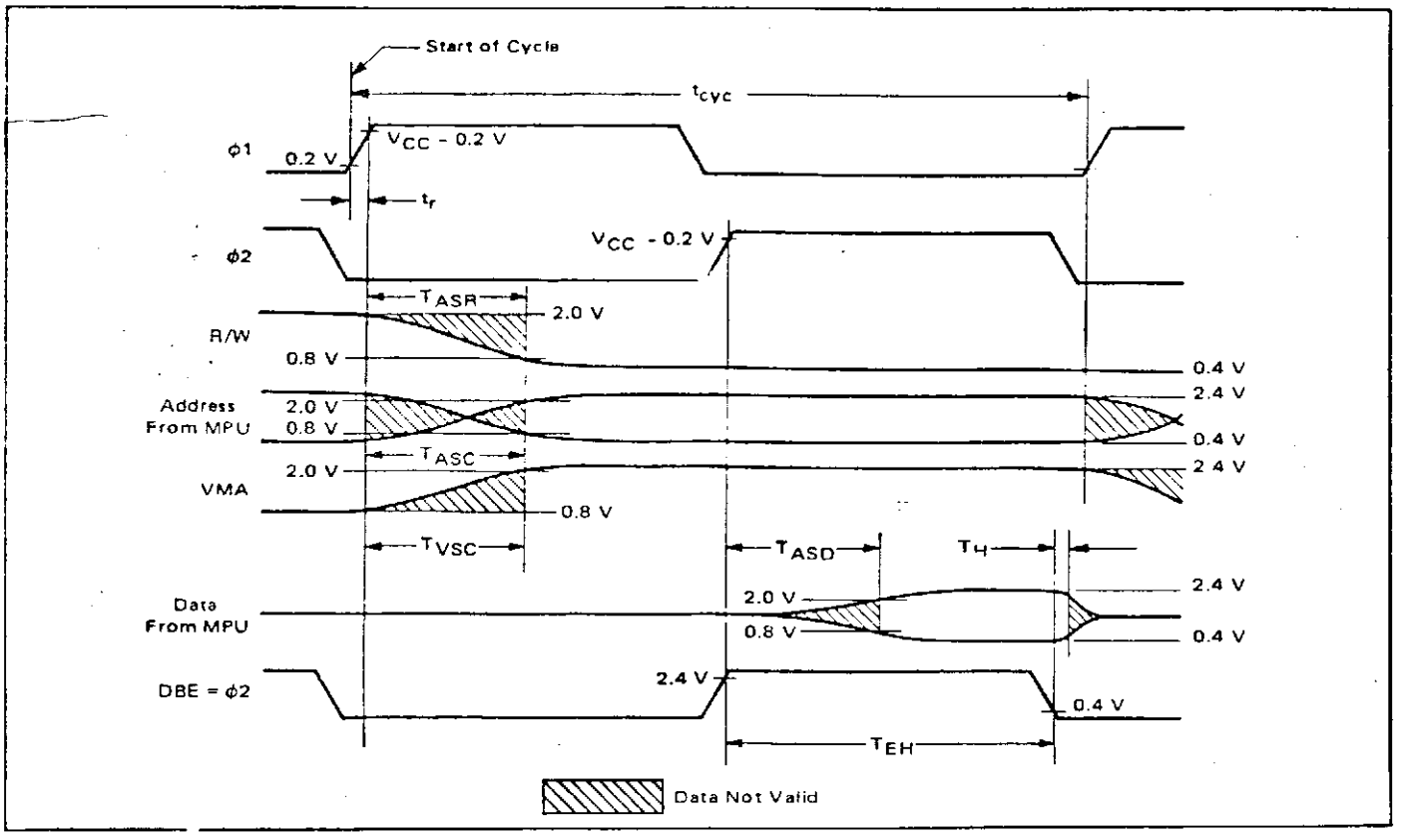


DIM	MILLIMETERS MIN	MILLIMETERS MAX	INCHES MIN	INCHES MAX
A	50.04	51.05	1.970	2.010
B	13.46	14.22	0.530	0.560
C	3.05	3.94	0.120	0.155
D	0.38	0.51	0.015	0.020
F	0.99	1.40	0.035	0.055
G	2.54 BSC		0.100 BSC	
H	0.89	1.40	0.035	0.055
J	0.20	0.28	0.008	0.011
K	3.05	3.68	0.120	0.145
L	14.86	15.87	0.585	0.625
M	–	15°	–	15°
N	0.51	1.14	0.020	0.045

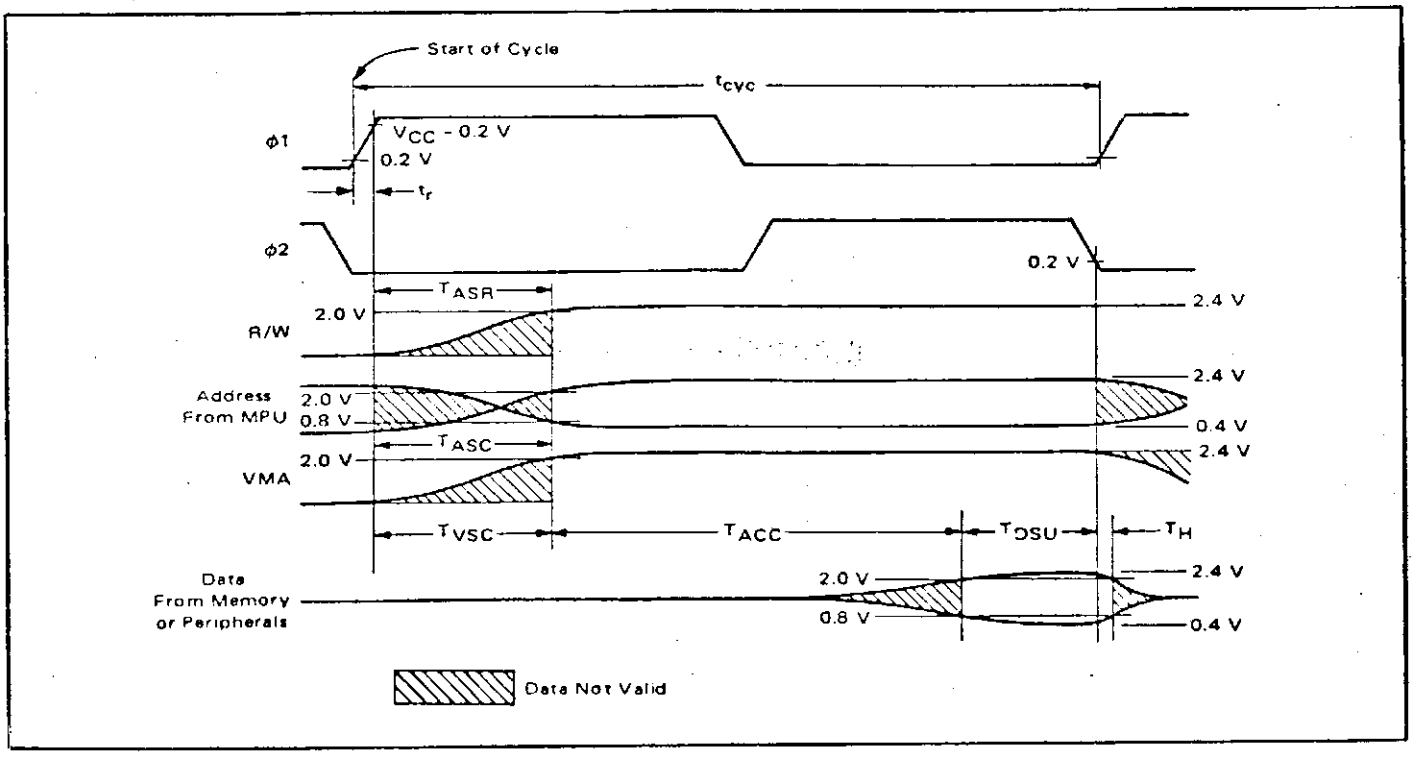
NOTES:
1. LEADS WITHIN 0.13 mm (0.005) RADIUS OF TRUE POSITION AT SEATING PLANE AT MAXIMUM MATERIAL CONDITION.
2. DIMENSION "L" TO INSIDE OF LEADS (MEASURED 0.51 mm (0.020) BELOW PACKAGE BASE)



– WRITE DATA IN MEMORY OR PERIPHERALS



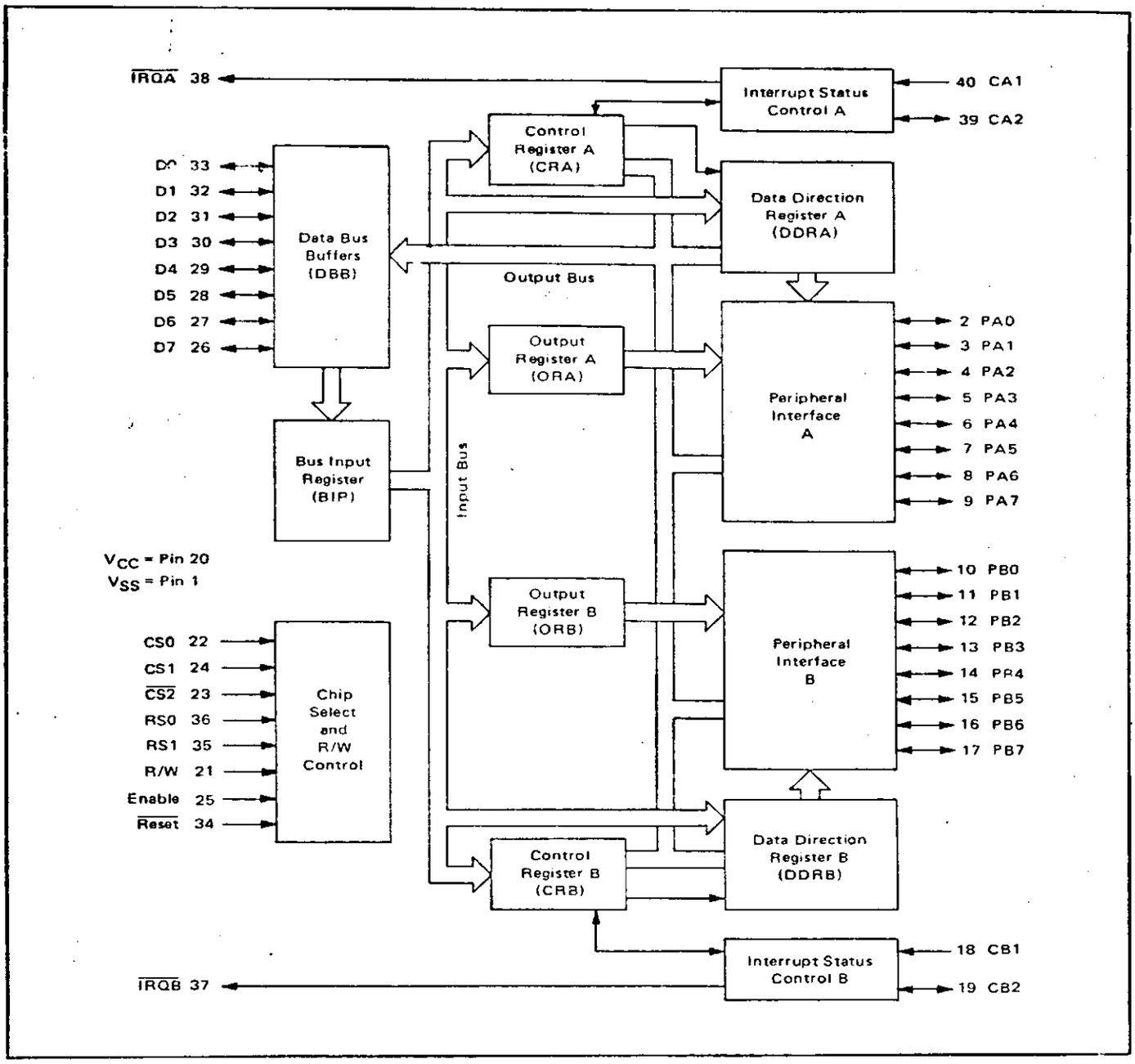
– READ DATA FROM MEMORY OR PERIPHERALS

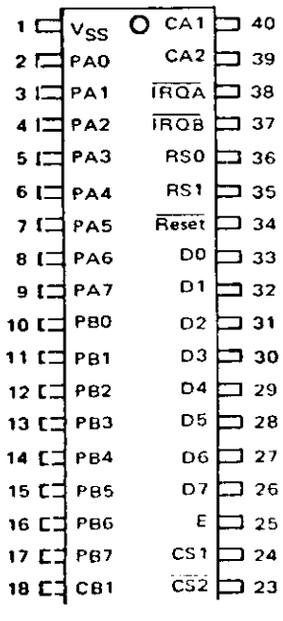


APÊNDICE IV

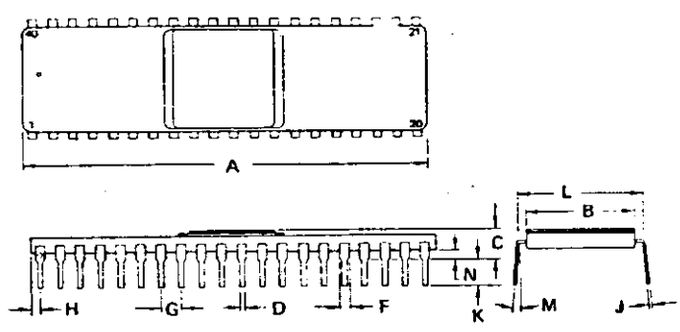
CIRCUITOS INTEGRADOS UTILIZADOS

EXPANDED BLOCK DIAGRAM





CASE 699-04



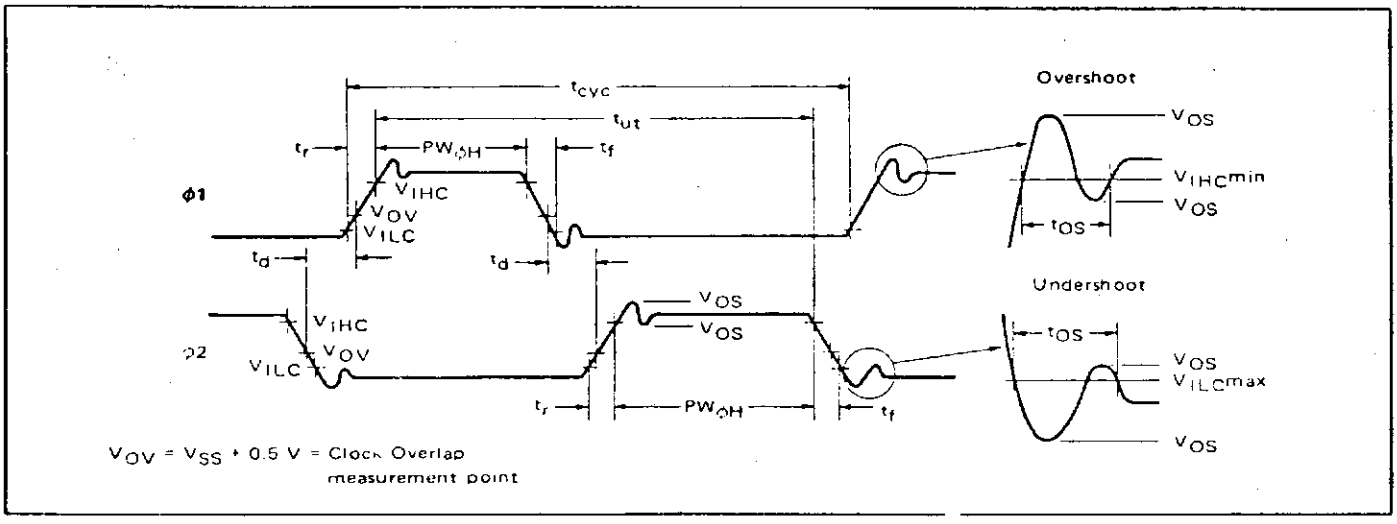
DIM	MILLIMETERS MIN	MILLIMETERS MAX	INCHES MIN	INCHES MAX
A	50.04	51.05	1.970	2.010
B	13.46	14.22	0.530	0.560
C	3.05	3.94	0.120	0.155
D	0.38	0.51	0.015	0.020
F	0.89	1.40	0.035	0.055
G	2.54 BSC		0.100 BSC	
H	0.89	1.40	0.035	0.055

NOTES:
1. LEADS WITHIN 0.13 mm (0.005) RADIUS OF TRUE POSITION AT SEATING PLANE, AT MAXIMUM MATERIAL CONDITION.
2. DIMENSION "L" TO INSIDE OF LEADS (MEASURED 0.51 mm

READ/WRITE TIMING Figures 2 and 3, f = 1.0 MHz, Loading = 130 pF and one TTL Load except VMA and BA Loading = 30 pF and one TTL Load.

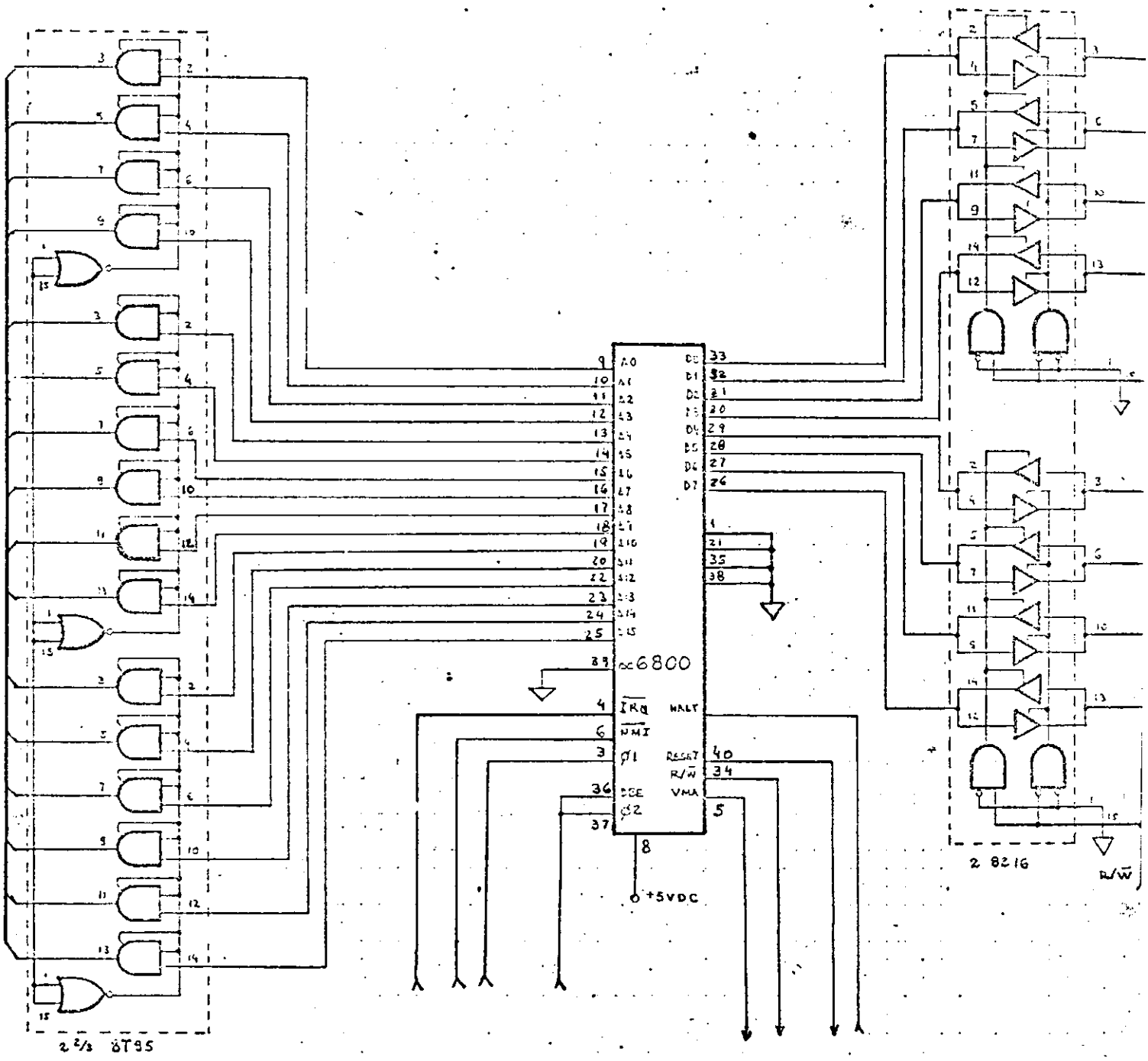
Characteristic	Symbol	Min	Typ	Max	Unit
Read/Write Setup Time from MPU	T_{ASR}	–	100	300	ns
Address Setup Time from MPU	T_{ASC}	–	200	300	ns
Memory Read Access Time $t_{cyc} - (T_{ASC} + T_{DSU} + t_r)$	T_{ACC}	–	–	575	ns
Data Setup Time	T_{DSU}	100	–	–	ns
Address Setup Time from MPU for VMA	T_{VSC}	–	150	300	ns
Data Hold Time	T_H	10	30	–	ns
Enable High Time for DBE Input	T_{EH}	470	–	–	ns
Data Setup Time from MPU	T_{ASD}	–	150	200	ns

FIGURE 1 – CLOCK TIMING WAVEFORM

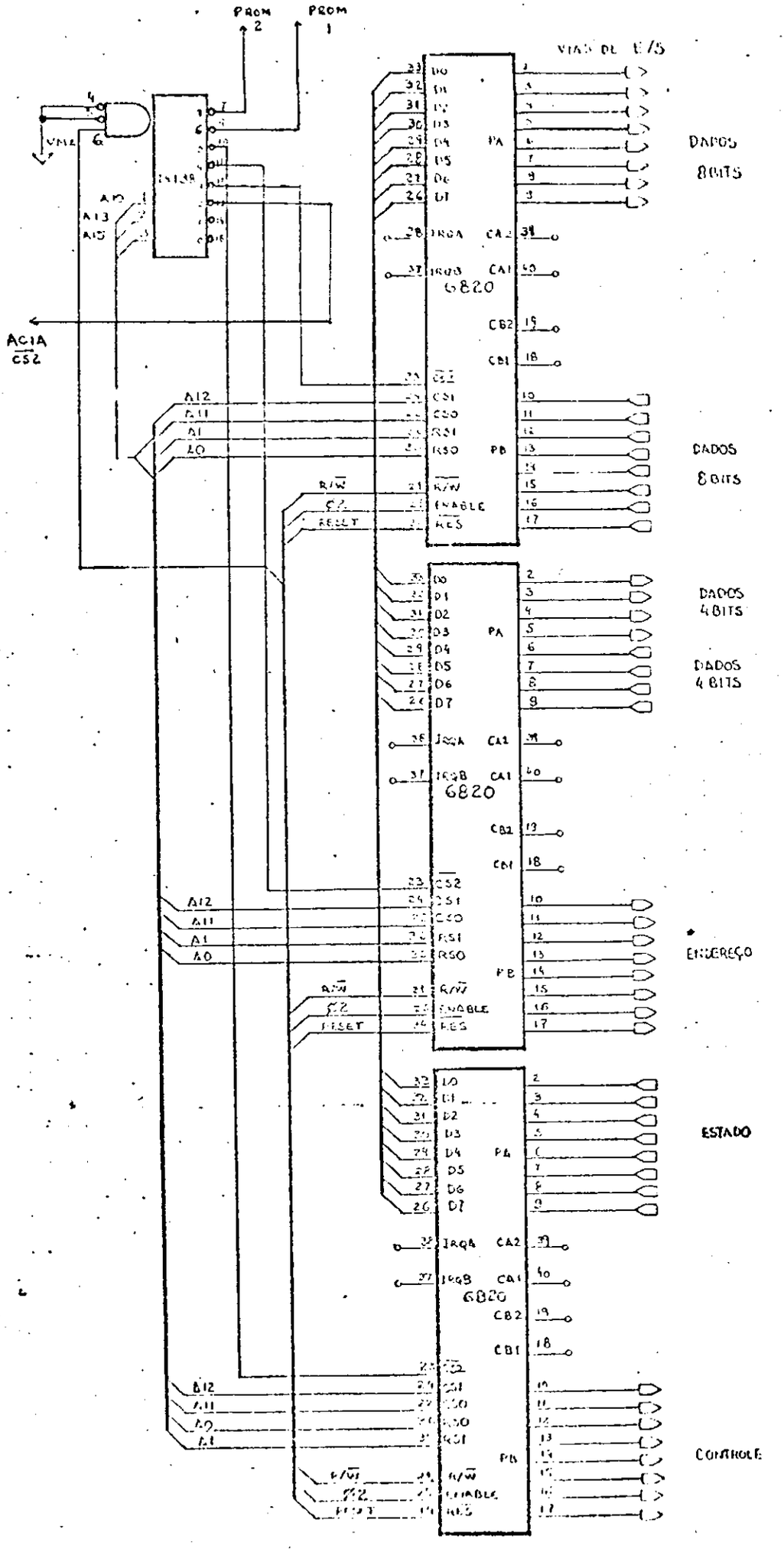


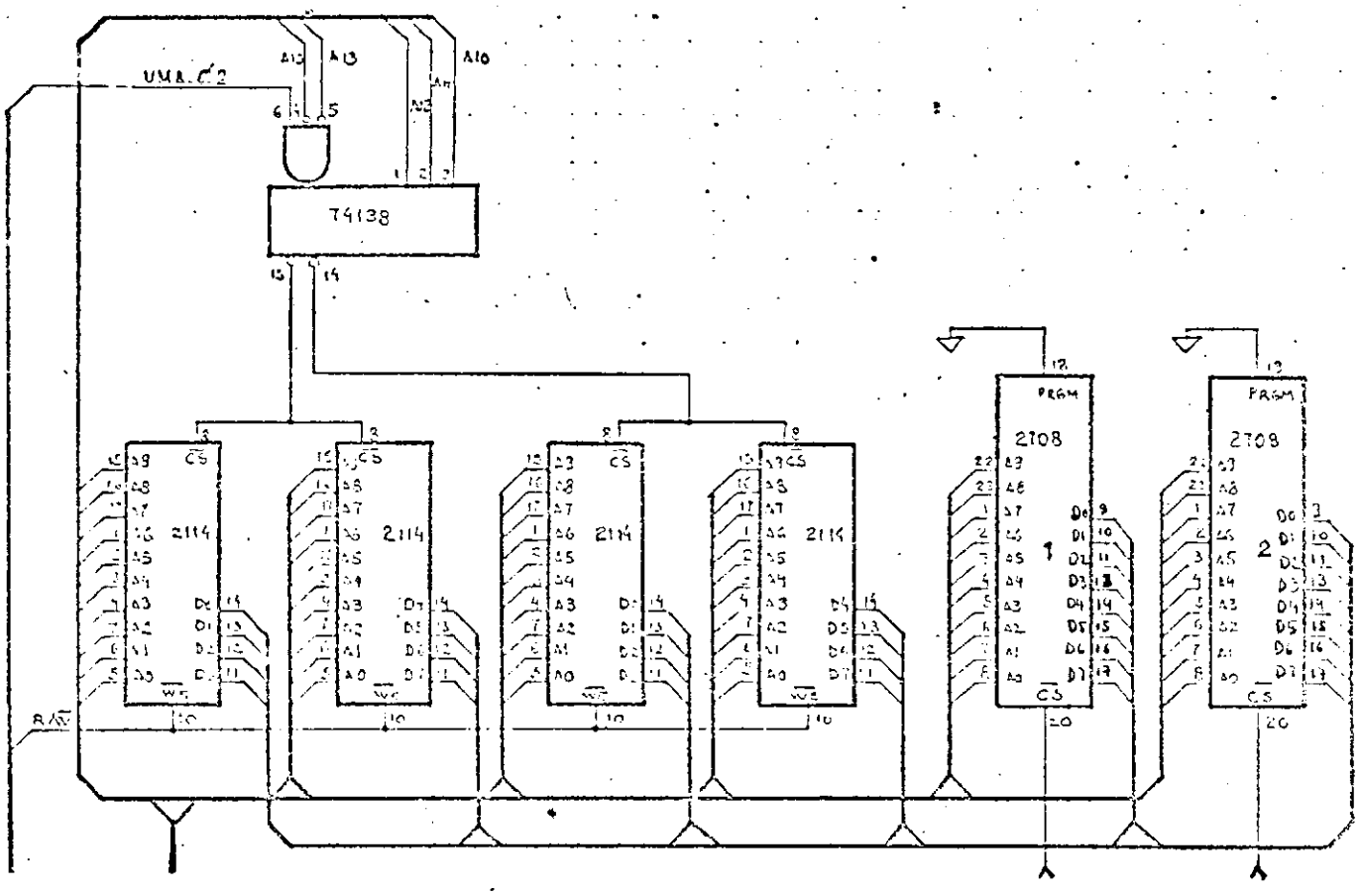
APÊNDICE VI

HARDWARE DO SISTEMA

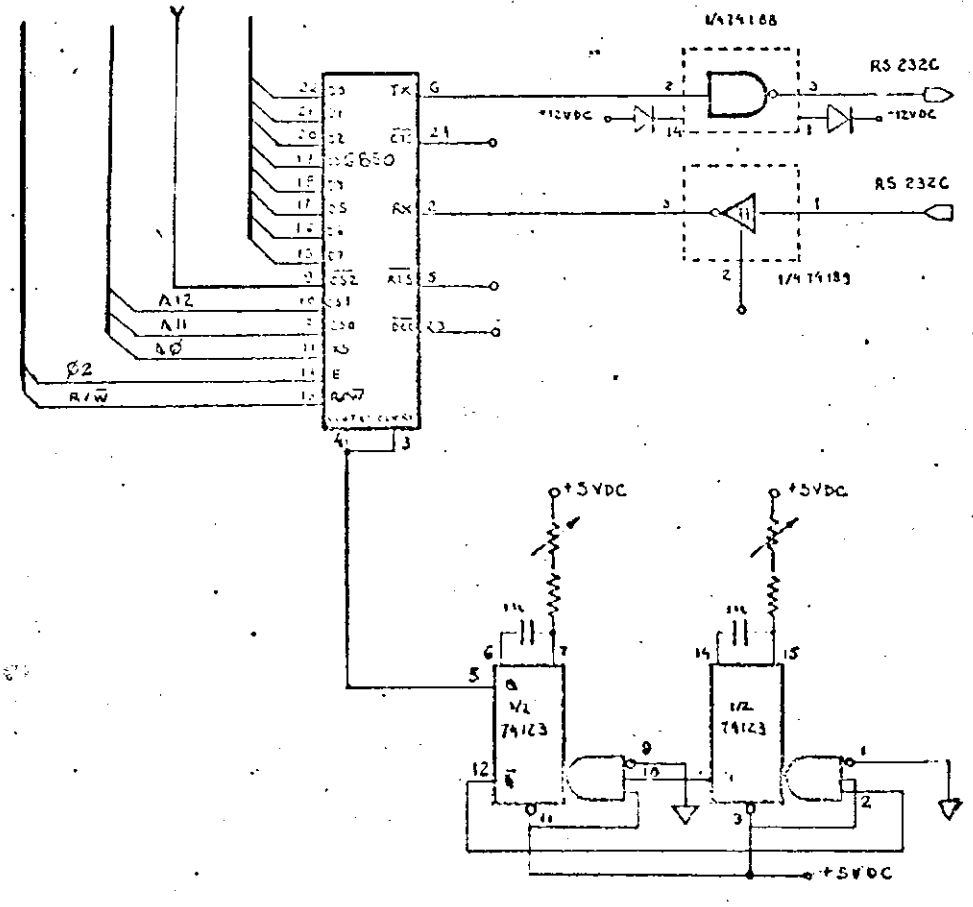


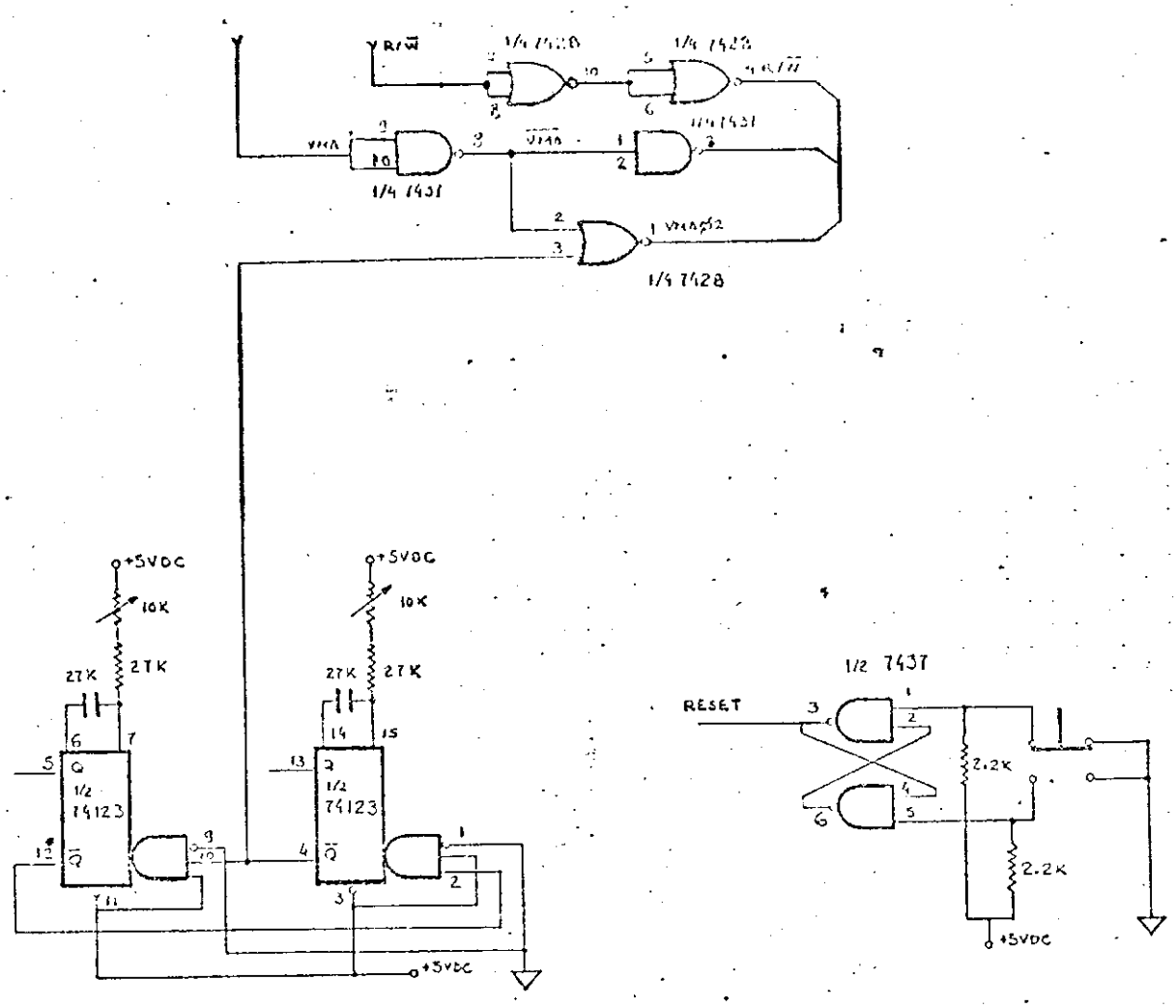
UCP


PROM 2
PROM 1
VMA
A15
A14
A13
A12
A11
A10
A9
A8
ACIA
CS2
A12
A11
A1
A0
R/W
E
RESET
6820
PA
PB
CA2
CA1
CB2
CB1
IRQA
IRQB
CS2
CS1
CS0
RS1
RS0
R/W
ENABLE
RES
DADOS 8BITS
DADOS 8 BITS
DADOS 4BITS
DADOS 4 BITS
ENDEREÇO
ESTADO
CONTROLE

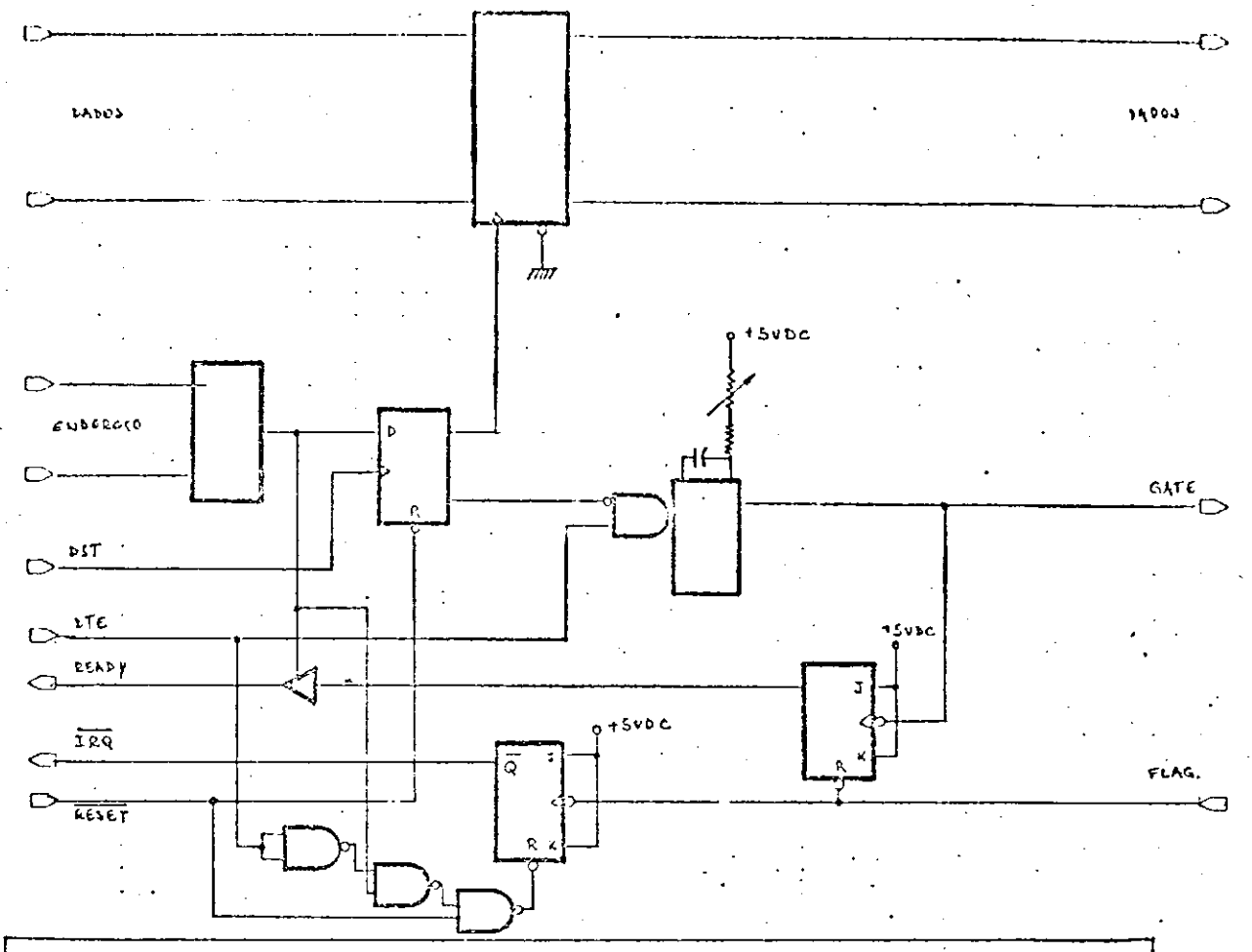

UMA. C2
A13
A12
A10
74138
2114
2708
PRGM
R/W

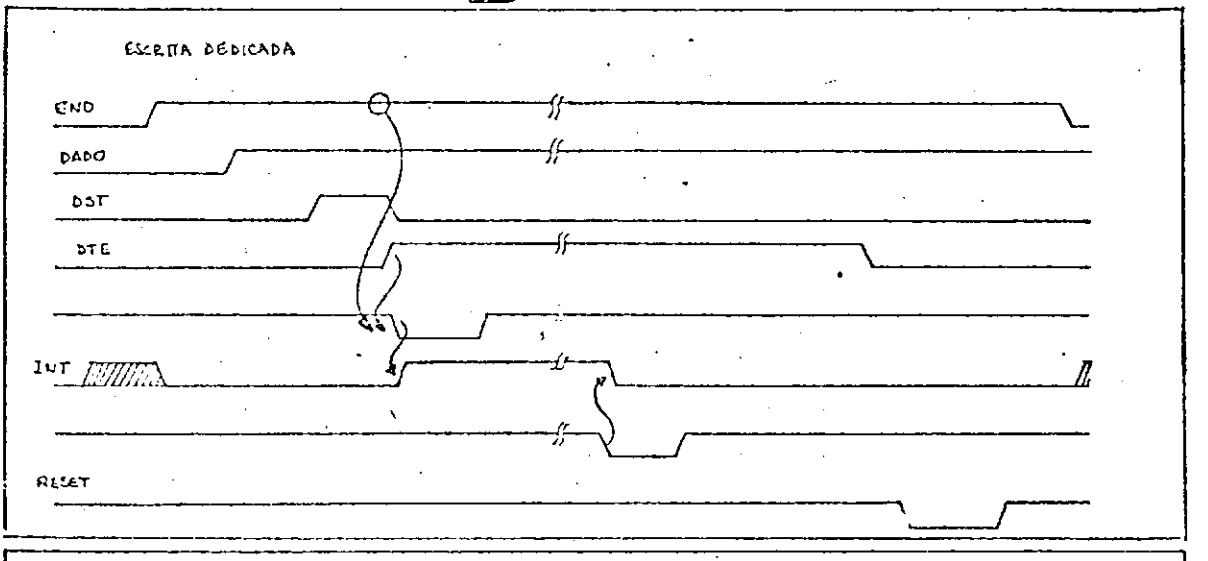
MEMORIA

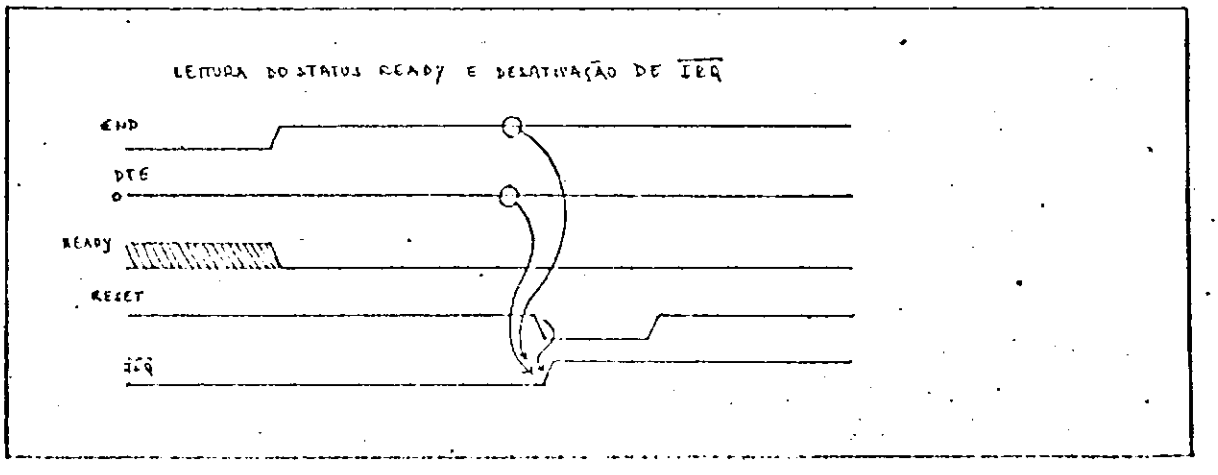

1/4 74188
RS 232C
+12VDC
-12VDC
6850
TX
RX
RTS
A12
A11
A0
Ø2
R/W
1/4 74189
+5VDC
74123



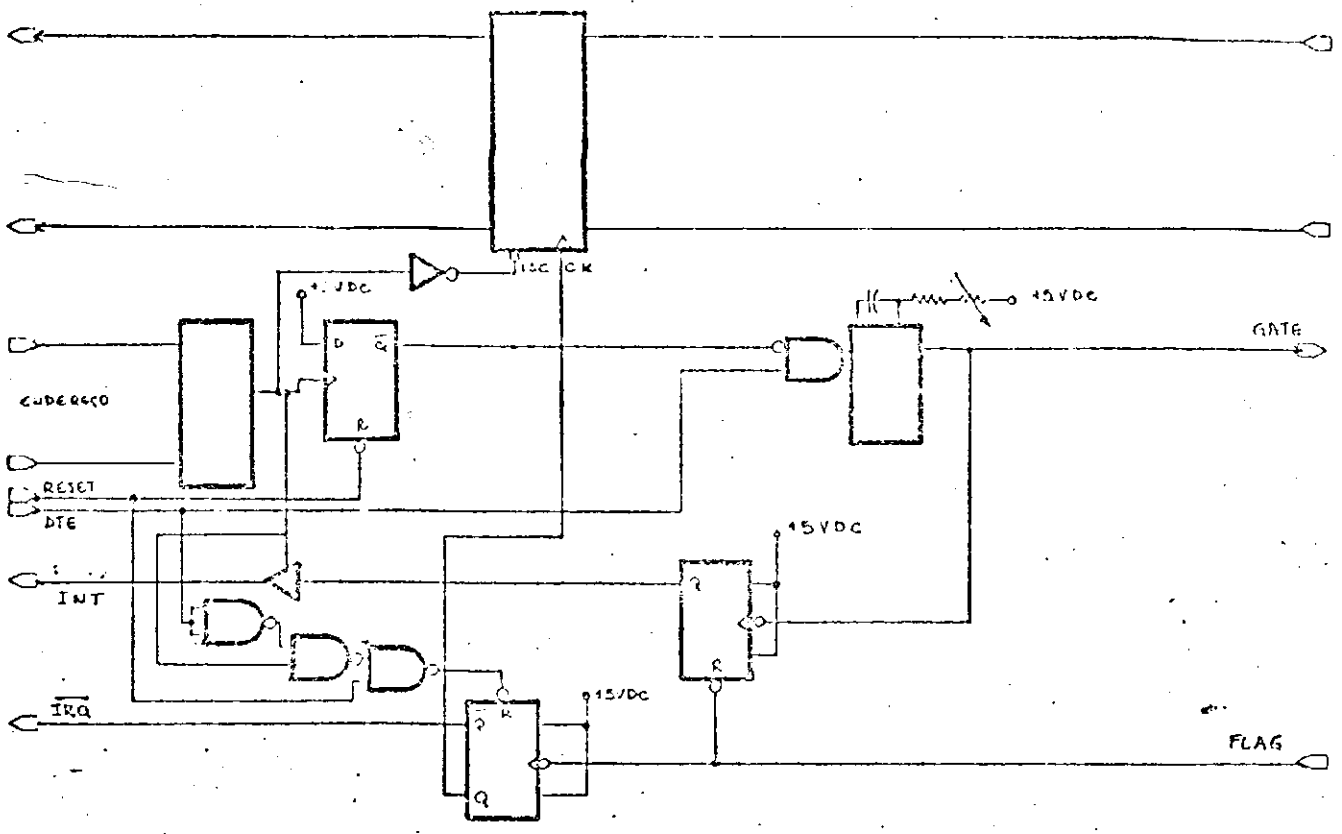
CLOCK . RESET & BUFFERS

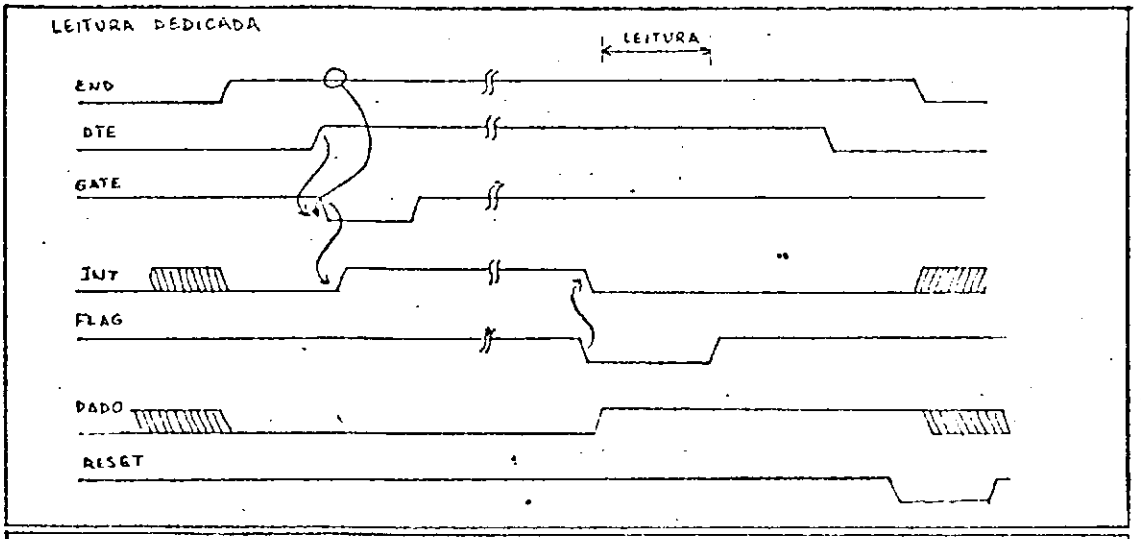


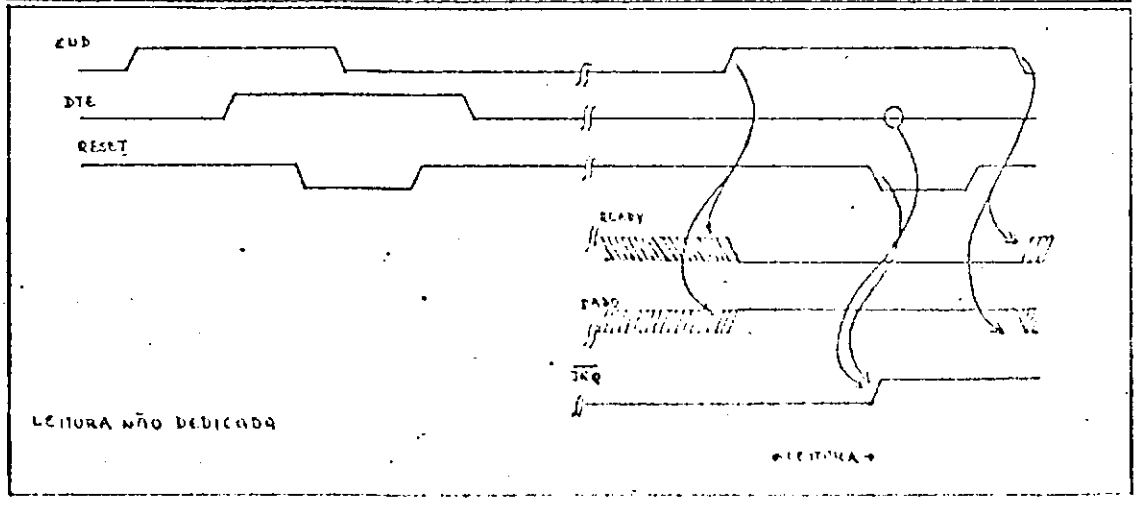




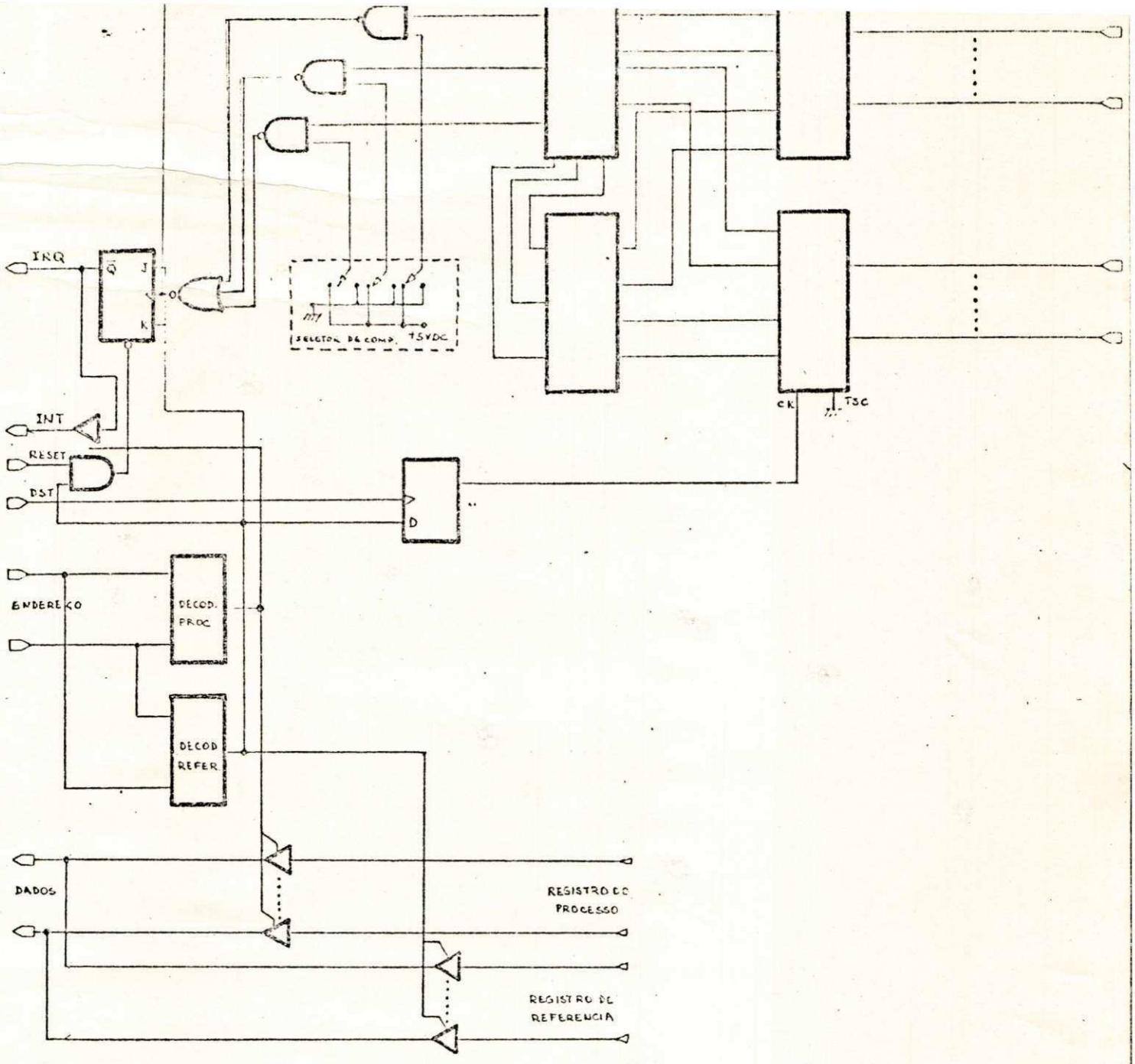
CARTÃO DE SAÍDA DIGITAL

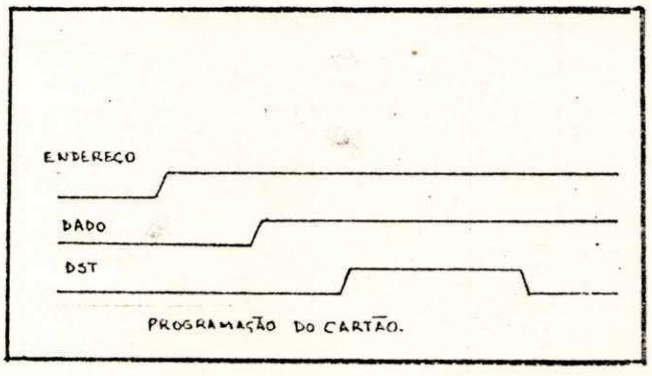




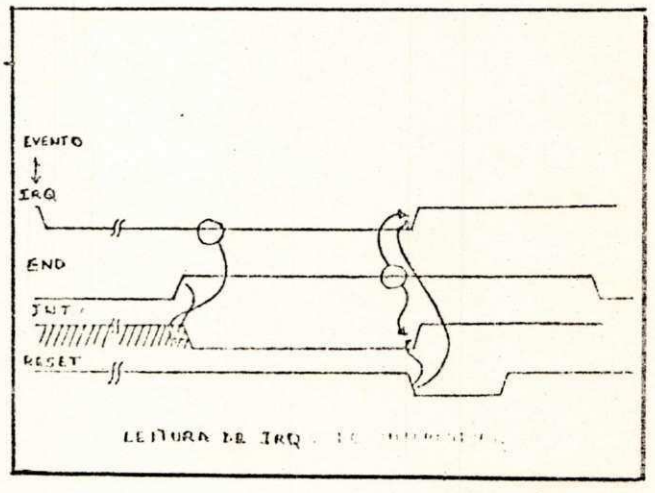


CARTÃO DE ENTRADA DIGITAL





PROGRAMAÇÃO DO CARTÃO.



LEITURA DE IRQ

CARTÃO DE DETECÇÃO DE EVENTOS