Alberto de Figueiredo Gontijo

PROTOTIPO DE UM SISTEMA DE DESENVOLVIMENTO PARA MICROPROCESSADORES 8088/8086

Dissertação apresentada ao curso de Pós-graduação em Engenharia Elétrica da Escola de Engenharia da Universidade Federal de Minas Gerais, como requisito parcial para obtenção do título de Mestre em Ciências.
Area de concentração: Automática
Orientador: Prof. Márcio Lage Siqueira
Universidade Federal de Minas Gerais

Belo Horizonte
Escola de Engenharia da UFMG
1989

"PROTÓTIPO DE UM

SISTEMA DE DESENVOLVIMENTO PARA MICROPROCESSADORES 8086/8088"

ENGº ALBERTO DE FIGUEIREDO GONTIJO

Dissertação de Mestrado submetida à banca examinadora, designada pelo Colegiado do Curso de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Minas Gerais, como parte dos requisitos necessários à obtenção do grau de Mestre em Ciência.

Aprovada em 24 de novembro de 1989.

Por:

Márcio Lage Siqueira (orientador)
Ms.C. CPGEE-UFMG/Prof. do CPGEE-UFMG

Fábio Gonçalves Jota
Ph.D. Univ. Oxford-GB/Prof. Deptº Física-UFMG

Paulo Fernando Seixas
Dr.D'Universite Toulouse-FR/Prof. do CPGEE-UFMG

Ronaldo Tadêu Pena
Ph.D. Univ. Texas-USA/Prof. do CPGEE-UFMG

Júlio Cezar David Melo
Ph.D. Univ. Texas-USA/Prof. do CPGEE-UFMG

AGRADECIMENTOS

Meus sinceros agradecimentos a todos aqueles que, de uma forma ou de outra, contribuiram para a realização desse trabalho, aos colegas, funcionários, professores da Escola de Engenharia da UFMG, ao CNPQ, pelo financiamento do projeto e principalmente ao Prof. Márcio Lage Siqueira, pela paciência e orientação.

Sou grato, também, a minha esposa Arlete, minhas filhas Layane e Laís, meus pais, irmãos e amigos, pelo incentivo e por darem especial sentido ao trabalho realizado.

Alberto de Figueiredo Gontijo

EPIGRAFE

"Penso noventa e nove vezes
e nada descubro;
deixo de pensar,
mergulho em profundo silêncio
e eis que a verdade se me revela"

Albert Einstein

INDICE

RESUMO

Foi projetado e construído um protótipo de um sistema de desenvolvimento para microprocessadores 8088/8086 da INTEL, denominado SD-16. Este sistema utiliza um microcomputador pessoal para emulação de um terminal de vídeo, geração e depuração de programas aplicativos e controle de um gravador de EPROMs. Uma das partes fundamentais do SD-16 é o kit de desenvolvimento KD-16, de baixo custo, baseado na CPU 8088, com as seguintes características: 32Kbytes de RAM, 128Kbytes de EPROM, 48 bits de E/S, 3 contadores programáveis de 16 bits, 8 níveis de interrupção, soquete para co-processador aritmético, interface série RS-232C, teclado e visor. O kit possui dois programas monitores, um para operação através de um pequeno teclado local e outro que permite comandar o KD-16 através de um terminal de vídeo usando a interface série RS-232C. O sistema possui também um gravador de EPROMs capaz de gravar memórias desde 2764 até 27512. O SD-16 pode ser utilizado como sistema de desenvolvimento em projetos e pesquisas na área de controladores digitais de processos ou como equipamento didático em aulas de laboratório nos cursos de graduação e pós-graduação. O KD-16 pode ser utilizado separadamente do SD-16, como controlador dedicado de baixo custo para processos industriais.

ABSTRACT

A simple microprocessor development prototype system, called SD-16, was designed and built for INTEL 8088/8086 microprocessors. The system makes use of a personal microcomputer for terminal emulation, design and debugging of aplication programs and control of an EPROM programmer. One important part of the SD-16 is the development kit, the KD-16, a low cost 8088-based microcomputer, with the following characteristics: 32 Kbytes of RAM; 128 Kbytes of EPROM; 48 bits for bidirectional I/O; three 16 bit programmable counters; 8 priority interrupts; arithmetic co-processor; RS-232C serial interface; keyboard and display. The kit has two monitor programs, one for small local keyboard operation and the other for video terminal operation. The system has also an EPROM programmer capable of programming memories from 2764 up to 27512. The SD-16 can be used as a development system on designs or research in the digital process control area, or in undergraduate and graduate microprocessor courses in laboratory subjects. The KD-16 can be also used separately from the SD-16, as a dedicated low cost industrial process controller.

INTRODUÇÃO

Nos dias de hoje, o computador vem sendo cada vez mais utilizado como ferramenta auxiliar de projeto nas diversas áreas da engenharia. O desenvolvimento nas áreas de controle digital de processos, tratamento digital de sinais e instrumentação, entre outras, é substancialmente facilitado pelos avanços da informática. A tendência crescente de utilização dos microprocessadores nas áreas citadas, reclama por novos métodos e equipamentos para auxílio aos projetistas. Entre os que dão esse apoio mais efetivamente, estão os "sistemas de desenvolvimento".

Um "sistema de desenvolvimento" é um equipamento que oferece um conjunto de ferramentas para desenvolver, testar, exercitar, e depurar o software e hardware de microprocessadores durante todas as fases de um projeto (HP 64000-UX, 1986).

Este equipamento, bastante complexo, caro e geralmente importado, não pode ser adquirido em qualidade e quantidade necessárias às aplicações nas áreas citadas, pela Escola de Engenharia da UFMG, no contexto econômico atual.

Atualmente, o Departamento de Engenharia Eletrônica da Escola de Engenharia da UFMG vem desenvolvendo vários trabalhos envolvendo controladores digitais, utilizando-se para isto de microprocessadores de 8 bits. Estes microprocessadores são inadequados para a implementação de controladores mais rápidos e mais complexos. Portanto, há necessidade de um sistema

de desenvolvimento que apoie os projetos que fazem uso de um microprocessador mais poderoso, a fim de atender às novas necessidades do Departamento.

Outro setor carente dos recursos oferecidos por este equipamento é o laboratório de ensino de microprocessadores, que atende aos cursos de graduação e pós-graduação.

O objetivo deste trabalho foi o de projetar e construir um "sistema de desenvolvimento" simplificado que respondesse às necessidades de ensino e pesquisa do Departamento de Engenharia Eletrônica da UFMG.

Este "sistema de desenvolvimento", deveria ter as seguintes características:

- ser de baixo custo;
- ter configuração versátil e expansível;
- utilizar componentes disponíveis no mercado brasileiro;
- apoiar a produção e depuração de programas aplicativos;
- dispor de fácil comunicação com o usuário;
- oferecer hardware básico de um microcomputador de 16 bits.

A dissertação está dividida em 4 capítulos e 3 apêndices:

No capítulo 1, vamos definir "sistema de desenvolvimento", descrever as características gerais dos sistemas de desenvolvimento comerciais e colocar nossa proposta de sistema de desenvolvimento moldado às necessidades e limitações do Departamento de Engenharia Eletrônica da Escola de Engenharia da UFMG.

No capítulo 2, apresentamos o kit de desenvolvimento KD-16 e os programas auxiliares de apoio ao kit: o programa "Monitor Série", o programa "Terminal" e o programa "Memintel".

No capítulo 3, descrevemos as placas periféricas de apoio ao kit KD-16 e a um microcomputador pessoal de 16 bits, assim como os programas que tornam estas placas funcionais, quando for o caso. São descritos: o "Módulo de Teclado e Visor" (com o programa "Monitor para Teclado"), a placa de "Interface Paralela e Temporização", a placa do "Gravador de EPROMs" (com o programa "Gravador"), e a placa "Rastreadora" (com o programa "Rastro"). Na descrição das placas citadas acima, procuramos mostrar o papel assumido por cada uma delas dentro da filosofia de projeto adotada.

No capítulo 4, juntamente com as conclusões, é feita uma avaliação do sistema projetado, apresentando suas vantagens, limitações e sugestões para trabalhos futuros.

O Apêndice A contém os circuitos esquemáticos e conectores de cada placa desenvolvida.

O Apêndice B contém as listas de material de cada placa do SD-16.

O Apêndice C apresenta as listagens dos programas desenvolvidos.

CAPITULO 1 - SISTEMAS DE DESENVOLVIMENTO

1.1 - Sistemas de Desenvolvimento - Definição

Logo nos primeiros anos da era dos microprocessadores, ficou clara a necessidade de se criar ferramentas apropriadas para apoiar o projeto e desenvolvimento de sistemas que utilizassem esta tecnologia.

A rápida evolução da microeletrônica colocou à disposição dos projetistas, em um período de tempo muito curto, componentes cada vez mais complexos. O software acompanhou este crescimento, reclamando por diferentes métodos de projeto e depuração.

Programar em linguagem Assembly e testar o hardware de equipamentos microprocessados são tarefas difíceis. Um dos primeiros equipamentos empregados para este fim foi o Analisador de Estados Lógicos. Entretanto, este não representava uma solução satisfatória, pois não respondia eficientemente a todas as necessidades dos projetistas tanto de software quanto de hardware.

Os sistemas de desenvolvimento surgiram justamente da necessidade de se ter um conjunto poderoso de ferramentas integradas que habilitassem ao usuário desenvolver, testar, exercitar e depurar o software e hardware de equipamentos baseados em microprocessadores, em todas as fases do desenvolvimento. Tipicamente, um sistema de desenvolvimento se constitui de uma configuração particular de um computador, projetada pa-

ra ser interligada a vários dispositivos, os quais fornecem facilidades para ajudar na realização das tarefas citadas (TSENG, 1982a).

Hoje em dia, a maioria dos fabricantes de microprocessadores oferece também um sistema de desenvolvimento para auxiliar os projetistas que lidam com sua família de chips. Essees sistemas são, geralmente, comandados por microprocessadores de uma geração anterior àquela que irão subsidiar.

A figura 1.1 (TSENG, 1982a) mostra o diagrama de blocos de um sistema de desenvolvimento típico (mono-usuário).

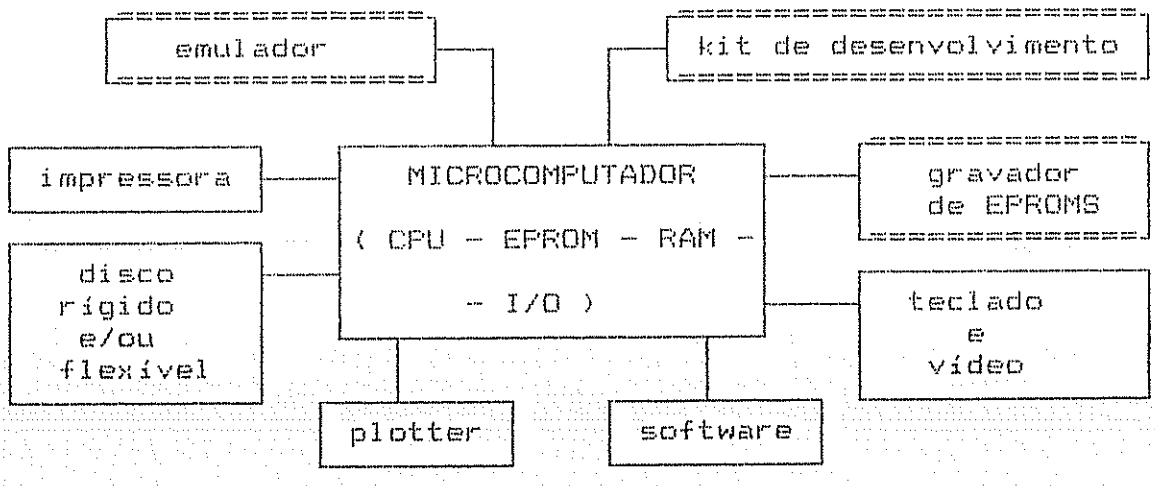


Figura 1.1 - Diagrama de blocos de um sistema de desenvolvimento

Esta configuração típica é composta de um microcomputador comum, acrescido dos blocos do gravador de EPROMs, do bloco emulador, do kit de desenvolvimento, e do software associado.

Temos, ao centro, a unidade de processamento do sistema de desenvolvimento. Conectada a ela encontramos memórias EPROM e RAM, dispositivos de entrada e saída, além de outros circuitos mais específicos como: controladores de acesso direto à memória e controladores inteligentes de periféricos. Normalmente, este conjunto é dotado de um barramento padrão, que permite maior flexibilidade de configuração e futuras expansões.

O bloco "teclado e vídeo" representa a interface com o operador, de modo que este possa agir interativamente no sistema.

Como memória de massa, temos os discos rígidos e/ou flexíveis. Uma impressora ajuda na documentação do trabalho.

Outro bloco imprescindível é o gravador de memórias EPROMs. Como são muitos os tipos de memórias existentes, alguns gravadores limitam o espectro de memórias utilizáveis a somente uma parte do universo disponível.

Um kit, com uma configuração básica de microprocessador, memória e periféricos, fornece ao projetista uma base de hardware, com capacidade para controlar pequenos sistemas. Normalmente, ele possue um programa monitor, que lhe confere habilidades de um mini-sistema de desenvolvimento.

O bloco emulador permite testar um software e/ou um hardware externos, em tempo-real. Este procedimento é conhecido como "depuração não intrusiva". Pode-se realizar simulações de circuitos ainda não implementados fisicamente, com o objetivo de testar, antecipadamente, parte do software e hardware.

Pode-se, inclusive, ter parte da memória do protótipo mapeada na memória do sistema de desenvolvimento. Esta característica permite a substituição de memórias EPROM por memórias RAM durante a fase de teste e depuração.

Dependendo do fabricante, encontramos desde simples programas monitores até sofisticados rastreadores/depuradores que permitem acompanhar o estado dos registradores e variáveis do sistema em estudo, referenciando cada uma delas por seu mnemônico definido no programa fonte.

No bloco do suporte de software encontramos o sistema operacional, os compiladores, os assemblers, os depuradores, os editores e, enfim, todo software que, de algum modo, apoia o trabalho de desenvolvimento.

1.2 - Sistemas de desenvolvimento comerciais

1.2.1 - Descrição geral

Existe uma grande variedade de produtos destinados a apoiar o desenvolvimento de sistemas baseados em microprocessadores. Dentro desse leque de opções oferecidas, encontramos produtos com as mais diferentes filosofias e as mais diferentes propostas. A figura 1.2 (KRUMMEL, 1977) apresenta o amplo espectro de ferramentas de apoio ao desenvolvimento de microprocessadores.

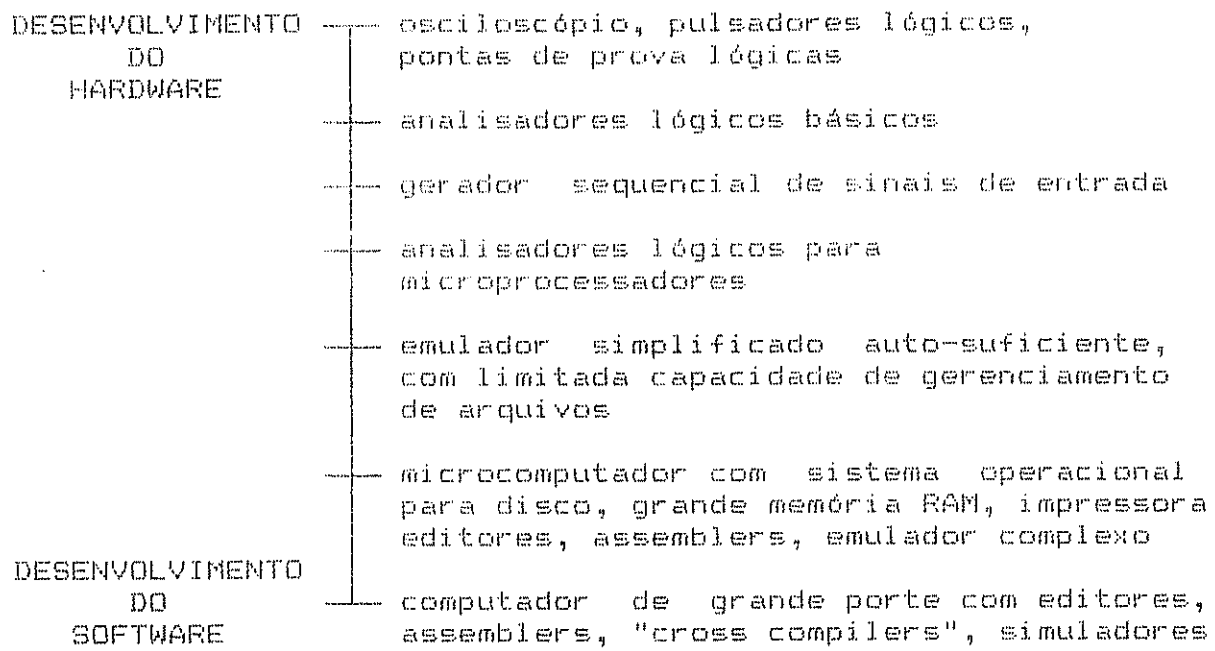


Figura 1.2 - Ferramentas de apoio para o desenvolvimento de equipamentos baseados em microcomputadores.

Podemos observar que o apoio ao desenvolvimento envolve desde computadores de grande porte em um extremo, até hardware sequencial no outro. Na parte superior da figura 1.2, temos os meios orientados para o desenvolvimento do hardware, e embaixo, os meios mais voltados para o desenvolvimento do software. Ao centro, temos o equilíbrio software/hardware conseguido com os emuladores.

Os meios de apoio ao desenvolvimento orientados mais para o hardware são adequados para depurar problemas na lógica do circuito depois que o problema tiver sido isolado. Eles são de pouca utilidade no início da depuração, pois não possuem um meio eficaz de excitar o hardware do sistema sob teste, que pode não estar funcionando. As operações básicas necessárias

são: ler/escrever nas memórias e ler/escrever nos periféricos. Nesta categoria, podemos citar os osciloscópios, os pulsadores e pontas de prova lógicas, os analisadores de assinaturas e os analisadores lógicos multicanais.

Para facilitar a interpretação dos inúmeros sinais coletados no equipamento sob teste e agilizar as conexões, surgiram os analisadores especializados para determinados microprocessadores. Estes analisadores, ao invés de apresentarem tabelas contendo números binários, de difícil interpretação pelo usuário, são capazes de reconstituir as instruções em linguagem Assembly, ao lado dos endereços e dados mostrados no formato hexadecimal. Sua conexão com o sistema sondado é feita por um conector que substitui o microprocessador diretamente em seu soquete, diminuindo a possibilidade de erros de ligação (OGDIN, 1977a).

Muitas vezes durante o trabalho de teste de aparelhos baseados em microprocessadores é necessário a sintetização de sinais sequenciais repetitivos, que não são convenientemente obtidos do ambiente onde este sistema irá trabalhar, seja por questões de segurança, praticabilidade ou custo. Neste caso pode-se fazer uso de um gerador sequencial de sinais de entrada, que cria testes repetitivos em tempo-real, impondo ao software e hardware condições críticas de teste (PANNACH, 1975; OGDIN, 1977b; OGDIN, 1978).

No extremo inferior do espectro da figura 1.2, temos os computadores de grande porte, mais adequados para solução de problemas de desenvolvimento do software. Por não serem

portáteis e apresentarem um elevado nível de complexidade, são direcionados fundamentalmente para uso por programadores ao invés de técnicos de nível médio. Estas características os tornam inadequados à depuração do hardware e em testes finais nas linhas de produção (BRISTOW, 1982).

A depuração e testes nestes sistemas de grande porte são realizados por programas simuladores ao invés de emuladores. A maioria dos programas simuladores oferecem pouca ajuda quando o software a ser desenvolvido deve ser executado e depurado em tempo-real. Apresentam, também, dificuldades para simular as interrupções de hardware. Os simuladores mais complexos e modernos permitem uma mudança progressiva da situação de simulação plena para uma situação de simulação parcial, na qual temos uma interação com um emulador diretamente interligado ao sistema alvo (MICHELIN, 1988; LEJEUNE, 1982).

Antes que os emuladores estivessem disponíveis para a depuração em tempo-real, os simuladores eram a principal ferramenta utilizada para o acompanhamento dos estados internos do processador.

O hardware dos equipamentos que serão produzidos em escala comercial devem ser minimizados de modo a baixar os custos de produção. Portanto, estes equipamentos não comportam a sobrecarga de ter um programa monitor implantado em sua memória, ou uma interface de comunicação provisória, que ajudem o trabalho de desenvolvimento durante a fase de teste e depuração. O desenvolvimento deste tipo de equipamento, exige uma maneira externa de perscrutar a operação e estados do sistema

alvo.

Uma vez que as ferramentas nos dois extremos do especto apresentado não interagem adequadamente com o hardware do usuário ou não conseguem excitar convenientemente o sistema alvo, foram desenvolvidos equipamentos que constituem a 1ã geração de emuladores. Estes primeiros emuladores, um tanto rudimentares, possuiam um único microprocessador que comandava, alternadamente, tanto o sistema emulador como o sistema do usuário (HORTON, 1985; KRUMMEL, 1977; MIHALIK, 1982a). Possuiam capacidade de executar comandos simples, que atuavam no hardware do usuário, por exemplo:

- verificar o conteúdo dos registros do microprocessador;
- carregar programas na memória;
- procurar um byte especificado na memória;
- encher a memória do sistema alvo com um byte especificado;
- ler ou escrever um byte na memória ou num dispositivo de E/S, continuamente;
- executar um teste de memória escrevendo e lendo "1'S" e "0'S" com auto-incremento;

O barramento do sistema emulador é interligado ao sistema alvo, através de buffers, diretamente no soquete da CPU do usuário. Suas linhas de endereço, dados e controle passam então a controlar o sistema alvo, simulando a presença da CPU (NATIONAL NS32CG16, 1989; USATEGUI, 1985a). Os comandos permitidos por este tipo de equipamento são bastante faceis de aprender, permitindo o uso dessa ferramenta por usuários

inexperientes ou técnicos com pouco conhecimento da linguagem Assembly.

A evolução destes sistemas rudimentares, associada a evolução dos microcomputadores, levou à implementação de sistemas de desenvolvimento bem mais completos. Nestes, temos incorporadas todas as facilidades e periféricos antes só disponíveis nos minicomputadores, como: discos rígidos de alta capacidade, impressoras rápidas, grande espaço de memória RAM, sistemas operacionais multi-usuário para operação em tempo-real, editores de texto, depuração à nível de programa fonte para linguagem de alto nível, etc.

Alguns sistemas de desenvolvimento utilizam estações de trabalho de alto desempenho para controle de módulos emuladores intercambiáveis, adequados para cada tipo de microprocessador. Esta implementação é bastante versátil pois fornece uma plataforma de trabalho eficiente para CAE (Computer-Aided Engineering) e CAD (Computer-Aided Design), além de apoiar o trabalho de teste e depuração em tempo-real. A capacidade de operação com múltiplos usuários facilita a integração do trabalho de uma equipe através do compartilhamento de arquivos e periféricos (ROMANO, 1988; HP Real Time Emulators, 1986).

Devido à alta confiabilidade requerida no trabalho de desenvolvimento, alguns sistemas tem incluso um sofisticado sistema de auto-teste, que é automaticamente acionado ao energizar o equipamento. Para ajudar a manutenção, alguns sistemas de desenvolvimento possuem embutido um microcontrolador independente com capacidade de executar programas de auxílio a ma-

nutenção, diminuindo ao máximo possível o tempo necessário para a realização desses serviços (KISTER, 1982).

Nos sistemas de desenvolvimento atuais são comuns os emuladores de 2ª geração, onde encontramos a divisão de tarefas entre diferentes microprocessadores (multi-processamento), ficando um microprocessador dedicado somente ao sistema do usuário. Esta arquitetura permite maior velocidade de operação, facilita a emulação em tempo-real e facilita o suporte de diferentes famílias de microprocessadores pela simples troca da placa do sistema emulador (KRUMMEL, 1977).

Para a execução de trabalhos extremamente sensíveis à interferência do emulador, foram desenvolvidos emuladores de 3ª geração, onde o microprocessador usado para emulação é especialmente fabricado, possuindo terminais extras que dão acesso a sinais internos ao chip comercial. Embora estes emuladores apresentem desempenho mais atraente, são menos comuns devido ao seu elevado preço (MIHALIK, 1982b).

Alguns fabricantes de sistemas de desenvolvimento fornecem kits com uma configuração básica de microprocessador, memória e periféricos, ou então, placas num barramento padrão, com interfaces paralelas, interfaces seriais, contadores/temporizadores e placas experimentais para "wire-wrap". Estes kits e placas permitem que os projetistas realizem testes em um hardware bem parecido com o hardware do sistema alvo, desde o início do trabalho de desenvolvimento.

Alguns kits possuem memória RAM e EPROM suficientes para dar um razoável suporte ao desenvolvimento e depuração de

software. Comumente, seus programas monitores permitem exami-nar e modificar o conteúdo dos registros do microprocessador e memória, ler e escrever nos circuitos periféricos, especificar pontos de parada na execução de um programa, realizar conver-sões de números hexadecimais para decimais. Outros, mais so-fisticados, permitem também a operação passo a passo, além de possuirem assemblers e desassemblers residentes em EPROM (NATIONAL MDS, 1989; OGDIN, 1978).

1.2.2 - Emuladores

Os emuladores constituem uma ferramenta essencial para estabelecer, eficientemente, a ligação entre o trabalho de desenvolvimento do software e a integração final hardware/ software. Além disso, o emulador permite que as equipes res-ponsáveis pelo desenvolvimento do hardware e do software tra-balhem com um mínimo de dependência na fase inicial do proje-to.

A emulação "in circuit", como o nome sugere, envolve a simulação de todas as operações da CPU na sua posição física normal. Pretende-se emular a CPU e, simultaneamente, monitorá-la através de circuitos auxiliares, os quais seriam indesejá-veis ou inviáveis de serem construídos no produto do usuário.

A emulação "in circuit" é implementada por um siste-ma composto de hardware e software. O hardware do emulador precisa se comportar o mais parecido possível com o micropro-cessador do sistema alvo. Este requerimento pode ser melhor

atendido com a utilização, no emulador, do mesmo tipo de microprocessador que o utilizado no sistema do usuário. Para poder enfrentar a carga extra das capacitâncias e indutâncias distribuidas inerentes ao cabo de conexão são utilizados buffers. Os atrasos associados aos buffers são compensados com a utilização de microprocessadores selecionados quanto a velocidade (BRISTOW, 1982).

O software de controle do emulador é composto, normalmente, de duas partes. A primeira parte fica residente no próprio emulador em memórias não voláteis (firmware). As rotinas deste firmware são executadas pelo microprocessador de emulação num modo de operação chamado "modo monitor" ou "modo de interrogação", e cuidam das tarefas básicas da interação entre o sistema do usuário e o emulador.

A segunda parte é um software carregado e executado na memória RAM do sistema de desenvolvimento, fazendo o interfaceamento entre o emulador e o usuário. Tipicamente esta interface é uma forma especializada de linguagem interpretada, que coloca uma variedade de comandos à disposição do usuário, além de traduzir de modo compreensível as informações fornecidas pelo firmware.

As informações trocadas entre o software de controle do emulador e o firmware, na verdade, não envolvem transmissão real de dados. O procedimento comum é a utilização de áreas compartilhadas de memória, formatadas em blocos de parâmetros, que são construidos e modificados pelos dois ambientes, o sistema de desenvolvimento e o sistema alvo.

A tabela 1.1 a seguir apresenta um resumo das necessidades básicas dos projetistas e a resposta dada a cada uma delas pelos sistemas emuladores (HP Real Time Emulators, 1986; NATIONAL NS32C016, 1989; LEJEUNE, 1982).

NECESSIDADES DOS PROJETISTAS	RESPOSTA DOS EMULADORES
Implementar o ambiente básico do usuário	- emulação "in circuit" - execução em tempo-real - recursos de mapeamento
Controlar o fluxo do programa	- pontos de parada por hardware
Observar o fluxo do programa	- rastreamento - operação passo a passo
Acompanhar variáveis	- mostrar conteúdos de memórias - mostrar conteúdos dos registros do microprocessador - mostrar dados de portas de E/S
Possibilitar trabalho em alto nível de abstração	- depuração simbólica - interpretação da memória
Ferramenta de fácil uso	- comandos poderosos - interpretador com suporte para macrocomandos e comandos compostos

Tabela 1.1 - Necessidades dos projetistas versus respostas dos emuladores.

1.2.3 - A questão do software

Muitos dos projetos de aparelhos baseados em microprocessadores têm como principal dificuldade a geração do software. O hardware dos microcomputadores usam, normalmente, arquiteturas bastante padronizadas e familiares, enquanto o

software possui particularidades que o torna único, consumindo mais tempo para sua produção e depuração. Este grande tempo necessário para a produção do software o torna a parte mais cara do projeto (MAGERS, 1980; TSENG, 1982b).

Se um certo produto deve ser comercializado em grande escala, é importante que se busque a minimização do hardware e, consequentemente, do tamanho da memória disponível para executar o software. O gráfico da figura 1.3 (MAGERS, 1980) mostra que, à medida que nos aproximamos da condição de 100% de utilização do espaço de memória em um sistema alvo, o custo da produção do software tende a subir devido à maior dificuldade em realizar modificações e acrescentar rotinas. Os requerimentos de minimização da memória e minimização de custos são, neste caso, conflitantes.

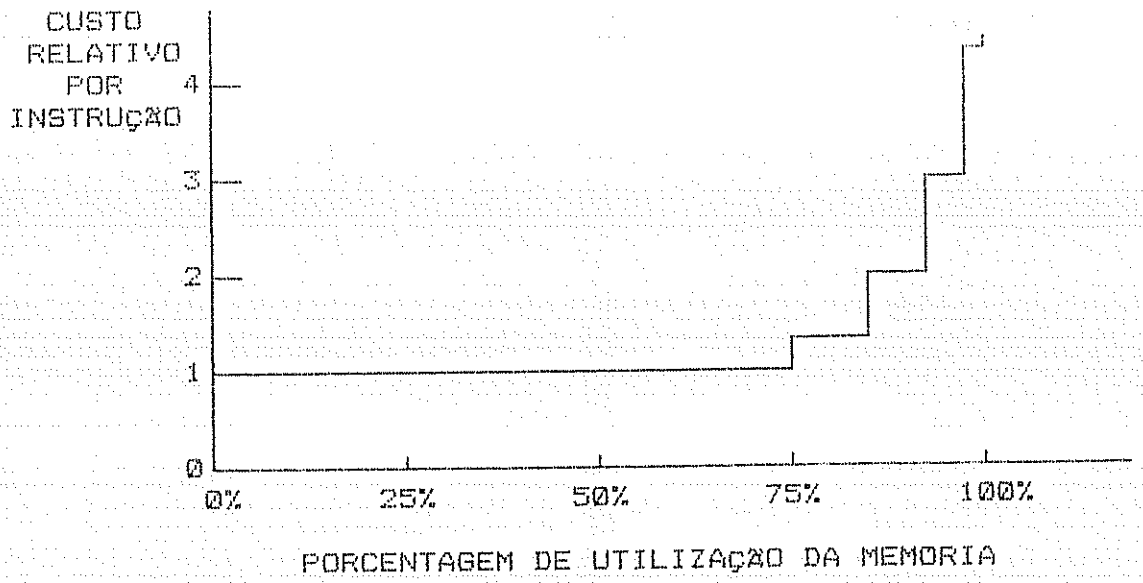


Figura 1.3 - Custo do software versus porcentagem de utilização da memória.

Para conseguir aumentar a produtivivade na produção do software, pode-se valer principalmente de duas estratégias: usar liguagens de alto nivel e usar boas ferramentas de apoio ao desenvolvimento.

O uso de compiladores para linguagens de alto nivel implica em vantagens e desvantagens:

VANTAGENS:

- maior produtividade do programador (de 3 a 10 vezes)
- menor custo de manutenção
- incentiva programas mais estruturados
- menor custo total, para produções em pequena escala
- melhora a documentação
- facilita o projeto de programas grandes
- grande capacidade de detecção de erros
- dá suporte para diferentes tipos de estruturas de dados

DESVANTAGENS:

- programas mais volumosos
- execução mais lenta
- mais distante do hardware

Atualmente, existe uma tendência para a utilização das linguagens "Pascal" e "C", para a geração do firmware dos aparelhos baseados em microprocessadores. Os compiladores otimizados destas linguagens conseguem atingir cerca de 80% do desempenho comparado ao código gerado em Assembly. Desse modo, só precisam ser feitos em Assembly os programas seriamente limitados em tamanho e velocidade (BRISTOW, 1982; TSENG, 1982c).

Um método de desenvolvimento de software bastante utilizado e interessante é a mistura das linguagens Pascal ou C com trechos em Assembly. Por exemplo: um programa pode ser implementado, num primeiro momento, em Pascal. Ao testar este programa, devem ser localizadas as rotinas mais críticas, procurando substitui-las por outras em Assembly. Esta "sintonização fina" de porções do programa original, tem a desvantagem de diminuir a facilidade de manutenção e portabilidade da linguagem de alto nível para as seções sintonizadas. Apesar dessas desvantagens, existem vantagens adicionais:

- retenção da estrutura "Top-Down" do Pascal ou C;
- código preliminar mais confiável;
- um aparelho protótipo pode tornar-se disponível mais cedo, devido a maior produtividade da linguagem de alto nível.

Para finalizar, é importante frisar que o software é um componente em constante aprimoramento, o que aumenta, sensivelmente, a importância de uma disciplinada documentação.

1.3 - Sistema de Desenvolvimento SD-16

Os sistemas de desenvolvimento comerciais são aparelhos bastante complexos, tipicamente voltados para apoiar o desenvolvimento de sistemas os mais diversos possíveis.

Os emuladores constituem parte importante num sistema de desenvolvimento comercial, principalmente em aplicações profissionais, onde a eficiência, rapidez, e confiabilidade

durante um trabalho de desenvolvimento podem significar perder ou ganhar uma fatia do mercado e muito dinheiro.

Por outro lado, podemos dizer que os emuladores são partes opcionais nos sistemas de desenvolvimento mais simples, porque é um equipamento de alto custo, podendo ter parte de suas habilidades desempenhadas por analisadores lógicos, ou mesmo, por software, através de programas monitores. Evidentemente, estas substituições implicam em menor eficiência no trabalho de desenvolvimento, e a impossibilidade de simulação do hardware acarreta uma certa defasagem entre o desenvolvimento do software e do hardware.

No Departamento de Engenharia Eletrônica da Escola de Engenharia da UFMG, percebemos que existe uma demanda mais ou menos definida em termos de hardware de microcomputador. Esta configuração típica é projetada e construida novamente a cada novo trabalho, levando assim a um grande desperdício de recursos e pulverização de esforços devido às dificuldades do desenvolvimento do hardware, antes que o usuário possa enfocar a questão principal do seu trabalho.

Concluimos que, se pudermos fornecer um sistema de microcomputador de uso geral com as características típicas necessárias às aplicações do Departamento de Engenharia Eletrônica, e além disso, fornecer um conjunto de complementos em software e hardware que facilitem o trabalho de desenvolvimento, teremos dado uma importante contribuição para incentivar o trabalho de ensino e pesquisa em nossa Escola.

Entre estes complementos, num primeiro momento, não precisaríamos incluir o emulador, pois de antemão, podemos reservar um espaço de memória suficiente para mantermos um programa monitor residente, capaz de realizar algumas das tarefas comuns aos emuladores, embora com menor eficiência e sem a capacidade de simulação.

No SD-16, como talvez 80% do hardware tipicamente utilizado já esteja presente no kit KD-16 e placas associadas, a capacidade de simulação do hardware torna-se secundária. A placa rastreadora e o programa monitor substituem diversas funções presentes nos emuladores como: transferência de programas para a memória do sistema alvo, especificação de pontos de parada durante a execução do programa, execução monitorada de programas passo a passo, verificação e modificação do conteúdo dos registros do microprocessador e memórias, e monitoração de acessos à memórias e periféricos.

Tendo em vista a exposição acima, esboçamos um protótipo de um sistema de desenvolvimento simplificado, mostrado na figura 1.4.

Na figura 1.4, vemos que existem dois blocos que centralizam as operações no sistema SD-16: o bloco "µC PESSOAL" e o bloco "KD-16". Estes blocos são capazes de controlar os blocos passivos que são: o bloco "GRAVADOR DE EPROM", o bloco "PLACA RASTREADORA" e o bloco "PLANTA A CONTROLAR".

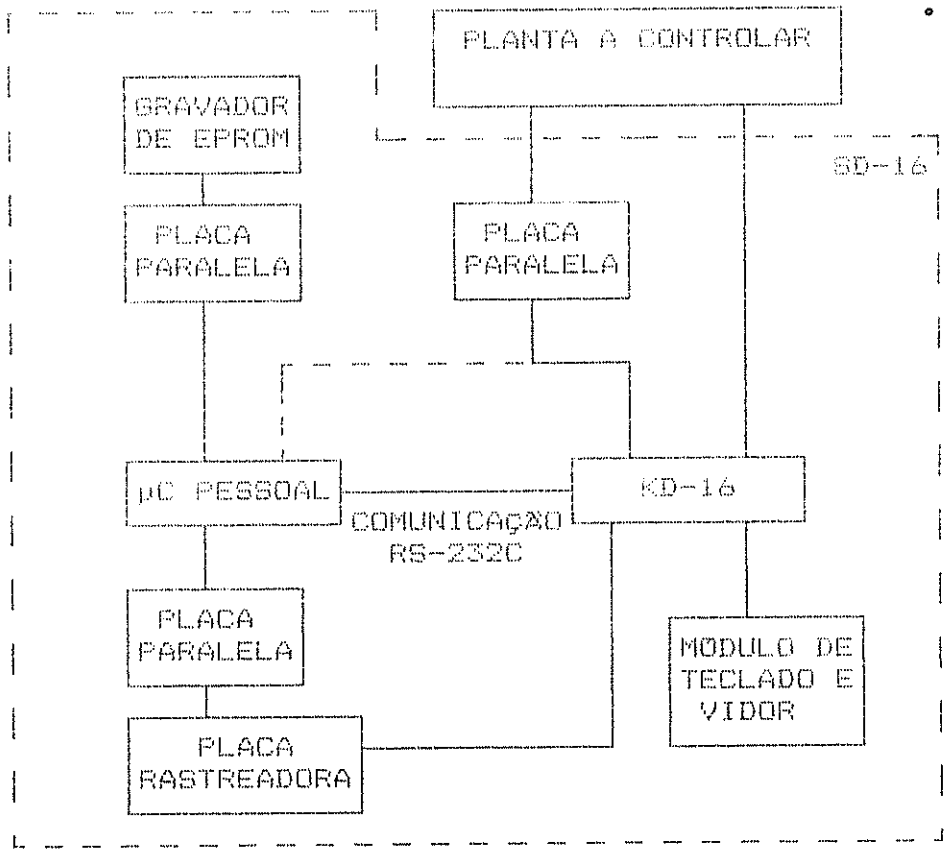


Figura 1.4 - Diagrama de blocos do sistema de desenvolvimento SD-16.

Os elos de ligação entre os centros de processamento e os blocos passivos são os blocos "PLACA PARALELA", que, de maneira muito conveniente, atendem tanto ao microcomputador pessoal quanto ao kit KD-16.

Esta dupla possibilidade de conexão nos dá a liberdade de realizar grande parte do trabalho de desenvolvimento de programas aplicativos utilizando o microcomputador pessoal, com seus poderosos programas depuradores, e posteriormente, trasladar o programa já sensivelmente depurado para o KD-16.

Para que este esquema de trabalho fique mais auto-

mático, escolhemos para o bloco "µC PESSOAL" um microcomputador compatível com o IBM-PC, que possui um microprocessador semelhante ao usado no KD-16 (IBM-PC, 1983; CIARCIA, 1982). Isto facilita a produção e teste de programas aplicativos desenvolvidos para o KD-16, dispensando o uso de simuladores do 8088 no microcomputador pessoal.

Outra vantagem dessa escolha, é que os microcomputadores compatíveis IBM-PC são comumente disponíveis em nossas Universidades. Como o custo do microcomputador pessoal assume elevados percentuais dentro do custo total do SD-16, fica bastante fácil a reprodução dos blocos restantes do sistema de desenvolvimento.

O SD-16 possui quatro programas que são executados no microcomputador pessoal de modo a permitir sua interação com o Gravador de EPROMs, Placa Rastreadora e o Kit KD-16:

- Programa GRAVADOR: O programa GRAVADOR, com a ajuda da placa de interface paralela, comanda as operações da placa do gravador de EPROMs. Possui 15 funções, além da função básica de gravação de memórias EPROM. é baseado na técnica de "menus" para facilitar sua utilização.
- Programa RASTRO: O programa RASTRO, que também faz uso da placa de interface paralela, permite o acompanhamento dos estados do barramento do Kit KD-16 através da tela e da impressora do microcomputador pessoal. Permite também o rastreamento seletivo segundo faixas de endereço e tipo de operação: se leitura/escrita, se memória/periférico.

- Programa TERMINAL: O programa TERMINAL transforma o microcomputador pessoal em um terminal de operador, para controle do Kit KD-16. Permite realizar transferências de programas aplicativos, dos arquivos em disco para a memória do KD-16 e vice-versa.
- Programa MEMINTEL: O programa MEMINTEL é um programa que converte arquivos binários para arquivos, no formato hexadecimal INTEL (XPTO, 1987). Os arquivos convertidos são usados nas transmissões de programas aplicativos para o KD-16.

No KD-16 existem dois programas monitores: um para permitir a operação pelo módulo de teclado e visor, e outro, que permite uma operação mais cômoda, usando um terminal de vídeo interligado via porta serial. Estes monitores são adaptações dos programas monitores do Kit SDK-86 da INTEL (INTEL, 1978a; INTEL, 1978b). Eles permitem a transferência de programas do microcomputador pessoal para o KD-16 e a execução de programas controlada passo a passo. Permitem também a leitura e escrita em memórias, periféricos e registros do microprocessador.

CAPITULO 2 - KIT DE DESENVOLVIMENTO KD-16

2.1 - Introdução

Atualmente, na Escola de Engenharia da UFMG, qualquer trabalho que envolva algum tipo de controle baseado em microprocessadores, certamente encontrará como entrave a construção de um pequeno sistema contendo um hardware básico de microcomputador.

As dificuldades são muitas, por exemplo: a obtenção de verbas, o atraso no recebimento das verbas, a dificuldade de aquisição de componentes, os prazos reduzidos para sua realização, a falta de pessoal especializado para o projeto e montagem das placas, a falta dos equipamentos de apoio para os testes finais, etc.

O tempo, o dinheiro e os esforços investidos antes que se possa dar início ao trabalho propriamente dito, podem muitas vezes, inviabilizá-lo. Assim sendo, entendemos ser fundamental o desenvolvimento de um kit de microcomputador que vá de encontro à maioria das necessidades dos usuários de microprocessadores na Escola de Engenharia. Listamos abaixo alguns requisitos para aplicações básicas:

- utilizar componentes encontrados no mercado nacional;
- usar componentes e placas de baixo custo;
- usar um microprocessador poderoso e com vasto apoio à programação e depuração;
- dispor de memórias RAM e EPROM de fácil expansão;

- dispor de interfaceamento paralelo para E/S;
- dispor de temporizadores e contadores de eventos;
- dispor de entradas para requisição de interrupções, para atendimento a eventos críticos;
- dispor de apoio ao processamento aritmético em ponto flutuante;
- dispor de interface de fácil comunicação com o usuário;
- permitir fácil reprodução do trabalho desenvolvido;

Para atender a estes requerimentos, criamos um kit de microcomputador de uso geral, denominado "KD-16". Ele pode ser utilizado como um kit de desenvolvimento, oferecendo um hardware básico sob controle do processador 8088-2.

A figura 2.1 apresenta o diagrama de blocos do KD-16.

2.2 - Descrição do Kit KD-16

A placa do KD-16 compõe-se dos seguintes componentes:

- microprocessador 8088-2;
- coprocessador aritmético 8087-2;
- gerador de relógio (cristal de até 24MHz) 8284;
- controlador de barramento 8288;
- controlador de interrupção 8259A;
- interface de 48 linhas paralelas 2 x 8255-A5;
- contador / temporizador 8253 ou 8254;
- interface de comunicação serial 8251A;
- adaptadores RS-232C para interface serial;

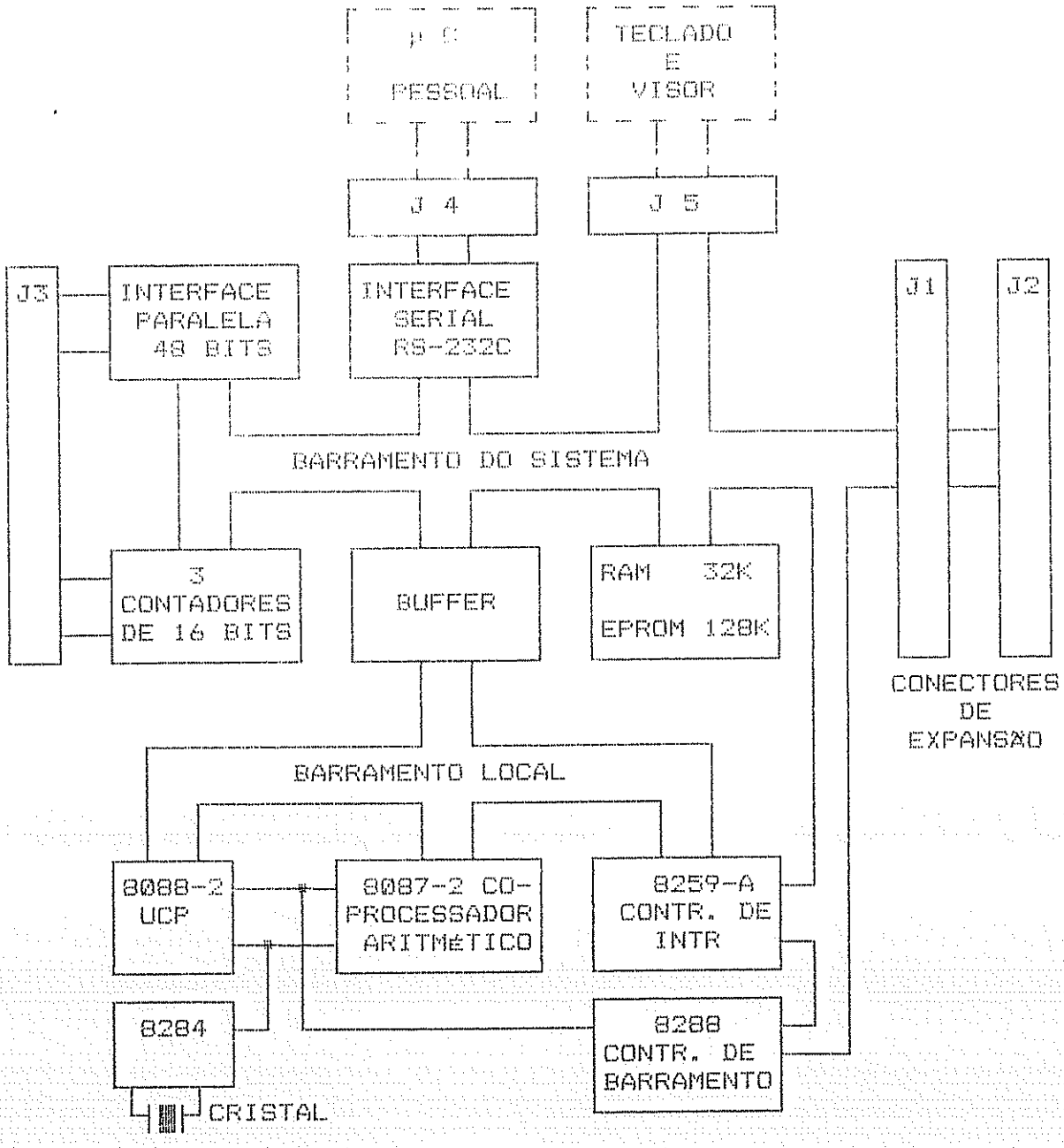


Figura 2.1 - Diagrama de blocos do KD-16

- EPROM (128 Kbytes) 2 x 27512;
- memórias RAM (32 Kbytes) 4 x 6264;
- 2 conectores de expansão (barramento compatível IBM-PC)

A escolha do microprocessador 8088-2 se deve a sua grande disseminação no meio acadêmico e industrial, além do enorme suporte de software para dar apoio à programação e depuração.

O fato de termos projetado o barramento do KD-16 no padrão de um microcomputador pessoal compatível com o IBM-PC, favorece o aproveitamento de algumas das placas comerciais disponíveis para o microcomputador pessoal, também para o KD-16.

No barramento local, mostrado no diagrama de blocos da figura 2.1, encontramos o microprocessador 8088-2, o coprocessador 8087-2 e o controlador programável de interrupções 8259A. Isolado desse primeiro grupo por meio dos buffers de endereço e dados, estão os outros periféricos internos à placa, as memórias EPROM e RAM, e os conectores de expansão.

O microprocessador está sendo utilizado em seu modo máximo, permitindo o uso do 8288 como controlador de barramento. Este circuito integrado recebe e decodifica as linhas de status "S0/", "S1/" e "S2/" do microprocessador, gerando as linhas de controle do barramento do sistema. Fornece também, sinais avançados de escrita às memórias e periféricos, possibilitando que maiores freqüências de relógio sejam usadas, sem a necessidade de geração de ciclos de espera. (OBS: "A/" significa, o sinal "A/" é ativo em "0").

Para a geração do relógio com tempos de transição e razão marca/espaço ideais ao 8088-2, usamos o 8284, que pode operar com cristal de até 24MHz.

Com as duas opções de memória RAM oferecidas (6116 e 6264), pode-se instalar desde 2K bytes até 32K bytes. Optamos por memórias RAM estáticas ao invés de memórias dinâmicas, porque simplificam os circuitos auxiliares, diminuindo o número de componentes na placa. Quanto às EPROMs, o kit comporta de 2K a 128K bytes em dois soquetes. Pode-se usar desde a 2716 (2K) até a 27512 (64K). A figura 2.2, apresenta, em blocos, as memórias EPROMs, RAMs e seus respectivos decodificadores.

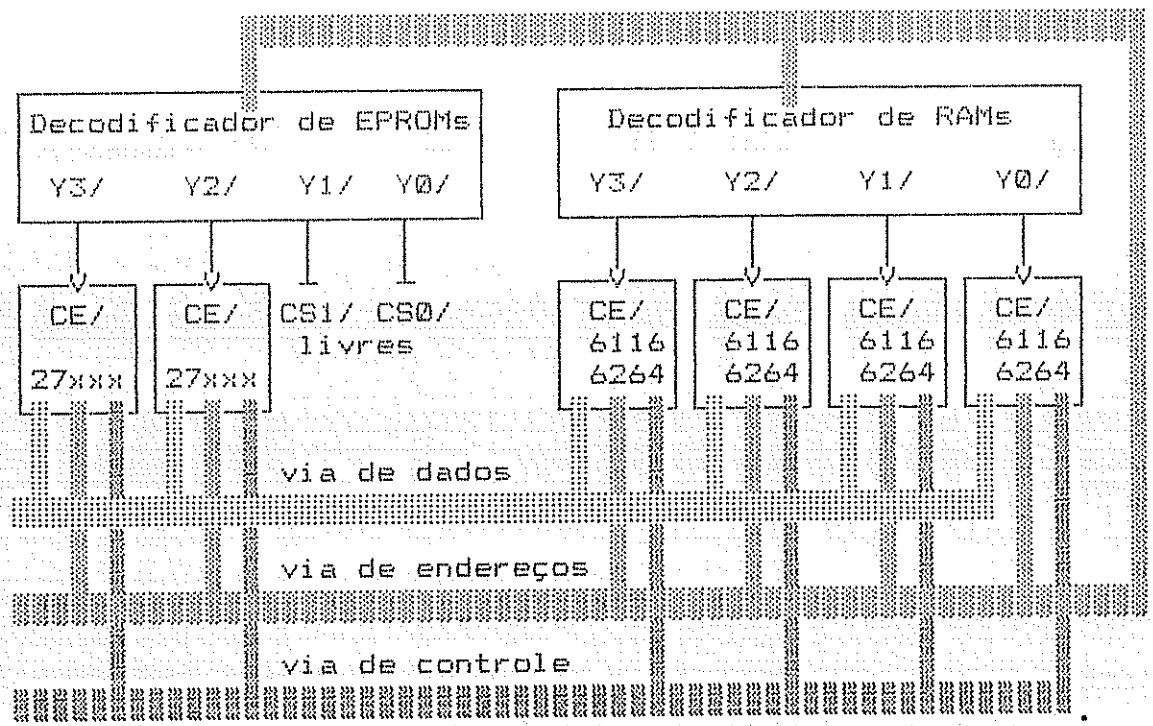


Figura 2.2 - Memória EPROM e RAM e Decodificadores.

A placa do KD-16 possui os seguintes periféricos:

- Duas interfaces paralelas 8255-A5, oferecendo 48 linhas bidirecionais programáveis, com três diferentes modos de operação e diversas configurações entrada e saída;
- Um contador/temporizador 8253-5, que possui três contadores programáveis de 16 bits, com possibilidade de operação em seis modos de contagem;
- Uma interface de comunicação serial 8251-A com seus adaptadores de entrada e saída RS-232C;
- Um controlador de interrupções 8259A, com oito linhas de entrada para requisição de interrupções, seis das quais disponíveis no barramento de expansão;
- Um controlador de teclado e visor 8279-5, que embora esteja num módulo externo à placa do KD-16, usa o mesmo barramento e o mesmo tipo de endereçamento que os outros periféricos internos.

Maiores detalhes dos circuitos integrados utilizados são encontrados nos catálogos de componentes da Intel, Texas e outras obras de referência (INTEL, 1980; INTEL, 1981a; INTEL, 1981b; INTEL, 1988; INTEL, 1981c; INTEL, 1979b; TEXAS, 1976; TEXAS, 1987; LIU, 1986).

A seleção desses periféricos foi feita diretamente através das linhas de endereço BA10 até BA15 do barramento, sem os decodificadores de endereçamento, visando diminuir assim ao máximo o número de componentes na placa. Por esse motivo, cuidados especiais devem ser tomados para acessar os periféricos.

Um contador binário de quatro bits produz as freqüências necessárias para quatro taxas de comunicação oferecidas, que são: 4800, 2400, 1200, 600 baud. Sua freqüência de entrada é obtida a partir de um divisor programável, que divide a freqüência do sinal do relógio principal, CLK, por "n". O ajuste do valor de "n" é feito com a ajuda da chave "S2", podendo variar de 0 a 15. Este circuito permite que se troque o cristal do 8284, mantendo disponíveis as diversas taxas de comunicação padronizadas.

Para a operação com até 8 MHz, que é a máxima freqüência que o 8088-2 pode operar, observamos o perfeito funcionamento de todos os circuitos integrados utilizados como periféricos internos. Entretanto, se formos acrescentar dispositivos mais lentos ao barramento padrão, devemos inserir no conector J1 (ver figura 2.1) da placa do KD-16, a placa geradora de ciclos de espera, descrita no item 2.7. Esta placa permite alongar os sinais de escrita e leitura por um período equivalente a um ciclo do sinal CLK adotado. Permite escolher também se este alongamento acontecerá somente para periféricos ou se também para as memórias.

A figura 2.3 apresenta o conjunto de periféricos ligados aos conectores J3, J4 e J5.

O conector "J5" de 20 pinos é reservado para a ligação do módulo de teclado e visor à placa do KD-16. Esse módulo, descrito no item 3.2, transforma o KD-16 em um sistema independente do terminal de vídeo ou pode servir como interface de uso geral para entrada e saída de dados.

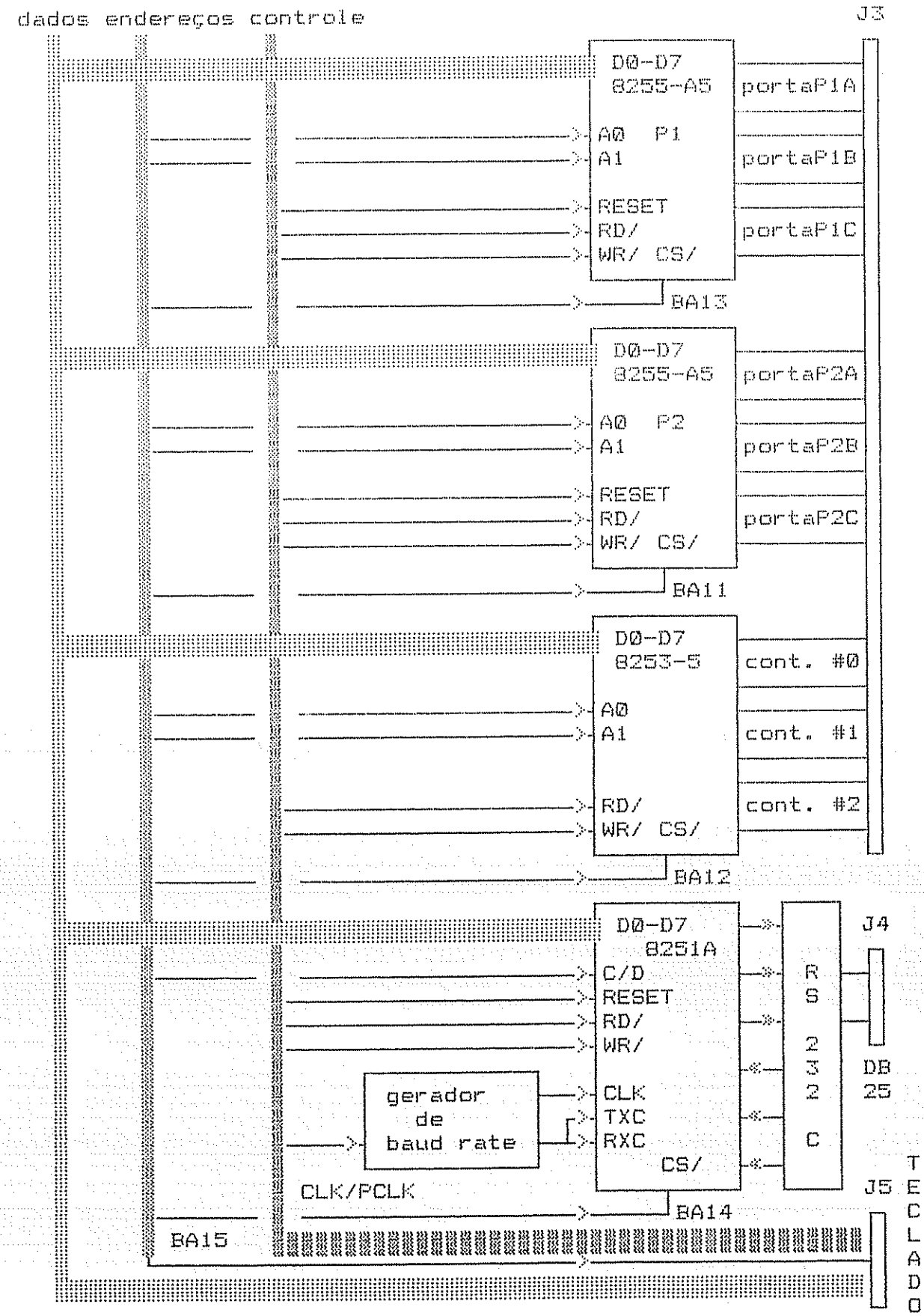


Figura 2.3 - Periféricos ligados aos conectores J3, J4 e J5.

Quando a placa do KD-16 contiver todos os circuitos integrados previstos, deve ser alimentada por uma fonte com as características apresentadas na tabela 2.1. Caso seja inserida alguma placa nos conectores do barramento de expansão, J1 ou J2, deve-se reforçar as exigências de corrente em relação às apresentadas na tabela 2.1.

O consumo de corrente da fonte +5V foi determinado pela soma das correntes máximas nos pinos VCC de cada circuito integrado, conforme especificação do fabricante, podendo tipicamente possuir valores menores.

Tensão	Tolelância	Corrente
+5V	±5%	3,5A
+12V	±5%	0,3A
-5V	±10%	0,3A
-12V	±10%	0,3A

Tabela 2.1 - Características da fonte necessária ao KD-16.

O circuito completo do KD-16 é apresentado no Apendice A (figuras AP.1, AP.2 e AP.3).

Para facilitar a localização dos componentes, a figura AP.4 (Apêndice A) apresenta a distribuição destes na placa do KD-16, vista pelo lado dos componentes. Podemos notar que existem diversos "jumpers" para selecionar opções de memória, velocidade de transmissão e uso de interrupções, além de 7 conectores. Estes "jumpers" e conectores dão maior flexibilidade e opções de uso ao Kit e serão descritos a seguir.

2.3 - Mapeamento e seleção de memórias

Até quatro circuitos integrados de memórias RAM podem ser inseridas nos soquetes denominados U9, U10, U11 e U12 (figuras AP.2 e AP.4, Apendice A). A tabela 2.2 apresenta os tipos de memórias RAM que podem ser utilizadas, o número de bytes disponíveis e o posicionamento necessário dos "jumpers" JP3, JP4 e JP5.

Seleção de memória RAM (6116 / 6264)				
Tipo	JP 3	JP 4	JP 5	Nº de bytes disponíveis
6116 6264	G-H G-F	J-I J-K	M-L M-N	4 x 2 K ---> 8K 4 x 8 K ---> 32K

Tabela 2.2 - Seleção do tipo de memória RAM

Até dois circuitos integrados de memória EPROM podem ser inseridos nos soquetes denominados U7 e U8 (figuras AP.2 e AP.4, Apendice A). A tabela 2.3 apresenta os tipos de memórias EPROM utilizáveis, o número se bytes disponíveis e o posicionamento necessário dos "jumpers" JP6, JP7, JP12 e JP13.

Seleção de memória EPROM (27xxx)					
Tipo	JP 6	JP 7	JP 12	JP 13	Nº de bytes
2716	P-O	S-R	F3-F1	F6-F4	2 X 2 K ---> 4 K
2732	P-Q	S-R	F3-F1	F6-F4	2 x 4 K ---> 8 K
2764	P-Q	S-R	F3-F1	F6-F4	2 x 8 K ---> 16 K
27128	P-Q	S-T	F3-F1	F6-F4	2 x 16 K ---> 32 K
27256	P-Q	S-T	F3-F2	F6-F4	2 x 32 K ---> 64 K
27512	P-Q	S-T	F3-F2	F6-F5	2 x 64 K ---> 128 K

Tabela 2.3 - Seleção do tipo de memória EPROM

Tanto as memórias RAM como as EPROM devem possuir um tempo de acesso inferior a 214ns, de modo a poderem trabalhar sem problemas, quando o KD-16 estiver operando a 8 MHz. Este tempo de acesso relativamente grande foi possível com a utilização dos sinais avançados **AIOWRC/** (Advanced I/O Write Command) e **AMWC/** (Advanced Memory Write Command) do controlador de barramento 8288.

Os mapas de memória estão representados nas figuras 2.4 e 2.5. Como podemos ver no esquema da figura AP.2, os bits de endereçamento não foram completamente usados na decodificação das memórias. Desse modo, existem múltiplos endereços para uma mesma posição de memória.

As memórias RAM ocupam toda a primeira metade do endereçamento possível, de 00000H até 7FFFFH (0 a 512K). Já as EPROMs ficam restritas entre E0000H e FFFFFH (896K a 1024K).

Devemos observar nas figuras 2.4 e 2.5d, que uma 27256 aparecerá duas vezes consecutivas dentro da faixa de endereçamento do soquete de U7. Repetição semelhante ocorre com os outros tipos de EPROMs, por exemplo: uma 2764 aparecerá oito vezes dentro da mesma faixa (ver figura 2.5f).

Caso seja necessário alguma expansão de memória, temos 384K livres na faixa compreendida entre 512K e 896K, que podem ser decodificados e utilizados a partir dos conectores do barramento J1 e J2.

Dois blocos de 64K consecutivos, iniciados em 768K e 832K, já se encontram decodificados, sendo habilitados pelos sinais CS0/ e CS1/ disponíveis no conector J6. Estes sinais

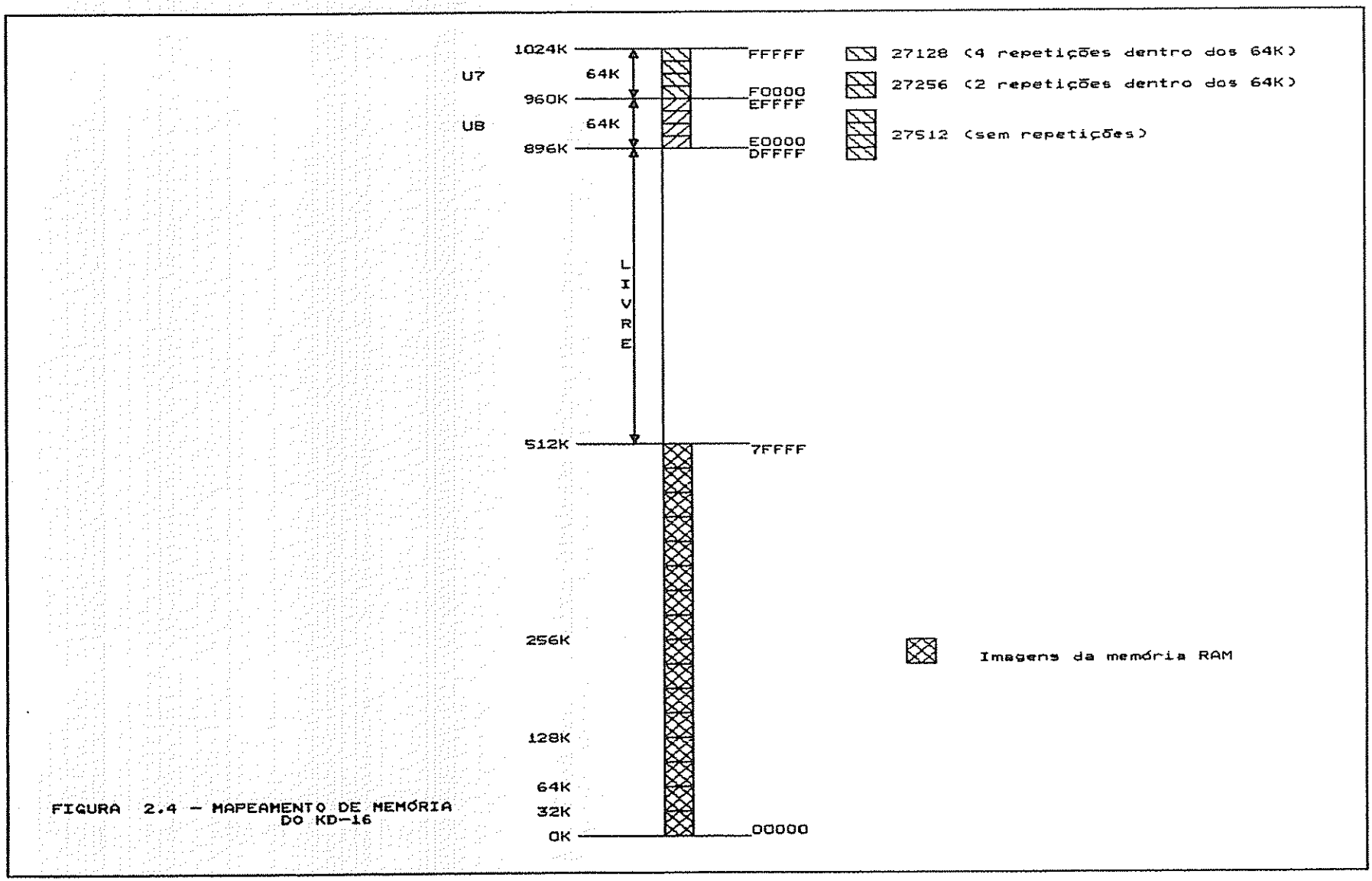


FIGURA 2.4 – MAPEAMENTO DE MEMÓRIA DO KD-16

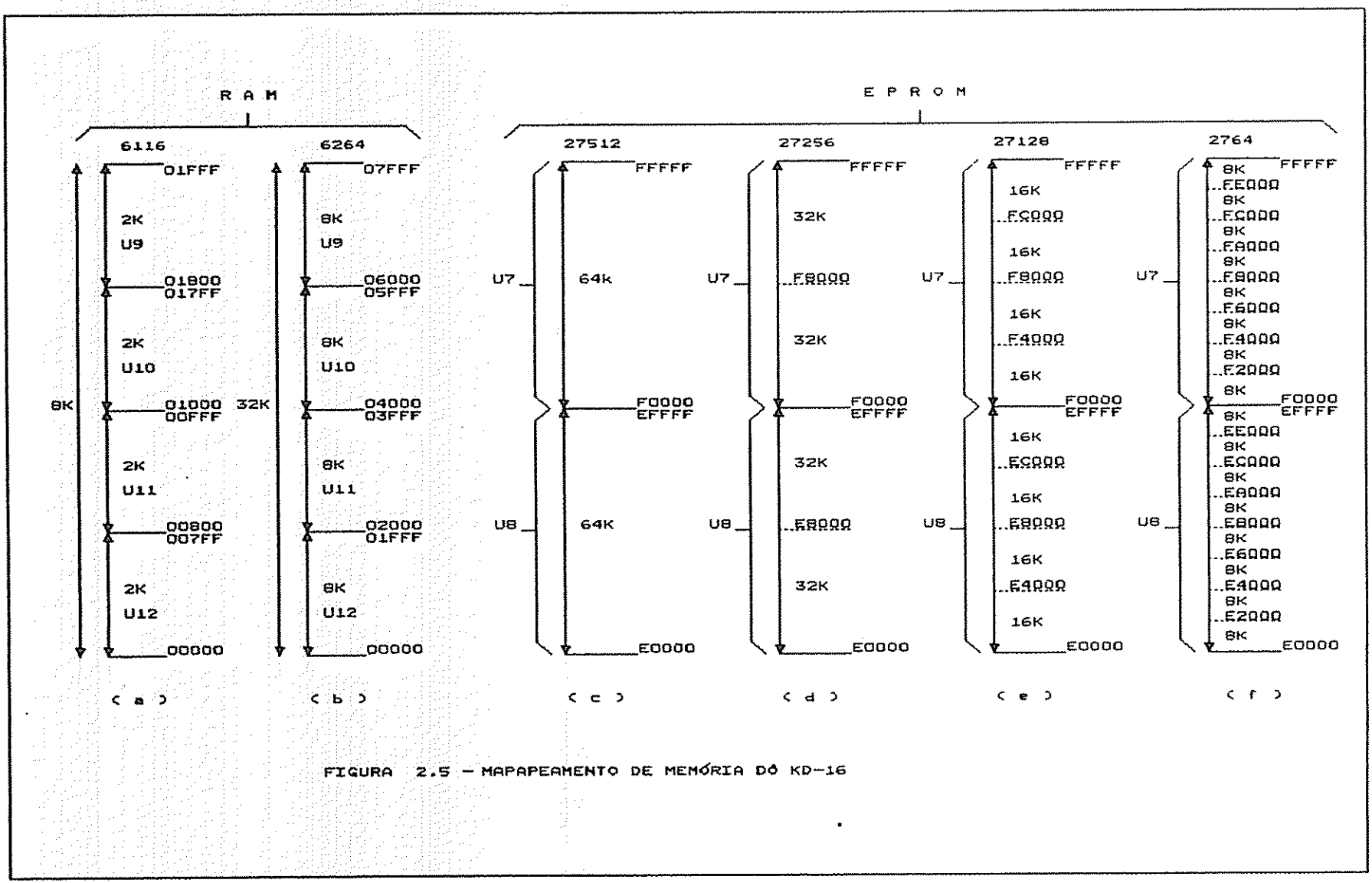


FIGURA 2.5 – MAPAPEAMENTO DE MEMÓRIA DO KD-16

pertencem ao decodificador 74LS139 (U6A) responsável pela decodificação das memórias EPROM.

2.4 - Seleção de periféricos

Para a seleção de periféricos, o KD-16 não possui decodificadores. Os bits da via de endereços de BA15 até BA10, são usados numa ligação direta com os pinos de seleção de cada integrado interno à placa (ver figuras 2.3, AP.1 e AP.3).

Esta implementação implica na possibilidade de habilitarmos mais que um periférico simultaneamente, caso haja um endereçamento inadequado. Além disso, a cada periférico caberá uma faixa de endereços, contendo repetidas imagens do endereçamento básico.

Os motivos que levaram a adoção desta implementação foram a economia de componentes e a possibilidade de ganharmos mais 40ns, aproximadamente, para o acesso aos circuitos integrados periféricos.

A tabela 2.4 mostra os endereços que **devem** ser utilizados para a seleção de cada periférico.

Caso seja necessário conectar alguma placa de expansão ao barramento do KD-16, deverão ser utilizados endereços livres entre FE00H e FFFFH (512 endereços). Evita-se assim superposições com o endereçamento dos periféricos internos.

ENDEREÇO	BA9	PERIFÉRICO
7C00H 7C01H	0	DADO 8279-5 COMANDO/STATUS
BC00H BC01H	0	DADO 8251A COMANDO/STATUS
DC00H DC01H DC02H DC03H	0	P. A 8255-A5 P. B P. C P1 COMANDO
EC00H EC01H EC02H EC03H	0	CONT.0 8253 CONT.1 CONT.2 MODO DE FUNC.
F400H F401H F402H F403H	0	P. A 8255-A5 P. B P. C P2 COMANDO
F800H F801H F802H F803H	0	8259A
FE00H : FFFFH	1	PLACAS NO BARRAMENTO DE EXPANSÃO

Tabela 2.4 - Endereços dos periféricos do KD-16.

OBSERVAÇÕES:

a) Os endereços fora das faixas indicadas na tabela não deverão ser utilizados, pois podem resultar numa dupla habilitação de periféricos.

b) BA9 = "0" -> periférico interno à placa do KD-16.
BA9 = "1" -> periférico no barramento de expansão do KD-16.

c) A tabela não mostra as faixas de redundância presentes no endereçamento dos periféricos.

2.5 Interface serial RS-232 C

Caso o usuário do KD-16 disponha de um computador pessoal ou de um terminal com interface de comunicação serial no padrão RS-232C, poderá utilizá-lo para controlar as operações do kit, quando este estiver sob o comando do programa monitor série decrito no item 2.7.

O conector "J4" (ver figuras AP.3 e AP.4) é usado para dar acesso à interface serial do KD-16. O circuito permite a seleção de uma entre quatro velocidades de comunicação: 4800, 2400, 1200 e 600 baud.

O circuito integrado responsável pelo controle da interface serial é o USART (Universal Synchronous/Asynchronous Receiver/Transmitter) 8251A, da INTEL (INTEL, 1980; INTEL, 1981c).

O USART é constituido de dois blocos de circuitos funcionalmente independentes, um para a transmissão e outro para recepção, o que torna possível a operação simultânea desses circuitos.

O microprocessador tem acesso a quatro registros internos do USART através da via de dados:

- o registro de comando, onde podemos escrever uma diretiva de controle para o 8251A;
- o registro de status, de onde podemos ler informações sobre os estados internos do USART;
- o registro ou buffer de transmissão, onde são escritos os bytes a serem serializados durante a transmissão;

- o registro ou buffer de recepção, de onde são lidos os bytes recebidos serialmente e convertidos para o formato paralelo.

As tarefas de conversão série-paralelo, paralelo-série e sinalização são totalmente controladas pelo 8251A.

O padrão RS-232C possui alguns sinais que normalmente são utilizados para controle e sincronização de um MODEM (MODulator/DEModulator) conectado à interface serial (OSBORNE, 1983). Dois desses sinais são utilizados no KD-16: o sinal CTS/ (Clear To Send) e DSR/ (Data Set Ready).

O sinal CTS/ é utilizado pelo computador pessoal para controlar o envio de dados pelo KD-16. Sempre que o computador pessoal colocar este sinal em nível alto, estará avisando ao KD-16 que ele não está pronto para receber novos dados. A habilitação permanente do envio de dados é conseguida interligando os pontos D-E pelo "jumper" JP2.

O sinal DSR/ é utilizado para monitorar a linha RXD/ (Receiver Data) pelo programa monitor série. Se o USART estiver sendo controlado por um outro programa qualquer, poderemos desconectar estas duas linhas desligando os pontos A-B, retirando o "jumper" JP1.

Os sinais que compõem o padrão RS-232C possuem níveis de tensão diferentes do padrão "TTL" (Transistor Transistor Logic), por isso colocamos adaptadores em cada linha de entrada e saída do conector "J4".

A obtenção das quatro velocidades de comunicação citadas, pode ser melhor entendida pelo diagrama de blocos mostrado na figura 2.6.

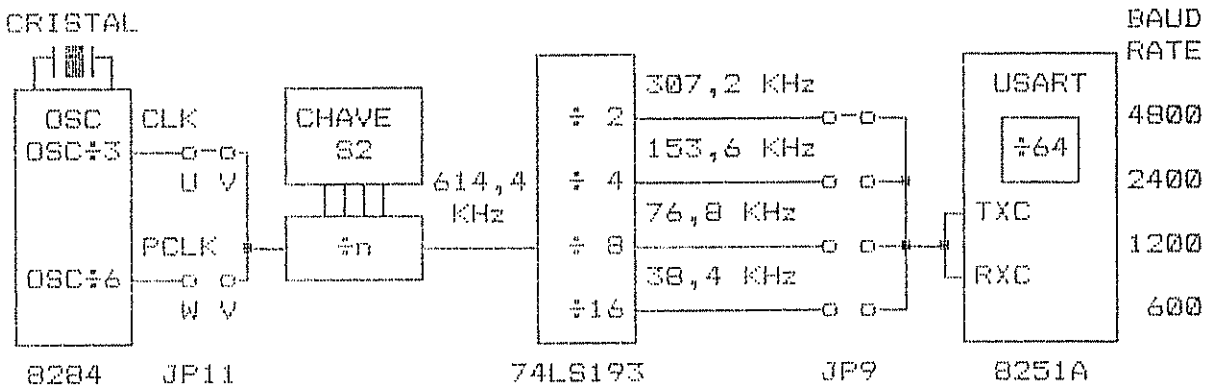


Figura 2.6 - Geração e seleção de "baud-rate" da porta serial.

Do lado esquerdo da figura 2.6, temos o gerador de relógio 8284, que produz o sinal "OSC" na freqüência do cristal de quartzo utilizado. O sinal "OSC" é dividido por 3 e por 6 dentro do 8284, gerando os sinais "CLK" e "PCLK", respectivamente. O "jumper" "JP11" seleciona qual desses sinais será dividido por "n", resultando numa freqüência a mais próxima possível dos 614,4 KHz necessários na entrada do contador binário 74LS193 (ver tabela 2.5). Para diferentes freqüências do sinal "OSC", teremos diferentes valores de "n" ajustados no divisor programável de forma a manter a freqüência de 614,4 KHz na entrada do contador binário.

JP 11	V-U OSC÷3	Seleciona freqüência entrada para o divisor por "n"
	V-W OSC÷6	

Tabela 2.5 - Opções para o "jumper" JP 11

Nas saídas QA, QB, QC e QD, teremos esta freqüência dividida por 2, 4, 8 e 16, respectivamente, correspondendo às taxas de 4800 a 600 baud. O "jumper" "JP9" tem a função de selecionar qual dessas saídas será conectada as entradas "RXC/" (Receiver Clock) e "TXC/" (Transmitter Clock) do 8251A (ver tabela 2.6). A freqüência selecionada será dividida por 64 internamente no USART, resultando no "baud-rate" escolhido.

JP 9	XA-YA	÷2 4800 baud	Seleção de uma das 4 velocidades de comunicação da interface serial
	XB-YB	÷4 2400 baud	
	XC-YC	÷8 1200 baud	
	XD-YD	÷16 600 baud	

Tabela 2.6 - Opções para o "jumper" JP 9

O sinal de 307,2 KHz da saída QA, além de ser utilizado pela USART para gerar a taxa de comunicação de 4800 baud, é também fornecido a outros pontos do KD-16, sob o nome de "PCK". O sinal PCK aparece nos conectores J3 e J6, como sinal de uso geral, e em J5, para controlar a velocidade de varredura dos multiplexadores do módulo de teclado e visor (ver item 3.2).

Para evitar erros de enquadramento ("Framing error") a freqüência do baud-rate gerado poderá ter um desvio máximo de 4% em relação ao valor nominal. A fórmula abaixo ajuda na determinação da faixa permitida para "n". Deve-se escolher o valor de "n" como o valor inteiro, compreendido entre $\sigma \pm 4\%$, que minimiza os erros na taxa de transmissão almejada.

A tabela 2.7 mostra como ajustar os 4 interruptores da chave S2, de modo a obter os diversos valores de "n".

$$\sigma = \frac{\text{OSC (Hz)}}{614400 \cdot X}$$

X=3 se JP11 liga V-U

X=6 se JP11 liga V-W

n	CHAVE S2 D	C	B	A	
1	lig	lig	lig	des	Para cada valor de "n" calculado, deve-se ajustar os interruptores da chave S2, conforme a indicação desta tabela
2	lig	lig	des	lig	
3	lig	lig	des	des	
4	lig	des	lig	lig	
5	lig	des	lig	des	
6	lig	des	des	lig	
7	lig	des	des	des	
8	des	lig	lig	lig	
9	des	lig	lig	des	<lig> interruptor ligado
10	des	lig	des	lig	
11	des	lig	des	des	
12	des	des	lig	lig	<des> interruptor desligado
13	des	des	lig	des	
14	des	des	des	lig	
15	des	des	des	des	

Tabela 2.7 - Ajuste do valor de "n" na chave S2

A tabela 2.8, apresenta três exemplos de como cristais de diferentes freqüências podem gerar, de modo aproximado, as taxas de transmissão e recepção necessárias.

OSC	JP 11 U-V	saída do div. p/n	JP 9 XA-YA	baud rate	erro %
24,000	8,000	615,4 (n=13)	307,69	4800	+0,16
18,432	6,144	614,4 (n=10)	307,20	4800	0,00
16,000	5,333	592,6 (n=9)	296,30	4630	-3,55
M H z		K H z			

Tabela 2.8 - Exemplo de como é possível obter 4800 baud a partir de cristais de freqüências diferentes.

O endereçamento dos buffers e registros do 8251A devem ser feitos como indicado na tabela 2.9.

OPERAÇÃO	ENDEREÇOS	
	BC01H	BC00H
LER	STATUS	BUFFER DE RECEPÇÃO
ESCREVER	COMANDOS	BUFFER DE TRANSMISSÃO

Tabela 2.9 - Endereçamento dos registros do USART 8251A

2.6 - Conectores do KD-16

No KD-16, encontramos sete conectores:

a) - J0 é a entrada para a fonte de alimentação (ver figura AP.7). O barramento escolhido requer quatro tensões de alimentação diferentes: +5V, -5V, +12V, -12V. A tensão de -5V não é usada internamente na placa do KD-16, e ±12V são usados apenas na interface série RS-232C;

b) - J1 e J2 são conectores parcialmente compatíveis com o barramento do IBM-PC (ver figura AP.5). A diferença existente é a ausência dos sinais relativos à requisição e reconhecimento do "DMA" (Direct Memory Acces). Estes conectores aceitam a placa de interface paralela e temporização descrita, à frente, no item 3.3;

c) - Em J3 temos as linhas bidirecionais das duas interfaces paralelas e as linhas dos contadores programáveis internos ao KD-16. Estão também disponíveis +5V e o sinal PCK de 307,2 KHz (ver figura AP.6);

d) - J4 é um conector padrão da interface RS232C de 25 linhas (tipo DB25), para comunicação serial entre o KD-16 e o microcomputador pessoal. Além das linhas de entrada e saída de dados, encontramos as linhas de controle DTR/, RTS/, DSR/ e CTS/ (ver figura AP.7).

e) - J5 é o conector para o Módulo do Teclado e Visor (ver figura AP.7). O consumo de corrente desse módulo já foi considerado na determinação das correntes listadas na tabela 2.1;

f) - J6 é um conector, para uso geral. Possui os seguintes sinais: +5V, GND, entradas de requisição de interrupção do 8259A, reset, CLK, PCK, RDY, as saídas livres CS0/ e CS1/ do decodificador de EPROMs e o sinal TEST/ do 8088 (ver figura AP.7).

2.7 - Placa geradora de estados de espera

Sempre que o microprocessador comandar um ciclo de acesso ao barramento do microcomputador e este possuir memórias ou periféricos que não consigam transferir informações na velocidade que o microprocessador impõe, será necessário um circuito que providencie a inserção de estados de espera ("WAIT"), para provocar o alongamento do comando de escrita ou leitura em atividade. O número necessário de estados de espera vai depender do tempo de acesso de cada dispositivo e dos diversos atrasos que ocorrem em buffers e decodificadores.

Basicamente, existem dois métodos para a implementação do circuito de requisição de estados de espera:

1)- A implementação mais clássica é ter a geração automática dos estados de espera. Quando um dispositivo selecionado recebe um comando (RD/, WR/, INTA/) e já teve tempo suficiente para completá-lo, ele sinaliza ao circuito gerador de es-

tados de espera para que habilite o microprocessador a termi-nar o estado de espera corrente e dê continuidade ao ciclo de máquina, finalizando-o.

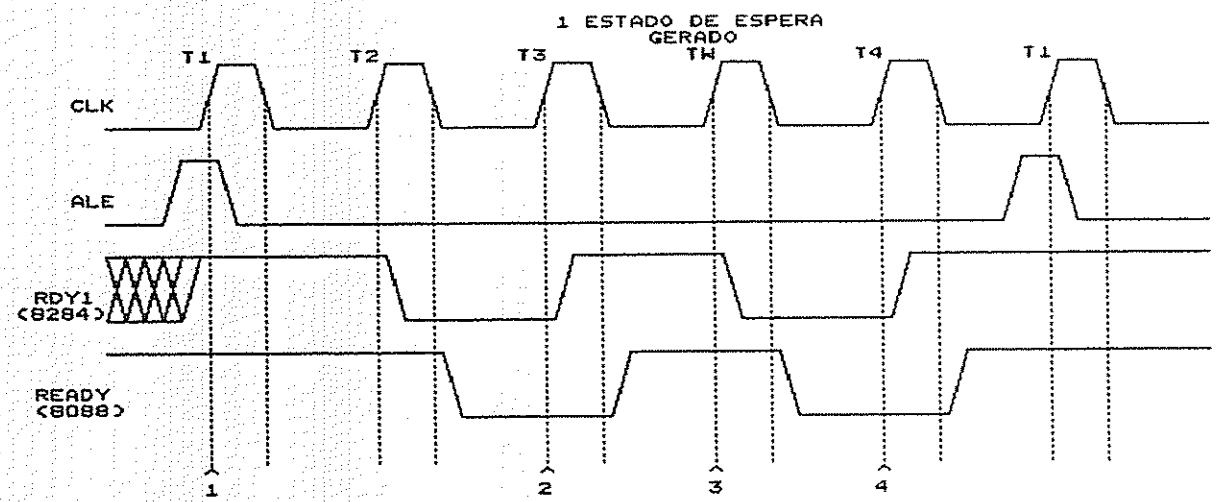
2)- Outro modo, é não ter a geração automática de estados de espera. Somente os dispositivos que não conseguem atender os requisitos de velocidade do sistema precisam provocar a in-serção dos estados de espera.

2.7.1 - Descrição do gerador de estados de espera

Nosso gerador de estados de espera, é uma adaptação da primeira modalidade de implementação. A diferença, é que ao invés de esperarmos que os dispositivos lentos enviem a ordem para terminar o estado de espera, esta é gerada automaticamen-te, após o período de um ciclo do sinal de relógio (CLK). Des-se modo, os sinais de controle do barramento MRD/ (Memory ReaD), IORD/ (I/O ReaD), MWR/ (Memory WRite) e IORW/ (I/O WRite), têm sua duração aumentada por um tempo igual a um pe-ríodo do sinal CLK utilizado.

Nas figuras AP.8 e 2.7 temos o circuito do gerador de ciclos de espera e seu diagrama de tempos, respectivamente.

No microprocessador 8088, os estados de espera são requisitados sincronamente, com a ajuda da lógica implementada internamente no gerador de relógio 8284. No KD-16 utilizamos a entrada "RDY1" do 8284 para iniciar o pedido de espera. Esta entrada está disponível no pino "A10" dos conectores "J1" e "J2".



1- ALE força RDY1 para "1"

2- Na subida de T3 o 8088 amostra a entrada READY, gerando TW

3- Na subida de TW o 8088 amostra a entrada READY, gerando T4

4- Nao interfere pedindo WAIT porque o estado atual não é T3

FIGURA 2.7 - DIAGRAMA DE TEMPOS DA PLACA GERADORA DE ESTADOS DE ESPERA

Um ciclo de máquina do 8088, é formado, normalmente, por quatro estados denominados: "T1", "T2", "T3", "T4". Para que, após o estado "T3", tenhamos um estado de espera, ao invés de "T4", é necessário que o sinal "READY" do 8088 assuma nível "0" até no máximo 8ns após a borda de descida de "T2", assim permanecendo durante todo o período de nível "0" do sinal de relógio. Para sair do estado de espera e fazer com que o próximo estado seja "T4", o sinal "READY" terá que assumir nível "1", 70ns antes da transição positiva de "Tw" (ver figura 2.7).

No KD-16, além da utilização do gerador de estados de espera para obtenção do alargamento dos sinais de controle, utilizamos também os sinais avançados **AIOWRC/** (Advanced I/O Write Command) e **AMWC/** (Advanced Memory Write Command), fornecidos pelo controlador de barramento 8288. Estes sinais são ativados um ciclo de relógio mais cedo que seus equivalentes não avançados, propiciando sinais de escrita mais largos, com duração igual à dos sinais de leitura. Uma vantagem da utilização destes sinais, é que eles permitem a utilização de dispositivos com tempo de acesso de até 216ns sem a necessidade de conectarmos a placa geradora de ciclos de espera ao KD-16.

A utilização destes sinais implica, entretanto, na impossibilidade da utilização de memórias dinâmicas e de dispositivos de E/S com captura de dados na borda de descida, pois o comando AMWC/ torna-se ativo antes que tenhamos dados válidos durante um ciclo de escrita.

A figura 2.8 mostra o efeito da placa geradora de estados de espera, quando inserida no conector "J1" do Kit KD-16, operando com um relógio de 8MHz.

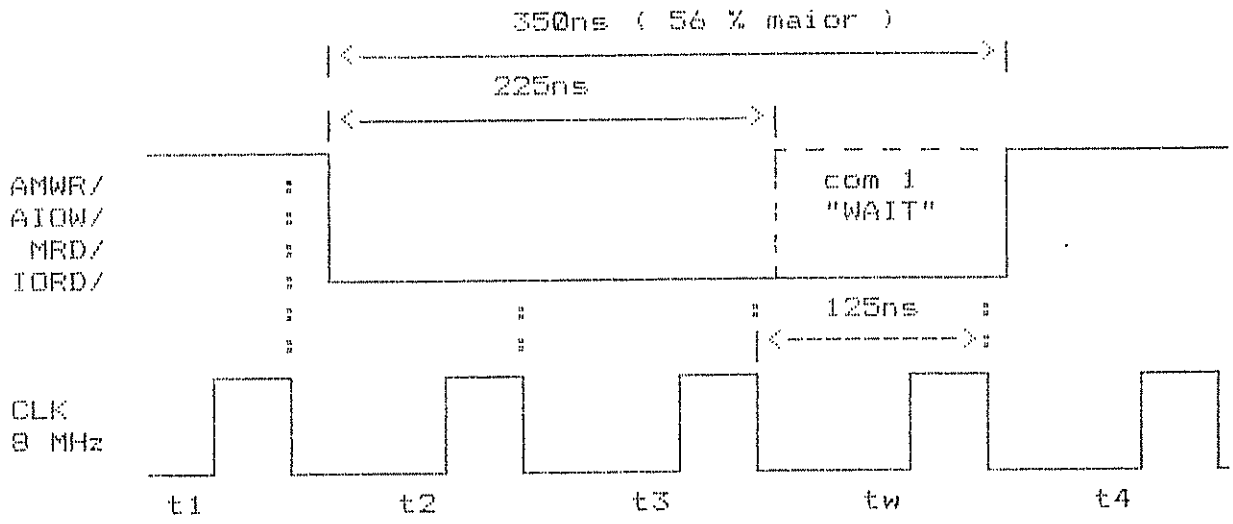


Figura 2.8 - Efeito do Gerador de Ciclos de Espera à 8 MHz.

A obrigatoriedade da utilização do conector "J1" para a conexão dessa placa, é porque somente ele possui o sinal de status "S2/". Com este sinal, consegue-se diferenciar desde o início do presente ciclo de máquina, se a operação será nos periféricos ou nas memórias. Assim, podemos escolher através do "jumper" "JP1" (ver figura AP.8), para quais dispositivos teremos a geração dos estados de espera.

Quando "JP1" estiver interrompido, teremos estados de espera tanto para dispositivos de E/S como para as memórias. Esta opção será raramente usada, pois as memórias utilizadas, tipicamente, possuem tempo de acesso entre 100ns e 200ns, não precisando portanto do alargamento dos sinais de comando.

Quanto aos periféricos, temos uma situação bem mais crítica. Com a ausência do decodificador de periféricos no KD-16, ganhamos mais 38ns para o tempo de acesso a estes dispositivos, totalizando 254ns (216ns + 38ns = 254ns). Entretanto, como seu tempo típico de acesso é da ordem de 300ns (15% a mais), precisaremos de pelo menos um estado de espera. Este estado de espera inserido, possibilita o uso de periféricos com tempo de acesso de até 379ns (254ns + 125ns = 379ns).

Optamos por não incluir o gerador de ciclos de espera na placa do KD-16, pelo mesmo motivo da eliminação do decodificador de periféricos, ou seja, a diminuição do número total de circuitos integrados na placa. Além disso, a intenção inicial era de operar com uma freqüência de relógio de 6,144MHz. A esta freqüência, os dispositivos periféricos dispõem de 329ns de tempo de acesso, não necessitando portanto do circuito gerador de estados de espera.

2.8 - Programa monitor série

O programa monitor série usado no KD-16 é uma adaptação do monitor para teclado do "SDK-86 System Design Kit" da INTEL (INTEL, 1978a). Neste item, descreveremos os aspectos gerais deste programa e seus comandos.

Quando, no soquete "U7" do KD-16, instalamos uma memória EPROM contendo o programa monitor série, e inicializamos o kit com a chave "RESET", passamos a operar o KD-16 a partir de um terminal de vídeo, usando o padrão de comunicação RS-232C. Este programa monitor ocupa um espaço de 4K bytes a partir do endereço "FF000H".

Após um reset, o programa monitor inicializa os registros do 8088, conforme mostrado abaixo:

CS=0000H	DS=0000H	SS=0000H	IP=0000H	FL=0000H	SP=0100H

É reservado, também, um bloco de 256 bytes em RAM (0000H a 00FFH) para uso do monitor (ver figura 2.9). Este bloco é usado para: a) armazenar os vetores de interrupção, b) como área de dados do monitor, e c) pilha ("Stack"). Após este bloco, temos a área de RAM livre para o usuário.

Ao se terminar a inicialização do kit, é enviado ao terminal de vídeo a mensagem "KD-16 MONITOR, V1.2", passando em seguida à condição de entrada de comandos. Esta condição é assinalada pelo caractere "." no início de uma linha, seguido do cursor piscante.

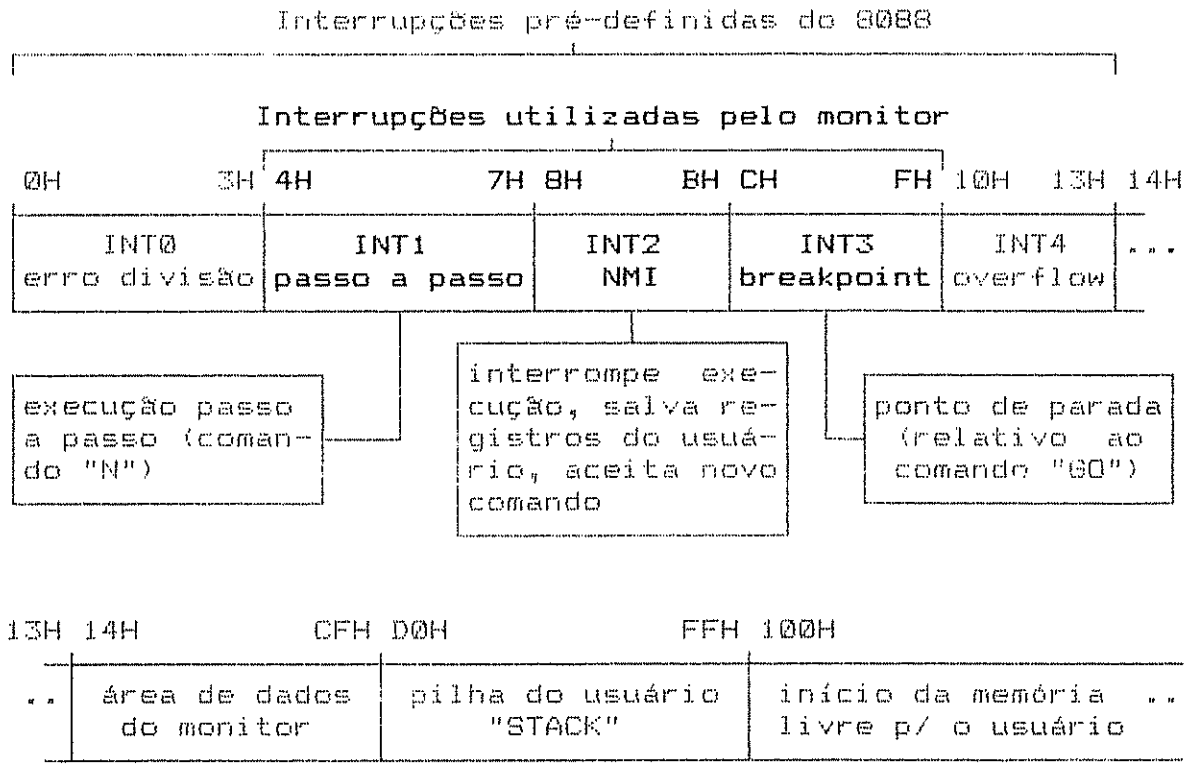


Figura 2.9 - Bloco reservado no início da memória RAM para uso do monitor série.

Em uma linha de comando, só é aceito um comando por vez. Cada comando é seguido por um certo número de parâmetros, que podem variar de 1 a 3, sendo delimitados por vírgulas. Qualquer número de espaços podem ser inseridos entre os parâmetros, se o usuário o desejar, não interferindo na interpretação do comando.

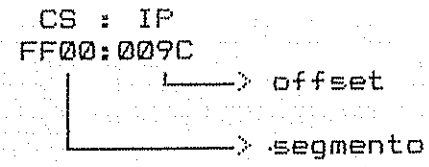
Para iniciar a execução de um comando deve-se terminar a linha pelo caractere "CR" (Carriage Return). Uma maneira, opcional, de iniciar a execução de alguns dos comandos ("S", "X", "I", "O", "N"), é acionarmos a tecla "," ao invés do "CR" (ver figura 2.21, página 68).

Qualquer comando pode ser interrompido, a qualquer momento, pelo caractere "CTRL C". Durante a entrada de parâmetros, caso sejam teclados caracteres inválidos ou o comando "CTRL C", será enviado à tela do terminal de vídeo um caractere "#", sinalizando o cancelamento do comando. O monitor série envia, em seguida, os caracteres de retorno de carro "CR" e alimentação de linha "LF" para o terminal de vídeo, e retorna à condição de espera de comandos.

Para qualquer comando ou parâmetro, só serão aceitos caracteres maiúsculos.

Existem dois tipos de parâmetros para os diversos comandos: a) números hexadecimais e b) abreviaturas dos nomes de registros do 8088 (ver tabela 2.10, página 59). Os números hexadecimais podem ter dois dígitos (bytes) ou quatro dígitos (palavras). Na entrada de bytes, se forem teclados mais que dois caracteres, somente os dois últimos serão aproveitados. Na entrada de palavras, somente os quatro últimos serão considerados.

Quando um parâmetro expressa um endereço, podemos ter duas formas: a completa e a resumida. Na forma completa entraremos com o segmento e o offset, como indicado no exemplo abaixo:



Na forma resumida, não entramos com valor do segmento, mas somente do offset. O valor utilizado para o segmento

será o do registro "CS", a menos que haja especificação contrária na descrição do comando.

2.8.1 - Comandos do monitor série

Descreveremos neste item os 10 comandos oferecidos pelo programa monitor série. Nesta descrição usaremos os seguintes símbolos:

[ ] Indica caractere(s) teclado(s)

[A] Indica que "A" é opcional

[A]* Indica uma ou mais ocorrências opcionais de "A"

<B> Indica que "B" é uma variável

OBSERVAÇÕES:

- Os comandos "S", "D", "I" e "O" (ver figura 2.21), realizam suas operações sobre bytes. Quando seguidos do caractere "W", atuam sobre palavras.
- Ao mostrar o conteúdo de um endereço no formato de palavra, o byte mais significativo da palavra é o conteúdo da memória apontada por "<endereço> + 1", enquanto o byte menos significativo expressa o conteúdo da memória apontada por "<endereço>".
- Ao modificarmos o conteúdo de uma palavra na memória, o byte mais significativo da palavra será armazenado na posição de memória apontada por "<endereço> + 1". O byte menos significativo, será armazenado na posição de memória apontada por "<endereço>".

- Um caractere "CR" teclado ao final de uma linha de comando sempre provoca a execução deste comando, e em seguida, seu término.
- Os parâmetros que especificam endereços, podem ser fornecidos na forma numérica pura (FD03:11BE), ou utilizando das abreviaturas dos nomes dos registros da CPU (SS:SP), ou formas mistas (DS:50 ou 0:IP).
- Os parâmetros que especificam endereços, podem ser fornecidos contendo operações aritméticas, a serem resolvidas pelo programa monitor (ES+10:2C).
- Um caractere "-" enviado pelo programa monitor após a impressão do conteúdo de uma memória ou registro indica que é esperada a entrada de um novo valor hexadecimal para esta memória ou registro. Se ao invés de um novo valor hexadecimal teclarmos uma "," ou "CR", o valor original não será alterado.

1) "S" VERIFICA/MODIFICA O CONTEÚDO DE POSIÇÕES DE MEMORIA:

S[W]<end.>,[[<novo dado>],]*<CR>

Após teclarmos o comando "S" e especificarmos o endereço da memória a ser examinada, devemos dar início à execução do comando com uma ",". Em resposta, será fornecido na tela o valor hexadecimal do byte endereçado, seguido do caractere "-" (ver figura 2.10). O caractere "-" indica que o programa monitor está a espera de um novo valor hexadecimal para

substituir o antigo, naquela posição de memória. Se teclarmos um novo valor hexadecimal seguido de uma "," ou "CR", teremos a atualização daquela posição de memória. Se o caractere teclado foi uma ",", além da atualização do conteúdo da memória, teremos o auto-incremento do endereço fornecido, permitindo assim, examinarmos e modificarmos bytes consecutivos na memória. Se o caractere teclado foi um "CR", teremos a atualização da memória e, em seguida, o término do comando. Caso não seja teclado nenhum valor hexadecimal, mas somente "," ou "CR", estaremos fazendo somente uma leitura do conteúdo da memória endereçada.

Será emitida mensagem de erro se houver a tentativa de alterar o valor de dados em EPROMs ou em endereços que não possuam memórias RAM instaladas.

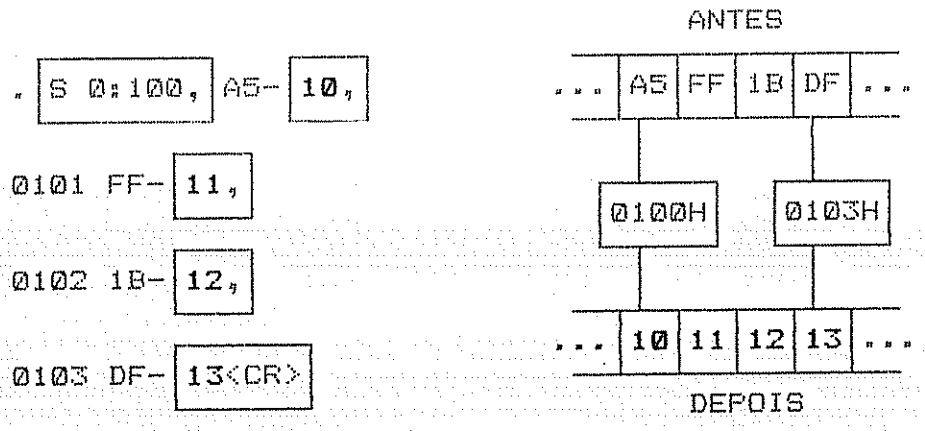


Figura 2.10 - Comando "S"

2) "X" EXAMINAR/MODIFICAR REGISTROS DO 8088:

```
X[<reg>][[<novo dado>],]*<CR>
```

O comando "X" permite que se examine e modifique o conteúdo dos 14 registros do microprocessador 8088. Estes registros são referenciados por suas abreviaturas, conforme a tabela 2.10.

NOME DO REGISTRO		ABREVIATURA
1º	acomulador	AX
2º	base	BX
3º	contador	CX
4º	dado	DX
5º	ponteiro de pilha	SP
6º	ponteiro de base	BP
7º	índice fonte	SI
8º	índice destino	DI
9º	segmento de código	CS
10º	segmento de dado	DS
11º	segmento de pilha	SS
12º	segmento extra	ES
13º	ponteiro de instruções	IP
14º	"flag"	FL

Tabela 2.10 - Abreviatura dos registros do 8088

Quando é necessário o exame do conteúdo de todos os registros, simultaneamente, basta teclarmos um "CR" depois do comando "X". Teremos a apresentação em tela dos 14 registros como mostrado na figura 2.11.

```
. X<CR>
 AX=01EF BX=0344 CX=0002 DX=89D3 SP=0010 BP=0010 SI=FFEA
 DI=1000 CS=FF00 DS=0000 SS=0010 ES=0000 IP=1EDF FL=105A
 .
```

Figura 2.11 - Comando "X" seguido de "CR".

Para se ter acesso individual a qualquer registro do 8088, devemos fornecer a abreviatura do nome do registro que se quer examinar (ver figura 2.12). O programa monitor apresentará o valor do conteúdo deste registro, habilitando o usuário a modificá-lo, se o desejar. O acionamento do caractere "," após o novo valor hexadecimal teclado provocará um avanço para o próximo registro na seqüência mostrada na tabela 2.10.

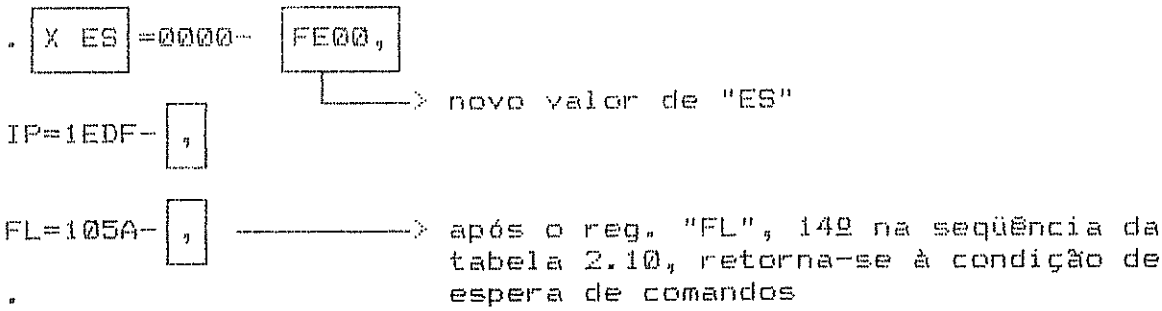


Figura 2.12 - Comando "X", edição do conteúdo de "ES" e verificação de "IP" e "FL"

3) "D" LISTA CONTEÚDO DA MEMORIA:

D[W]<end. inicial>[,<end. final>]<CR>

Este comando é usado para listar na tela do terminal de vídeo o conteúdo de um bloco de memória delimitado por "<end. inicial>" e "<end. final>" (ver figura 2.13).

Caso, em "<end. inicial>" não seja especificado o valor do segmento, será utilizado o valor do segmento "CS". O parâmetro "<end. final>" só poderá conter um offset, pois o valor do segmento será sempre relativo àquele especificado em "<end. inicial>".

O endereço inicial deverá ter um offset menor que o endereço final, ou causará mensagem de erro.

```
. D SP:D , 1F<CR>
000D E8 BA EE
0010 87 B0 FF EA BA 46 4D 8A FB 8B F7 74 07 FB 80 43
.
```

Figura 2.13 - Comando "D"

4) "M" MOVER BLOCO NA MEMORIA:

```
M<end. inicial>,<end. final>,<end. destino><CR>
```

Este comando é usado para mover blocos de dados na memória do KD-16 (ver figura 2.14). O bloco é delimitado por "<end. inicial>" e "<end. final>" (inclusive). Os bytes são movidos para posições consecutivas de memória, começando por "<end. destino>".

Assim como no comando "D", na especificação do endereço final, devemos fornecer somente o offset, pois o valor do segmento é o mesmo que o especificado no endereço inicial. Se não fornecermos o valor do segmento no 1º e 3º parâmetros, será utilizado o valor do segmento "CS".

```
. M FF00:0 , FFF , 10:0
.
```

Figura 2.14 - Deslocamento do programa monitor para a área livre da RAM.

5) "I" ENTRADA EM UMA PORTA PERIFÉRICA:

```
I[W]<end. porta>,[,]*<CR>
```

Este comando permite a leitura do valor presente em uma porta periférica, cujo endereço é "<end. porta>", e a impressão deste na tela do terminal de vídeo. A saída na tela acontecerá a cada acionamento da tecla "," (ver figura 2.15). O acionamento da tecla "CR" termina o comando.

```
. I FFFB,
00 ,
BB ,
27 <CR>
.
```

Figura 2.15 - Comando "I".

6) "O" SAIDA EM PORTA PERIFÉRICA:

```
O[W]<end. porta>,<dado>[,<dado>]*<CR>
```

Este comando permite escrever o valor da variável "<dado>" na porta de saída endereçada por "<end. porta>".

Caso seja necessário enviar mais que um valor à porta periférica de saída, deve-se usar uma vírgula ao invés do "CR" no final da linha de comando (ver figura 2.16).

```
. O FFFB,00,
- FF,
- 55<CR>
.
```

Figura 2.16 - Comando "O".

7) "G" EXECUTAR PROGRAMA :

```
G[<end. inicial>][,<ponto de parada>]<CR>
```

Este comando permite que se transfira o comando do 8088, do programa monitor para o programa do usuário.

Ao acionarmos a tecla "G" na linha de comando, será impresso na tela o atual conteúdo do registro "IP". Caso seja necessário alterá-lo, deve-se entrar com o endereço inicial. A transferência para o programa do usuário terá início com o acionamento da tecla "CR".

Após a entrada do endereço inicial, podemos fornecer ainda, um endereço de parada de execução (ver figura 2.17). A instrução neste endereço será substituida, temporariamente, pela interrupção por software "INT3", que ocupa um único byte. Quando, durante a execução do programa do usuário, for executada a interrução INT3, serão salvos os registros do 8088, voltando o controle para o programa monitor. Deste modo, o usuário poderá fazer verificações com os comandos do monitor e sondar o estado do seu programa no ponto escolhido.

Como, na especificação de pontos de parada existe uma substituição de instrução, não se deve usar este artifício em programas armazenados em memórias EPROM.

```
. G  1B3- 7E  10:0,2F<CR>
BR @0010:002F
.
```

Figura 2.17 - Execução de um programa em 10:0 com especificação de parada no endereço 10:2F.

8) "N" EXECUÇÃO PASSO A PASSO:

```
N[<end. inicial>],[[<end. inicial>],]*<CR>
```

Este comando é usado para executar uma única instrução do usuário. Após a execução dessa instrução, o controle do KD-16 retornará para o programa monitor, permitindo a avaliação da instrução executada (ver figura 2.18).

Ao acionarmos o comando passo a passo "N", teremos na tela a indicação do atual conteúdo do ponteiro de instruções. Pode-se alterar este valor e, em seguida, ordenar a execução de uma instrução pelo acionamento da tecla ",". A cada passo, pode-se utilizar dos diversos comandos do programa monitor para sondar a evolução da execução do programa.

```
. N  0000- 00  100,
0102- 8E  ,
0104- BA  ,
0107- B0  ,
0109- BA  <CR>
.
```

Figura 2.18 - Execução passo a passo de um programa no endereço 0:100H.

9) "R" LER ARQUIVO HEXADECIMAL:

R[<deslocamento>]<CR>

Este comando permite que se carregue a memória do KD-16 com o conteúdo de um arquivo hexadecimal. A variável "<deslocamento>", quando especificada, é somada aos endereços de carga fornecidos, permitindo um deslocamento do arquivo na memória (ver figura 2.19).

Serão emitidas mensagens de erro quando se tentar carregar um arquivo em memória inexistente ou em memórias EPROMs, e quando os bytes de verificação de erro gravados no arquivo detectarem alguma inconsistência nos dados recebidos.

```
. R 1FF<CR>
.
```

Figura 2.19 - Transferência de um arquivo para a memória com um deslocamento de 512 bytes.

10) "W" ARMAZENAMENTO DE DADOS NUM ARQUIVO HEXADECIMAL:

```
W<end. inicial>,<end. final>[,<end. de execução>]<CR>
```

Este comando permite que o conteúdo de um bloco de memória do KD-16 seja armazenado num arquivo hexadecimal (ver figura 2.20).

O fornecimento do endereço inicial e final do comando "W" segue as mesmas recomendações que os comandos "D" e "M" já descritos.

```
. W 100,1FF,120<CR>
.
```

Figura 2.20 - Armazenamento de 256 bytes delimitados por CS:100H e CS:1FFH, com endereço de execução CS:120H.

2.9 - Programa Terminal

O programa "TERMINAL", desenvolvido em Pascal, transforma o microcomputador pessoal compatível IBM-PC em um terminal de operação do KD-16. A ligação entre os dois aparelhos é feita pela porta de comunicação serial "COM1", do microcomputador pessoal e o conector "J4" do KD-16, usando o padrão RS-232C.

Basicamente, o trabalho do programa TERMINAL consiste em direcionar os caracteres recebidos pela interface serial para a tela do microcomputador pessoal e providenciar a transmissão dos caracteres teclados no microcomputador pessoal para o KD-16, através da interface serial.

Os caracteres recebidos pelo KD-16 são devolvidos para o microcomputador pessoal e, ao mesmo tempo, interpretados pelo programa monitor série. Caso exista algum erro no comando ou dado teclado, o KD-16 responderá enviando o caractere indicador de erro: "#".

Quatro caracteres teclados no microcomputador pessoal são interceptados pelo programa TERMINAL, não sendo enviados ao KD-16:

" ? " - Fornece na tela o quadro mostrado na figura 2.21, contendo os comandos disponíveis e a sintaxe utilizada;

"CTRL R" - Lê um arquivo do disco para a memória do KD-16. Este comando pede o nome do arquivo desejado, que deverá estar no formato de comunicação INTEL, apro-

priado para o KD-16. Caso seja necessário, pode-se fornecer uma constante a ser somada ao endereço de carga especificado no arquivo. Esta constante possibilita deslocar o arquivo na memória;

"CTRL W" - Armazena um bloco de memória do KD-16 em um arquivo no disco. O programa requisitará ao usuário, o nome do arquivo que receberá os dados, assim como os endereços limites do bloco de memória. O formato de comunicação INTEL para a saída é o mesmo que o utilizado no comando "CTRL R";

" Q " - Abandona o programa TERMINAL, retornando ao DOS.

```
Comandos do KD-16 e programa TERMINAL:

 S -  S[W]<end.>,[[<novo conteúdo>],]*<CR>
 X -  X[<reg>][[novo conteúdo>],]*<CR>
 D -  D[W]<end. inicial>[,<end. final>]<CR>
 M -  M<end. inicial>,<end. final>,<end. destino><CR>
 I -  I[W]<end. porta>,[,]*<CR>
 O -  O[W]<end. porta>,<dado>[,<dado>]*<CR>
 G -  G[<end. inicial>][,<end. parada>]<CR>
 N -  N[<end. inicial>],[[<end. inicial>],]*<CR>

^R -  Lê um arquivo do disco para o KD-16
^W -  Grava um arquivo com o conteúdo da memória do KD-16
^S -  Para impressão na tela
^Q -  Continua impressão na tela
^C -  Abandona comando
 ? -  Pede menu
 Q -  Retorna ao DOS

Se o nome dos arq. for "ERRO" -> Abandona comando ^W ou ^R

 [A]   "A" opcional
 [A]*  uma ou mais ocorrências opcionais de "A"
```

Figura 2.21 - "Menus" do KD-16 e programa TERMINAL.

Os comandos "CTRL R" e "CTRL W" utilizam um buffer de dados, implementado na memória do microcomputador pessoal, para permitir o armazenamento intermediário de caracteres. No comando CTRL W, por exemplo, os dados são transferidos do KD-16 para o buffer e, em seguida, do buffer para o arquivo em disco. Esta implementação visa minimizar o tempo de utilização do disco, principalmente quando selecionamos baixas velocidades de comunicação serial.

A figura 2.22 apresenta as ligações necessárias para o cabo de comunicação serial. Nesta figura, temos o sinal de saída "RTS/" (Request To Send) do microcomputador pessoal ligado à entrada "CTS/" (Clear To Send) do KD-16. Esta ligação permite controlar o envio de dados do KD-16 para o microcomputador pessoal (OSBORNE,1983).

O sinal RTS/, comandado pelo programa TERMINAL, assumirá nível "1" sempre que o microcomputador pessoal receber um "CR" (Carriage Return) ao fim de uma linha. Este nível "1", aplicado à entrada CTS/ do KD-16, bloqueará o envio de outros caracteres, até que o sinal RTS/ assuma novamente o nível "0". A necessidade desta paralisação se deve à relativa lentidão do microcomputador pessoal para realizar o rolamento da tela (SCROLL), quando o cursor está na última linha.

Se desfizéssemos a ligação RTS/ --> CTS/, seriam perdidos alguns caracteres no início de cada linha. Esta perda seria tanto maior, quanto maior fosse a velocidade de comunicação.

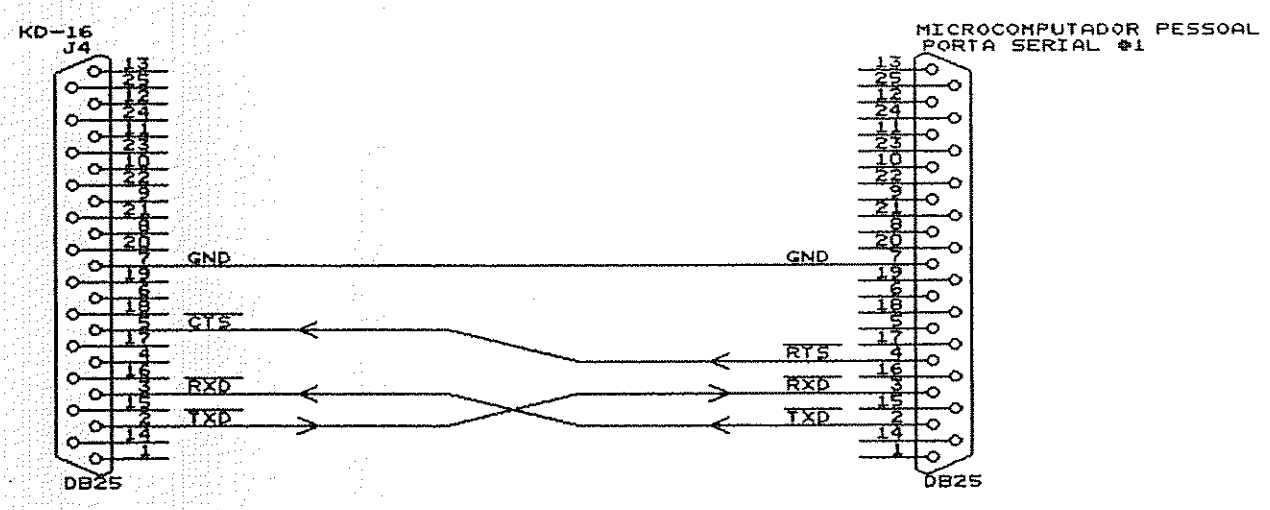


FIGURA 2.22 – CABO DE COMUNICAÇÃO SERIAL

2.10 - Programa Memintel

Quando utilizamos, em um microcomputador pessoal compatível IBM-PC, compiladores para linguagens como "Pascal", "C" ou macroassemblers, podem ser gerados dois tipos de arquivos executáveis, os arquivos do tipo ".COM" e ".EXE".

O usuário do SD-16 desejará transferir os programas contidos nestes arquivos para o kit KD-16, para teste e depuração. Entretanto, na transferência de arquivos de programas, comumente, são utilizados formatos padronizados de comunicação, com objetivo de controlar o endereço onde o programa deverá ser carregado e a verificação de erros de transmissão. Os arquivos transmitidos são arquivos do tipo texto, contendo somente caracteres ASCII.

Dentre os vários formatos de comunicação padronizados, estamos utilizando o formato INTEL para o microprocessador 8086/8088 (XPTO, 1987). O programa Memintel serve para converter o conteúdo binário dos arquivos tipo ".COM" e ".EXE" para um arquivo texto no formato de comunicação adotado. Depois de convertidos estes arquivos podem ser carregados no KD-16 pelo comando "CTRL R". A figura 2.23 apresenta a estrutura deste formato de comunicação.

Este formato de comunicação permite que também sejam enviados o endereço de carga e o endereço de execução do programa.

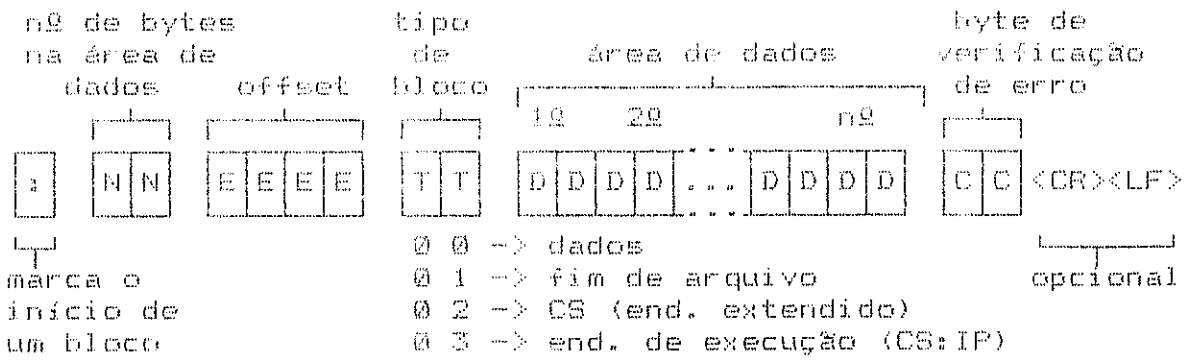
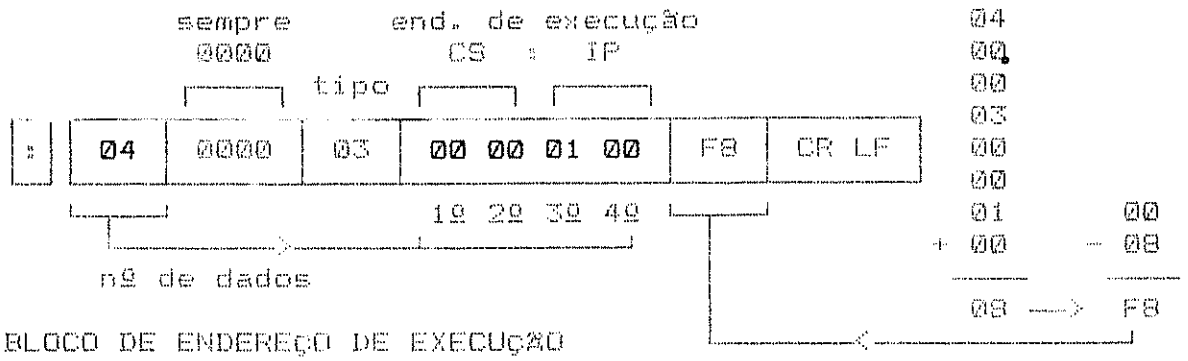


Figura 2.23 - Estrutura do formato de comunicação INTEL para o 8086/8088.

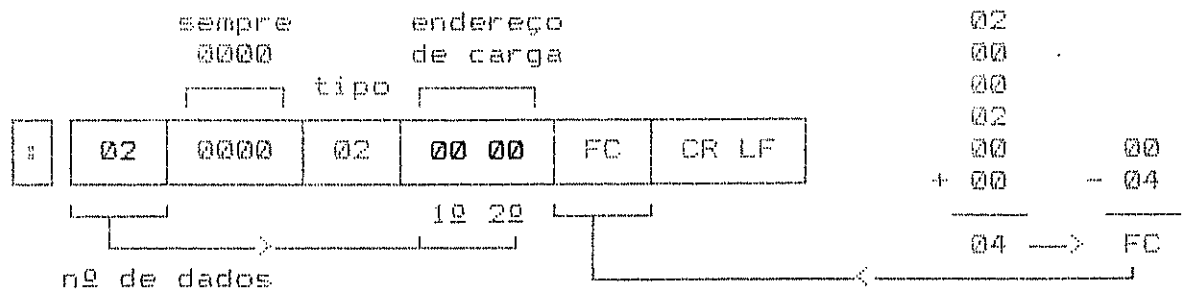
Os dados são transferidos em blocos, cujo tamanho pode variar de 11 a 45 caracteres. Cada bloco é iniciado pelo caractere ":", podendo, opcionalmente, ser finalizado pelos caracteres "CR" e LF". Os caracteres opcionais "CR" e LF", quando utilizados, têm função exclusivamente de aumentar a legibilidade da listagem ou impressão destes dados.

Durante a transferência de um arquivo, podemos encontrar até quatro tipos diferentes de blocos: o bloco de dados, o bloco de fim de arquivo, o bloco de endereço de carga e o bloco de endereço de execução. A figura 2.24 apresenta um exemplo de cada um destes blocos.

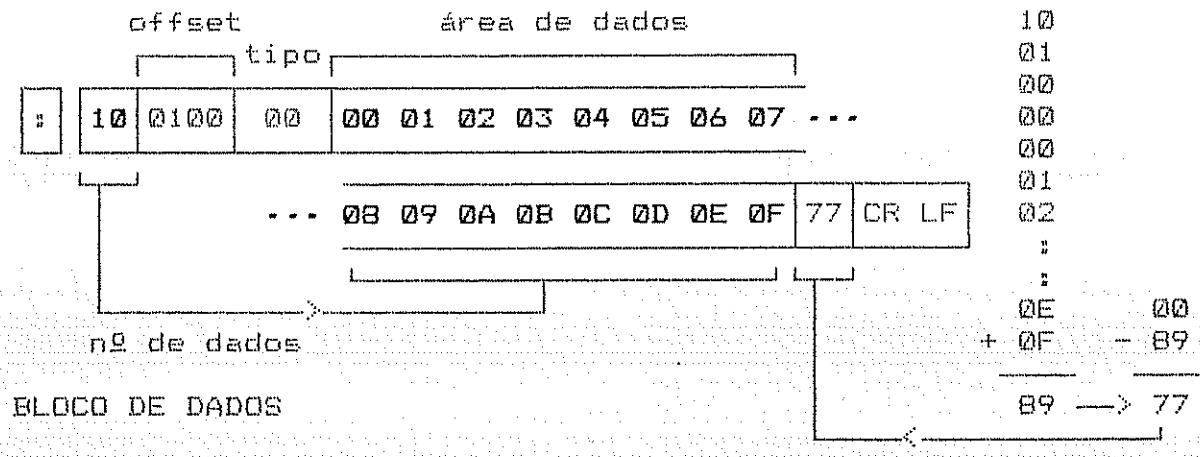
O tipo de cada um destes blocos pode ser identificado pelo conteúdo do 4º byte do bloco (caracteres 8º e 9º). Os bytes do programa a ser transferido aparecem a partir do 10º caractere. Os dois últimos caracteres de um bloco são usados para verificação de erros, formando um byte que é o complemento de 2 da soma de todos os bytes enviados depois do caractere ":" até o byte de verificação de erro.



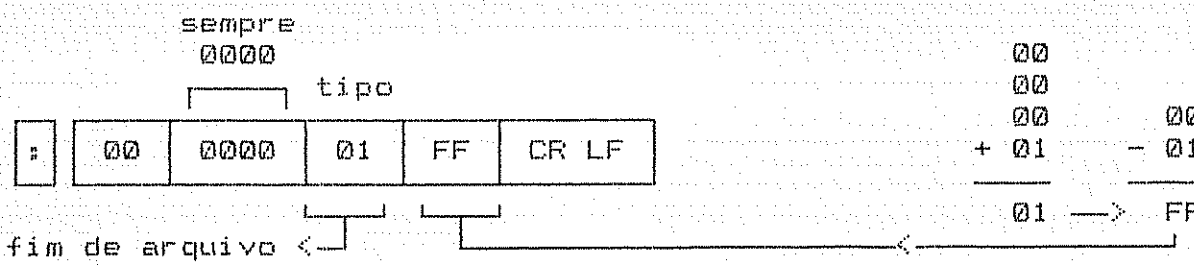
BLOCO DE ENDEREÇO DE EXECUÇÃO



BLOCO DE ENDEREÇO DE CARGA



BLOCO DE DADOS



BLOCO DE FINALIZAÇÃO DO ARQUIVO

Figura 2.24 - Tipos de blocos do formato INTEL adotado.

O programa Memintel aceita o nome do arquivo a ser convertido, como um parâmetro fornecido diretamente na linha de comando do DOS. Como opção, podemos executá-lo omitindo o nome do arquivo, deixando para fornece-lo num campo reservado, na tela do conversor Memintel (ver figura 2.25).

No conversor Memintel, além do nome do arquivo a ser convertido, podemos modificar os valores pré-estabelecidos para os endereços de carga e execução.

Um arquivo de entrada "arq.mem" resultará num arquivo de saída com uma nova extensão, "arq.int", que caracteriza a tradução realizada. A linha denominada "STATUS", informa ao usuário sobre o andamento do processo de conversão, e sobre a ocorrência de erros durante a tentativa de execução.

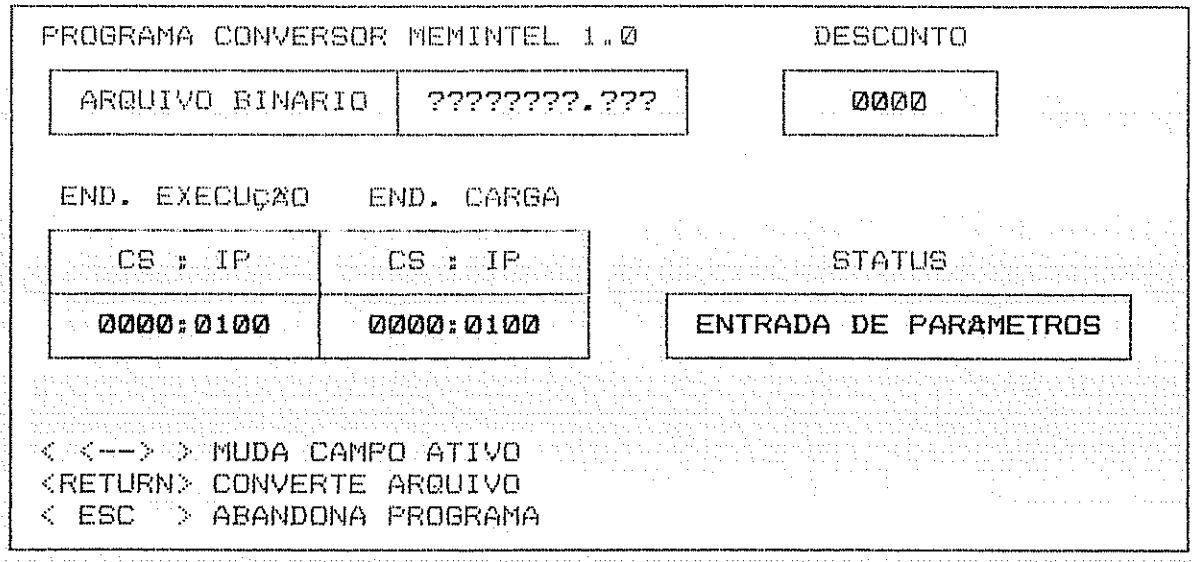


Figura 2.25 - Tela do conversor Memintel.

O grau de intervenção do usuário no processo de conversão de um arquivo pode variar de caso para caso, dependendo de como ele foi gerado (tipo de compilador) e de parâmetros

fornecidos no programa fonte (ex: pseudo-instrução "ORG 0H").

Um programa escrito originalmente em linguagem de montagem ou "Assembly", que resulte em um arquivo final do tipo ".COM", já se encontra armazenado no disco no formato "imagem de memória", ou seja, byte a byte equivalente ao que estará na memória durante a execução. Este tipo de arquivo pode ser convertido pelo programa Memintel sem maiores preocupações em relação ao seu conteúdo. Devemos somente cuidar para que sejam corretamente ajustados os endereços de carga e execução nos campos correspondentes, na tela do programa Memintel.

Um programa escrito em Assembly que resulte em um arquivo final do tipo executável ".EXE", provavelmente, apresentará um bloco de bytes iniciais, que não corresponde ao programa propriamente dito. Este bloco é chamado de cabeçalho do arquivo ".EXE". O cabeçalho contem informações complementares, utilizadas somente pelo sistema operacional do microcomputador pessoal, para que ele consiga colocar o programa em execução. Dentro do cabeçalho encontramos, entre outras coisas, uma tabela de realocação de variáveis, após a qual, temos o início do programa. A tabela de realocação de variáveis é usada pelo sistema operacional para fazer a realocação do programa, instalando-o em sua real posição de execução. Esta realocação precisa ser efetuada pois, um programa ".EXE" pode possuir diversos segmentos, o que torna impraticável armazená-lo diretamente no formato "imagem de memória". Um programa ".EXE", antes da realocação, referencia as suas variáveis como se o programa fosse ser carregado no início da memória

(segmento 0).

A 5ª palavra de um arquivo ".EXE" (bytes 9 e 10) contem um número hexadecimal que, multiplicado por 16, expressa o número de bytes do cabeçalho. O valor contido nesta palavra pode ser usado para descartar este cabeçalho, inútil em um ambiente estranho ao sistema operacional do microcomputador pessoal (MILLER, 1988; QUADROS, 1988; NORTON & SOCHA, 1988; NORTON, 1988; NORTON, 1985).

O modo mais fácil de se obter o tamanho do cabeçalho, é através da utilização do programa utilitário "EXEMOD", fornecido no DOS, que informa prontamente diversas características dos programas do tipo ".EXE". A informação sobre o tamanho do cabeçalho pode ser obtida da linha identificada por: "Header size".

No programa Memintel, o campo da tela denominado "DESCONTO", pode ser utilizado para dar entrada no número de bytes a serem descartados do início do arquivo.

A tecla "ESC" interrompe a execução da conversão a qualquer momento.

CAPITULO 3 - PLACAS PERIFÉRICAS E DE APOIO

3.1 - Introdução

Neste capítulo são apresentadas quatro placas que realizam funções de apoio ao desenvolvimento, como a placa do gravador de EPROMs e a placa rastreadora, além de interfaces de E/S e temporização.

Na introdução de cada um dos sub-itens que descrevem as diversas placas, ressaltamos a utilidade de cada uma delas dentro da filosofia de implementação do SD-16. Os diagramas de blocos mostram a estrutura interna de cada placa e sua inter-ligação ao kit KD-16 ou microcomputador pessoal.

3.2 - Módulo de teclado e visor

3.2.1 - Introdução

Embora o KD-16 seja mais comodamente operado através do computador pessoal juntamente com o programa monitor série, é interessante permitir sua operação usando uma interface de baixo custo.

O módulo de teclado e visor aqui apresentado, é um acessório do kit KD-16. Quando conectado ao kit, estando este sob o comando do programa monitor para teclado, permite que ele funcione de modo auto-suficiente, tornando-se uma opção econômica para aqueles que não dispõem de um microcomputador pessoal como interface para o usuário. O baixo custo deste mó-

dulo facilita o uso do kit em aulas de laboratório de técnicas digitais da graduação.

O programa monitor para teclado, descrito no item 3.2.3, permite ao usuário do KD-16 usar o teclado para carregar um programa na memória RAM, executá-lo e depurá-lo, através da execução passo a passo. Permite entrar e sair com bytes e palavras nas portas paralelas e demais periféricos. Habilita, também, a verificação e modificação do conteúdo das memórias e registradores do 8088.

Independentemente de sua utilização especializada junto ao programa monitor para teclado, este módulo fornece ao usuário uma interface prontamente utilizável para entrada e saída de dados no KD-16.

Na figura 3.1 temos o diagrama de blocos do módulo de teclado e visor. Nas figuras AP.9 e AP.10 (Apêndice A) encontramos seus diagramas esquemáticos.

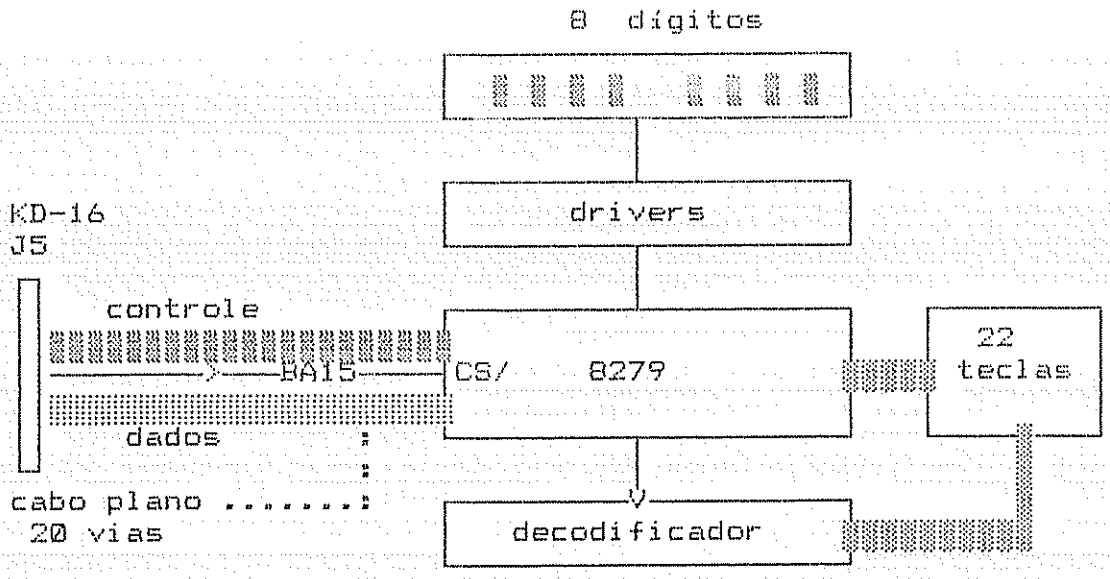


Figura 3.1 - Diagrama de blocos do modulo de teclado e visor

3.2.2 - Descrição do módulo de teclado e visor

Como o módulo de teclado e visor é montado em uma caixa separada do KD-16, sua conexão ao kit é feita pelo conector "J5", através de um cabo plano de 20 condudores. (Ver conectores do KD-16, figura 2.9).

Neste cabo temos:

- 8 vias de dados, BD0 até BD7;
- 2 bits da via de endereço BA0 e BA15;
- sinais de comando IORD/, IOWR/, RESET;
- sinal de relógio PCK, de 307,7KHz;
- sinal de pedido de interrupção IRQ;

O circuito integrado responsável pelo controle desse módulo é o 8279-5 da INTEL (INTEL, 1980; INTEL, 1981). O 8279-5 é uma interface programável, com duas porções controladoras:

- A parte controladora do teclado serve de interface com uma matriz de até 64 teclas. Faz a eliminação dos ruídos de chaveamento e o armazenamento de até 8 entradas tecladas. Qualquer tecla acionada ativa a linha "IRQ" (Interrupt ReQuest), para requisição de interrupção ao microprocessador.
- A parte controladora do visor trabalha por multiplexação, podendo acionar mostradores alfanuméricos ou simples LEDs indicadores. Existe uma memória RAM com capacidade de 16 x 8 bits, para armazenagem dos dados a se-

rem apresentados. Pela via de dados, o microprocessador tem acesso ao barramento interno do 8279-5 para ler e escrever nesta memória.

Em nosso circuito, o teclado é composto de 22 teclas organizadas de forma matricial, contendo os algarismos 0-9, A-F e 6 teclas de controle (+ - . , : REG).

As saídas "SL0" até "SL3" fornecem uma contagem binária que será utilizada pelos decodificadores 7445 e 74LS156. O primeiro comanda a habilitação de cada um dos mostradores de 7-segmentos. O segundo faz uma varredura nas colunas do teclado, de modo que se houver algum acionamento, teremos o retorno dessa informação ao 8279-5 pelas linhas "RL0" até "RL7". Quando detectado um acionamento, os valores da coluna e linha ativos serão utilizados na geração do código da tecla, que é armazenado numa memória interna, tipo "primeiro a entrar, primeiro a sair" ("FIFO" First In First Out). As linhas A0-A3 e B0-B3 são utilizadas no controle do acendimento dos segmentos do mostrador.

A tabela 3.1 indica os códigos gerados pelo acionamento de cada uma das teclas.

Quanto ao visor, são usados dois grupos de quatro dígitos hexadecimais. Um grupo para apresentação de endereços e outro para dados.

"0" - 00H	"8" - 08H	"." - 10H
"1" - 01H	"9" - 09H	"," - 11H
"2" - 02H	"A" - 0AH	"-" - 12H
"3" - 03H	"B" - 0BH	"+" - 13H
"4" - 04H	"C" - 0CH	";" - 14H
"5" - 05H	"D" - 0DH	"REG" - 15H
"6" - 06H	"E" - 0EH	
"7" - 07H	"F" - 0FH	

Tabela 3.1 - Códigos gerados pelas teclas do módulo de teclado e visor

O endereço das memórias RAM do visor estão relacionadas aos mostradores de 7-segmentos, conforme mostrado na figura 3.2.

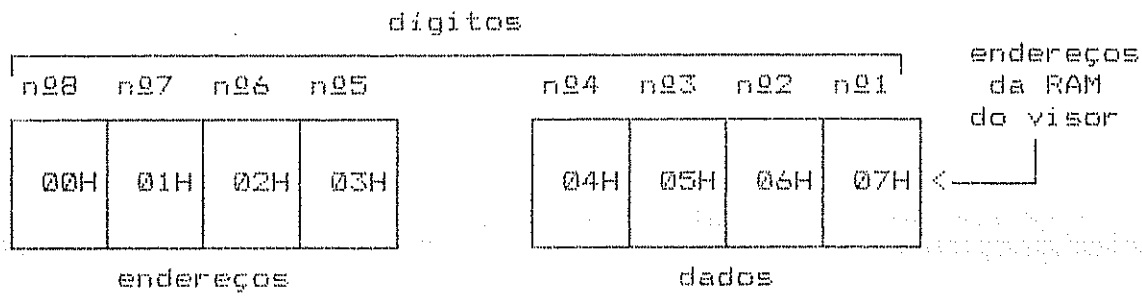
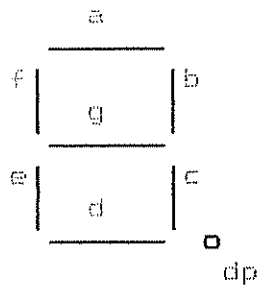


Figura 3.2 - Endereço das memóris RAM do visor, relacionados a cada mostrador de 7-segmentos

Cada um dos segmentos de cada mostrador é aceso ou apagado conforme os níveis lógicos assumidos por cada bit dos bytes armazenados nas memórias RAM do visor. A correspondência entre os segmentos e cada bit é mostrada na figura 3.3.



D7	D6	D5	D4	D3	D2	D1	D0
dp	g	f	e	d	c	b	a

Figura 3.3 - Correspondência entre segmentos e os bits dos dados armazenados na RAM do visor

Visando as aplicações onde é importante um pronto atendimento às ordens dadas pelo teclado, existe a possibilidade de conectarmos o sinal de pedido de interrupção ("IR0") do 8279-5 a uma das duas linhas de requisição de interrupção, "IR2" ou "IR3", do KD-16. A seleção de qual dessas linhas será usada é feita pelo "Jumper" "JP10", localizado na placa do KD-16 (ver tabela 3.2).

JP 10	Z1-Z2	IR0 -> IR2	Possibilita ao 8279-5 gerar interrupções usando o controlador 8259A
	Z1-Z3	IR0 -> IR3	

Tabela 3.2 - Opções para o "jumper" JP 10

O endereçamento dos registros do 8279-5 devem ser feitos como indicado na tabela 3.3.

OPERAÇÃO	ENDEREÇOS		
	7C01H	7C00H	
LER	STATUS	DADOS RAM DO VISOR	DADOS FIFO DO TECLADO
ESCREVER	COMANDOS	DADOS RAM DO VISOR	

Tabela 3.3 - Endereçamentos dos registros do 8279-5

3.2.3 - Programa Monitor para Teclado

O programa monitor usado no KD-16 é uma adaptação do monitor para teclado do "SDK-86 System Design Kit" da INTEL (INTEL, 1978a). Neste item, descreveremos os aspectos gerais deste programa e seus comandos.

Quando, no soquete "U7" do KD-16, instalamos uma memória EPROM contendo o programa monitor para teclado, passamos a operar a partir do pequeno módulo de teclado e visor descrito no item 3.2.2. Este programa monitor ocupa, aproximadamente, 3200 bytes a partir do endereço "FF000H".

Ao ser inicializado pelo "RESET", o microprocessador buscará instruções no endereço "FFFF0", que possui uma instrução de salto para o início do programa monitor propriamente dito.

Após a inicialização do controlador 8279-5 e de algumas variáveis do programa monitor, este emitirá uma mensagem de "pronto" ao usuário, como mostrado na figura 3.4.

Figura 3.4 - Mensagem inicial do programa monitor para teclado

A não ser após o acionamento da chave "RESET" do KD-16, quando temos a mensagem mostrada na figura 3.4, a mensagem de "pronto" do monitor para teclado será conforme mostrado na figura 3.5. Somente o segmento "g" do oitavo mostrador de 7 segmentos ficará aceso.

Figura 3.5 - Mensagem de "pronto" do programa monitor para teclado

As mensagens das figuras 3.4 e 3.5 indicam que é esperada uma tecla de comando. As teclas de "0" até "9", além de habilitarem a entrada de valores numéricos, permitem a seleção de um dos dez comandos disponíveis.

Estas teclas de múltiplas funções (até três funções) têm seu significado definido pelo contexto em que são usadas. Na figura 3.6 temos um exemplo de tecla de múltiplas funções, a tecla com o algarismo "2".

No item 3.2.3.1 são descritos os comandos do programa monitor para teclado.

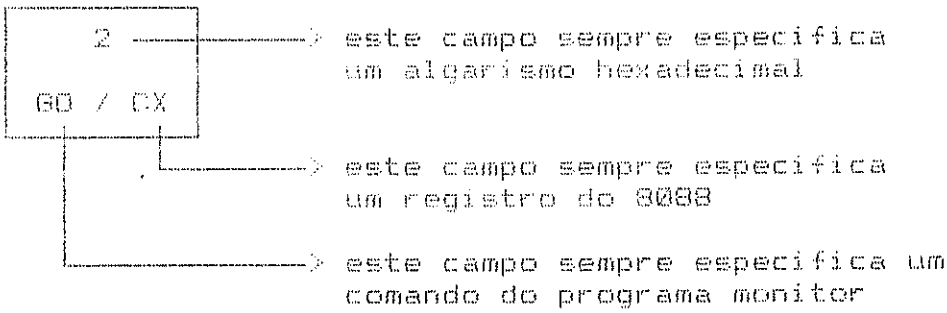


Figura 3.6 - Exemplo de tecla de múltiplas funções

Quando acionada uma tecla correspondente a um comando qualquer, serão acesos de 1 a 3 pontos decimais nos mostradores do campo de endereços. Estes pontos indicam o número de parâmetros requeridos por aquele comando. A medida que estes parâmetros são fornecidos, haverá a redução do número de pontos decimais.

A tecla "." dá início a execução do comando previamente escolhido, finalizando-o em seguida.

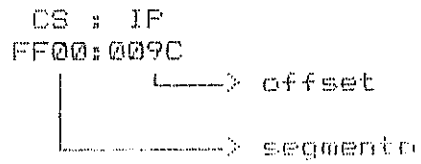
A tecla "," possui funções diferentes, dependendo da situação em que é utilizada:

- nos comandos que precisam de mais que um parâmetro de entrada, serve como separador de parâmetros;
- nos comandos que acessam memórias, permite incrementar o endereço fornecido e repetir a ação do comando para a nova posição apontada;
- nos comandos que acessam registros do 8088, permite avançar para o próximo registro, também repetindo a ação do comando vigente.

As teclas "+" e "-" permitem ao usuário realizar

operações aritméticas com números hexadecimais, facilitando o cálculo de endereçamentos relativos.

A tecla ":" é usada para separar a entrada de endereços em duas parcelas: o valor do segmento e o valor do offset. Exemplo:



Em uma especificação de endereço, a parte relativa ao segmento é sempre opcional. Caso não seja fornecido o segmento, será assumido o valor corrente do segmento "CS".

Ao teclarmos um endereço, os dígitos entrarão um a um no campo de endereços do visor. Caso sejam teclados mais que quatro algarismos, somente os quatro últimos serão utilizados, e os excedentes serão descartados.

Como mostrado no exemplo da figura 3.6, as teclas hexadecimais de "0" até "D" também são utilizadas para a seleção de um dos 14 registros do 8088. A tecla "REG" deverá ser acionada antes da tecla hexadecimal de múltipla função, de modo que esta seja corretamente interpretada, selecionando o registro desejado.

Qualquer erro cometido pelo usuário na seqüência de teclas permitida para um comando, implicará numa mensagem de erro e no término do comando em atividade.

No início da memória RAM, temos 256 posições reservadas para utilização como área de dados e pilha do programa monitor.

A área de pilha é inicializada fazendo "SS:SP" igual a "0000:0100". Desse modo, a área livre utilizável pelo usuário começa a partir do endereço "00100H" (inclusive).

Os vetores das três primeiras interrupções do 8088 ("INT1", "INT2" e "INT3"), são inicializados pelo programa monitor e utilizados, respectivamente, para as seguintes tarefas:

- comando passo a passo "ST";
- interrupção não mascarável "NMI";
- pontos de parada do comando "GO".

3.2.3.1 - Comandos do programa monitor para teclado

Neste item descreveremos resumidamente a operação de cada comando do programa monitor. Para maiores detalhes, veja o guia do usuário do SDK-86 (INTEL, 1978b).

Na descrição dos comandos usaremos os seguintes símbolos:

| | Indica o acionamento de uma das teclas

[A] Indica que "A" é opcional

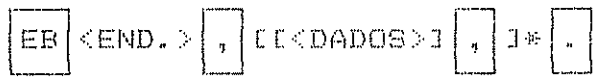
[A]* Indica uma ou mais ocorrências opcionais de "A"

<B> Indica que "B" é uma variável

1) **"EB" EXAMINAR BYTE:** Permite ler e modificar o conteúdo da memória endereçada (ver figura 3.7).

OBS: - A tecla "," incrementa o endereço, mostrando o conteúdo do byte subsequente.

- A tecla "." termina o comanto "EB" corrente, habilitando o monitor a executar outros comandos.
- Enquanto algum byte está sendo examinado, qualquer acionamento de teclas de algarismos hexadecimais modificará o conteúdo deste byte.



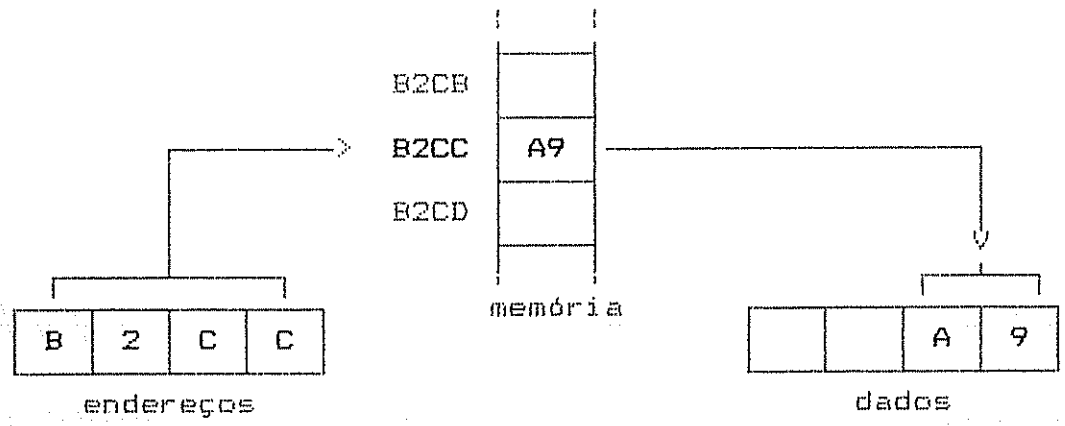


Figura 3.7 - Ação do comando **"EB"**

2) **"EW" EXAMINAR PALAVRA:** Permite, com um só comando, ler dois bytes a partir de memórias com endereços consecutivos (ver figura 3.8).

Estes dois bytes formam uma palavra, que é apresentada no campo de dados do visor. O usuário pode modificar esta palavra, se quiser. As demais observações do comando "EB" também valem para este comando.

EW <END.> [, [[<DADOS>] ,]* .]

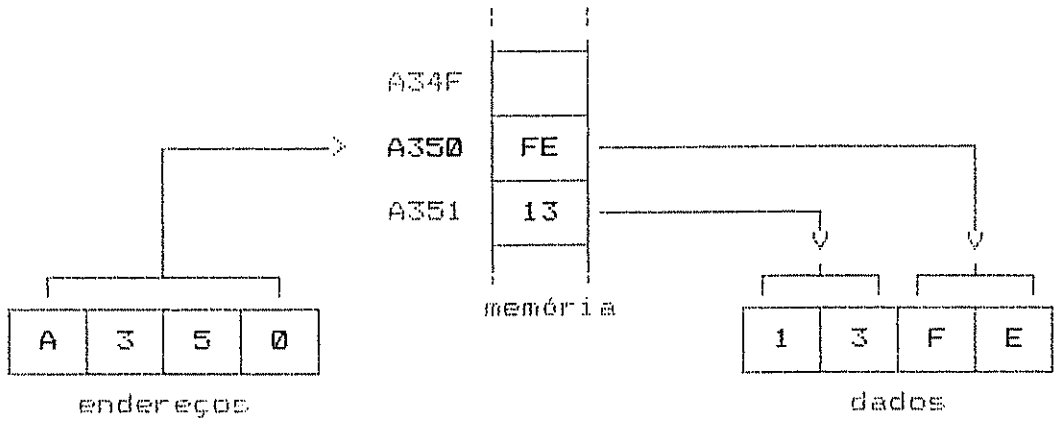


Figura 3.8 - Ação do comando **"EW"**

3) **"ER" EXAMINAR REGISTRO** - Este comando é usado para examinar o conteúdo dos registros do 8088. Se necessário, permite fazer modificações deste conteúdo (ver figura 3.9).

OBS: - Após a tecla "ER", o programa monitor interpreta a próxima tecla acionada como o nome do registro a ser examinado.

- Acionamentos opcionais da tecla "," permitirão o exame sequencial dos registros, a partir do 1º registro selecionado. A ordem de apresentação é a seguinte: AX, BX, CX, DX, SP, BP, SI, DI, CS, DS, SS, ES, IP e FL.

ER <TECLA COM O NOME DO REGISTRO>[[<DADOS>] ,]*[.]

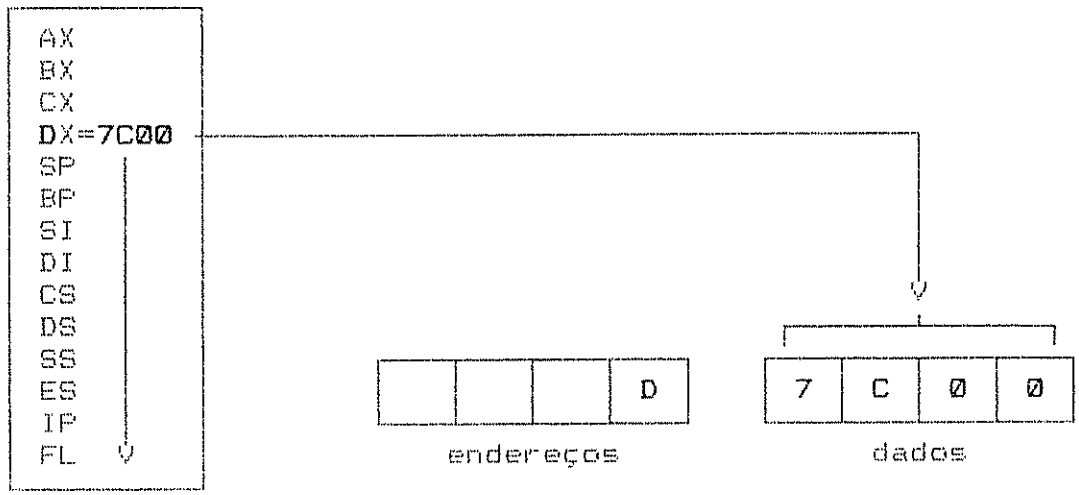
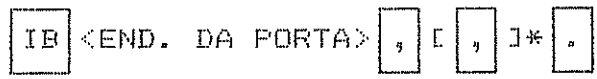


Figura 3.9 - Ação do comando "ER"

4) **"IB" ENTRA BYTE:** Permite ler um byte proveniente de uma porta de entrada (ver figura 3.10).



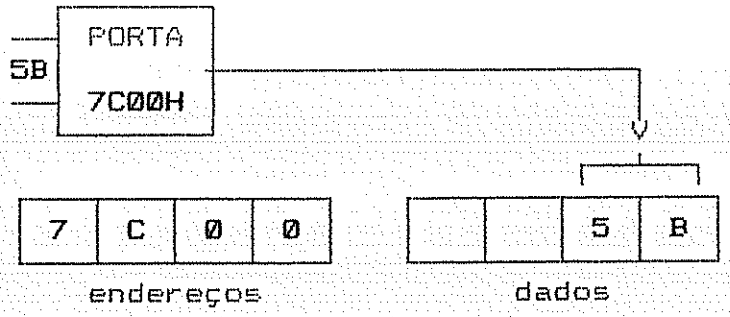
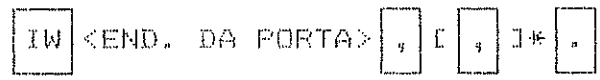


Figura 3.10 - Ação do comando **"IB"**

5) **"IW" ENTRADA DE PALAVRA:** Permite, com um só comando, ler dois bytes a partir de portas de entrada com endereços consecutivos (ver figura 3.11).

Estes dois bytes formam uma palavra que é apresenta-

da no campo de dados do visor.



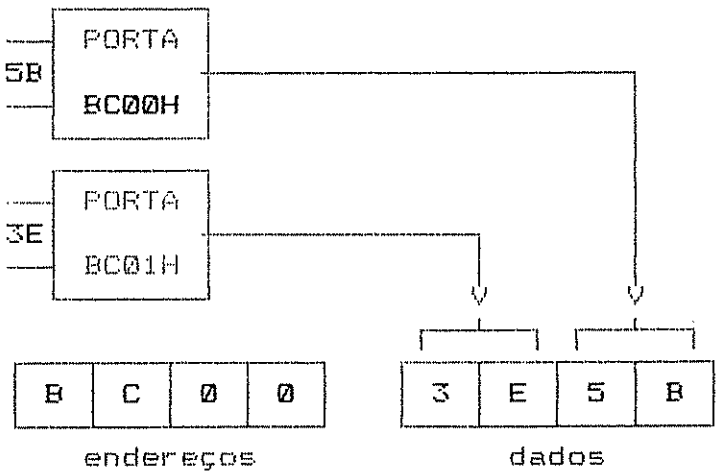


Figura 3.11 - Ação do comando **"IW"**

6) **"OB" SAIDA DE BYTE:** permite escrever um byte em uma porta de saída (ver figura 3.12).

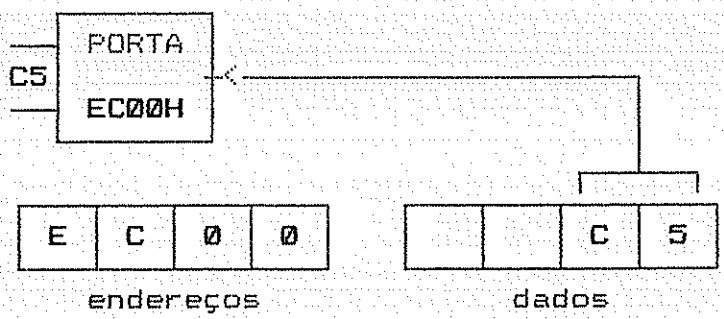


Figura 3.12 - Ação do comando **"OB"**

7) **"OW" SAIDA DE PALAVRA:** permite escrever uma palavra em duas portas de saída com endereços consecutivos (ver figura 3.13).

OW <END. DA PORTA> , <DADO>[, <DADO>]* .

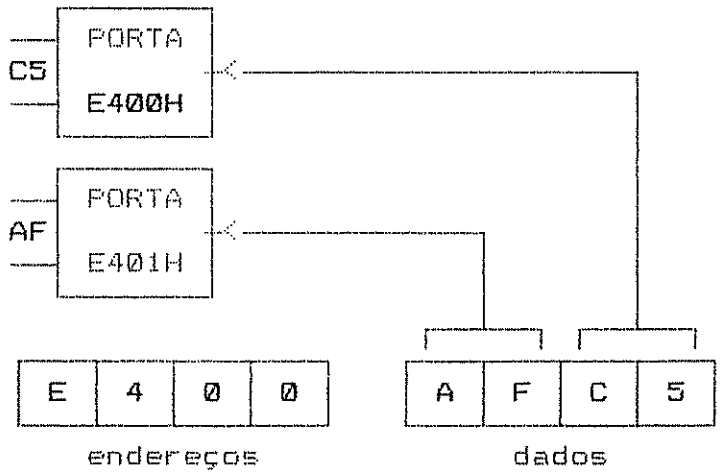


Figura 3.13 - Ação do comando **"OW"**

8) **"GO" EXECUTAR:** Permite transferir o controle do 8088 para o programa do usuário.

Este comando aceita, opcionalmente, a especificação de um endereço de um ponto de parada (breakpoint). O primeiro byte da instrução onde ocorrerá a paralisação será substituído, temporariamente, pela instrução "CC" (INT3). Quando esta instrução é encontrada, é executada uma rotina que salva todos os registros do 8088 e coloca o programa monitor em condições de receber comandos. Os comandos "EB", "EW" e "ER" podem serem usados para acompanhar a execução do programa do usuário.

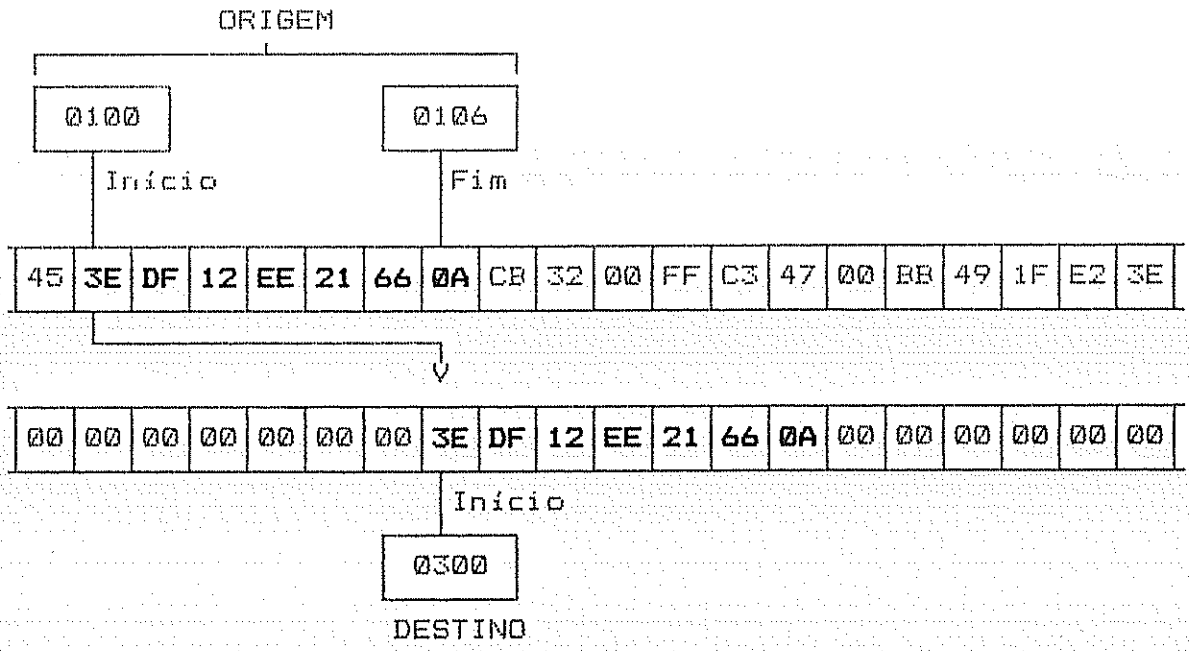
GO [<END.>][,<END. DO PONTO DE PARADA>][.]

9) **"MV" MOVER DADOS:** Permite mover um bloco de dados dentro da memória (ver figura 3.14).

Este comando precisa de três parâmetros de entrada, portanto, ao ser selecionado, causará o acendimento de três pontos decimais no campo de endereços. O tamanho máximo do bloco a ser movido é de 64Kbytes.

MV <END. INICIAL DA ORIGEM> , <END. FINAL DA ORIGEM> , ...

... <END. INICIAL DO DESTINO> .



MV <0100> , <0106> , <0300> .

Figura 3.14 - Ação do comando **"MV"**

10) **"ST" EXECUÇÃO PASSO A PASSO:** Permite que as instruções na memória sejam executadas uma a uma, a cada acionamento da tecla "ST".

Após a execução de cada instrução, o controle do microprocessador retorna para o programa monitor, permitindo que o usuário faça verificações dos registros e memórias.

OBS: - Este comando não deve ser usado com programas que utilizem a instrução "INT3" (breakpoint), pois causará a execução passo a passo do próprio programa monitor.

- Instruções que alteram o conteúdo dos registros "SS" e "SP" não podem ser processados com este comando.

[ST] [<END. DE EXECUÇÃO>] [,] [[<END. DE EXECUÇÃO>] [,]]* [.]

3.3 - Placa de interface paralela e temporização

3.3.1 - Introdução

Nas fases iniciais do desenvolvimento de software para aplicações em controle, é mais interessante que os testes possam ser feitos usando o próprio microcomputador pessoal, ao invés do KD-16. Fazendo isto, evitamos o demorado processo de conversão do programa compilado para o formato de comunicação INTEL e sua transmissão serial para o KD-16. A capacidade de trabalhar com arquivos em disco, usando o sistema operacional, aumenta bastante a produtividade, facilitando as atualizações após cada modificação e recompilação. Além disso, os depura-

dores disponíveis no microcomputador pessoal são mais fáceis de operar e mais poderosos, permitindo que o usuário trabalhe a nível do programa fonte. Numa fase mais avançada, quando os programas desenvolvidos já estiverem suficientemente testados e depurados, podem ser adaptados e transferidos para o KD-16.

Para que esta idéia seja colocada em prática, é necessário que o microcomputador pessoal tenha capacidade de interfaceamento semelhante à do kit KD-16.

O KD-16 possui portas paralelas que servem de interface com os dispositivos externos, além de contadores e temporizadores de eventos. Para que no microcomputador pessoal tenhamos estas mesmas facilidades, projetamos e construimos a placa de interface paralela e temporização.

Uma vez que o microcomputador pessoal utilizado neste projeto é compatível com o IBM-PC, a placa de interface paralela e temporização será compatível também com o barramento do KD-16, tornando-se um acessório que permite expandir o número de portas de E/S disponíveis.

Outra utilidade dessa placa, é de servir de interface controladora para o gravador de EPROMs descrito no item 3.4.

Na placa de interface paralela e temporização encontramos:

- 2 interfaces paralelas programáveis, 8255-A5 da INTEL;
- 2 contadores/temporizadores programáveis, 8253-5 também da INTEL (ou opcionalmente 8254);
- 1 oscilador controlador a cristal de quartzo.

Na figura 3.15 temos o diagrama de blocos da placa de interface paralela e temporização. Nas figuras AP.11 e AP.12 (Apêndice A) temos seus diagramas esquemáticos.

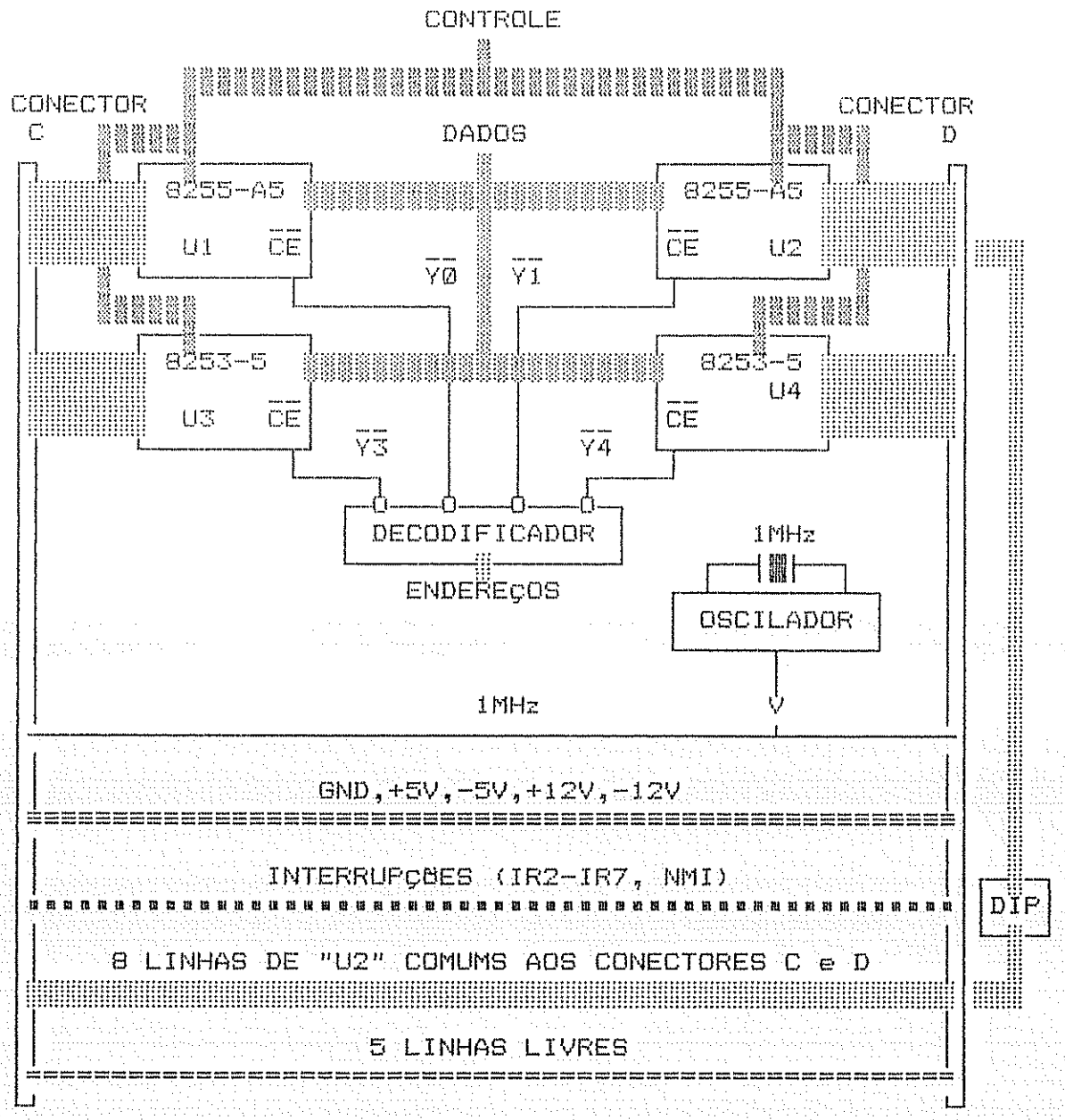


Figura 3.15 - Diagrama de blocos da placa de interface paralela e temporização.

3.3.2 - Descrição da placa de interface paralela e temporização

Descreveremos neste item os circuitos das figuras 3.15, AP.11 e AP.12.

As duas interfaces paralelas 8255-A5 tornam disponíveis ao usuário até 48 linhas programáveis, distribuidas em seis portas de 8 linhas cada. Estas linhas podem funcionar como entradas ou saídas dependendo da conveniência para diferentes aplicações. Em comunicações paralelas de 8 bits, dispõe-se, também, de sinais para sincronização ("handshake") e geração de interrupção.

Acrescentamos a esta placa dois contadores/ temporizadores 8253-5, pois temporizar intervalos, contar eventos, gerar cadências de pulsos e diferentes freqüências são tarefas normalmente necessárias em controle. Cada 8253-5 é constituído de três contadores/temporizadores programáveis de 16 bits, que podem operar em seis modos distintos.

Opcionalmente o 8253-5, pode ser substituido pelo 8254, que consegue trabalhar com freqüências até quatro vezes superiores, sendo pino a pino compatível com o primeiro.

Outra facilidade oferecida por esta placa é a geração de um sinal com freqüência estável controlado a cristal de quartzo de 1MHz. Sinais de freqüência estável normalmente são requeridos pelas aplicações que se utilizam dos contadores/temporizadores. Preferimos não fazer uso do sinal de relógio "CLK", prontamente oferecido no barramento do microcomputador pessoal, porque dependendo do fabricante e das condições

selecionadas pelo usuário, poderemos encontrar o "CLK" com valores de freqüência variando de 4,7MHz até 10MHz.

Para melhorar o desempenho do microprocessador, quanto ao atendimento de dispositivos conectados à placa de interface paralela e temporização, tornamos disponível ao usuário as entradas de requisição de interrupção oferecidas no conector do barramento do microcomputador compatível IBM-PC. Algumas dessas interrupções já possuem função definida se a placa estiver operando no microcomputador pessoal. Por exemplo, o sinal "NMI", quando ativado, indica erro de paridade na memória. Entretanto, no KD-16 todos esses sinais estão livres para o usuário.

A saída dos sinais das portas paralelas e dos contadores é feita através de dois conectores de 60 vias, para cabo plano, denominados "C" e "D".

Cada conector fornece:

- os sinais de um 8255-A5;
- os sinais de um 8253-5;
- 7 linhas de interrupção do barramento;
- o sinal "RESET" ou "RESET/";
- a freqüência padrão gerada na placa;
- as linhas de alimentação (GND, +5V, -5V, +12V e -12V)

A figura 3.16 apresenta os sinais desses conectores.

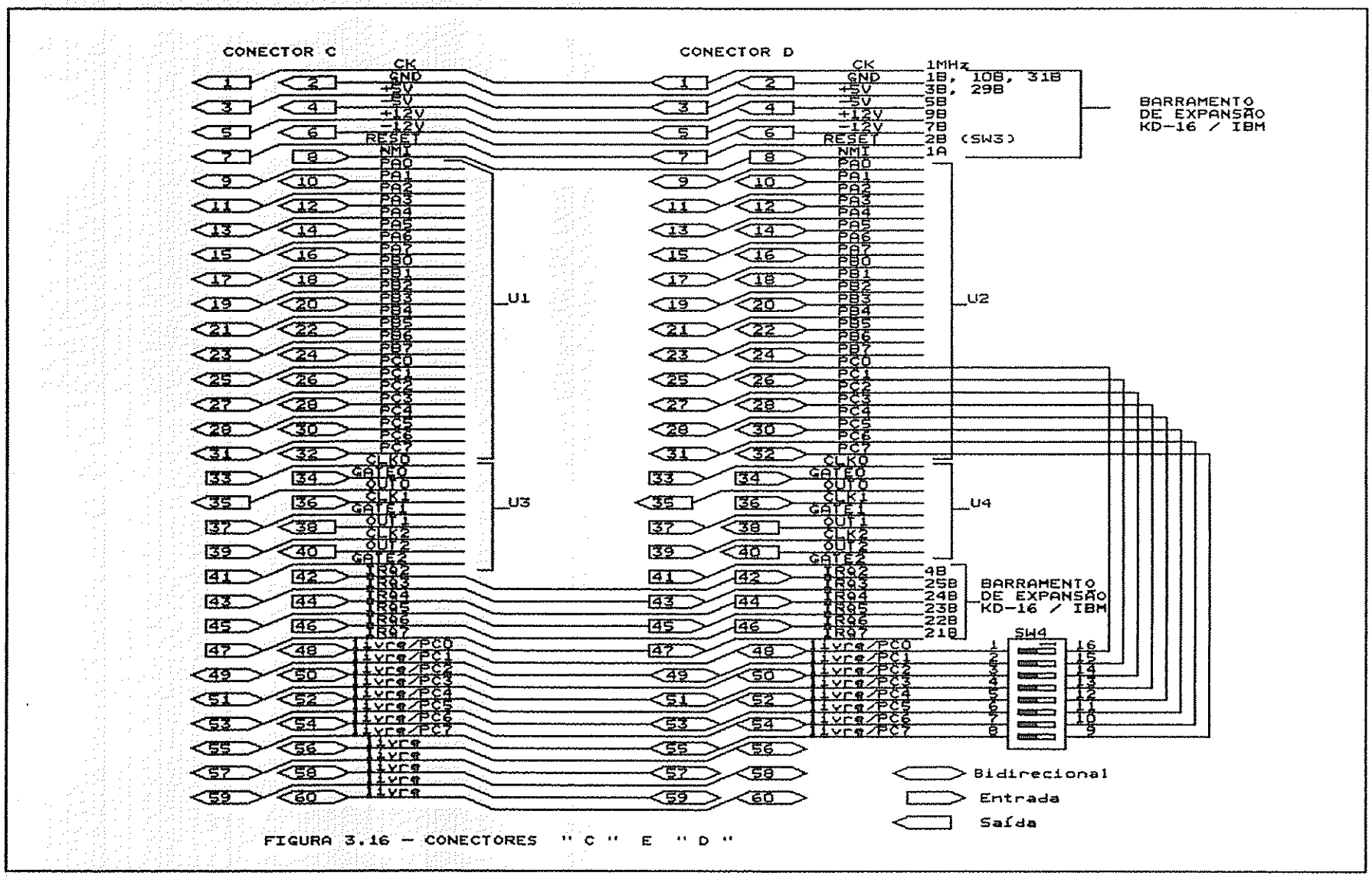


FIGURA 3.16 – CONECTORES " C " E " D "

Algumas aplicações necessitam de um número maior de linhas como é o caso, por exemplo, da placa do gravador de EPROMs que requer 30 linhas. Sendo assim, para evitar que sejam usados os dois conectores para suprir o número requerido de linhas, usamos uma chave miniatura para interligar ao conector "C" mais 8 linhas pertencentes ao conector "D" (porta C de U2). Estas 8 linhas só estarão disponíveis no conector "C" se os interruptores da chave "SW4" estiverem fechados. Estando estes interruptores abertos, estas linhas ficarão livres.

As linhas deixadas livres interligam os dois conectores dentro da placa, permitindo a troca de sinais entre diferentes montagens, através dos cabos planos.

3.3.3 - Endereçamento dos periféricos

Para garantir que os componentes da placa de interface paralela e temporização sejam endereçados, sem superposição, em qualquer computador compatível IBM-PC, escolhemos a faixa reservada para protótipos para seu acesso. Como cada placa precisa de 16 endereços diferentes para o seu completo uso, a faixa de 32 endereços reservados (300H a 31FH) comportam duas dessas placas (ver tabela 3.4). A seleção de qual faixa será ocupada é feita pelo "jumper" "SW2" (1ª faixa, posição 30XH ou 2ª faixa, posição 31XH).

	KD-16	IBM-PC	PERIFÉRICO
FAIXA 1	FF00	0300	U1 - 8255-A5 - porta A
	FF01	0301	U1 - 8255-A5 - porta B
	FF02	0302	U1 - 8255-A5 - porta C
	FF03	0303	U1 - 8255-A5 - registro de controle
	FF04	0304	U2 - 8255-A5 - porta A'
	FF05	0305	U2 - 8255-A5 - porta B'
	FF06	0306	U2 - 8255-A5 - porta C'
	FF07	0307	U2 - 8255-A5 - registro de controle
	FF08	0308	U3 - 8253-5 - contador #0
	FF09	0309	U3 - 8253-5 - contador #1
	FF0A	030A	U3 - 8253-5 - contador #2
	FF0B	030B	U3 - 8253-5 - modo de operação
	FF0C	030C	U4 - 8253-5 - contador #0
	FF0D	030D	U4 - 8253-5 - contador #1
	FF0E	030E	U4 - 8253-5 - contador #2
	FF0F	030F	U4 - 8253-5 - modo de operação
FAIXA 2	FF10	0310	U1 - 8255-A5 - porta A
	FF11	0311	U1 - 8255-A5 - porta B
	FF12	0312	U1 - 8255-A5 - porta C
	FF13	0313	U1 - 8255-A5 - registro de controle
	FF14	0314	U2 - 8255-A5 - porta A'
	FF15	0315	U2 - 8255-A5 - porta B'
	FF16	0316	U2 - 8255-A5 - porta C'
	FF07	0317	U2 - 8255-A5 - registro de controle
	FF18	0318	U3 - 8253-5 - contador #0
	FF19	0319	U3 - 8253-5 - contador #1
	FF1A	031A	U3 - 8253-5 - contador #2
	FF1B	031B	U3 - 8253-5 - modo de operação
	FF1C	031C	U4 - 8253-5 - contador #0
	FF1D	031D	U4 - 8253-5 - contador #1
	FF1E	031E	U4 - 8253-5 - contador #2
	FF1F	031F	U4 - 8253-5 - modo de operação

Tabela 3.4 - Endereços dos periféricos nas placas de interface paralela e temporização

Como podemos notar, um mesmo periférico da placa de interface paralela e temporização precisa ser endereçado diferentemente caso ela seja inserida em um dos conectores do bar-

ramento do microcomputador compatível IBM-PC ou do KD-16.

A diferença é causada pela necessidade de, no KD-16, mantermos as linhas de endereços de A10 até A15 em nível "1", para que nenhum dos periféricos internos sejam habilitados simultaneamente aos da placa de interface paralela e temporização. Como foi visto no capítulo 2, o kit KD-16 não possui decodificador de periféricos, sendo estes habilitados diretamente pelos bits da via de endereços (ver figura 3.17).

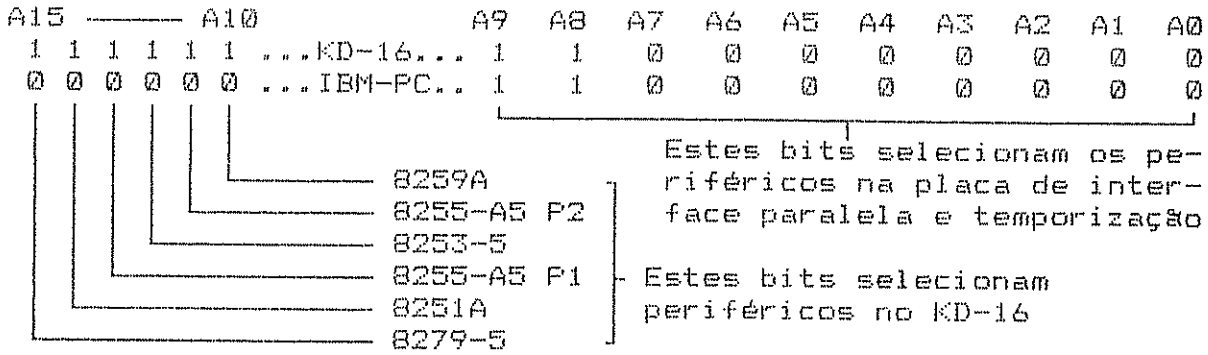


Figura 3.17 - Endereçamento da porta "A" ("U1", FAIXA 1) no KD-16 (FF00H) e no IBM-PC (0300H)

Com a decodificação implementada, estamos permitindo uma redundância no endereçamento da placa de interface paralela e temporização:

- de 340H a 35FH no IBM-PC
- de FF40H a FF5FH no KD-16

Porém isto não leva a qualquer dano, uma vez que esta faixa de endereçamento não é alocada para nenhum periférico no microcomputador compatível IBM-PC, nem no KD-16.

3.4 - Gravador de EPROMs

3.4.1 - Introdução

Tendo o usuário do KD-16 desenvolvido e depurado um programa para uma aplicação específica, desejará armazená-lo de modo não volátil numa memória EPROM (Erasable Programmable Read Only Memory). Para permitir isto, projetamos e construímos um gravador de memórias EPROM.

Existem diversos gravadores de memórias EPROM disponíveis no comércio. Alguns deles são implementados em uma placa de circuito impresso projetada para ocupar um dos conectores do barramento de expansão de um microcomputador pessoal. Geralmente, possuem um cabo multi-veias fazendo a conexão entre a placa principal e uma placa secundária. Esta placa secundária fica fora do microcomputador, como mostrado na figura 3.18, facilitando o acesso para o usuário. Ela é constituída basicamente de um soquete do tipo "zero-força", que recebe o circuito integrado de memória EPROM a ser gravado.

Na placa principal encontramos dispositivos capazes de armazenar o endereço do byte a ser gravado, o valor do byte propriamente dito e ainda diversos bits que controlam as muitas variáveis e modos de funcionamento que as diferentes memórias EPROM exigem.

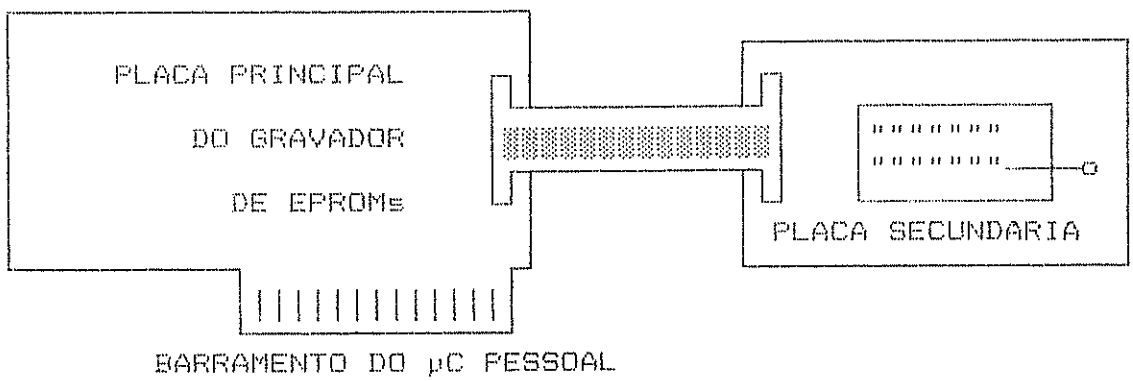


Figura 3.18 - Uma implementação comum em gravadores de EPROM comerciais

Visando conseguir melhor aproveitamento do hardware e maior versatilidade, optamos por projetar nosso gravador de EPROMs aproveitando as entradas e saídas da placa de interface paralela e temporização descrita no item 3.3. Esta implementação é ilustrada pela figura 3.19.

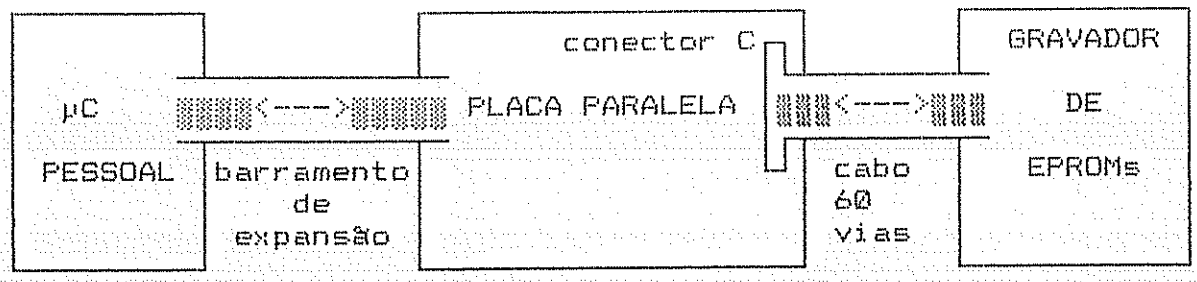


Figura 3.19 - Gravador de EPROMs como acessório da Placa de Interface Paralela e Temporização.

A placa do gravador de EPROMs pode ser encarada como um acessório da placa de interface paralela e temporização, sendo a ela conectada através de um cabo plano de 60 vias.

Diferindo um pouco dos gravadores de EPROM comerciais, temos nosso gravador de EPROMs fora do computador, juntamente com o soquete "zero-força".

O gravador é controlado por um programa interativo, descrito no item 3.4.3, executado no microcomputador pessoal. Para maior facilidade de operação, utilizamos a técnica de "menus", para evitar que o usuário tenha que decorar uma grande quantidade de comandos.

A figura 3.20 abaixo, mostra o diagrama de blocos do Gravador de EPROMs.

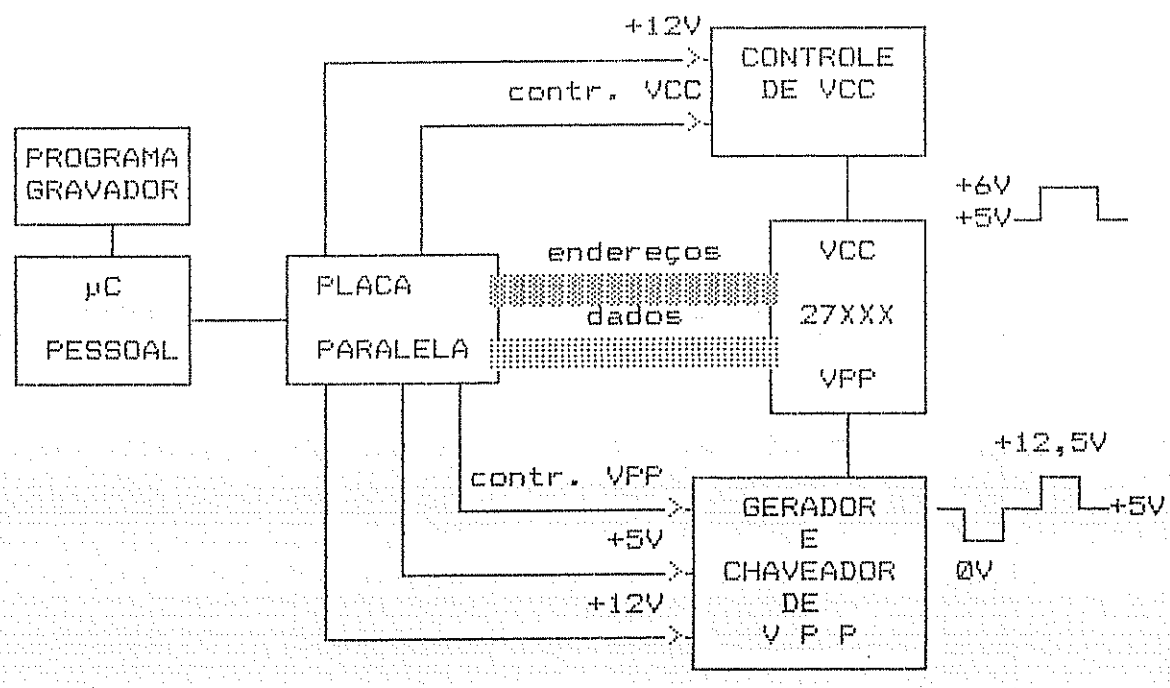


Figura 3.20 - Diagrama de blocos do Gravador de EPROMs.

3.4.2 - Descrição do gravador de EPROMs

Com a evolução da tecnologia dos fabricantes, alguns tipos de memórias são hoje considerados obsoletos e de difícil aquisição no comércio especializado. Não é necessário portanto construir um gravador de EPROMs que atenda todo o conjunto de tipos existente, já que quanto mais universal for um gravador de EPROMs, mais complexo e caro este se tornará.

Fazendo uma seleção das memórias utilizáveis, vamos nos ater aos tipos listados na tabela 3.5.

Memória	Capacidade
2764	8 K x 8
27128	16 K x 8
27256	32 K x 8
27512	64 K x 8

Tabela 3.5 - Memórias utilizáveis.

Com esta restrição o número de componentes e a área da placa do gravador serão reduzidos, e consequentemente, o custo final do equipamento.

A figura 3.21, a seguir, apresenta a pinagem das memórias utilizáveis neste gravador.

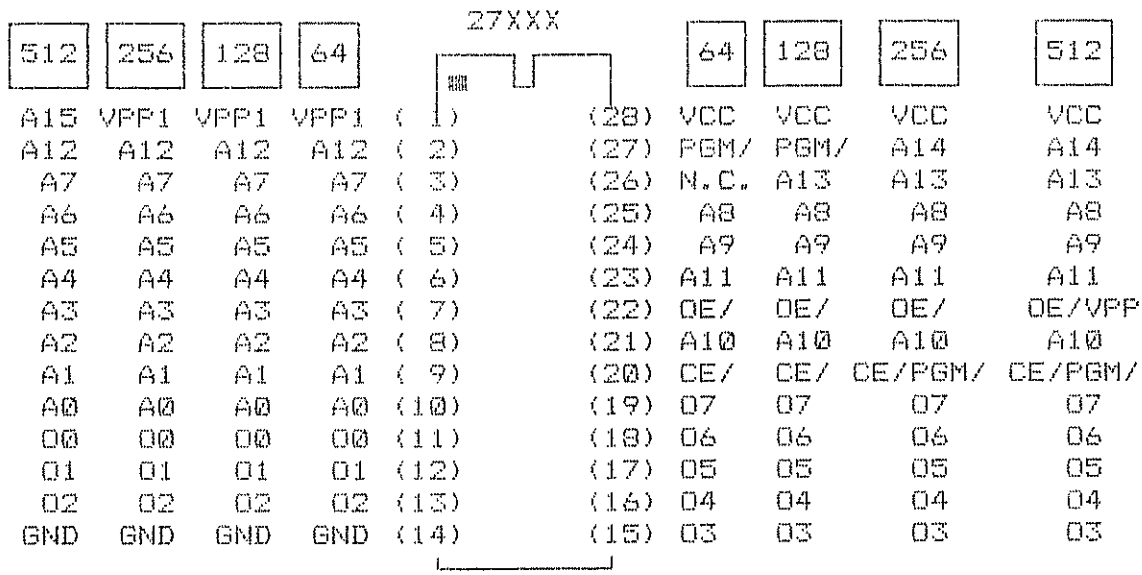


Figura 3.21 - Pinagem das memórias 2764/128/256/512.

Existem diversos fabricantes destas memórias; entretanto, para o desenvolvimento do projeto do gravador, baseamo-nos principalmente nos dados fornecidos no catálogo da INTEL (INTEL, 1988).

Como mostrado no diagrama de blocos da figura 3.20, utilizamos a placa de interface paralela e temporização como interface controladora do gravador de EPROMs. A tabela 3.6 e a figura 3.22 apresentam a alocação de bits das diversas portas disponíveis no conector "C" da placa de interface paralela e temporização, para as funções de endereçamento, controle e dados, necessárias à placa do gravador.

A chave DIP SW4 da placa de interface paralela e temporização deverá ter seus contatos fechados, de modo que o conector "C" possua o número de bits necessários para as funções citadas acima (30 bits).

CONECTOR C PINO	FUNÇÃO
2 - G N D	G N D
3 - + 5 V	máx 120 mA
5 - + 1 2 V	máx 400 mA
9 - porta A U1 bit 0	DADOS - D0
10 - porta A U1 bit 1	DADOS - D1
11 - porta A U1 bit 2	DADOS - D2
12 - porta A U1 bit 3	DADOS - D3
13 - porta A U1 bit 4	DADOS - D4
14 - porta A U1 bit 5	DADOS - D5
15 - porta A U1 bit 6	DADOS - D6
16 - porta A U1 bit 7	DADOS - D7
17 - porta B U1 bit 0	ENDEREÇO - A0
18 - porta B U1 bit 1	ENDEREÇO - A1
19 - porta B U1 bit 2	ENDEREÇO - A2
20 - porta B U1 bit 3	ENDEREÇO - A3
21 - porta B U1 bit 4	ENDEREÇO - A4
22 - porta B U1 bit 5	ENDEREÇO - A5
23 - porta B U1 bit 6	ENDEREÇO - A6
24 - porta B U1 bit 7	ENDEREÇO - A7
25 - porta C U1 bit 0	ENDEREÇO - A8
26 - porta C U1 bit 1	ENDEREÇO - A9
27 - porta C U1 bit 2	ENDEREÇO - A10
28 - porta C U1 bit 3	ENDEREÇO - A11
29 - porta C U1 bit 4	ENDEREÇO - A12
30 - porta C U1 bit 5	ENDEREÇO - A13
31 - porta C U1 bit 6	ENDEREÇO - A14_PGM/
32 - porta C U1 bit 7	ENDEREÇO - A15_VPP1
48 - porta C' U2 bit 0	CONTROLE - TTL 22
49 - porta C' U2 bit 1	CONTROLE - TTL 1
50 - porta C' U2 bit 2	CONTROLE - OE/_VPP22e
51 - porta C' U2 bit 3	CONTROLE - LIG_6V
52 - porta C' U2 bit 4	CONTROLE - HABILITA/
53 - porta C' U2 bit 5	CONTROLE - CE/_PGM/

Tabela 3.6 - Alocação de bits para o Gravador de EPROMs.

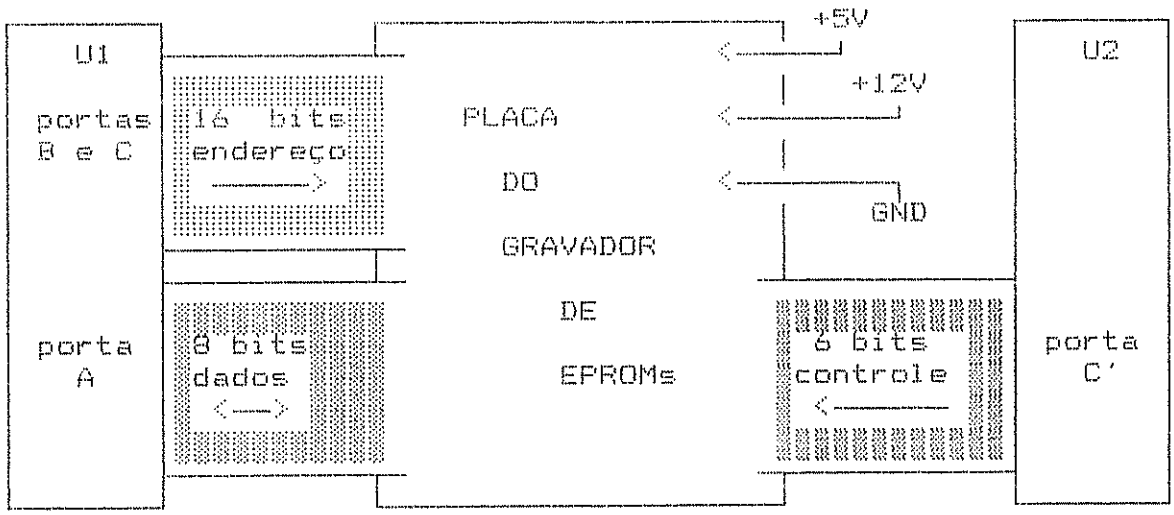


Figura 3.22 - Portas de controle do Gravador de EPROMs.

Para que possamos entender alguns dos aspectos dos gravadores de memórias EPROM, é importante que aprofundemos um pouco no papel das tensões Vpp (tensão de gravação) e Vcc (tensão de alimentação da memória) no processo de gravação.

A tabela 3.7 apresenta, comparativamente, algumas características de três processos utilizados para gravação de memórias EPROM.

Três objetivos são perseguidos por todos estes processos:

1 - usar um pulso de programação o mais curto possível

2 - garantir a retenção dos dados

3 - manter a "programabilidade" (possibilidade de ser regravada)

Para atingir estes objetivos, alguns artifícios são recomendados pelos fabricantes. Passaremos então a descrevê-los.

Método de gravação	Vpp	Vcc	Tempo mínimo p/ gravação		Velocidade relativa
			1 byte	64 Kbytes	
Tradicional	12V ±0,5V	5V ±0,25V	50ms	54,6 minutos	12X / 444X
Algoritmo Inteligente	12,5V ±0,5V	6V ±0,25V	(1ms+3ms)	4,4 minutos	37 X
Algoritmo de pulso rápido	12,75V ±0,25V	6,25V ±0,25V	0,1ms	0,11 minutos	

Tabela 3.7 - Características típicas de três processos de gravação de memórias EPROM

Considerando as colunas **"Vpp"** e **"Tempo mínimo p/ gravação"**, percebemos que, quanto mais aumentamos o valor de Vpp, menor o tempo necessário para conseguirmos uma gravação confiável.

Um bit gravado com "0" é caracterizado por um número suficiente de elétrons presos na porta flutuante do transistor FET que compõe a célula de memória (ver figura 3.23). Estes elétrons repelem os portadores minoritários do substrato tipo "P" impedindo a formação do canal quando a porta de seleção é polarizada positivamente. O transistor FET comporta-se portanto como uma chave desligada.

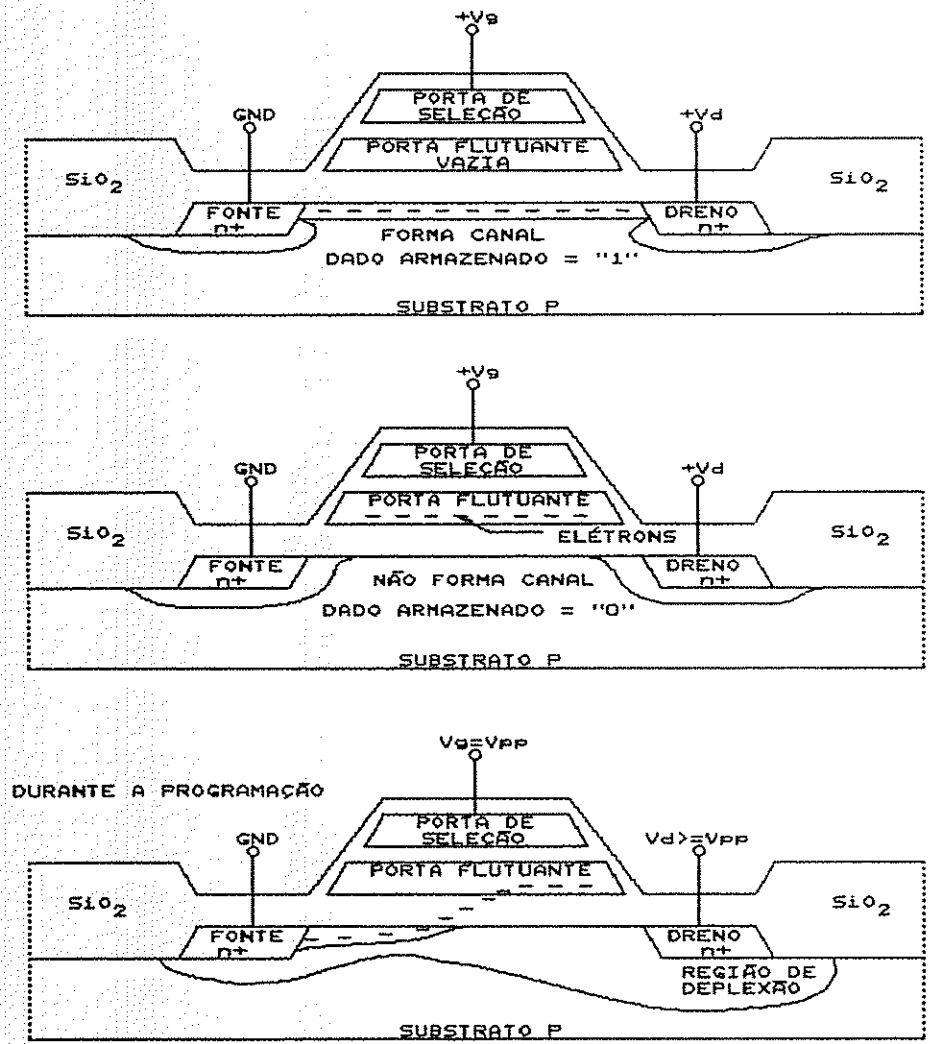


FIGURA 3.23 – CÉLULA DE MEMÓRIA EPROM

Durante a gravação, um aumento na tensão Vpp aumenta a energia suprida aos elétrons que atravessam o canal, aumentando a probabilidade de que um maior número deles atinja a porta flutuante. Isto acarreta num menor tempo para colocar um número suficiente de cargas nesta porta.

Pela observação da coluna "Vpp" da tabela 3.7, percebe-se a necessidade de uma fonte de tensão estabilizada, ajustável na faixa de 12V até 12,75V, para que possamos trabalhar com qualquer um dos métodos de gravação. A tensão (nominal) de 12V, disponível no barramento do microcomputador pessoal, não pode ser usada diretamente, pois possui uma tolerância de ±5%, podendo apresentar qualquer tensão entre 11,4V e 12,6V.

Para conseguirmos uma fonte de Vpp confiável, foi construído o gerador de Vpp, apresentado na figura AP.13 (Apêndice A). Este circuito é composto de um oscilador baseado no circuito integrado 555, transistores reforçadores e um dobrador de tensão com diodos e capacitores. Consegue-se obter, a partir dos +12V, uma tensão de +23,4V à vazio e cerca de 20,4V à 100mA. A freqüência de oscilação escolhida foi de 20KHz, permitindo um bom compromisso entre as perdas de comutação e a capacitância necessária para filtragem. Finalmente, um regulador eletrônico estabiliza o valor dessa tensão para o valor nominal de Vpp, conforme o método de gravação utilizado. O ajuste do valor de Vpp pode ser feito pelo usuário, através do trimpot multivoltas TP2, conforme indicações da tabela 3.8 e da figura 3.24.

Ajuste	Carga de teste	Ligar carga entre os pinos	Tensão na carga de teste
TP1 (Vcc)	100Ω 1/2W	28-14	Ajustar conforme colunas Vpp e Vcc da tabela 3.7
TP2 (Vpp)	390Ω 1/2W	1-14 ou 22-14	

Tabela 3.8 - Ajuste de tensões da placa do Gravador de EPROMs

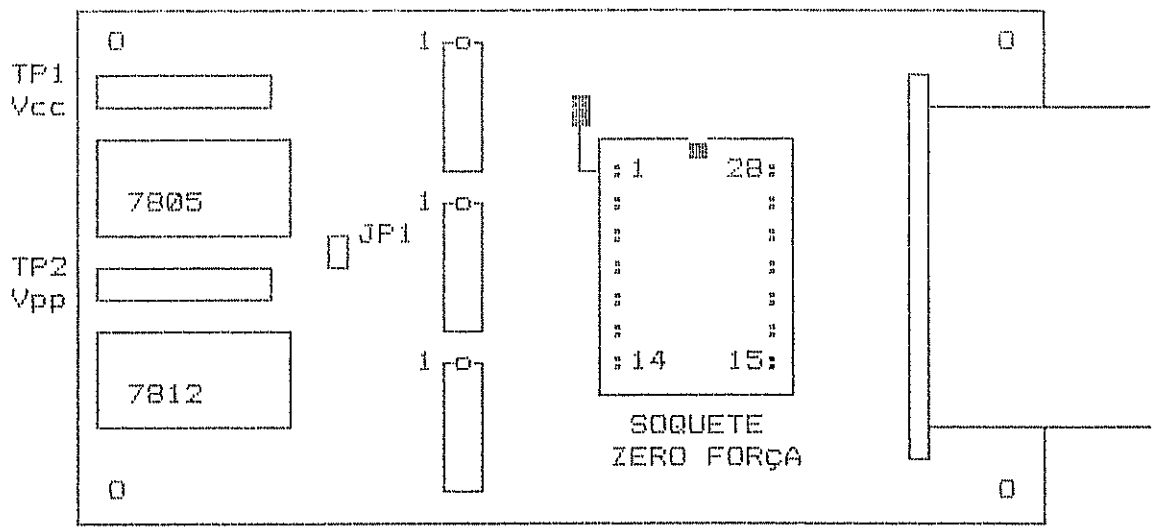


Figura 3.24 - Localização dos trimpots de ajuste das tensões do Gravador de EPROMs

A tensão de gravação, ajustada no gerador de Vpp pode assumir no máximo 13V. Esta tensão é insuficiente para se proceder à gravarção das versões mais antigas das memórias propostas na tabela 3.5, pois necessitam de 21V. Caso seja necessário gravar estas memórias, deve-se retirar o "jumper" "JP1" na placa do gravador e alimentar o circuito chaveador de Vpp com uma fonte externa.

Considerando agora as colunas **"Vcc"** e **"Tempo mínimo p/ gravação"** da tabela 3.7, vemos que, quanto menor o tempo gasto para gravar um byte, maior a tensão de Vcc requerida durante o processo de gravação e verificação.

Como vimos, os elétrons presos na porta flutuante tendem a dificultar a passagem de corrente através do canal do transistor FET (ver figura 3.23). Ao elevarmos o Vcc, existe uma tendência de aumentarmos a corrente neste canal, contrapondo-se ao efeito de repulsão dos elétrons. Sob estas condições, os bits marginalmente gravados são mais facilmente identificados e podem receber um reforço pela aplicação de um novo pulso de gravação.

Para comutarmos a tensão de Vcc, do valor normal (5V), para o valor mais elevado (6V), projetamos o circuito chaveador de Vcc, também apresentado na figura AP.13. Neste circuito, as duas tensões fornecidas são derivadas da tensão de 12V do barramento. O ajuste do valor mais elevado de Vcc pode ser feito pelo trimpot multivoltas "PT1", conforme a tabela 3.8 e figura 3.24.

Os pinos 1 e 22 do soquete "zero-força" possuem múltipla função. Ora funcionam como entradas padrão "TTL", trabalhando com níveis lógicos "0" e "1", ora como entradas de tensão de gravação, operando com aproximadamente +12,5V. Os circuitos que comandam as tensões nesses pinos (A15_Vpp e OE/Vpp) são apresentados na figura AP.14 (Apêndice A).

Cada transistor funciona como uma chave que é aberta ou fechada sob o comando do decodificador. De um lado dos terminais das chaves, temos as tensões correspondentes a GND, Vcc e Vpp. O outro lado de cada uma das chaves é ligado em comum e alimenta o pino de múltipla função (ver figura 3.25).

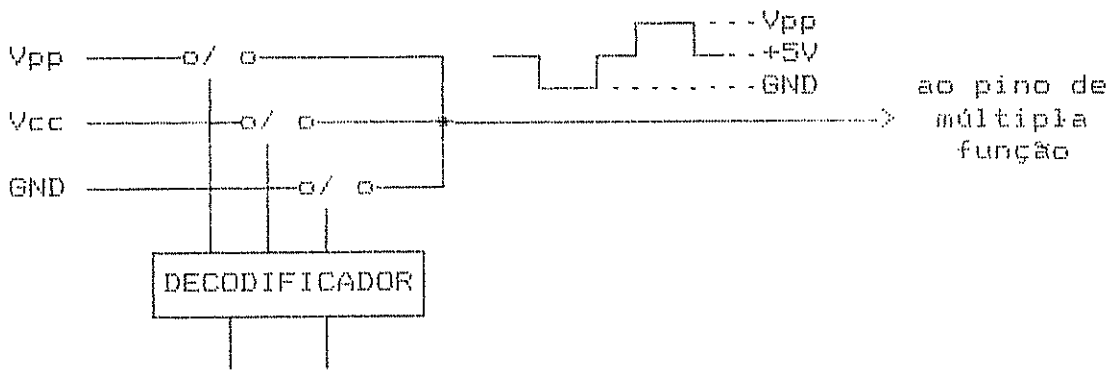


Figura 3.25 - Circuito chaveador de tensões

A figura AP.15 (Apêndice A), mostra as ligações ao soquete da EPROM.

3.4.3 - Programa GRAVADOR

O programa GRAVADOR foi desenvolvido usando a linguagem Pascal, devendo ser executado no microcomputador pessoal. Este programa, juntamente com a placa de interface paralela e temporização e a placa do gravador de EPROMs, permite gravar quatro tipos de memórias: 2764, 27128, 27256 e 27512.

Para gravação destas memórias, estamos utilizando o "Algoritmo Inteligente", cujas características básicas foram apresentadas na tabela 3.7. Embora este algoritmo apresente somente uma eficiência média, é facilmente implementado com a liguagem Pascal, o que levou à sua utilização.

A figura 3.26 apresenta a tela inicial do programa GRAVADOR. Os caracteres impressos em negrito indicam os diversos campos existentes para a entrada dos parâmetros necessários. O cursor piscante indica na tela qual é o campo ativo para entrada destes parâmetros. A mudança do campo ativo é comandada pelas teclas de seta à esquerda e seta à direita ("<-" e "->").

Gravador de EPROM	1.0	Tipo de memória:2764 (64/128/256/512)	
Arquivos de Trabalho		Endereços de Trabalho	
Fonte	Destino	Eprom	Buffer
????????.???	????????.???	Inicial: 0000 Final: 1FFF	Inicial: 0000 Final: 1FFF
		Função:	
0-Copia eprom ——>buffer 1-Grava buffer ——>eprom 2-Copia arquivo——>buffer 3-Grava buffer ——>arquivo 4-Compara eprom<——>buffer 5-Verifica eprom apagada 6-Assinatura eprom/buffer 7-Lista conteúdo do buffer		8-Liga/Desliga impressora(DESL) 9-Edita conteúdo do buffer A-Procura bloco no buffer B-Move bloco no buffer C-Conversores Mem<——>Intel D-Diretório(A:\) E-Enche buffer com byte F-Terminal KD-16	
<->Muda o campo ESC-Abandona programa ENTER-Inicia função			

Figura 3.26 - Tela de entrada do programa GRAVADOR

Para melhor entendimento da tela inicial vamos dividi-la em cinco blocos de trabalho (ver figura 3.27).

1)- Bloco de seleção da EPROM: serve para apresentação e seleção dos tipos de memórias a serem trabalhadas.

2)- Bloco de Arquivos de trabalho: serve para especificarmos os nomes dos arquivos em disco que servirão para carregar e descarregar dados do buffer de trabalho.

3)- Bloco de endereços de trabalho: serve para especificar intervalos de endereçamento na EPROM e no buffer, dentro dos quais atuarão as diversas funções selecionadas pelo usuário.

4)- Bloco de seleção de funções: serve para especificação de uma das 16 funções disponíveis.

5)- Bloco de informações ao usuário: serve para interagir com o usuário mostrando opções de operação, requisitando parâmetros, listando dados, apontando erros de operação e fornecendo informações importantes.

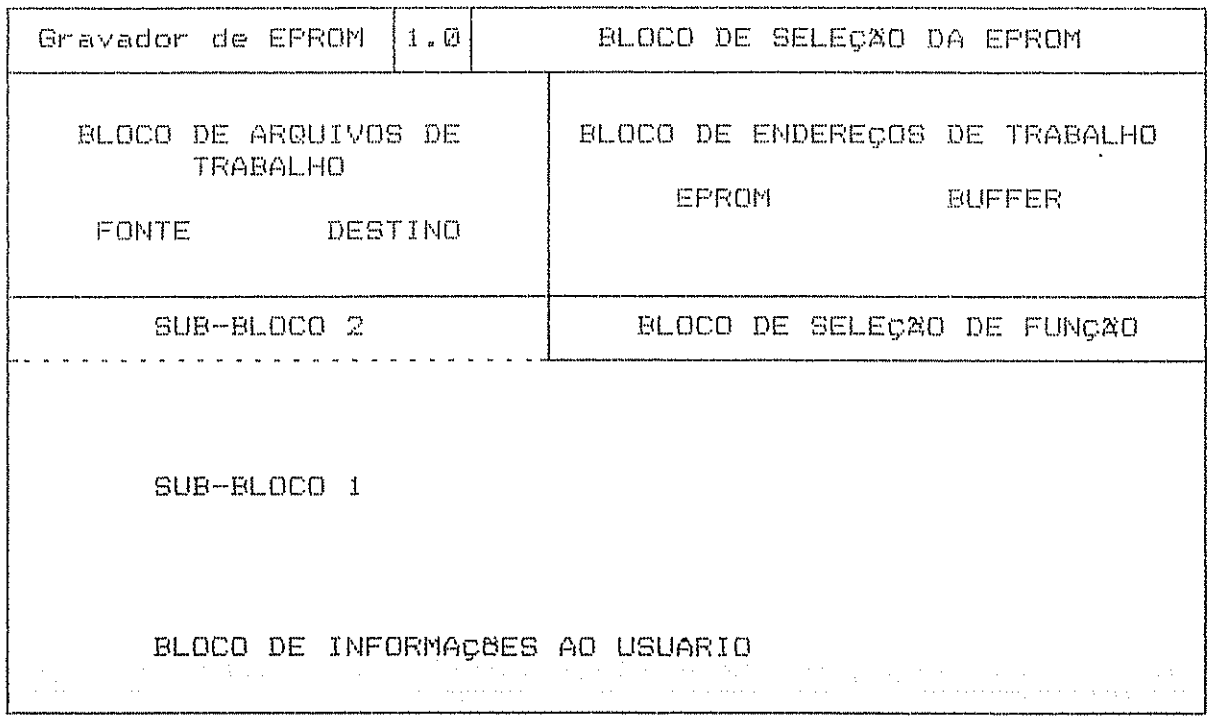


Figura 3.27 - Tela do programa GRAVADOR dividida em blocos

O buffer de trabalho citado acima é usado como área intermediária para armazenamento de dados. Ele é na verdade uma variável indexada unidimencional da linguagem Pascal. Cada termo desta variável foi definido como sendo um byte. Esta implementação é bastante simples, porém acarreta um inconveniente: este buffer compartilha um espaço dentro do segmento de dados ao lado de outras variáveis do programa GRAVADOR; conseqüentemente seu tamanho máximo não pode ser de 64Kbytes que seriam necessários para armazenar todo o conteúdo de uma me-

mória 27512. Os endereços limites do buffer são "0000H" e "F000H", correspondendo a 61441 bytes. Qualquer função que tente operar acima do limite superior (F000H) implicará em uma mensagem de erro, que será apresentada no bloco de informações ao usuário.

Os limites iniciais e finais dos endereços de trabalho, tanto da EPROM, quanto do buffer, são especificados nos campos especialmente reservados para isto, após as inscrições "Inicial:" e "Final:". Estes limites definem áreas sujeitas à atuação das funções disponíveis no programa GRAVADOR. Os bytes correspondentes aos dois endereços limites também estão inclusos nas áreas de atuação das funções (ver figura 3.28).

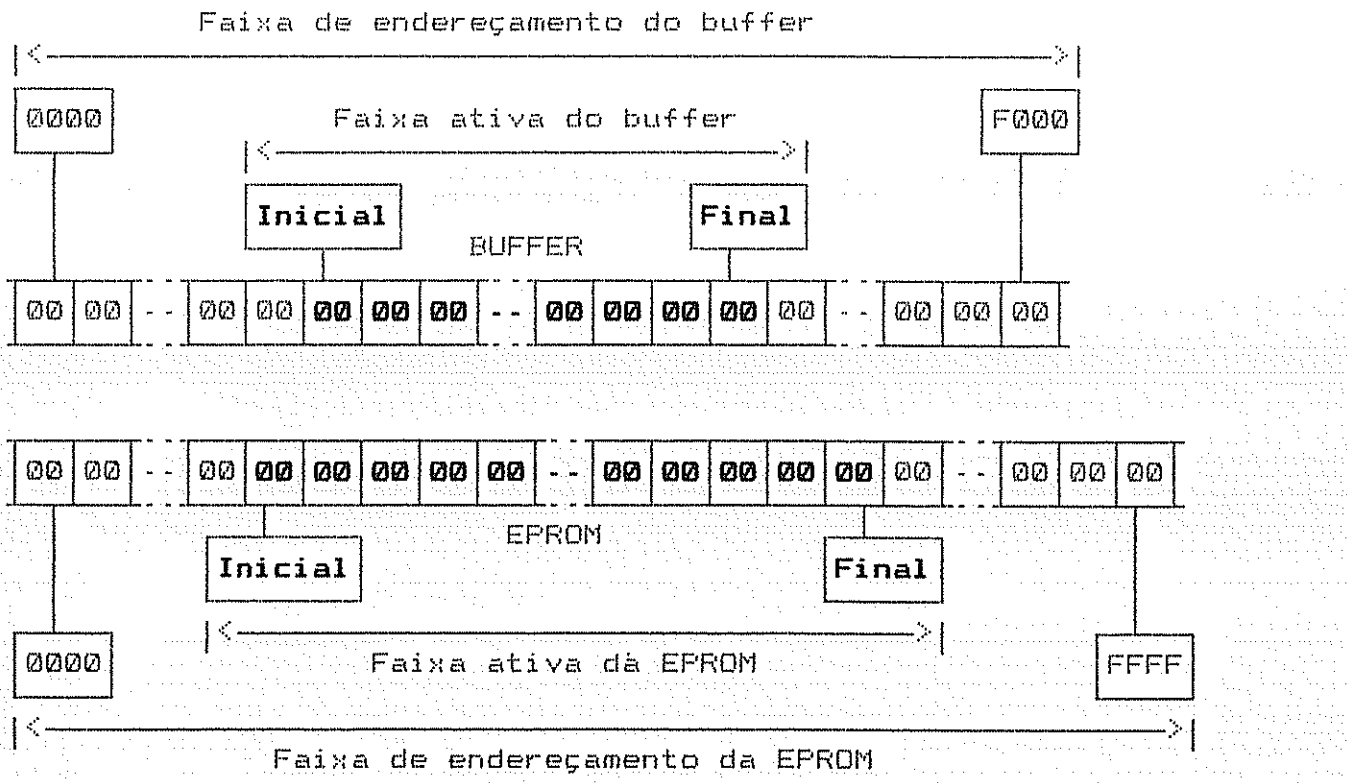


Figura 3.28 - Areas de trabalho na EPROM e no Buffer

A seleção do tipo de memória a ser gravado é feita deslocando-se o campo ativo para o bloco de seleção da EPROM e em seguida, teclando o primeiro algarismo de qualquer dos quatro números mostrados entre parênteses (64/128/256/512). Ao teclarmos um destes quatro algarismos, automaticamente, teremos os limites dos endereços de trabalho da EPROM e do buffer ajustados para conter o número de bytes correspondentes a cada tipo de EPROM. Caso não desejemos atuar sobre toda a faixa ativada automaticamente, podemos alterar novamente estes limites conforme a necessidade.

Para especificarmos os nomes dos arquivos de trabalho, também será necessário movimentarmos o cursor para o campo correspondente. A tecla "ESPAÇO" apagará as interrogações presentes nestes campos, habilitando a entrada de um nome de arquivo aceito pelo DOS. Se o arquivo especificado for um arquivo fonte, ele deverá estar presente no diretório corrente. Caso não exista este nome no diretório corrente, será emitida uma mensagem de erro. Se for um arquivo destino, ele será aberto pelo próprio programa GRAVADOR, desde que haja espaço suficiente no disco para conter o bloco a ser transferido.

A seleção de uma das funções é habilitada quando o campo ativo é o bloco de seleção de funções. O acionamento de uma das teclas de "0" até "F", causará a impressão da função escolhida dentro do quadro intitulado "Função:". Esta função permanecerá selecionada até que outra função seja escolhida pelo usuário. A tecla "ENTER" dará início a execução da função selecionada.

3.4.3.1 - Descrição das funções do programa GRAVADOR

Neste item descreveremos cada uma das funções do programa GRAVADOR e apresentaremos as possíveis mensagens geradas. As funções oferecidas são as seguintes:

0 - Copia EPROM no buffer;

1 - Grava buffer na EPROM;

2 - Copia arquivo no buffer;

3 - Grava buffer no arquivo;

4 - Compara EPROM com buffer;

5 - Verifica EPROM apagada;

6 - Fornece "Assinatura" da EPROM e do buffer;

7 - Lista conteúdo do buffer;

8 - Liga/desliga a impressora;

9 - Edita conteúdo do buffer;

A - Procura bloco no buffer;

B - Move bloco no buffer;

C - Executa conversão de arquivos:

- do formato imagem de memória para formato INTEL;

- do formato INTEL para formato imagem de memória;

D - Verifica/Troca o Diretório corrente;

E - Enche o buffer com um valor hexadecimal especificado;

F - Executa o programa "Terminal".

0-COPIA EPROM ——> BUFFER: O conteúdo da área ativa da EPROM, compreendido entre o endereço inicial e final, será copiado no buffer, a partir de seu endereço inicial especificado (ver figura 3.29).

Possíveis mensagens:

- ERRO: BLOCO DA EPROM > BLOCO DO BUFFER
- BLOCO DA EPROM COPIADO NO BUFFER

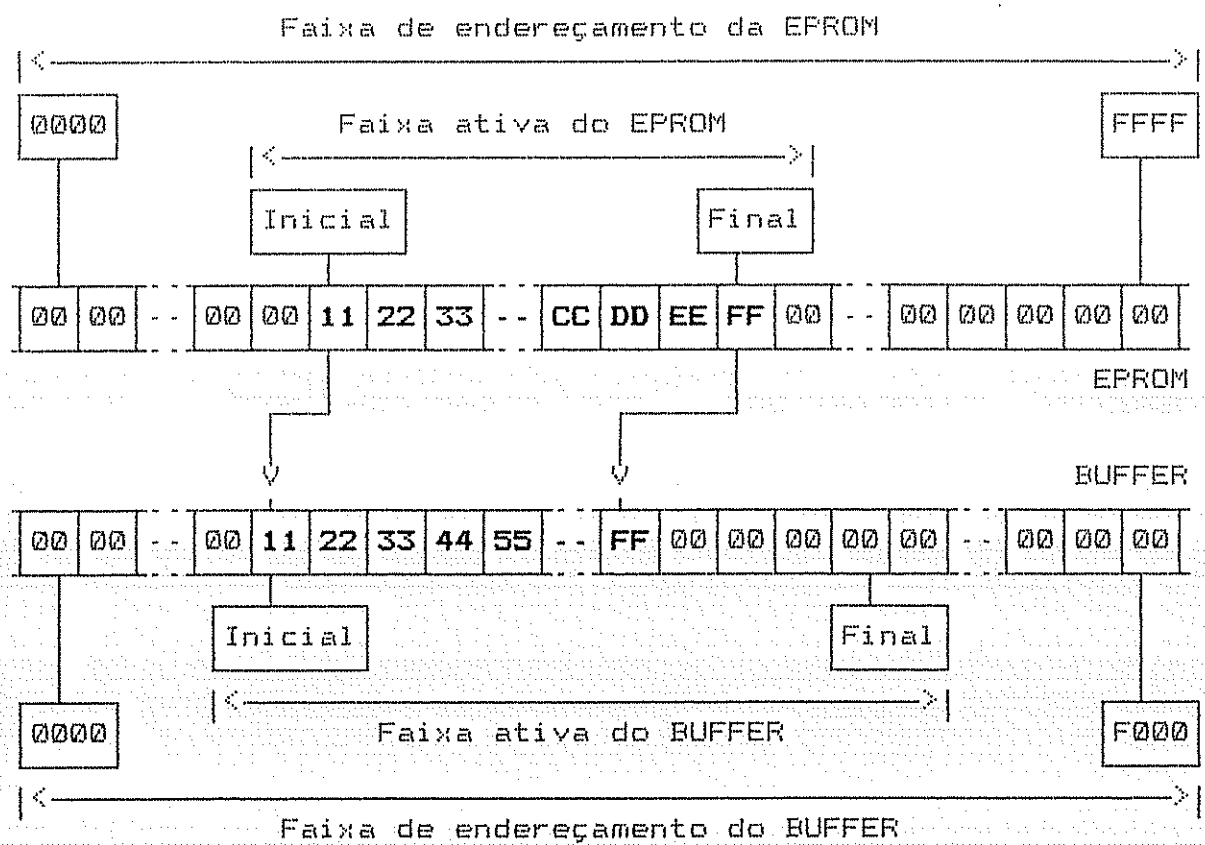


Figura 3.29 - Função "COPIAR EPROM ——> BUFFER"

1-GRAVA BUFFER ——> EPROM: O conteúdo da área ativa do buffer, compreendido entre o endereço inicial e final, será gravado na EPROM, a partir de seu endereço inicial especificado (ver figura 3.30).

Possíveis mensagens:

- ERRO: BLOCO DO BUFFER > BLOCO DA EPROM
- ERRO: GRAVAÇÃO NÃO OK!
- GRAVAÇÃO CONCLUIDA E OK!

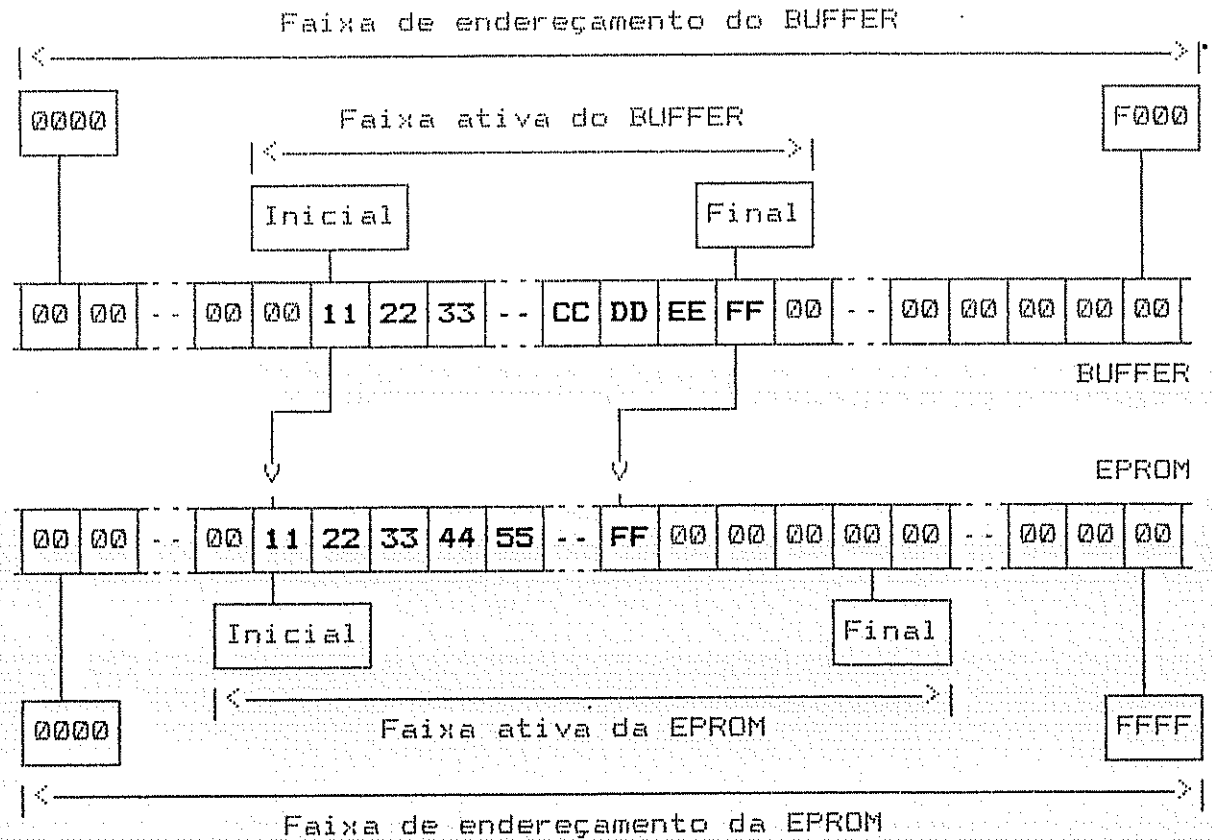


Figura 3.30 - Função "GRAVAR BUFFER ——> EPROM"

OBS: - O sub-bloco 2 de informações ao usuário, apresentará o número de bytes a serem gravados.

- Caso alguma das células de memória não retenha o byte correspondente após 25 tentativas de gravação, será emitida uma mensagem de erro, e a função será abortada.

2-COPIA ARQUIVO NO BUFFER: O arquivo fonte, especificado no bloco de arquivos de trabalho, será carregado no buffer a partir de seu endereço inicial especificado (ver figura 3.31).

O conteúdo de um arquivo é acessado seqüêncialmente byte a byte do primeiro ao último. Para possibilitar que os bytes iniciais de um arquivo sejam descartados, é possível especificar um deslocamento, que posicionará o ponteiro de acesso um certo número de bytes à frente da seqüência normal de leitura. O programa GRAVADOR permite que o usuário especifique qual é o valor do deslocamento desejado. O valor pré-estabelecido para este deslocamento é "0000", o que implica na transferência de todo o conteúdo do arquivo para o buffer.

Qualquer problema na abertura de um arquivo implicará numa mensagem de erro ao usuário.

Possíveis mensagens:

- ERRO: NOME DO ARQUIVO INVALIDO
- ENTRE COM O DESLOCAMENTO (HEX)
- ERRO NA ABERTURA DO ARQUIVO
  VERIFIQUE SE O ARQ. EXISTE NO DIRETORIO CORRENTE
- ERRO: DESLOCAMENTO INCORRETO
- ERRO: ARQUIVO > BLOCO DO BUFFER
- CARGA DO ARQUIVO <**NOME**> FOI CONCLUIDA E OK!

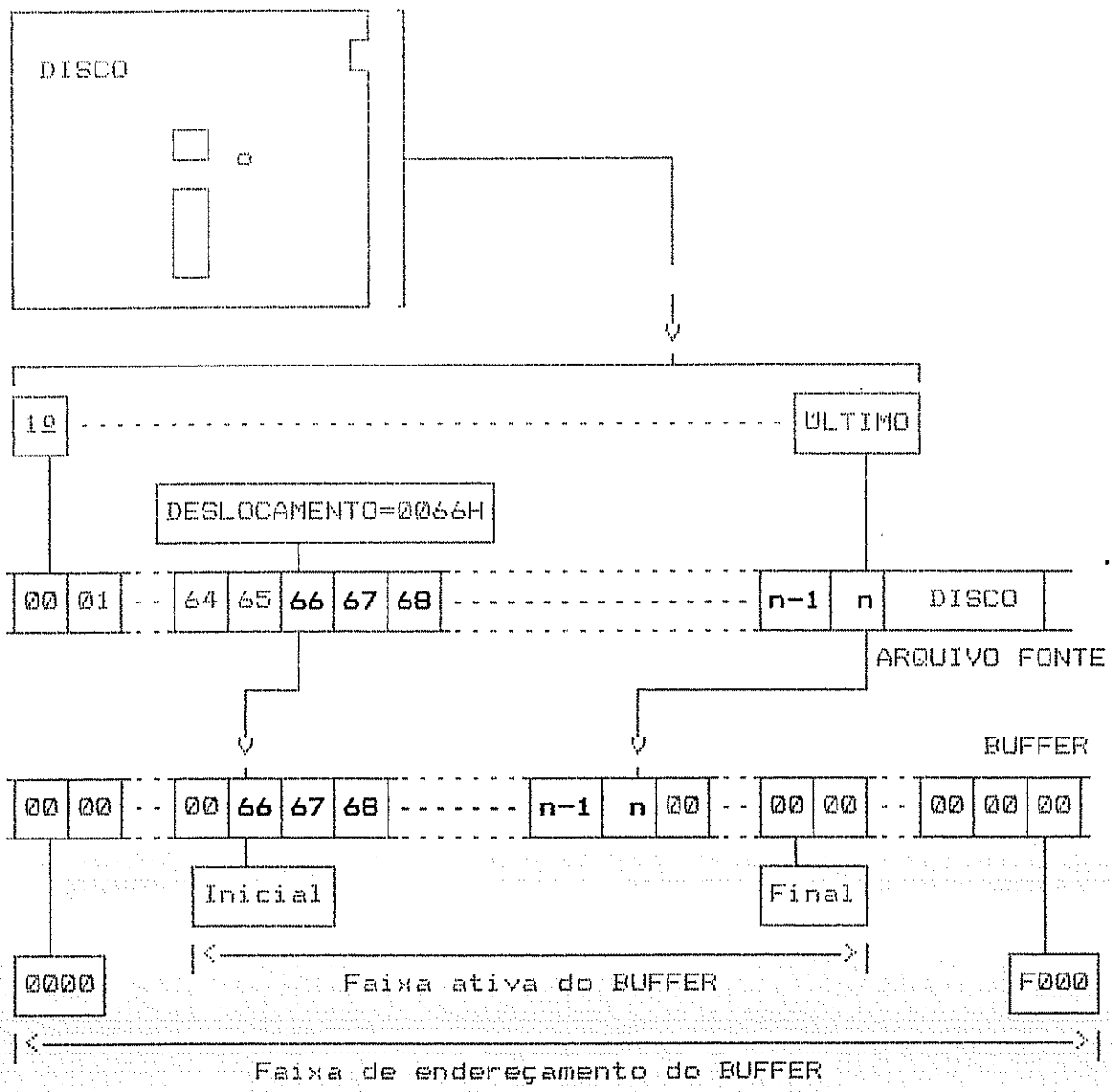


Figura 3.31 - Função "COPIA ARQUIVO ——> BUFFER"

3-GRAVA BUFFER ——> ARQUIVO: O conteúdo da área ativa do buffer, compreendido entre o endereço inicial e final, será gravado no arquivo destino, cujo nome está definido no bloco de arquivos de trabalho (ver figura 3.32).

Possíveis mensagens:

-ERRO: NOME DO ARQUIVO INVALIDO!

-ERRO NA ABERTURA DO ARQUIVO!
VERIFIQUE SE O ARQ. ESTA PROTEGIDO OU SE O DISCO ESTA CHEIO

-GRAVAÇÃO DO BLOCO DE BUFFER PARA O ARQUIVO <NOME> OK!

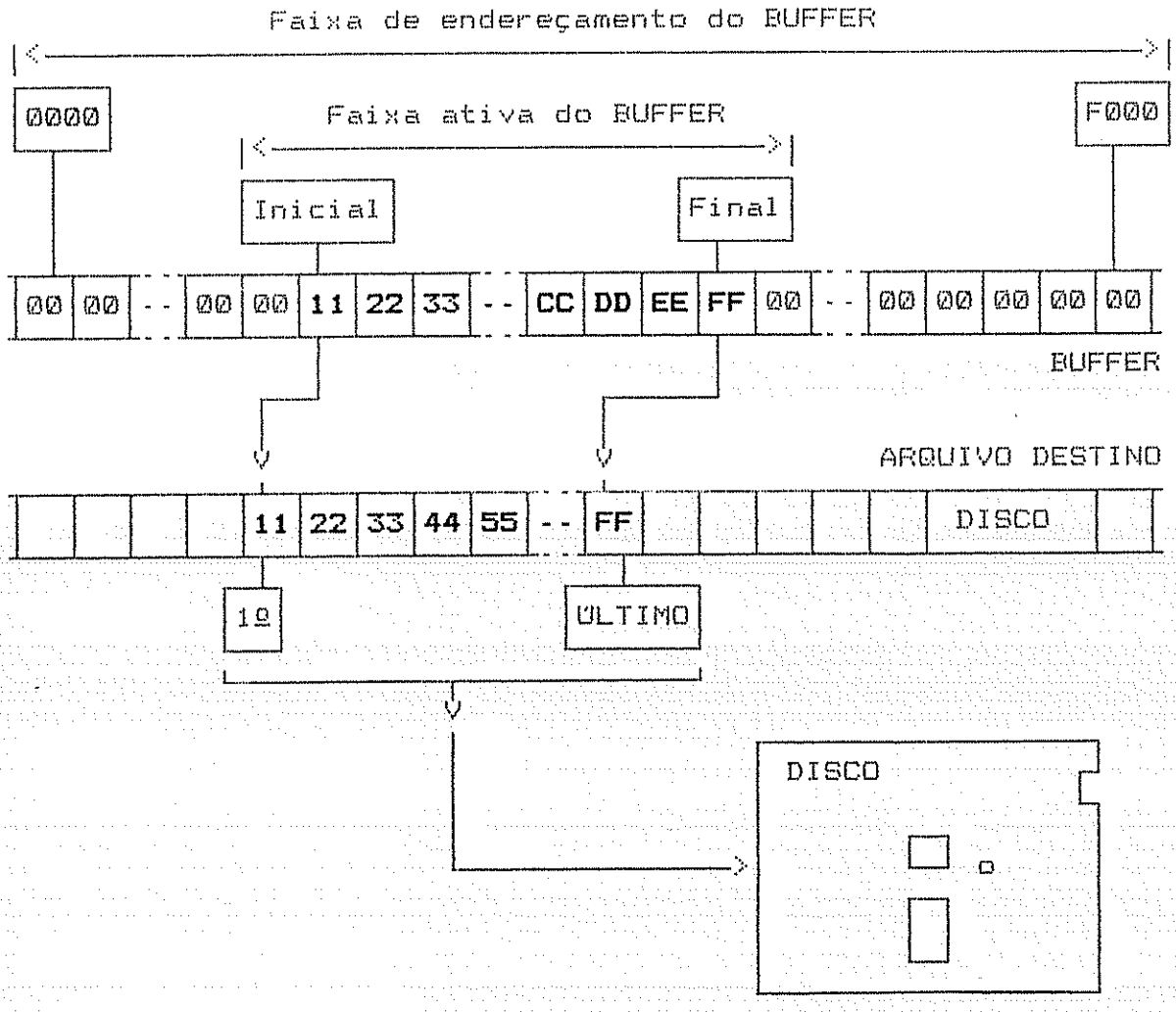


Figura 3.32 - Função "GRAVA BUFFER ——> ARQUIVO"

4-COMPARA EPROM <—> BUFFER: Compara byte a byte o conteúdo da memória EPROM com o conteúdo do buffer. O número de bytes dos dois blocos a serem comparados devem ser iguais (ver figura 3.33). As diferenças encontradas serão listadas na tela.

Possíveis mensagens:

-ERRO: TAMANHO DIFERENTE DOS BLOCOS!

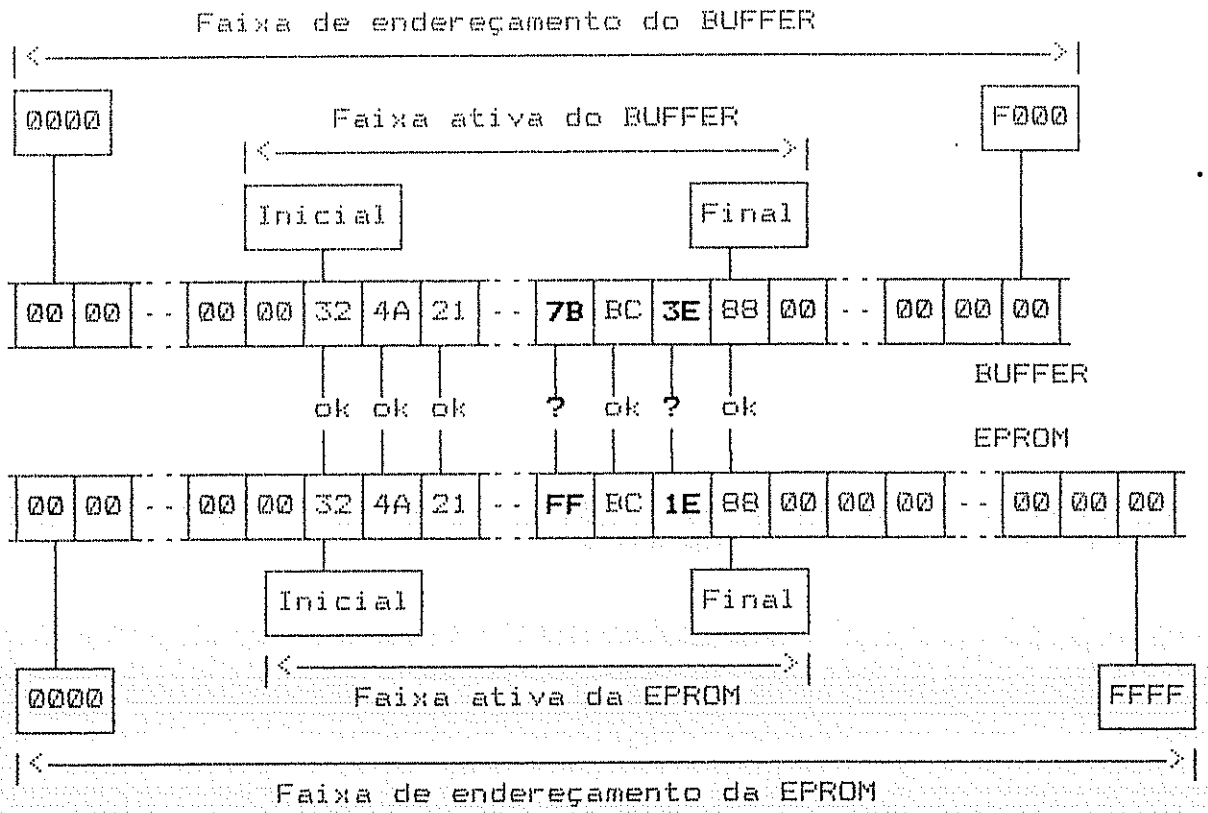


Figura 3.33 - Função compara EPROM <--> BUFFER.

5-VERIFICA EPROM APAGADA: Verifica se cada byte compreendido entre o endereço inicial e final é igual a "FF" (ver figura 3.34).

Possíveis mensagens:

- ESTA EPROM NÃO ESTA TOTALMENTE APAGADA
- ESTA EPROM ESTA APAGADA NO BLOCO ESPECIFICADO

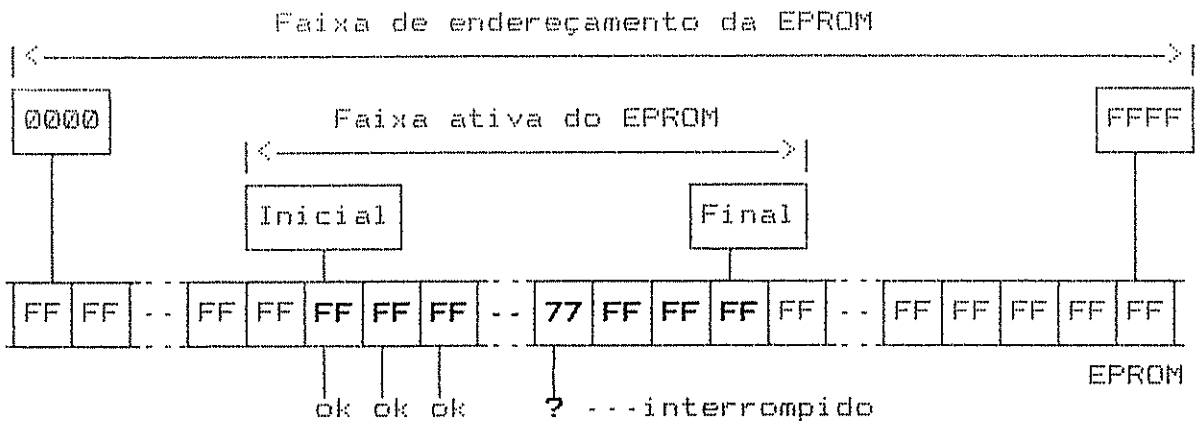


Figura 3.34 - Função "VERIFICA EPROM APAGADA"

6-ASSINATURA EPROM/BUFFER: Esta função fornece dois números hexadecimais, um para a EPROM e outro para o buffer, os quais refletem o conteúdo das faixas ativas de cada um destes dispositivos. Estes números constituem uma maneira rápida de verificar e/ou identificar as memórias gravadas (ver figura 3.35).

Possíveis mensagens:

- ASSINATURA DA EPROM = <VALOR>

  ASSINATURA DO BUFFER = <VALOR>

OBS: Se a soma dos valores dos bytes do buffer ou da EPROM excederem a capacidade de representação possível com um byte, os bits de "vai um" serão desprezados.

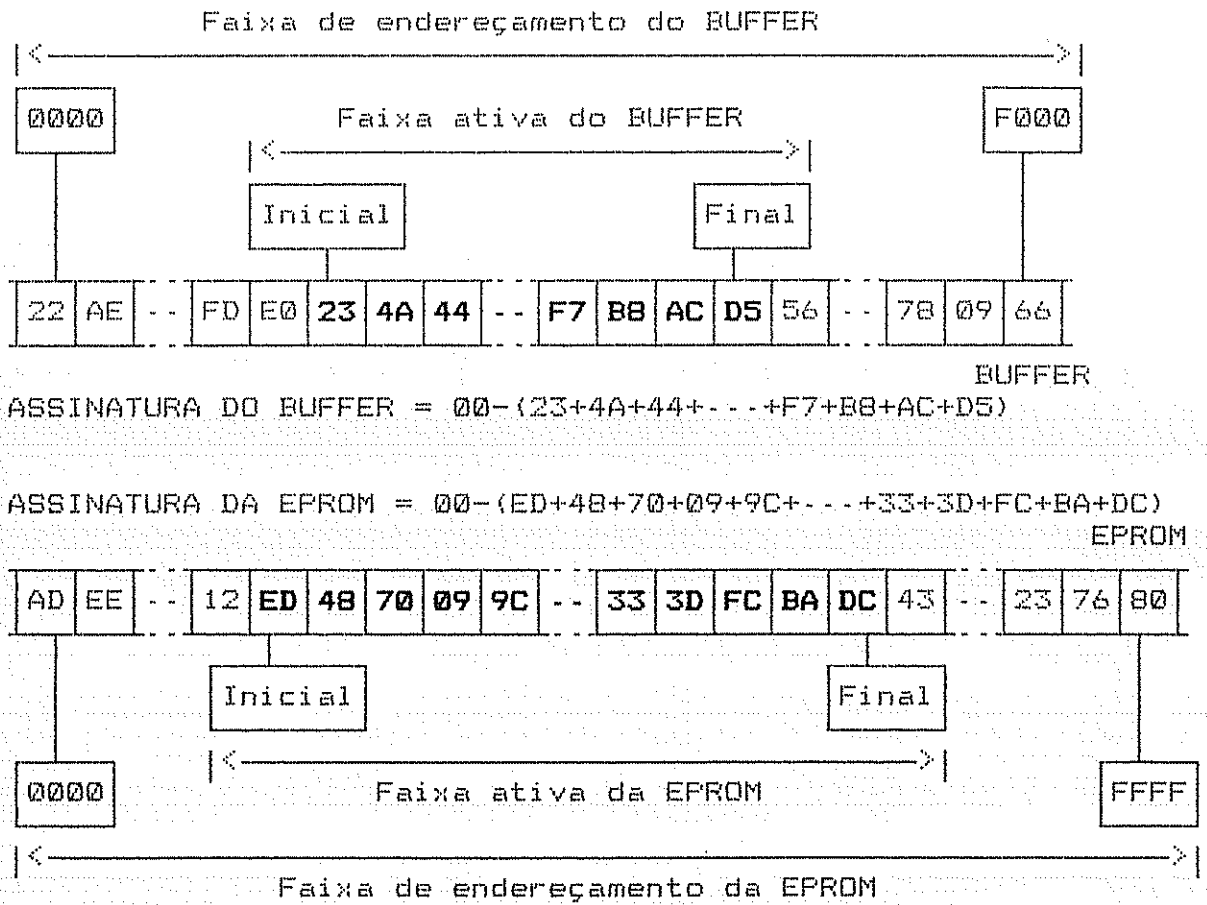


Figura 3.35 - Função "ASSINATURA EPROM/BUFFER"

7-LISTA CONTEÚDO DO BUFFER: Lista o conteúdo da faixa ativa do buffer mostrando 16 bytes por linha. Estes bytes são mostrados na forma hexadecimal e em ASCII (ver figura 3.36).

Gravador de EPROM	1.0	Tipo de memória:2764 (64/128/256/512)	
Arquivos de Trabalho		Endereços de Trabalho	
Fonte	Destino	Eprom	Buffer
????????.???	????????.???	Inicial: 0000 Final: 1FFF	Inicial: 0000 Final: 1FFF
ESC P/ PARAR		Função:7-LISTA CONTEÚDO DO BUFFER	

```
END. 00 01 02 03 ----- 09 0A 0B 0C 0D 0E 0F   0123456789ABCDF
0000 30 30 30 30 ----- 30 30 30 30 30 30 30   0000-----000000
0010 31 31 31 31 ----- 31 31 31 31 31 31 31   1111-----111111
0020 72 72 72 72 ----- 72 72 72 72 72 72 72   rrrr-----rrrrrr
0030 40 40 40 40 ----- 40 40 40 40 40 40 40   @@@@-----@@@@@@
0040 00 00 00 00 ----- 00 00 00 00 00 00 00   ..........-.....
0050 2B 2B 2B 2B ----- 2B 2B 2B 2B 2B 2B 2B   ++++-----++++++
0060 41 41 41 41 ----- 41 41 41 41 41 41 41   AAAA-----AAAAAA
0070 51 51 51 51 ----- 51 51 51 51 51 51 51   QQQQ-----QQQQQQ
0080 61 61 61 61 ----- 61 61 61 61 61 61 61   aaaa-----aaaaaa
```

Figura 3.36 - Função "LISTA CONTEÚDO DO BUFFER"

8-LIGA/DESLIGA IMPRESSORA: Deslocando-se o campo ativo para o bloco de seleção de funções e teclando-se o número "8", estaremos selecionando a função que habilita ligar e desligar a operação da impressora. A cada acionamento da tecla "ENTER", comutamos o estado da impressora de desligada para ligada e vice-versa.

9-EDITA CONTEÚDO DO BUFFER: Habilita a alteração, byte a byte, do conteúdo da faixa ativa do buffer (ver figura 3.37).

Após a ativação da função, deve-se acionar a tecla "ENTER", o que provocará a apresentação do 1º endereço da área de ativa do buffer, o valor hexadecimal do byte neste endereço e o código ASCII correspondente. Qualquer tecla que corresponda a um dígito hexadecimal válido, modificará o valor presente naquele endereço apresentado. A tecla "ENTER" causará o avanço para a próxima posição dentro do buffer. A tecla "ESC" provocará o abandono da edição.

Gravador de EPROM	1.0	Tipo de memória:27**64** (64/128/256/512)	
Arquivos de Trabalho		Endereços de Trabalho	
Fonte	Destino	Eprom	Buffer
????????.???	????????.???	Inicial: **0000** Final: **1FFF**	Inicial: **0000** Final: **1FFF**
< ESC > ABANDONA		**Função:9-Edita conteúdo do buffer**	

ENDEREÇO	HEX	ASCII	HEX	ASCII
0000	00	.	30	0
0001	00	.	31	1
0002	00	.	32	2
0003	00	.	33	3
0004	00	.	34	4
0005	00	.	35	5
0006	00	.	36	6
0007	00	.	37	7
0008	00	.	38	8

Figura 3.37 - Função "EDITA CONTEÚDO DO BUFFER"

A-PROCURA BLOCO NO BUFFER: Procura uma seqüência hexadecimal ou ASCII dentro da área ativa do buffer.

Se a seqüência existir, esta função fornece o endereço do primeiro byte ou caractere da seqüência encontrada.

Possíveis mensagens:

- ESCOLHA O TIPO DE PROCURA:

  H - SEQÜÊNCIA HEX

  A - SEQÜÊNCIA ASCII

- ENTRE COM A SEQÜÊNCIA DE CARACTERES ASCII:

  (NO MAX. 16 CARACTERES)

  TERMINE COM < ENTER >

- ENTRE COM A SEQÜÊNCIA DE BYTES HEX:

  (NO MAX. 16 CARACTERES)

  TERMINE COM < ENTER >

- PROCURANDO...

- SEQÜÊNCIA PROCURADA FOI ENCONTRADA A PARTIR DO

  ENDEREÇO: <END.>

- SEQÜÊNCIA < NÃO > ENCONTRADA

B-MOVE BLOCO NO BUFFER: Permite mover N bytes dentro da área ativa do buffer (ver figura 3.38).

Possíveis mensagens:

- CARACTERE INVALIDO
- ERRO: PARÂMETROS INVALIDOS!
- TRANSFERÊNCIA REALIZADA!

A figura 3.38 apresenta o pedido de movimentação de "FF" bytes do endereço "0000H", para "1000H", dentro do buffer.

Gravador de EPROM	1.0	Tipo de memória:2764 (64/128/256/512)	
Arquivos de Trabalho		Endereços de Trabalho	
Fonte	Destino	Eprom	Buffer
????????.???	????????.???	Inicial: 0000 Final: 1FFF	Inicial: 0000 Final: 1FFF
		Função:B-Move bloco no buffer	

ENTRE COM:<END.INIC.ORIGEM>,<END.INIC.DESTINO>,<Nº DE BYTES>
VALORES EM HEX. TERMINE ENTRADA DE PARÂMETROS COM <ENTER>
0000,1000,FF

Figura 3.38 - Função "MOVE BLOCO NO BUFFER"

C-CONVERSORES MEM <—> INTEL: Permite que o usuário selecione e execute, de dentro do programa GRAVADOR, os programas conversores de arquivos (ver figura 3.39).

Os nomes especificados no bloco de arquivos de trabalho, serão utilizados como parâmetros de entrada nos programas requisitados.

Os programas "Memintel" e "Intelmem" serão buscados no diretório que estava ativo por ocasião da chamada do programa GRAVADOR. Este procedimento permite que todos os programas utilitários fiquem concentrados em um único disquete.

Gravador de EPROM	1.0	Tipo de memória:2764 (64/128/256/512)	
Arquivos de Trabalho		Endereços de Trabalho	
Fonte	Destino	Eprom	Buffer
NOME.MEM	NOME.INT	Inicial: 0000 Final: 1FFF	Inicial: 0000 Final: 1FFF
		Função:C-Conversores Mem<-->Intel	

OS PROGRAMAS CONVERSORES SERÃO CARREGADOS DO DIRETORIO **A:**

ESCOLHA O TIPO DE CONVERSOR:

I - Intel ---> Mem
M - Mem ---> Intel

<ESPAÇO> Abandona comando

Figura 3.39 - Função "CONVERSORES MEM <—> INTEL"

D-DIRETORIO: Permite que o usuário verifique o conteúdo do diretório corrente ou que especifique um outro (ver figura 3.40).

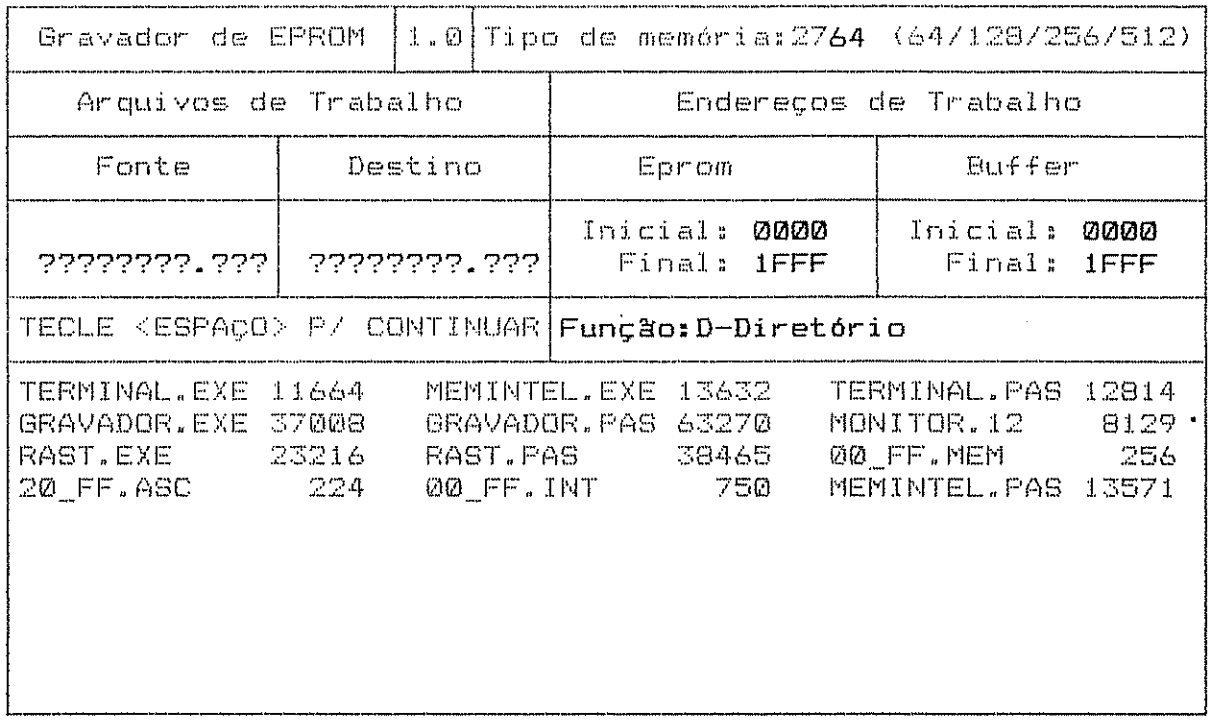


Figura 3.40 - Função "DIRETORIO"

E-ENCHE BUFFER COM BYTE: Permite que a área ativa do buffer seja preenchida com o valor hexadecimal especificado pelo usuário.

Possíveis mensagens:

- ENTRE COM O BYTE (HEX): <BYTE>
- BLOCO DE BUFFER CHEIO COM <BYTE>

  TECLE <ESPAÇO> PARA VOLTAR AO MENU

F-TERMINAL KD-16: Permite que executemos o programa TERMINAL, sem sair do programa GRAVADOR.

3.5 - Placa rastreadora

3.5.1 - Introdução

Durante a fase de teste e integração do software e hardware de um sistema de controle implementado com o KD-16, ou mesmo durante o trabalho de manutenção deste, é importante que o usuário tenha condições de saber precisamente o que está acontecendo, a qualquer momento, no barramento do seu micro-computador. Este acompanhamento permitirá checar como flui o programa em execução e sua interação com o hardware.

Para realizar este acompanhamento de maneira eficiente, podem ser utilizados três tipos de equipamentos comerciais:

1) Analisador Lógico Convencional - Permite visualisar, simultaneamente, diversos sinais do circuito analisado, como se fosse um osciloscópio multi-canal, ou acompanhar os estados das linhas na forma tabular, através de números binários. O conjunto de bits monitorados formam a "palavra de entrada", que normalmente é comparada com um outro conjunto de bits que formam a chamada "palavra de disparo". Comumente, os bits da palavra de disparo são definidos pelo usuário por meio de chaves no painel frontal. Estas chaves podem assumir três posições: "1", "0" ou "inativo". Uma vez identificada a palavra de disparo nas linhas monitoradas, são capturadas mais algumas palavras de entrada, que ficam

armazenadas em um "buffer" circular, constituído de memórias estáticas de alta velocidade. A análise destas informações permite avaliar o funcionamento do software e hardware. Como este aparelho foi idealizado para análise de circuitos lógicos em geral, não realiza maiores interpretações dos dados coletados, ficando esta tarefa a cargo do usuário. O grande número de "1's" e "0's" apresentados na tela na forma tabular dificultam a análise dos dados. (HEWLETT-PACKARD JOURNAL, 1973, 1974, 1975b, 1975c)

2) Analisador Lógico para Microprocessadores - Funciona de modo semelhante ao analisador lógico convencional descrito no item anterior. Entretanto, como se destina especificamente ao trabalho de manutenção e depuração de aparelhos com microprocessadores, este é capaz de traduzir a seqüência de palavras que ocorrem no barramento do microprocessador para outra forma mais conveniente para o usuário. Uma dessas formas de tradução, é a apresentação da seqüência de instruções capturadas através de mnemônicos da linguagem Assembly do microprocessador em uso, habilitando um nível de legibilidade e interpretação bem maior que o conseguido com as tabelas de números binários. Outra facilidade normalmente oferecida, é a substituição do grande número de pontas individuais de teste por um conector que se encaixa no soquete do microprocessador, diminuindo a possibilidade de erros de ligação. (HEWLETT-PACKARD JOURNAL, 1977a)

3) Emulador - Este aparelho, já descrito anteriormente (ver item 1.2.2), possui entre seus diversos modos de funcionamento a capacidade de operar como um poderoso analisador lógico para microprocessadores. Uma das diferenças entre o emulador e os analisadores lógicos citados anteriormente, é a possibilidade de se construir comandos múltiplos, usando uma sintaxe apropriada, o que facilita bastante o trabalho do usuário. Por exemplo: se o conteúdo da memória "00A8H" possuir o valor "FFH", então, acompanhar a execução do programa na faixa de endereços de "385FH" até "3A1EH".

Para possibilitar o acompanhamento dos estados consecutivos do barramento do KD-16 durante a execução de um programa, foi desenvolvida a placa rastreadora, que constitui um acessório do SD-16.

A figura 3.41, apresenta um esboço da placa rastreadora. As figuras AP.16 e AP.17 (Apêndice A) apresentam os diagramas esquemáticos.

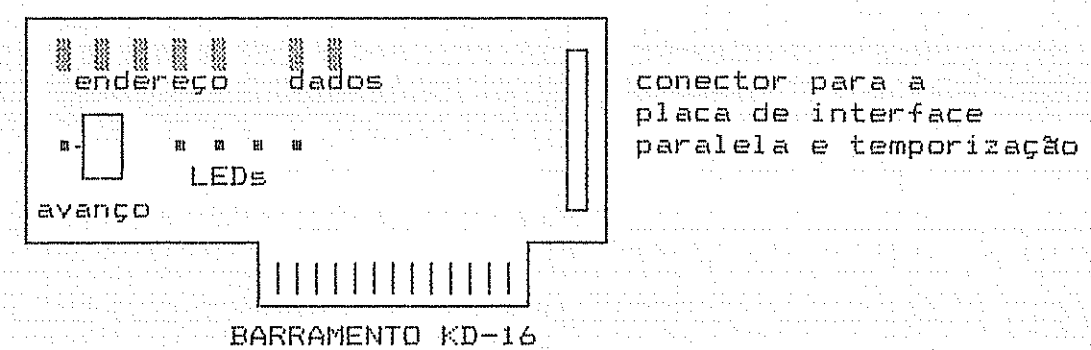
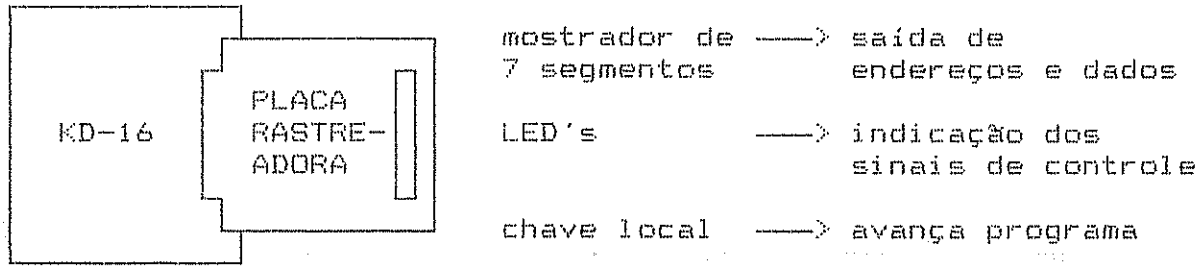


Figura 3.41 - Placa Rastreadora.

O tipo de trabalho realizado por esta placa pode ser classificado como intermediário entre o trabalho de um analisador lógico de uso geral e de um analisador específico para microprocessadores. Classificamos como intermediário, porque, embora ele forneça os estados das linhas de endereço, dados e controle, não realiza a tradução dos códigos capturados para a linguagem Assembly.

A placa rastreadora possui dois modos de operação: o modo manual e o modo assistido pelo microcomputador pessoal (ver figura 3.42).

a) Modo de operação manual:



b) Modo de operação assistido pelo microcomputador pessoal:
vídeo ——> endereços / dados / controle
teclado ——> comandos do rastreador
impressora ——> quando ativada, acompanha o rastreamento

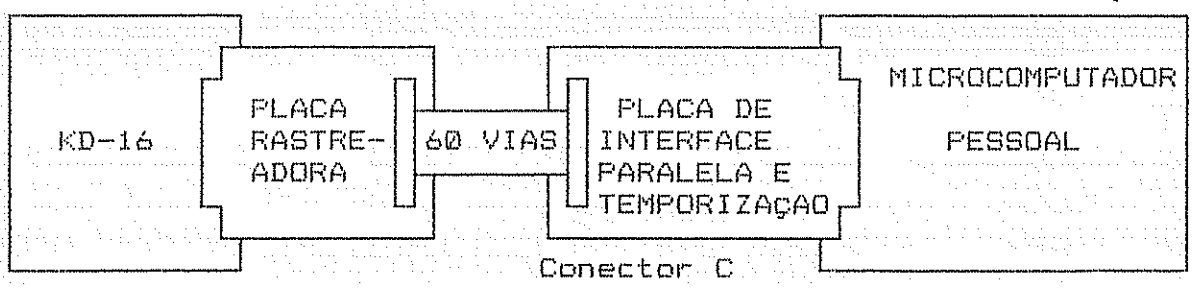


Figura 3.42 - Dois modos de operação da Placa Rastreadora.

3.5.2 - Operação manual

A operação manual é necessária quando pretendemos fazer o acompanhamento do barramento do KD-16 e não dispomos de um microcomputador pessoal funcionando como terminal para facilitar esta operação.

Sua operação se baseia na detecção do início de um ciclo de acesso ao barramento, através do sinal ALE (Address Latch Enable), que provoca repetidos pedidos de estados de espera ("Wait") (USATEGUI, 1985b). A liberação deste estado de espera e o consequente avanço do programa, dependerá de uma ordem dada pelo usuário através do acionamento de uma tecla na própria placa rastreadora. O diagrama de estados, mostrado na figura 3.43, ilustra os ciclos de funcionamento dessa placa.

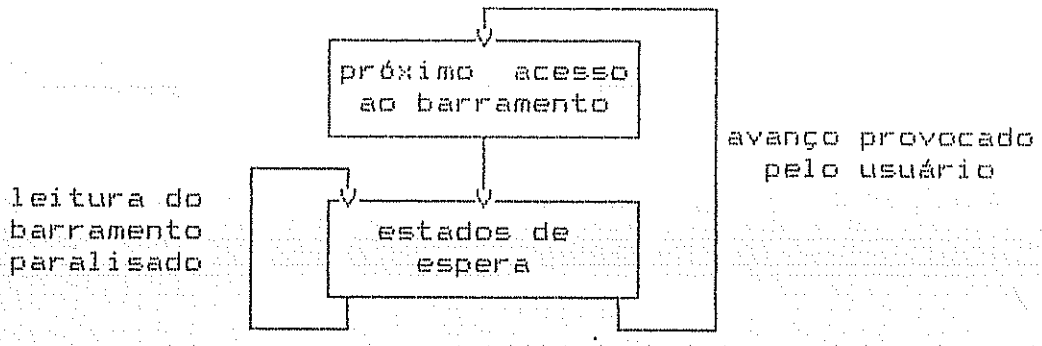


Figura 3.43 - Diagrama de estados do barramento do KD-16 operando com a Placa Rastreadora.

Durante os estados de espera, os bits de endereços (BA0 até BA19) e os bits de dados (BD0 até BD7) são apresentados por 7 dígitos hexadecimais, através de decodificadores "9368" e mostradores de 7 segmentos. Os sinais de controle, MRD/, MWR/, IORD/ e IOWR/ são indicados por LEDs que se acen-

dem de acordo com o sinal de controle ativo (ver circuito da figura AP.17).

Como no modo de operação manual não existe nenhuma forma de comparação dos sinais de entrada com palavras de disparo definidas pelo usuário, então, todos os estados do barramento serão mostrados um a um, a cada acionamento da chave de avanço.

A figura 3.44 mostra o diagrama de tempos da placa rastreadora. Os diversos passos numerados de 1 a 8, descrevem um ciclo completo de paralisação/captura/liberação do barramento do KD-16.

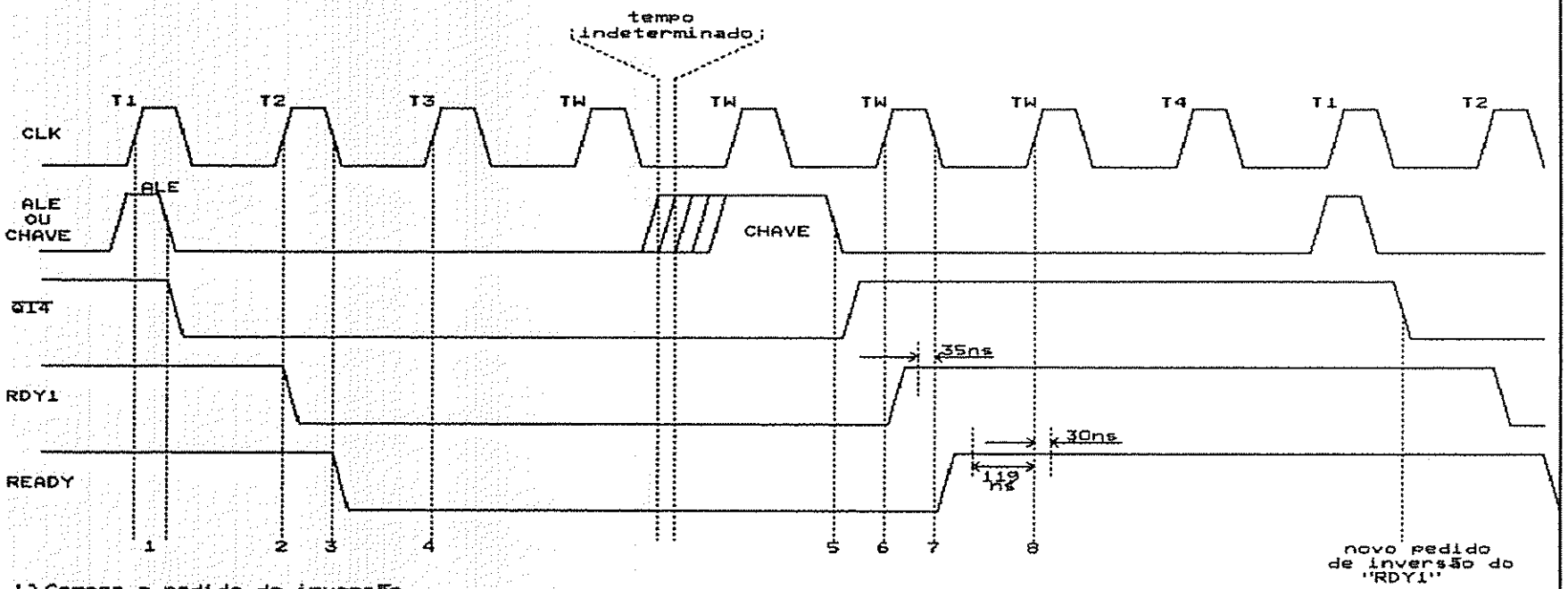
3.5.3 - Operação assistida por um microcomputador pessoal

No segundo modo de operação, temos o microcomputador pessoal auxiliando o usuário na tarefa de rastreamento.

Para que o microcomputador pessoal receba as informações de endereços, dados, e controle do barramento do kit KD-16, foi utilizada a placa de interface paralela e temporização, descrita no item 3.3 , conectada à placa rastreadora (ver figura 3.42). Esta implementação adota uma concepção modular, que é mais econômica, maximiza o uso da placa de interface paralela e simplifica a placa rastreadora.

O controle do conjunto "placa rastreadora / placa de interface paralela", fica a cargo do programa "Rastro", especialmente desenvolvido para esta tarefa.

A alocação de bits da placa de interface paralela, para cada sinal do KD-16, é mostrada na figura 3.45.



1) Começa o pedido de inversão do estado do sinal "RDY1"

2) O pino $\overline{QI4}$ do flip-flop 74LS76 permanecerá em zero até que haja uma borda de descida comandada pela chave de avanço

3) O sinal "READY" sinaliza para o 8088 o pedido de espera ("wait")

4) O 8088 amostra o "READY" e sabe que o estado futuro será TW

5) A chave de avanço é acionada, causando a mudança de QI4, de "0" ——> "1", começando o pedido de inversão do sinal "RDY1"

6) Na subida de "CLK", o nível "1" de QI4 é transferido para o sinal "READY". Isto deve acontecer até 35ns antes da descida do sinal "CLK"

7) 8ns após a descida do sinal "CLK", temos o "READY" em nível alto

8) Na subida do último TW, teremos o 8088 amostrando o sinal "READY", sendo o próximo período o T4. O sinal "READY" deve ser alto pelo menos 119ns antes da subida do último TW, devendo permanecer alto pelo menos 30ns depois da subida do último TW.

FIGURA 3.44 — DIAGRAMA DE TEMPOS DA PLACA RASTREADORA

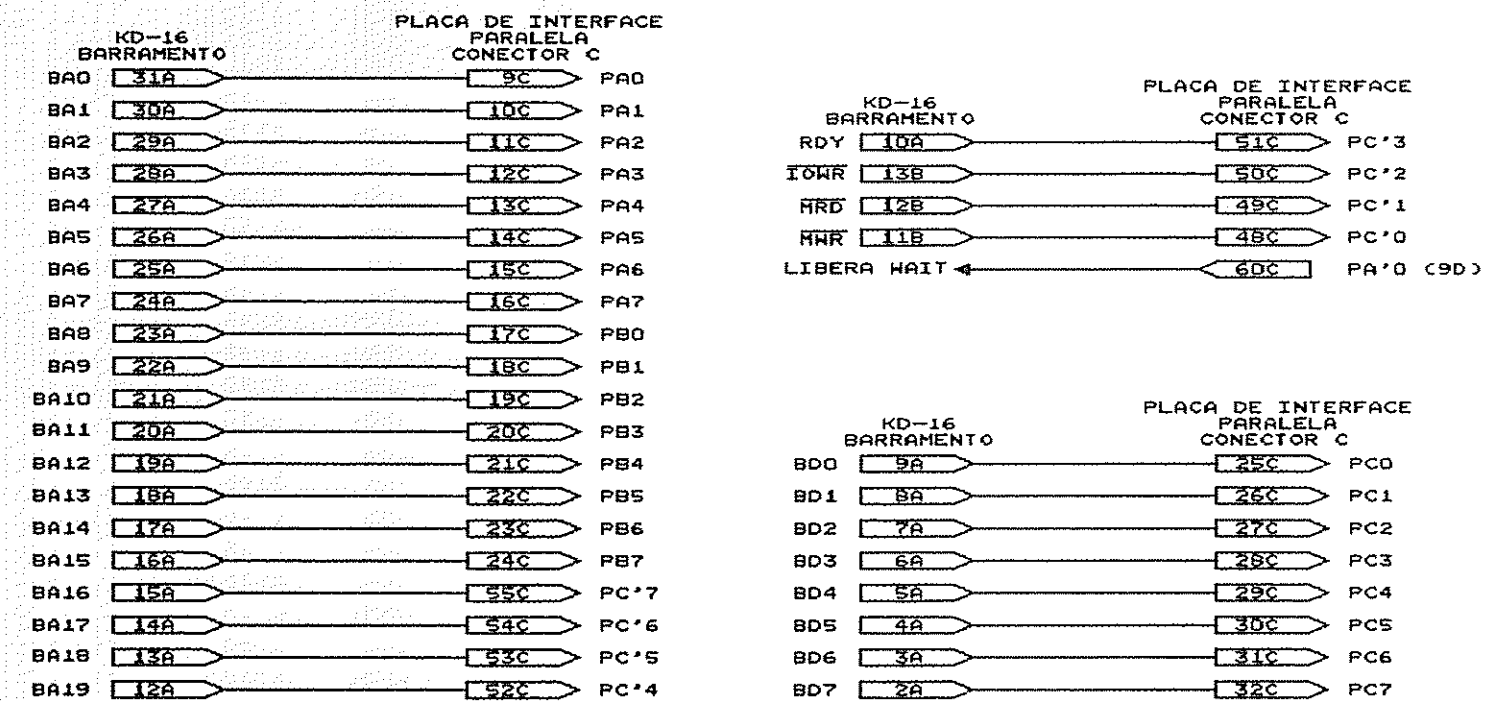


FIGURA 3.45 – LIGAÇÃO DA PLACA RASTREADORA À PLACA DE INTERFACE PARALELA E TEMPORIZAÇÃO

O programa Rastro usa uma das linhas das portas paralelas para monitorar o sinal "RDY" do barramento do KD-16. Quando o sinal "RDY" assume nivel "0", temos a indicação de que o barramento está paralisado. Outras 31 linhas das portas paralelas, também programadas como entradas, realizam a leitura das vias de endereços, dados e controle. A última linha usada, funcionando como saída, simula o acionamento da tecla de avanço da placa rastreadora, liberando o 8088 do estado de espera em que se encontra. O próximo acesso ao barramento provocará nova paralisação e portanto nova oportunidade para proceder a leitura e análise do barramento.

3.5.4 - Programa RASTRO

O programa RASTRO foi desenvolvido usando a liguagem Pascal, devendo ser executado no microcomputador pessoal.

A tela de entrada do programa RASTRO pode se vista na figura 3.46.

Nesta tela podemos identificar três blocos principais:

- o bloco intitulado "Faixa rastreada", onde o usuário entra com os limites de até três faixas independentes de rastreamento;
- o bloco intitulado "Barramento", onde temos a apresentação da última ocorrência de endereço, dado e controle dentro de cada uma das faixas definidas;
- o bloco de auxílio ao usuário, com os possíveis comandos do programa RASTRO.

Rastreador passo a passo para o kit KD-16							1.0
Faixa rastreada		Barramento					
Inicio	Fim	Endereço	Dado	IORD	IOWR	MRD	MWR
00000	FFFFF	FFFFF	FF			▓▓	
▒▒	▒▒	▒▒	▒▒	▒▒	▒▒	▒▒	▒▒
▒▒	▒▒	▒▒	▒▒	▒▒	▒▒	▒▒	▒▒

P- Próximo passo ▒▒ --> Desabilitado

I- Liga impressora ESC- Abandona programa

H- Habilita/Desabilita <->- Muda campo ativo

Figura 3.46 - Tela do programa RASTRO.

Para conseguirmos uma placa rastreadora de baixo custo é importante que grande parte do trabalho seja delegado ao software, o que necessariamente provoca a diminuição da velocidade de execução, afastando-nos da condição de operação em tempo real. Desse modo, após a leitura do barramento do KD-16, verifica-se, por software, se o endereço lido está incluso nas faixas definidas pelo usuário.

Caso o endereço esteja fora das faixas delimitadas, é gerado um pulso que simula o acionamento da tecla de avanço e passa-se novamente à monitoração da linha "RDY", que sinaliza nova paralisação. Mesmo neste caso, onde ocorre uma rápida liberação do KD-16, são necessários vários ciclos de espera para que o programa RASTRO tenha tempo para verificar se o endereço lido esta dentro de alguma das faixas rastreadas.

Em caso positivo, passa-se a uma segunda fase de verificação:

- se o comando detectado estiver habilitado nesta faixa rastreada, atualiza-se os campos relativos ao endereço, dado e comando, para posteriormente proceder a liberação do KD-16.
- se o comando detectado não estiver habilitado nesta faixa rastreada, o programa RASTRO simplesmente comandará a liberação do KD-16.

A figura 3.46 mostra que, de início, temos somente uma faixa de rastreamento ativada. Os limites dessa primeira faixa, denominados "Início" e "Fim", são inicializados com os valores "00000" e "FFFFF", respectivamente. As quatro colunas destinadas aos sinais de controle estarão também ativadas. Nessa condição, todos os acessos ao barramento serão capturados e apresentados na primeira faixa.

A segunda e terceira faixas estão, a princípio, desativadas. Este estado é representado pela marcação pontilhada "▒▒▒▒▒▒▒▒" nos diversos campos das linhas desativadas.

Se for do interesse do usuário incluir a segunda e/ou terceira faixa no trabalho de rastreamento, bastará deslocar o campo ativo, através das teclas seta à esquerda e seta à direita ("<-" e "->"), para qualquer coluna do bloco "Faixa rastreada", e teclar "H" para habilitar aquela faixa. Um novo acionamento tornará a desativa-la. A condição inicial da faixa recém habilitada é a mesma que a apresentada para a primeira faixa, na tela de entrada do programa RASTRO.

A tecla "I", serve para ligar/desligar a impressora. A linha que descreve a ação da tecla "I", no bloco de auxílio ao usuário, será alterada de "Liga impressora", para "Desl. impressora", ou vice-versa, a cada acionamento da tecla "I". Enquanto a impressora estiver ligada, as faixas de rastreamento ativadas provocarão a impressão dos estados das linhas de endereço, dados, e controle, sempre que, na tela, alguma faixa for atualizada.

Sempre que um acesso ao barramento for identificado e provocar atualização de algum campo, em qualquer das três faixas, teremos o programa RASTRO esperando que o usuário acione a tecla "P" para dar prosseguimento à execução do programa no KD-16. Caso a tecla "P" fique permanentemente acionada, o programa rastreado avançará rapidamente.

A tecla "ESC" causa a interrupção do programa RASTRO e o retorno ao sistema operacional, mediante confirmação.

Como exemplo de utilização da placa rastreadora, no modo assistido pelo microcomputador pessoal, faremos a inicialização das três faixas de rastreamento como indicado na figura 3.47. O rastreamento pode ser acompanhado na figura 3.48, onde temos uma seqüência de acessos ao barramento do KD-16, relativos a execução do programa monitor, após o acionamento da chave "RESET".

Rastreador passo a passo para o kit KD-16							1.0
Faixa rastreada		Barramento					
Inicio	Fim	Endereço	Dado	IORD	IOWR	MRD	MWR
FF0A8	FF0FF			▒▒▒	▒▒▒		▒▒▒
0BC00	0BF01					▒▒▒	▒▒▒
00000	000FF			▒▒▒	▒▒▒		

P- Próximo passo ▒▒▒ --> Desabilitado

I- Desl. impressora ESC- Abandona programa

H- Habilita/Desabilita <->- Muda campo ativo

OBS: 1ª faixa -----> início do programa monitor
2ª faixa -----> registros da porta serial 8251A
3ª faixa -----> área do "STACK"
Impressora ---> habilitada

Figura 3.47 - Exemplo de utilização da placa rastreadora

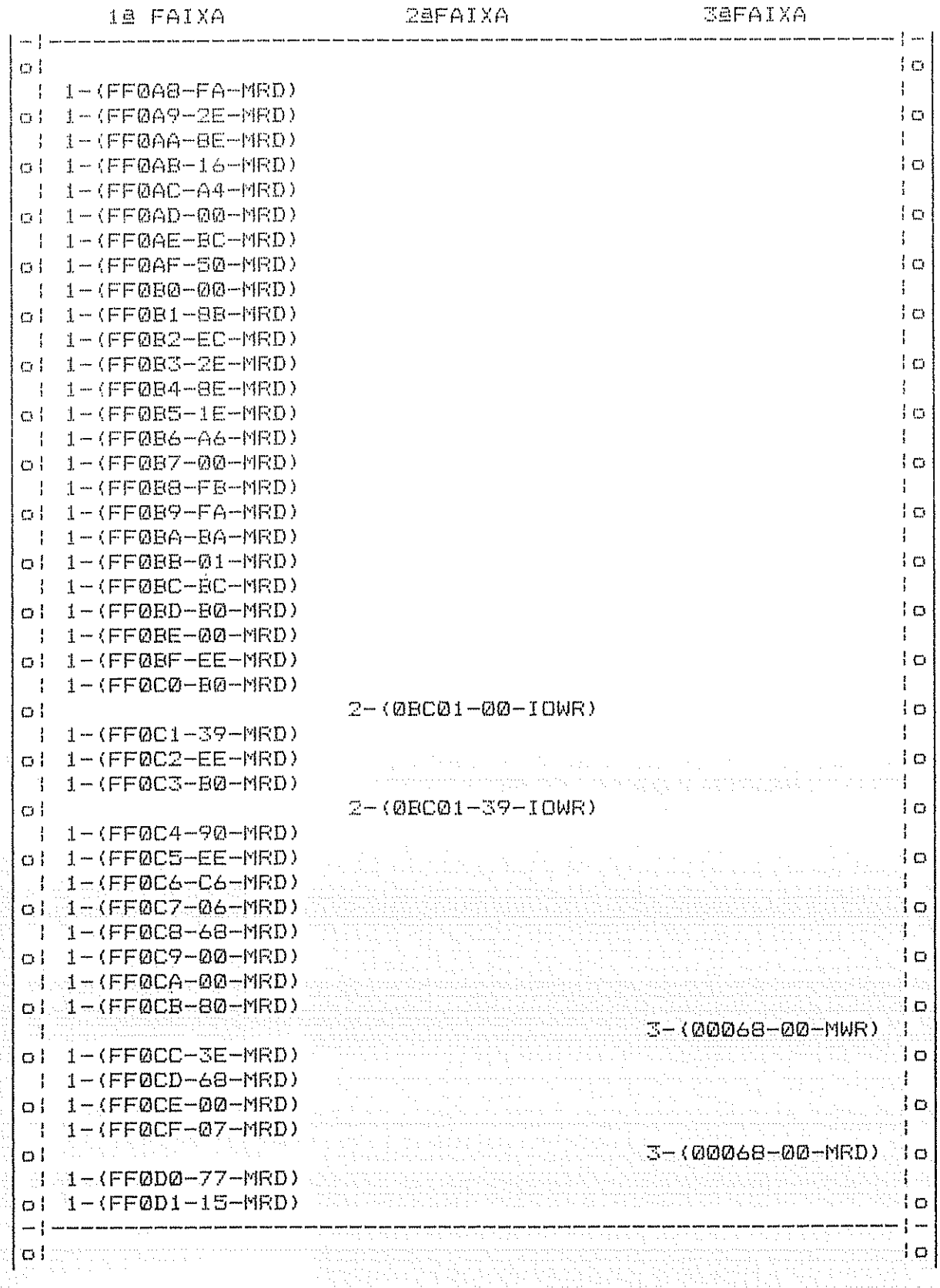


Figura 3.48 - Rastreamento acompanhado pela impressora.

CAPITULO 4 - CONCLUSÕES

No Brasil, assim como em todo mundo, estamos vivendo um período de automação crescente. Este avanço da automação reclama permanentemente por especialistas com experiência em sistemas de controle baseados em microcomputadores. Em resposta a isto encontramos uma ênfase cada vez maior no estudo dos microprocessadores e sistemas de controle digitais, nos cursos de graduação e pós-graduação.

Este trabalho de mestrado, sem pretensões de competir com os sofisticados sistemas de desenvolvimento disponíveis no mercado, almejou oferecer ao estudante de engenharia e ao projetista de sistemas digitais, uma base de hardware tipicamente utilizada no Departamento de Engenharia Eletrônica da Escola de Engenharia da UFMG, com preço acessível e com razoável apoio para produção e depuração do software.

4.1 - Objetivos alcançados

Apresentaremos neste item alguns objetivos que foram alcançados com o SD-16:

- **Apresentar baixos custos:** Especificar ou comparar custos em nosso pais é uma tarefa bastante difícil, dada a característica extremamente irregular de variação dos preços dos componentes e a elevada taxa de inflação. Entretanto, numa tentativa de estabelecer uma base de comparação, a tabela 4.1 fornece os preços aproximados de cada uma das placas que

compõem o SD-16, baseado em preços médios do mercado norte americano e britânico (BYTE, 1990 / ELECTOR ELECTRONICS, 1990). Os preços cotados supõem as placas completas, incluindo conectores, circuitos integrados, etc; a única exceção é o circuito integrado co-processador aritmético 8087-2 ($120) da placa do KD-16, que não foi incluido no preço fornecido.

PLACAS DO SD-16		PREÇO EM DOLARES
kit KD-16		$ 130
Interface Paralela e Temporização		$ 25
Gravador de EPROMs		$ 45
Teclado e Visor		$ 65
Placa Rastreadora	completa	$ 85
	sem decodificadores e mostradores de 7 seg.	$ 20

Tabela 4.1 - Preços das placas que compõem o SD-16.

- **Usar componentes facilmente encontrados:** Foi preocupação constante no projeto de cada placa, a utilização de componentes comumente encontrados no mercado nacional, facilitando assim a reprodução do SD-16.

- **Utilização de um microprocessador de maior desempenho:** O microprocessador escolhido, 8088-2, apresenta um desempenho relativo médio de 5 a 6 vezes superior, comparado aos microprocessadores de 8 bits (Z80 e 8085) anteriormente utilizados no Departamento de Engenharia Eletrônica da Escola de Engenharia da UFMG. Em aplicações que envolvem grande volume

de cálculos com aritmética de ponto flutuante, pode-se acelerar o processamento em até 100 vezes, desde que seja usado o co-processador 8087-2 em conjunto com o 8088-2 (INTEL, 1979a; BENCHMARK, 1981). Além disso, sua maior velocidade e capacidade de endereçamento permitem a implementação de projetos mais sofisticados.

- **Utilização de um microcomputador pessoal versátil:** A adoção de um microcomputador compatível com um IBM-PC, amplamente difundido em nossas universidades, facilita bastante a reprodução do SD-16 com um mínimo de gastos adicionais. A ampla variedade de software de boa qualidade disponível para este tipo de microcomputador pode ser vantajosamente aproveitado para a produção de programas aplicativos, com eficiência, rapidez e confiabilidade. A adoção do barramento do microcomputador IBM-PC no kit KD-16 permite que a placa de interface paralela e temporização possa ser facilmente intercambiada entre estes dois sistemas. Esta interface comum, permite que os trabalhos iniciais de teste e depuração sejam realizados em um microcomputador pessoal compatível com o IBM-PC (um sistema mais eficiente), passando posteriormente à operação com o KD-16 (um sistema de menor custo).

- **Atender às necessidades básicas de hardware:** O SD-16 supre as necessidades mais comuns de hardware de microcomputador no Departamento de Engenharia Eletrônica, e além disso oferece facilidades para acompanhamento e depuração de programas.

- **Apoio ao ensino:** O SD-16 tem sido usado com sucesso no ensino de sistemas de microprocessadores, na pós-graduação.

4.2 - Sugestões para aprimoramento do SD-16

Tendo em vista que o SD-16 é um sistema protótipo, vamos aqui sujerir algumas modificações e aperfeiçoamentos para futuros trabalhos:

- **Desenvolvimento de um emulador:** Tendo em vista o desenvolvimento e depuração em tempo real em sistemas que não admitem a presença de monitores estranhos ao sistema alvo, seria de grande ajuda ao estudante e pesquisador dispor de um emulador para o 8088 e 8086. Este emulador poderia ser um acessório do microcomputador pessoal, visando reduzir os custos.

- **Modificações no KD-16:** Uma placa aperfeiçoada do KD-16 poderá conter internamente o gerador de ciclos de espera, o decodificador de periféricos, permitir a utilização de memórias RAM de até 32K bytes por chip (62256) e aumentar a densidade da distribuição de componentes, diminuindo a área da placa. Uma versão do KD-16 em placas tipo "eurocard", por exemplo, encontraria boa receptividade.

- **Gravador de EPROMs:** A fonte de Vpp do gravador de EPROMs pode ser alterada para permitir a gravação de memórias mais antigas, com Vpp superiores a 12,5V. Além disso, o programa Gravador poderia ser alterado de modo que o usuário pudesse escolher a metodologia de gravação que desejasse.

- **Melhorar a placa rastreadora:** A placa rastreadora por nós construida impõe grandes restrições à velocidade de operação do kit KD-16 durante o acompanhamento de um programa. A causa é que, praticamente, todo o trabalho de rastreamento da execução do programa, é feito por software. A modificação do circuito da placa, com a identificação por hardware de eventos no barramento, aumentará bastante a eficiência de operação, permitindo o acompanhamento da execução do programa em tempo real.

- **Construção de placas de conversores A/D e D/A:** O fornecimento de placas contendo conversores A/D e D/A completará o interfaceamento do KD-16 com as plantas sob seu controle.

APÊNDICES

Apêndice A: Circuitos esquemáticos

Apêndice B: Listas de material

Apêndice C: Listagem dos programas desenvolvidos

APÊNDICE A

CIRCUITOS ESQUEMATICOS

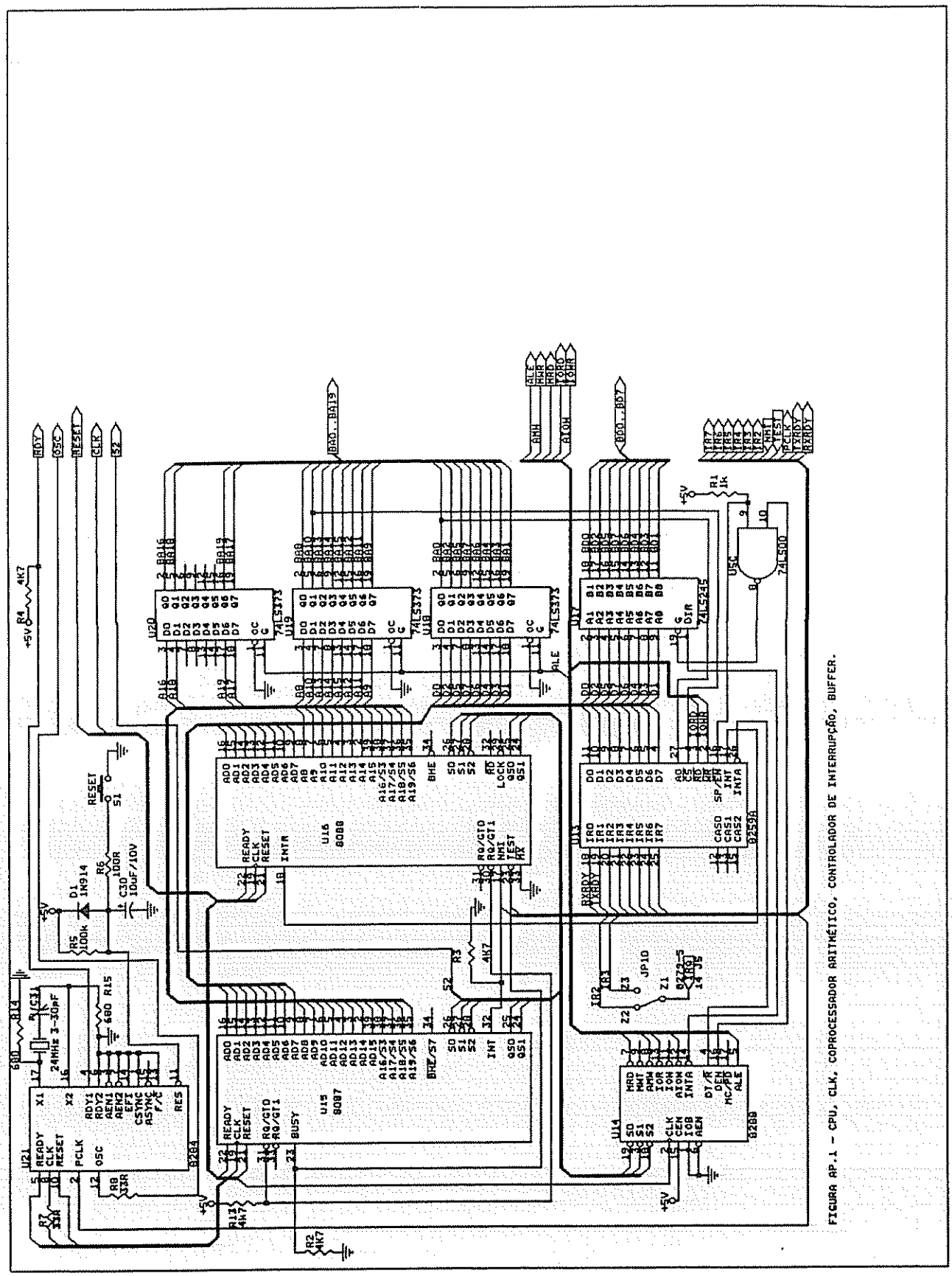


FIGURA AP.1 – CPU, CLK, COPROCESSADOR ARITMÉTICO, CONTROLADOR DE INTERRUPÇÃO, BUFFER.

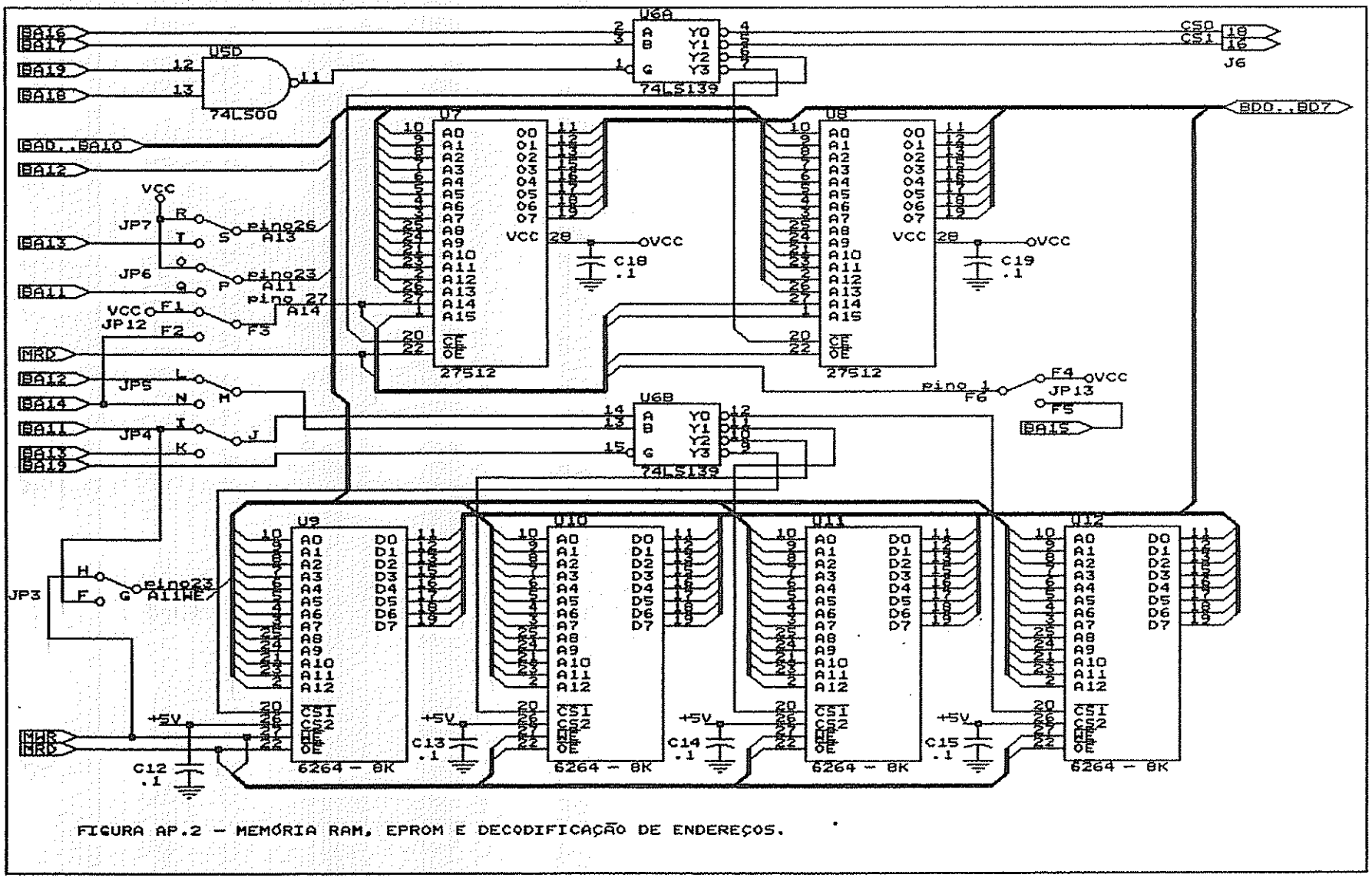


FIGURA AP.2 – MEMÓRIA RAM, EPROM E DECODIFICAÇÃO DE ENDEREÇOS.

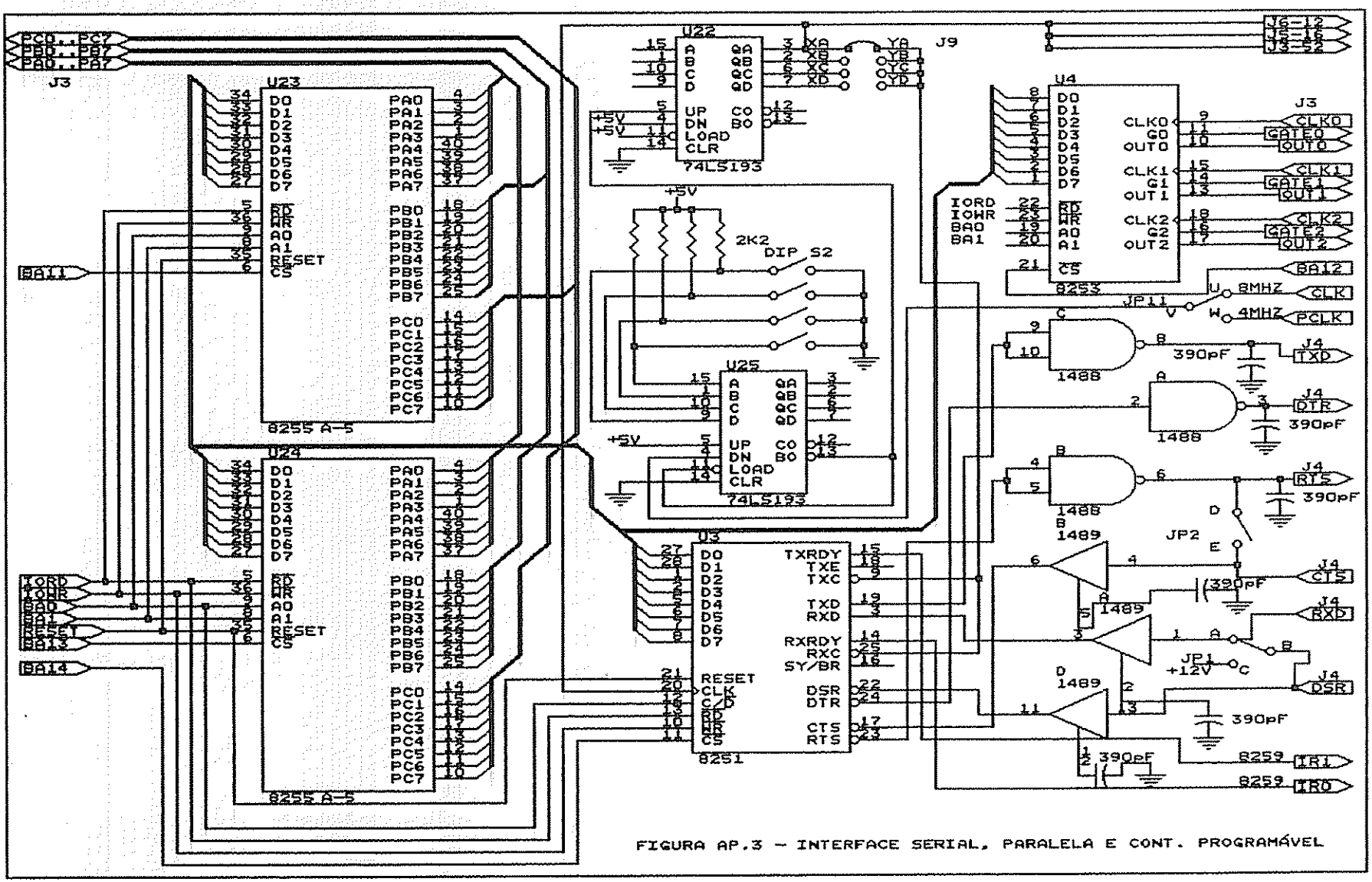


FIGURA AP.3 – INTERFACE SERIAL, PARALELA E CONT. PROGRAMÁVEL

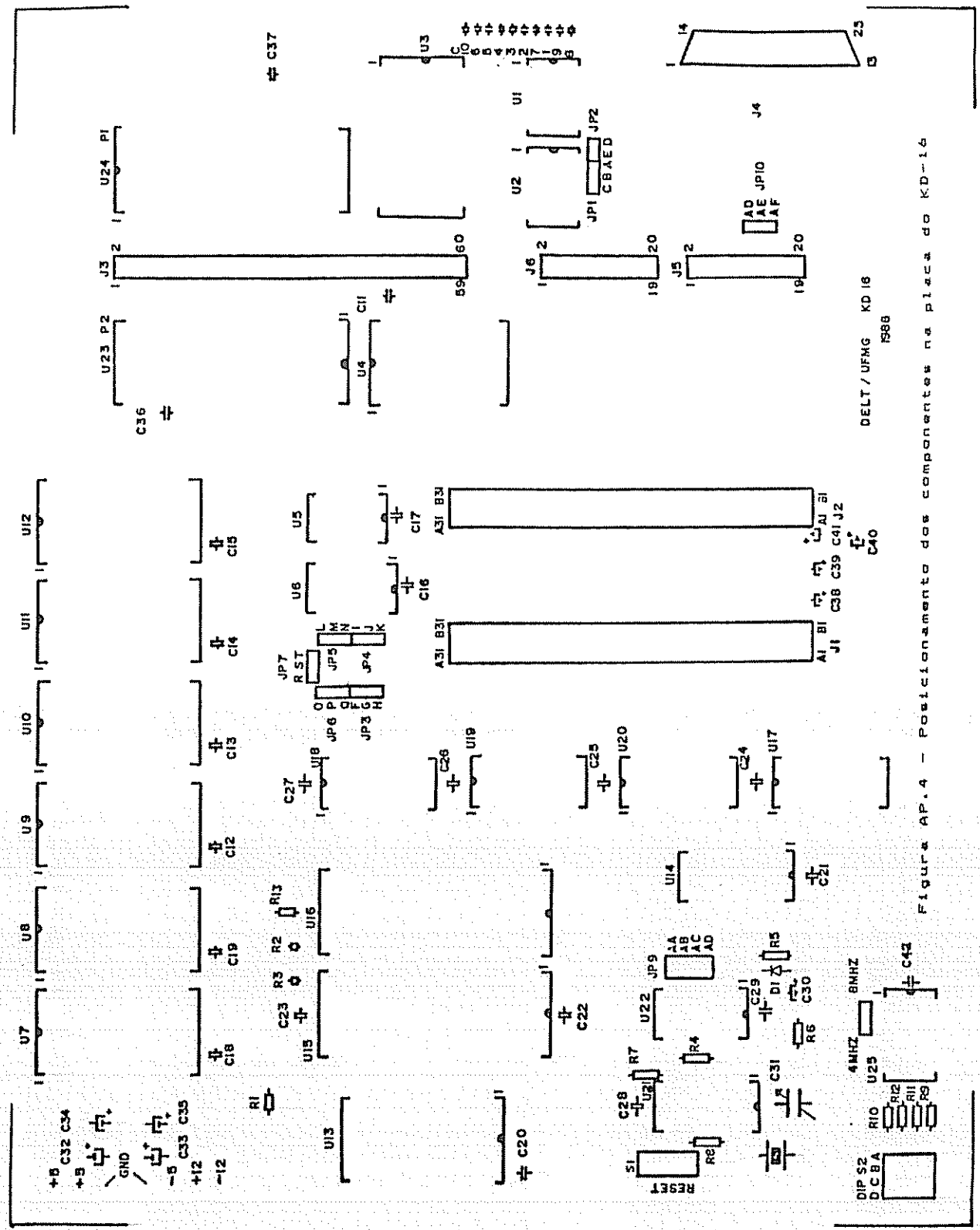


Figura AP.4 - Posicionamento dos componentes na placa do KD-16

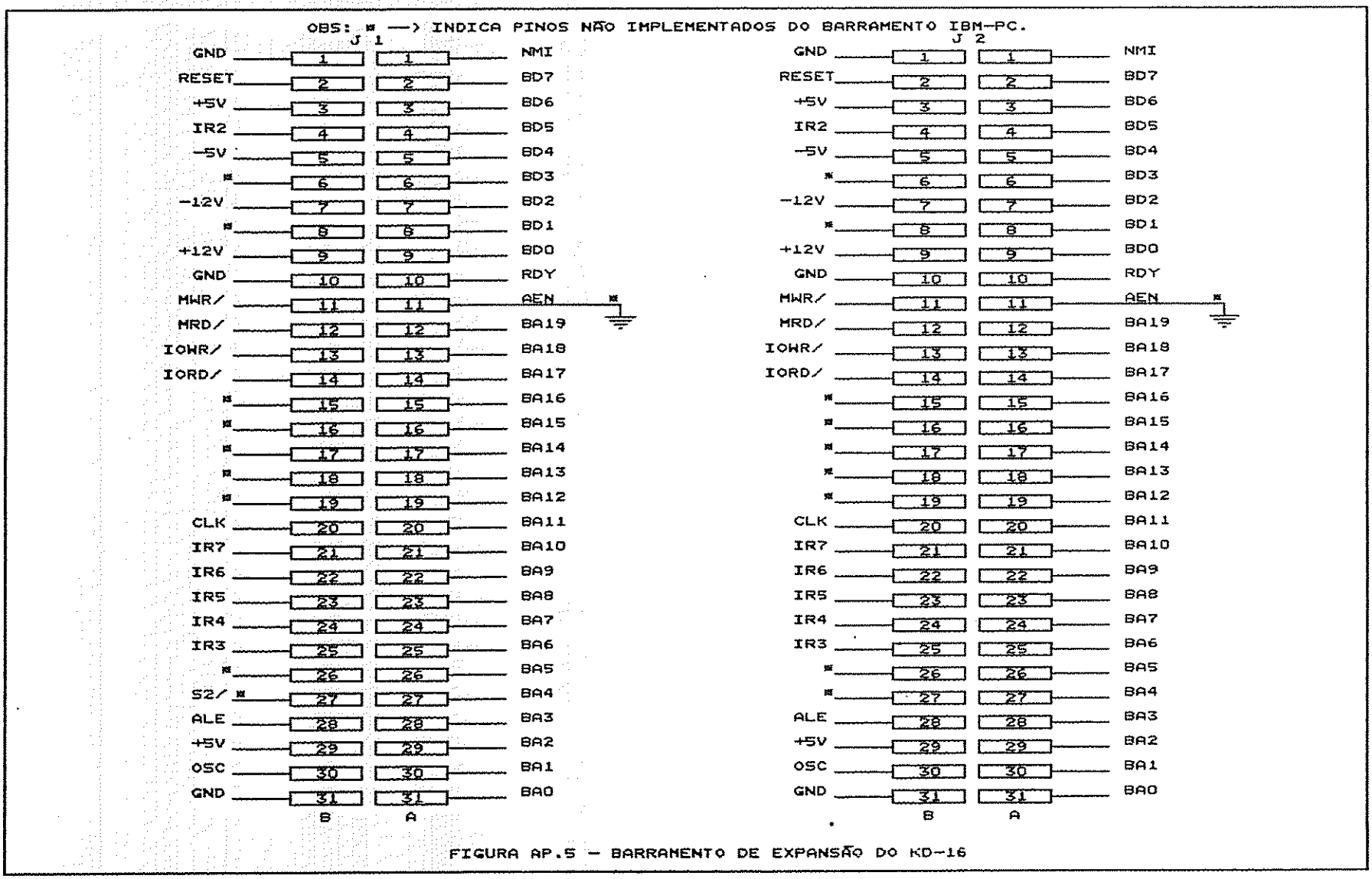


FIGURA AP.5 – BARRAMENTO DE EXPANSÃO DO KD-16

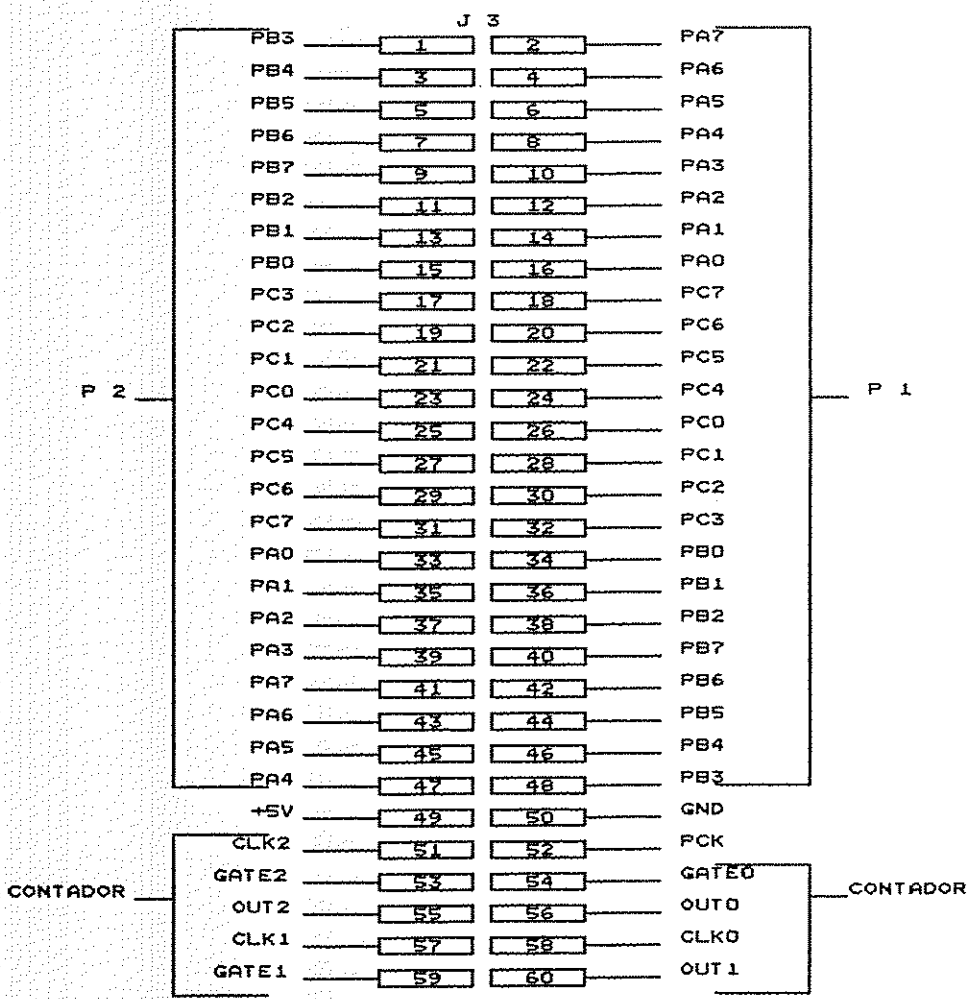


FIGURA AP.6 – CONTADORES E PORTAS PARALELAS DO KD-16

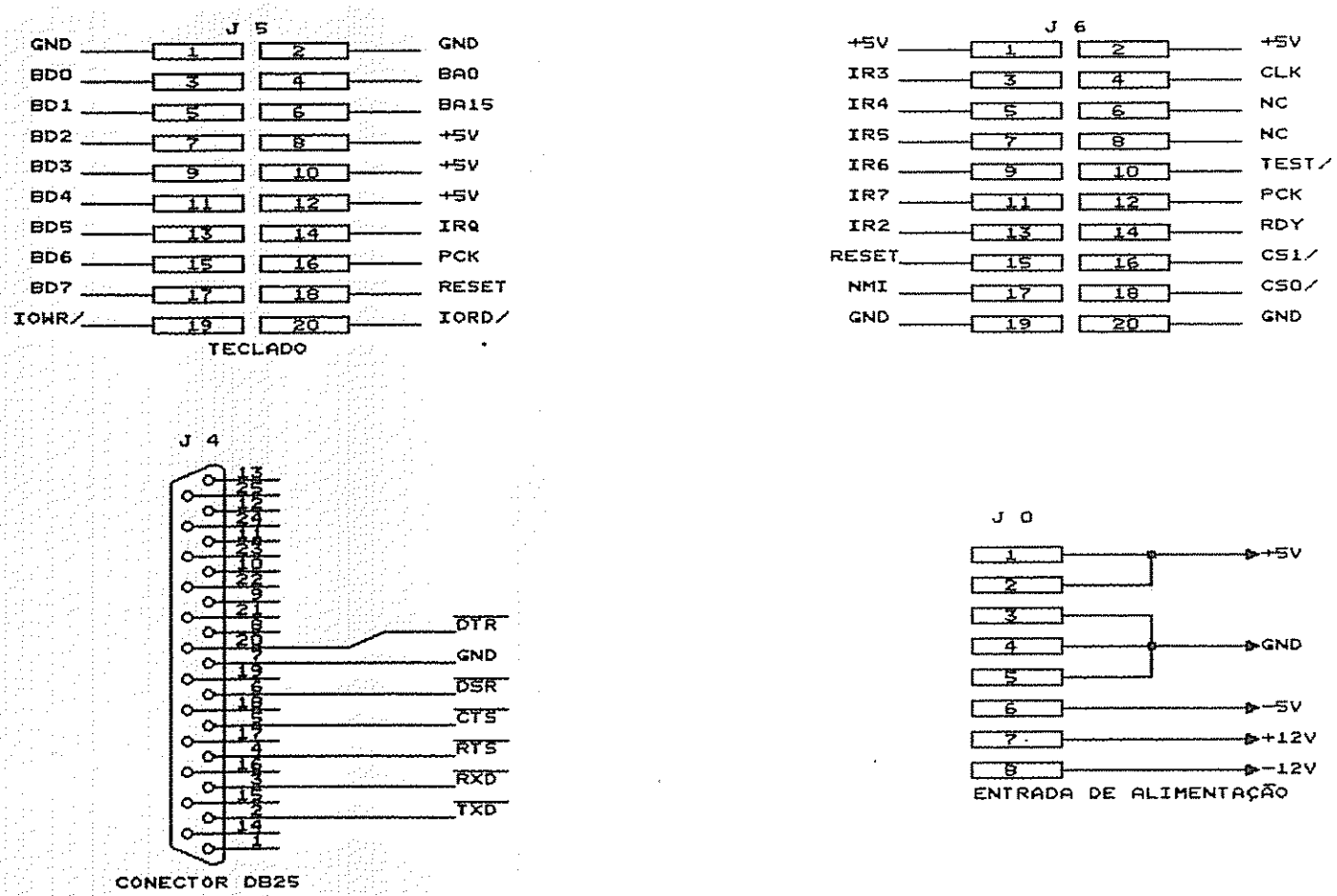


FIGURA AP.7 – DIVERSOS CONECTORES DO KD-16

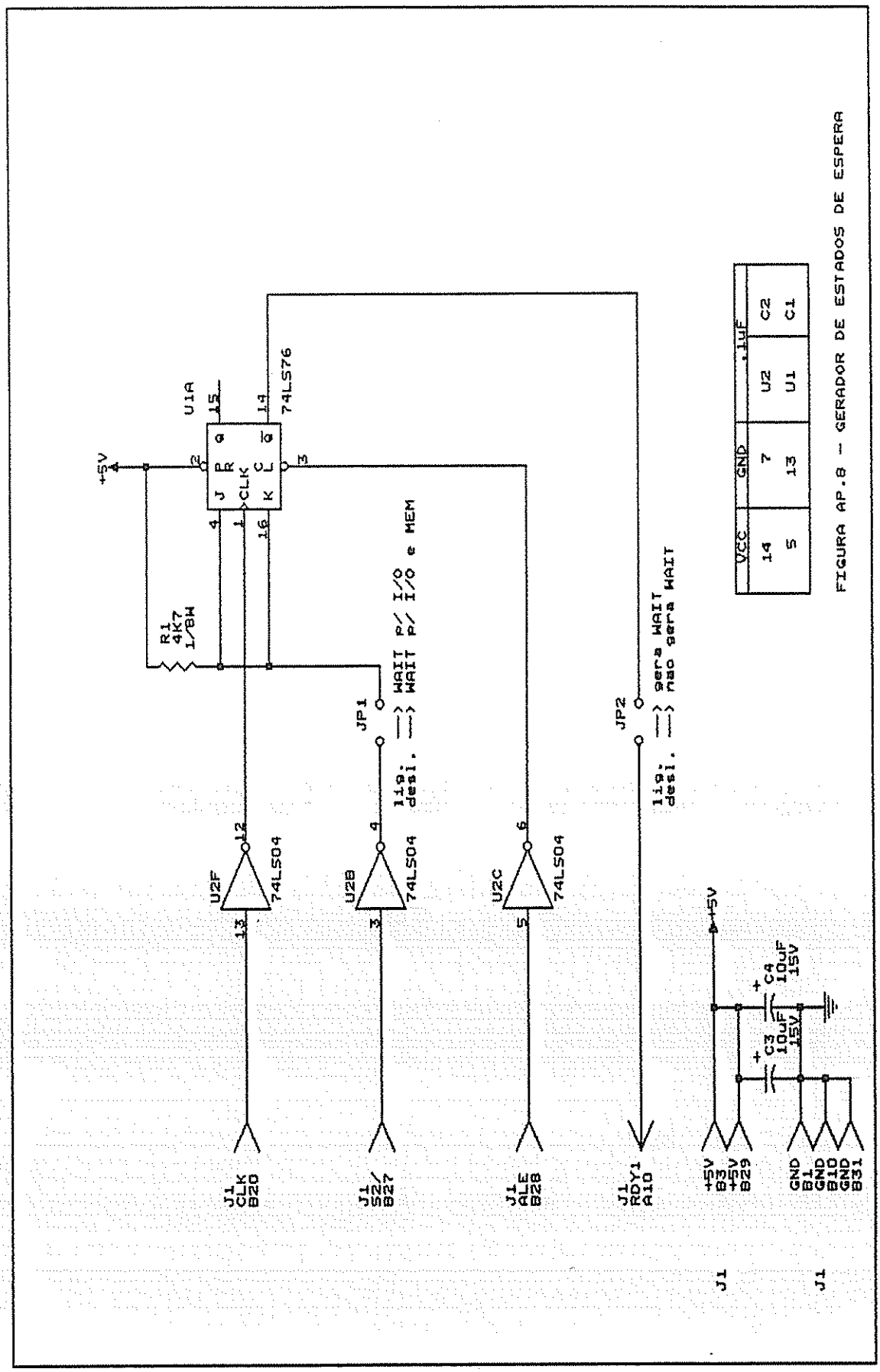


FIGURA AP.8 -- GERADOR DE ESTADOS DE ESPERA

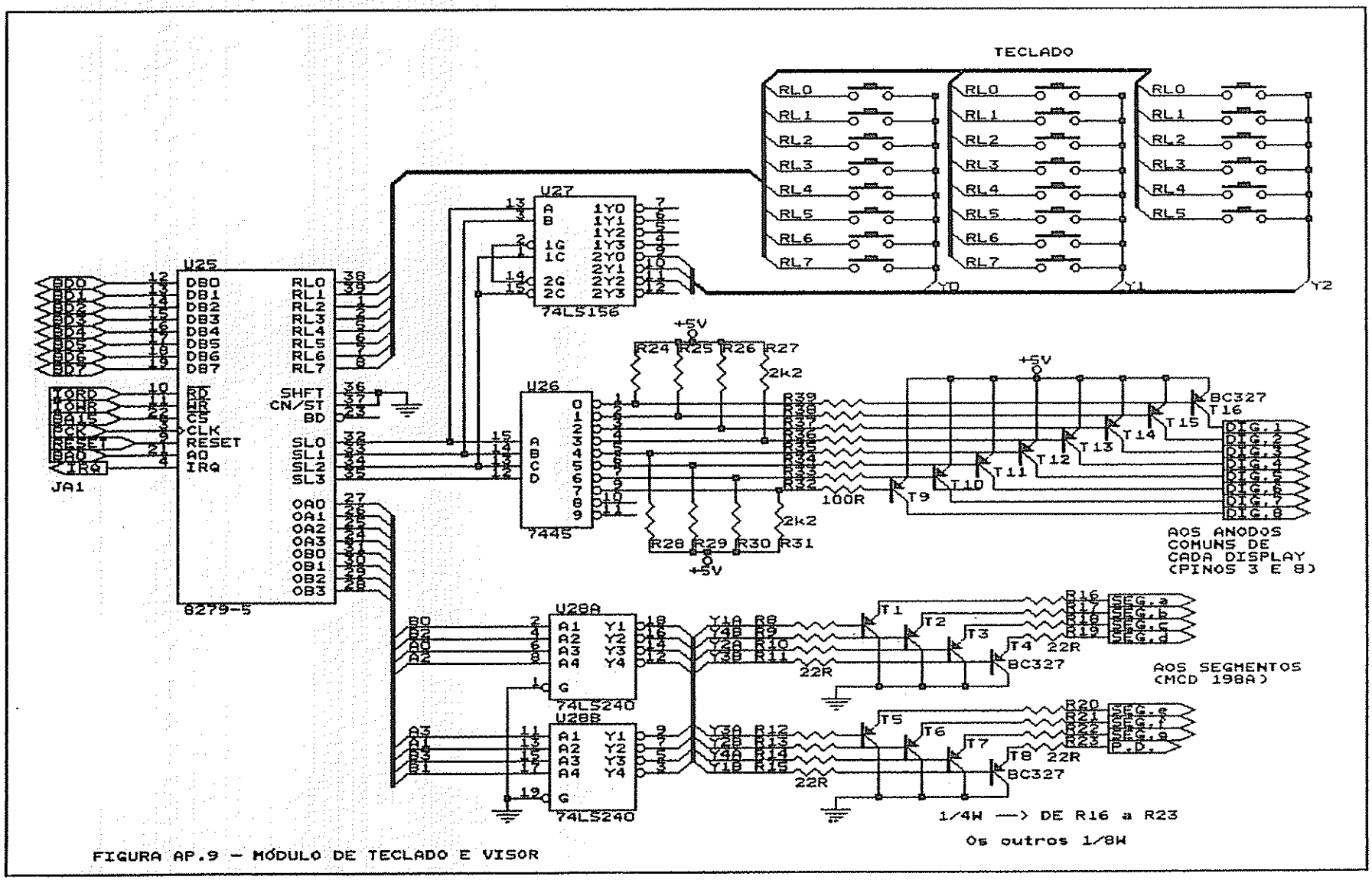


FIGURA AP.9 — MÓDULO DE TECLADO E VISOR

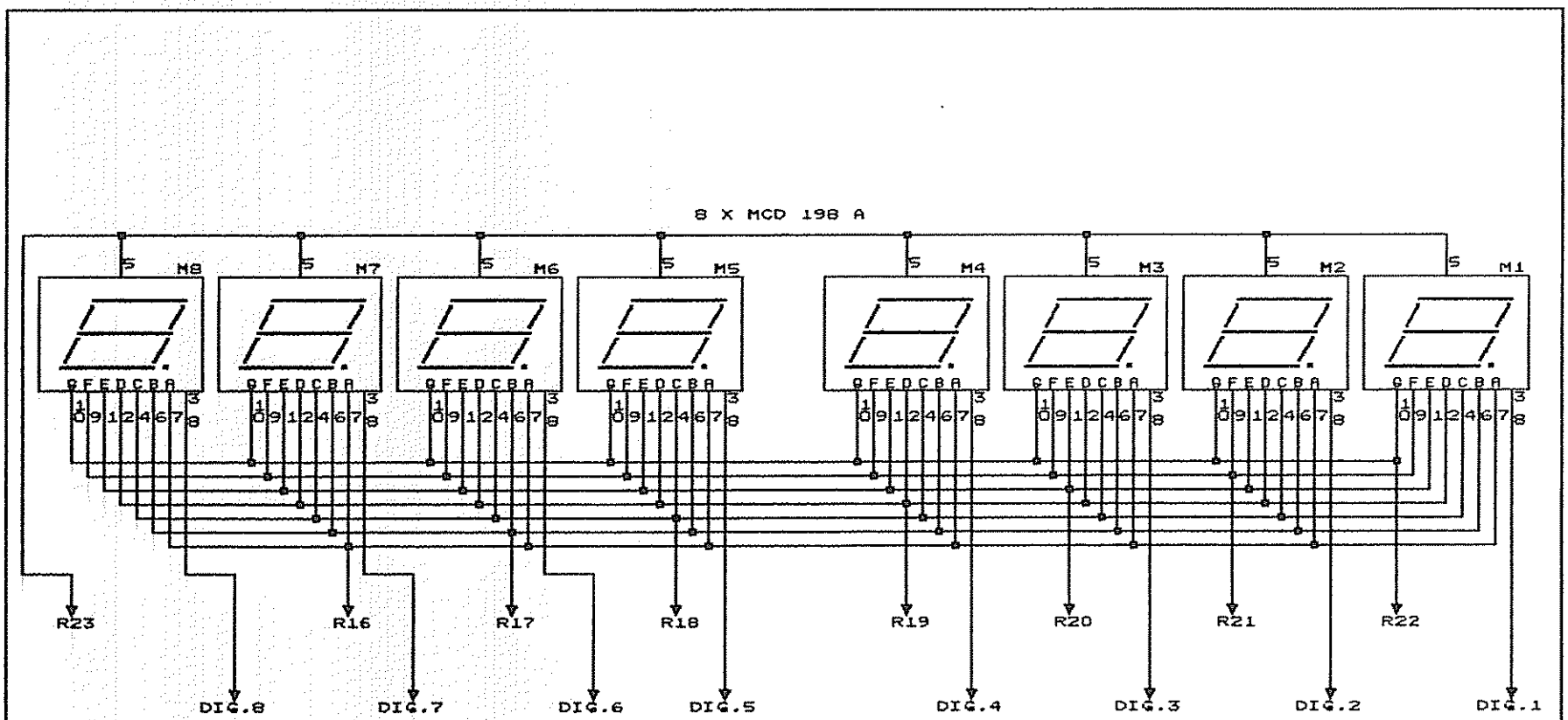


FIGURA AP.10 – MOSTRADORES DE 7 SEGMENTOS DO MÓDULO DE TECLADO E VISOR

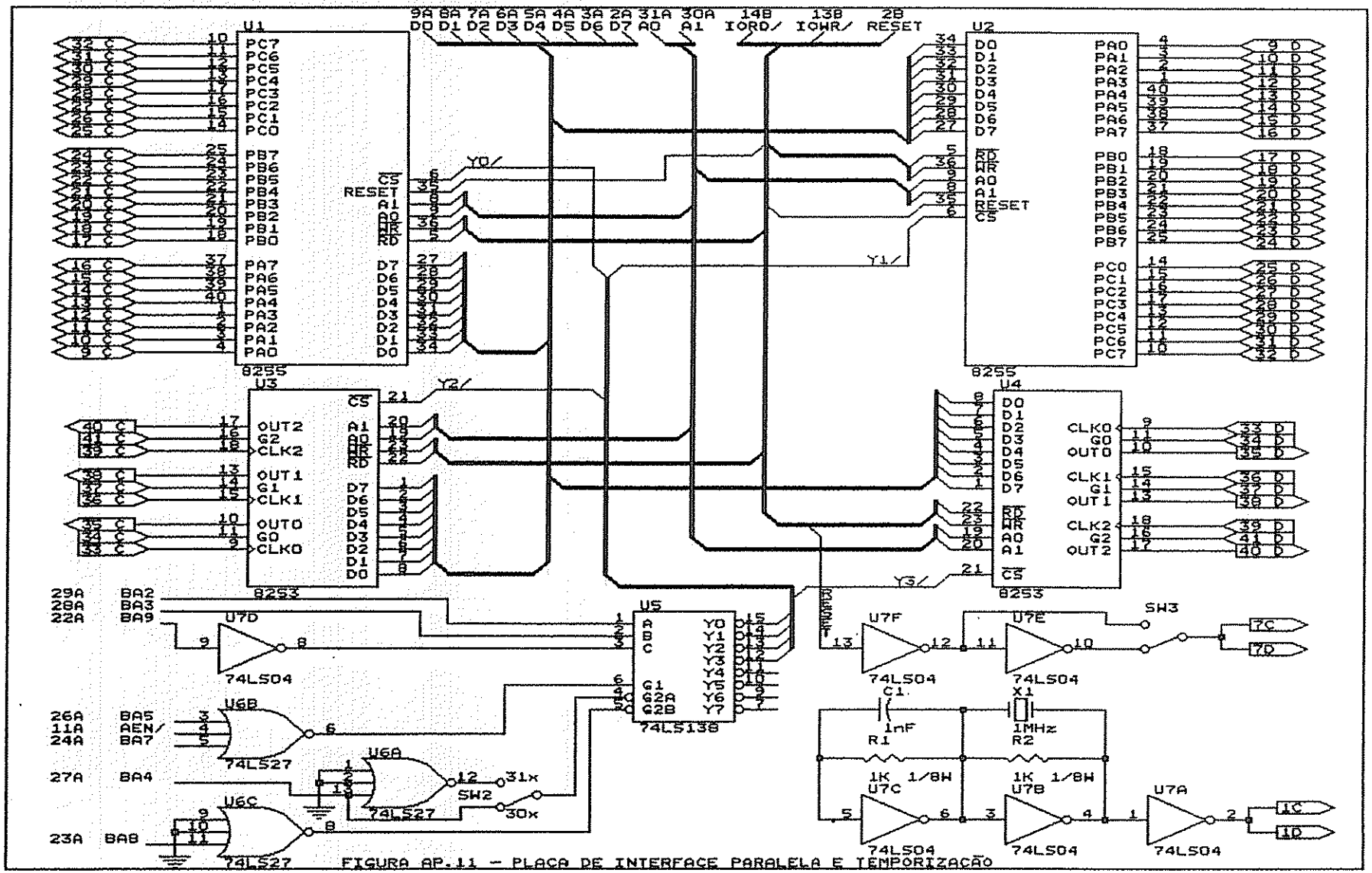


FIGURA AP.11 – PLACA DE INTERFACE PARALELA E TEMPORIZAÇÃO

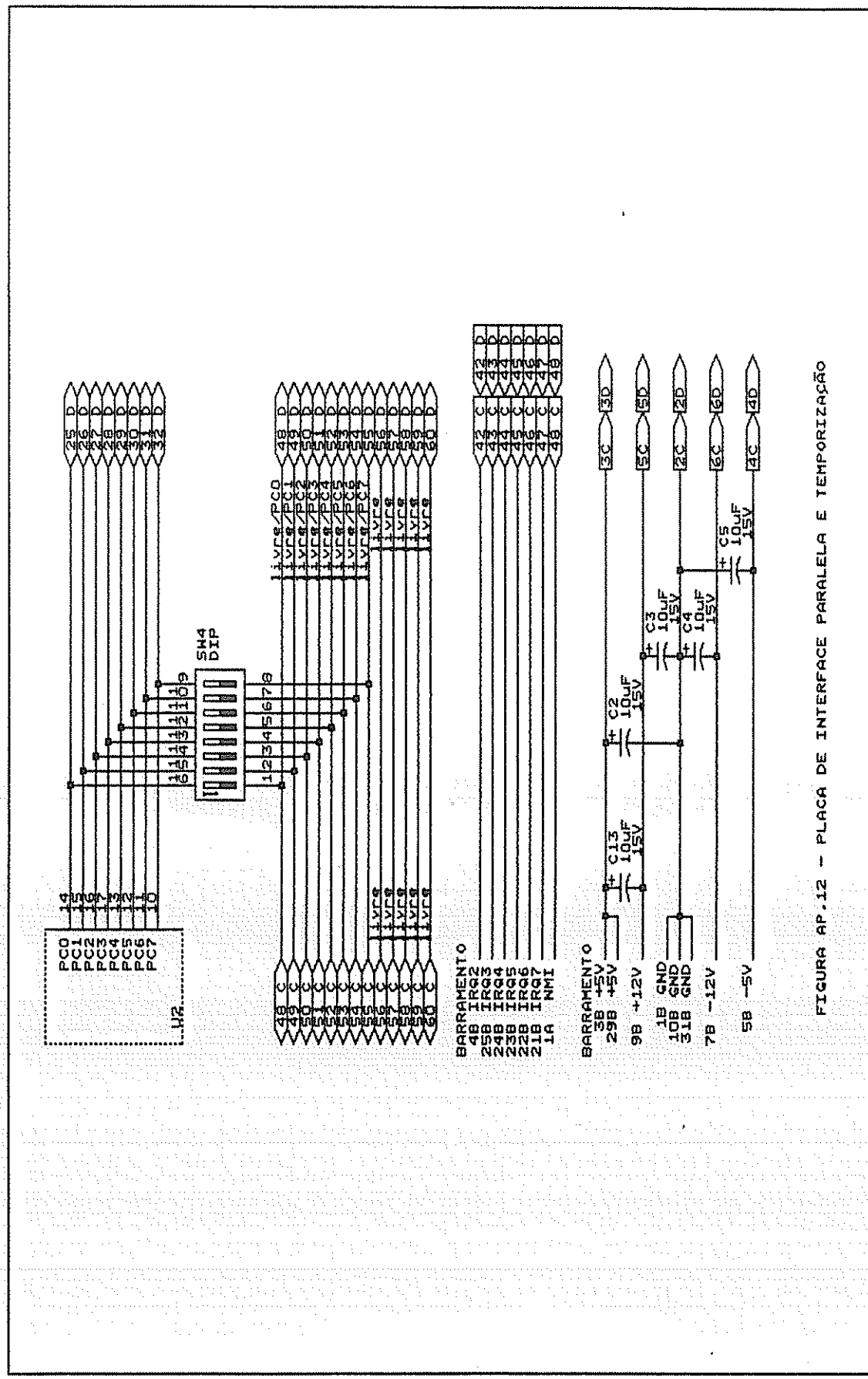
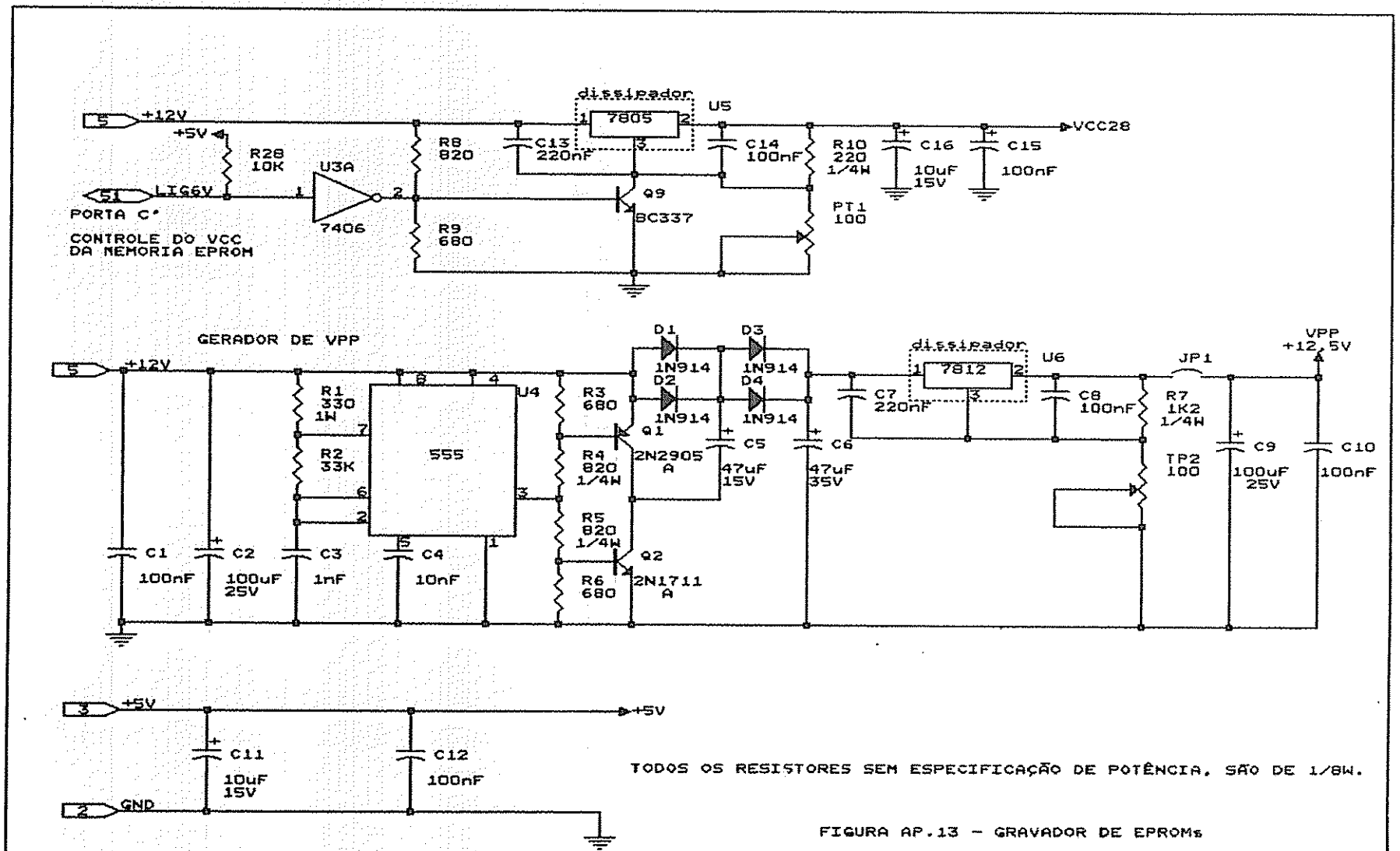
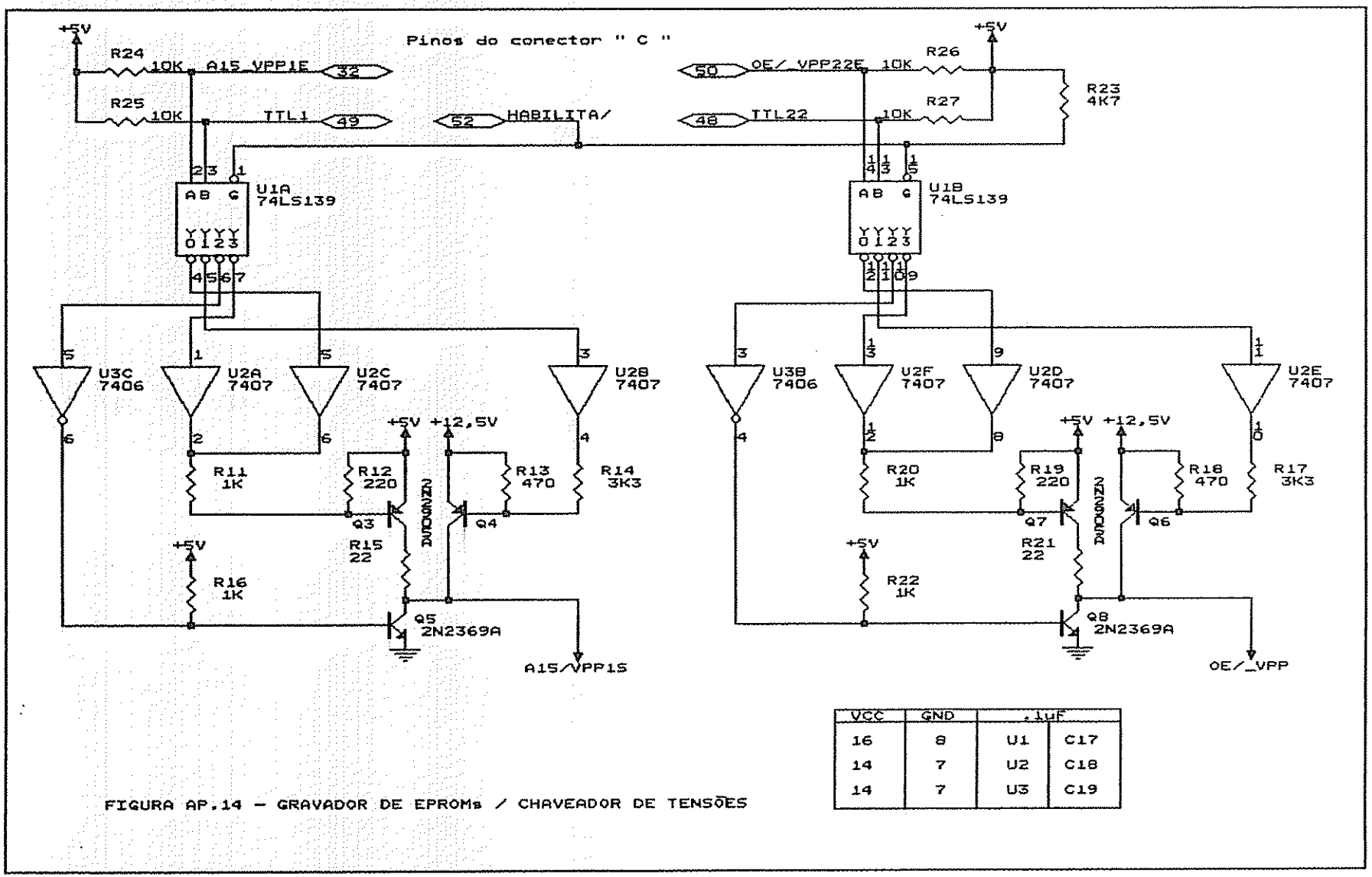


FIGURA AP.12 - PLACA DE INTERFACE PARALELA E TEMPORIZAÇÃO



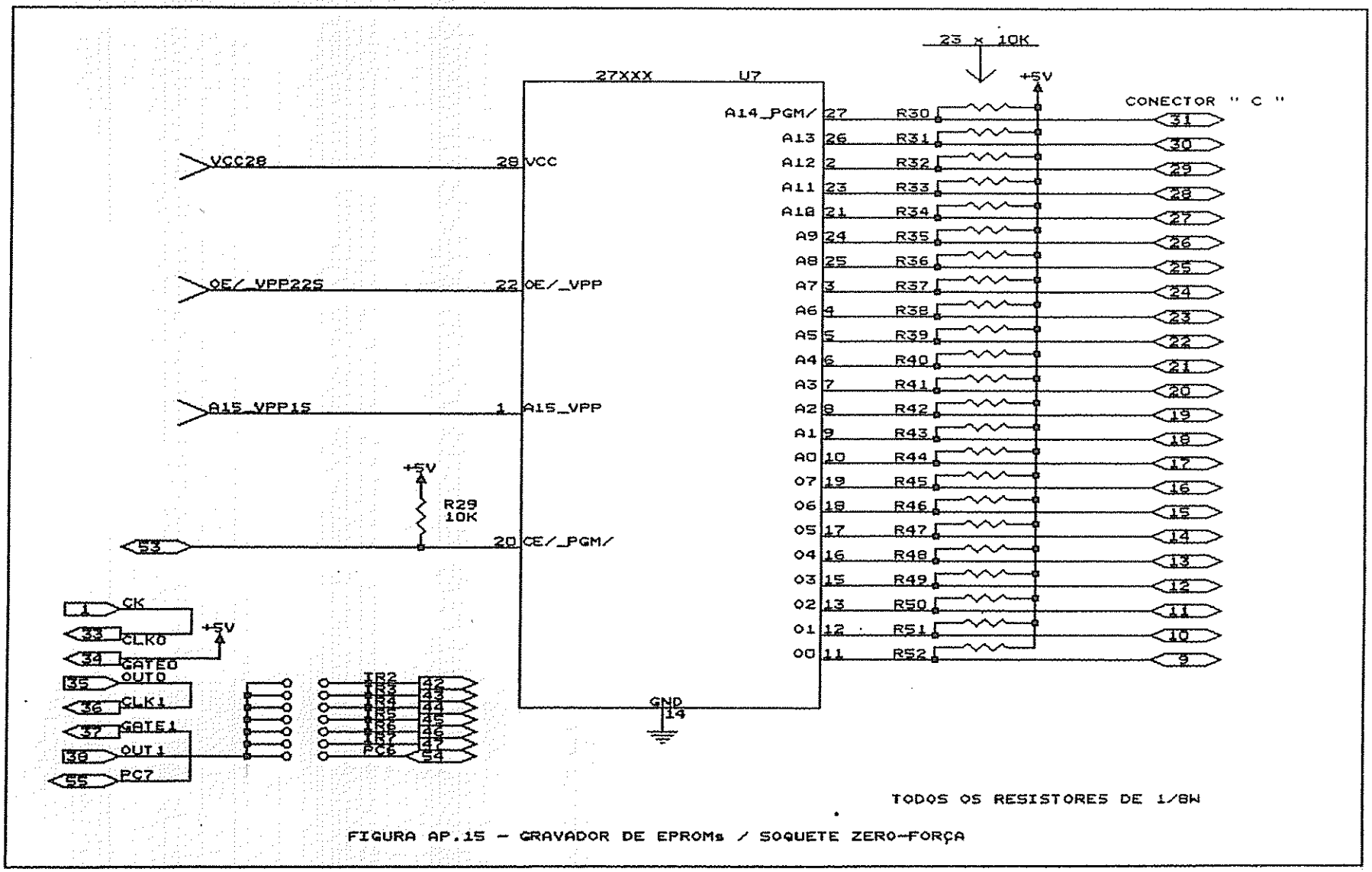
TODOS OS RESISTORES SEM ESPECIFICAÇÃO DE POTÊNCIA, SÃO DE 1/8W.

FIGURA AP.13 - GRAVADOR DE EPROMs



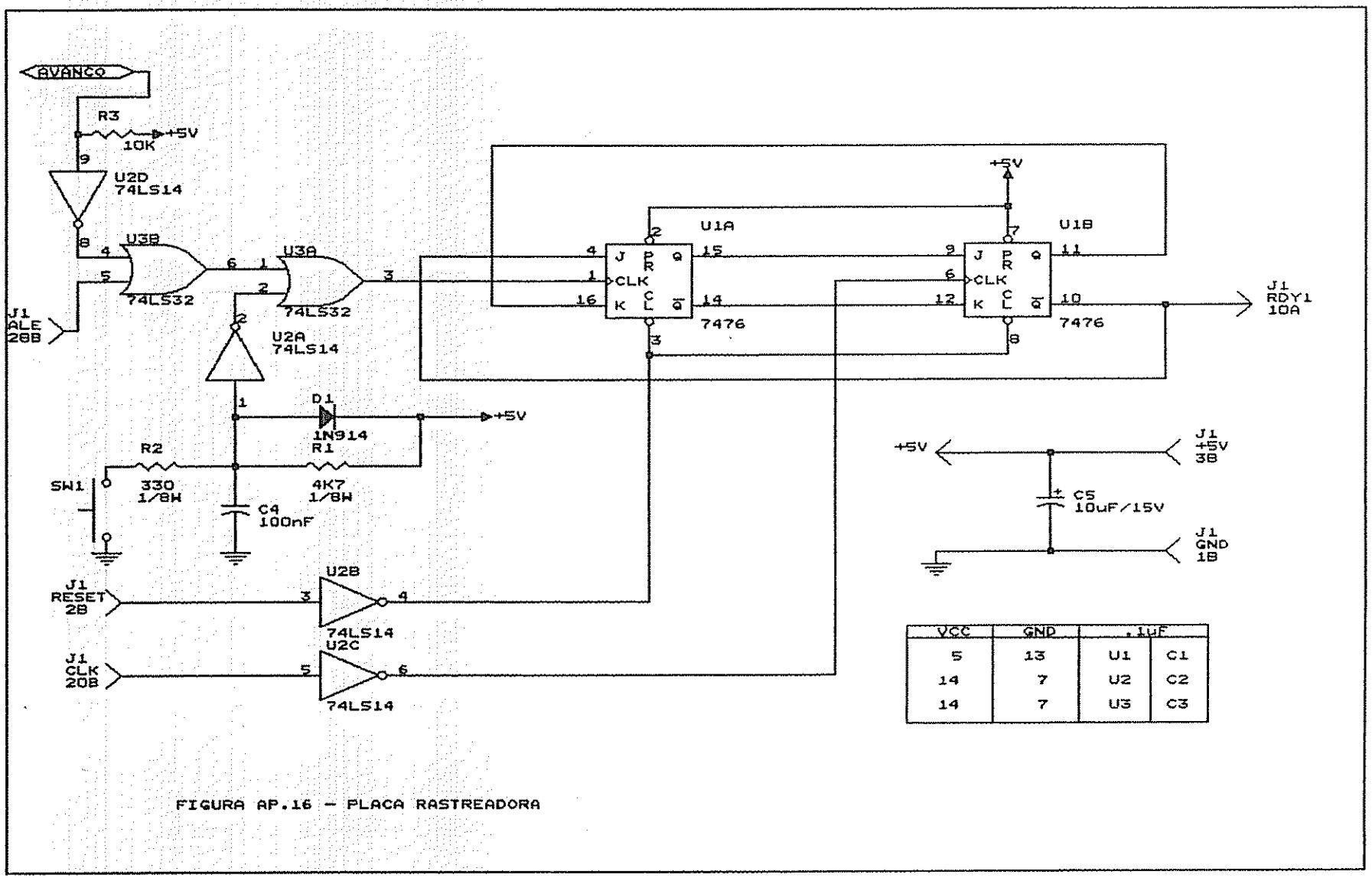
VCC	GND	.1uF	
16	8	U1	C17
14	7	U2	C18
14	7	U3	C19

FIGURA AP.14 – GRAVADOR DE EPROMs / CHAVEADOR DE TENSÕES



TODOS OS RESISTORES DE 1/8W

FIGURA AP.15 – GRAVADOR DE EPROMs / SOQUETE ZERO-FORÇA



VCC	GND	.1uF	
5	13	U1	C1
14	7	U2	C2
14	7	U3	C3

FIGURA AP.16 – PLACA RASTREADORA

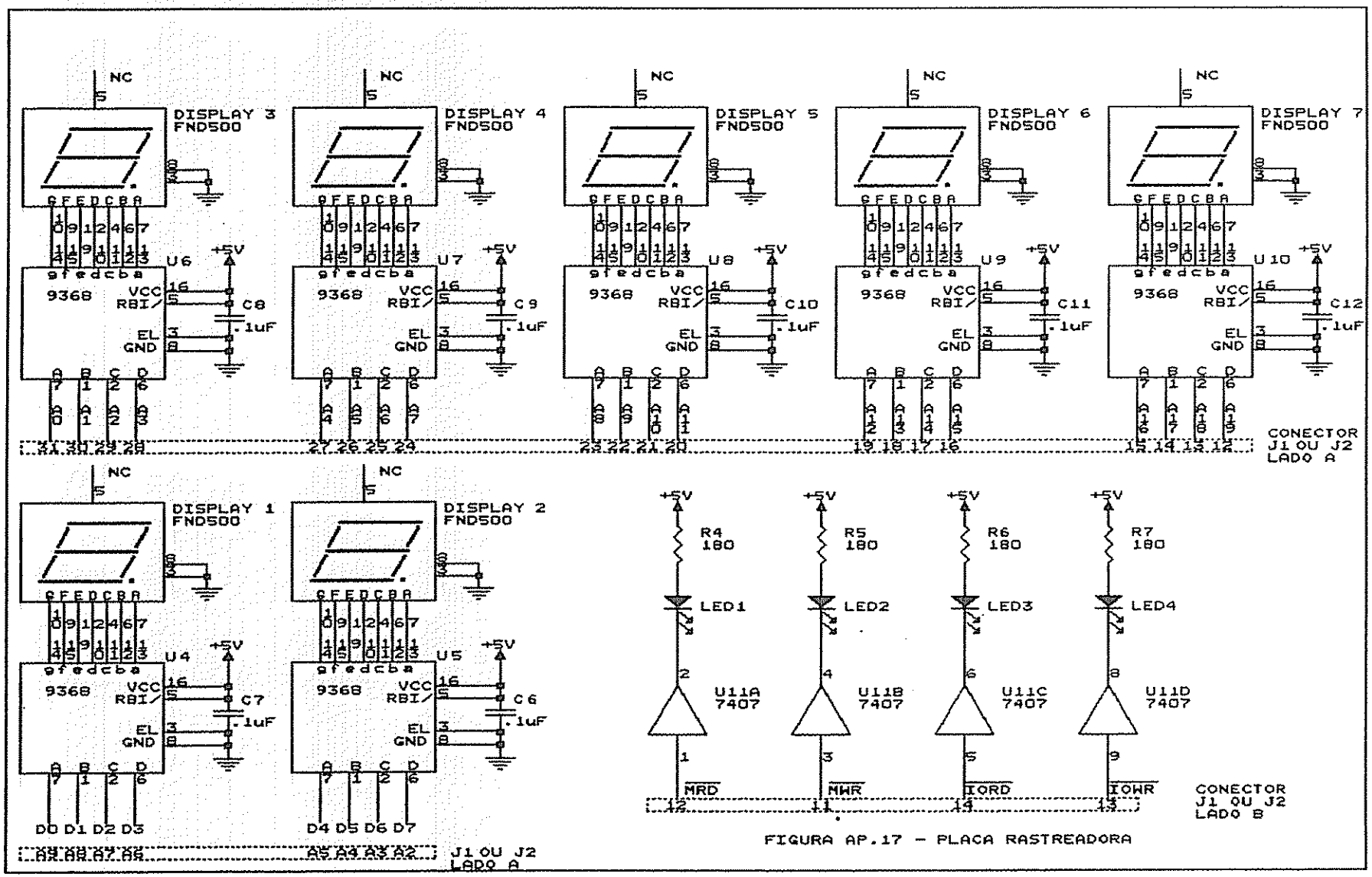


FIGURA AP.17 - PLACA RASTREADORA

APÊNDICE B

LISTAS DE MATERIAL

ITEM #	DESIGNAÇÃO NO ESQUEMA	COMPONENTE	QUANT	OBSERVAÇÕES
1	U16	8088-2	1	CPU
2	U15	8087-2	1	COPROCESSADOR
3	U21	8284	1	GERADOR DE CLOCK
4	U14	8288	1	CONTROLADOR DE VIA
5	U13	8259-A	1	CONTR. DE INT.
6	U18,U19,U20	74LS373	3	LATCH BUFFER
7	U17	74LS245	1	BUF. BIDIRECIONAL
8	U6	74LS139	1	DUPLO DECOD. 2-4
9	U5	74LS00	1	4 NÃO-E, 2 ENTR.
10	U9,U10,U11, U12	6116/6264	4	RAM (150n)2K/8K
11	U7,U8	2764/27128	2	EPROM (150n)8K/16K
12	U23,U24	8255-A5	2	PORTA PARAL. NEC
13	U4	8253-5	1	CONT.PROG. NEC
14	U3	8251-A	1	USART NEC
15	U1	1488	1	RS-232 SAIDA
16	U2	1489	1	RS-232 ENTRADA
17	U22,U25	74LS193	2	CONTADOR BIN.
18	C7..C29 C36,C37	.1µF/50V	25	CERÂMICO/DISCO
19	C30, C32..C35, C38..C41	10µF/15V	9	ELETROLITICO TÂNTALO
20	R2,R3,R4,R7	4K7 Ω 5% 1/8W	4	FILME DE CARBONO
21	R1	1K Ω 5% 1/8W	1	FILME DE CARBONO
22	R14,R15	1K Ω 5% 1/8W	1	FILME DE CARBONO

Tabela AP.1 - Lista dos componentes da Placa KD-16.

ITEM #	DESIGNAÇÃO NO ESQUEMA	COMPONENTE	QUANT	OBSERVAÇÕES
23	R8,R9	33Ω 5% 1/8W	2	FILME DE CARBONO
24	R5	100K Ω 5% 1/8W	1	FILME DE CARBONO
25	R6	100 Ω 5% 1/8W	1	FILME DE CARBONO
26	R10-R13	2K2 Ω 5% 1/8W	4	FILME DE CARBONO
27	D1	1N914	1	OPÇÃO: 1N4148
28	C31	3-30 pF	1	CAP. VAR. MINI
29	XTAL	24MHz	1	CRISTAL DE QUARTZO
30	C1-C6	390 pF	6	CERÂMICO/DISCO
31	S1	CHAVE DE CONTATO MOMENTÂNEO	1	1 POLO INVERSOR 1A-120V, MARCA JOTO OU MARGIRUS
32	DIP-S2	CHAVE DIP	1	4 CHAVES
33		SOQUETE	3	14 PINOS
34		SOQUETE	3	16 PINOS
35		SOQUETE	1	18 PINOS
36		SOQUETE	5	20 PINOS
37		SOQUETE	1	24 PINOS
38		SOQUETE	8	28 PINOS
39		SOQUETE	4	40 PINOS
40	J0	CONECTOR 8 PINOS, COM PARAFUSO	1	SOLDA DIST. ENTRE PINOS 5 mm
41	J5,J6	CONECTOR 20 PINOS,MACHO 180 GRAUS,FILA DUPLA,SEM POLARIZAÇÃO,CONTATOS QUADRADOS	2	FABRICANTE: AMP DO BRASIL (P/CABO PLANO)

Tabela AP.1 - Lista dos componentes da Placa KD-16.

ITEM #	DESIGNAÇÃO NO ESQUEMA	COMPONENTE	QUANT	OBSERVAÇÕES
42	J3	CONECTOR 60 PINOS,MACHO 180 GRAUS,FILA DUPLA,SEM POLARIZAÇÃO,CONTATOS QUADRADOS	1	FABRICANTE: AMP DO BRASIL (P/ CABO PLANO)
43	J1-J2	CONECTOR 62 PINOS,PASSO 2,54mm,	2	SOLDA (BARRAMENTO IBM PC)
44	J4	CONECTOR DP-25 FÊMEA,90 GRAUS	1	SOLDA,PASSO 2,54mm
45	PLACA KD-16	PLACA DE CIRC. IMPRESSO,DUPLA FACE,FUROS METALIZADOS,FIBRA DE VIDRO	1	

Tabela AP.1 - Lista dos componentes da Placa KD-16.

ITEM #	DESIGNAÇÃO NO ESQUEMA	COMPONENTE	QUANT	OBSERVAÇÕES
1	U1	74LS76	1	FLIP-FLOP JK, SENSIVEL A BORDA DE DESCIDA, COM PRESET E CLEAR
2	U2	74LS04	1	6 INVERSORES
3	R1	2K2 Ω 5% 1/8W	1	FILME DE CARBONO
4	C1,C2	.1 µF/50V	2	CERÂMICO/DISCO
5	C3,C4	10µF/15V	2	ELETROL.TÂNTALO
6		SOQUETE	1	16 PINOS SOLDA
7		SOQUETE	1	14 PINOS SOLDA
8		CONECTOR 2 PINOS MACHO 180 GRAUS SEM POLARIZAÇÃO CONTATOS QUADRADOS	2	FABRICANTE: AMP DO BRASIL P/ SELEÇÃO DO MODO DE OPERAÇÃO
9	PLACA WAIT	PLACA DE CIRC. IMPRESSO FIBRA DE VIDRO FUROS METALIZADOS BANHADOS A OURO	1	

Tabela AP.2 - Lista dos componentes do Gerador de "WAIT".

ITEM #	DESIGNAÇÃO NO ESQUEMA	COMPONENTE	QUANT	OBSERVAÇÕES
1	U1	8279-5	1	CONTR. TECLADO
2	U2	74LS240	1	BUFFER INVERSOR
3	U3	7445	1	DECOD. BCD-DECIMAL
4	U4	74LS156	1	DECODIFICADOR
5	Q1-Q16	BC 327-C	16	
6	M1-M8	MCD 198 A	8	7 SEG. ANODO COMUM
7		22 Ω 5% 1/8W	8	FILME DE CARBONO
8		2K2 Ω 5% 1/8W	8	FILME DE CARBONO
9		100 Ω 5% 1/8W	8	FILME DE CARBONO
10		22 Ω 5% 1/4W	8	FILME METALICO
11		.1µF/50V	4	CERÂMICO/DISCO
12		10µF/15V	12	ELETROL. TÂNTALO
13		TECLA NORMAL ABERTA	22	TECLADO 0..9, A..F , 6 ESPECIAIS
14		SOQUETE	2	16 PINOS SOLDA
15		SOQUETE	1	10 PINOS SOLDA
16		SOQUETE	1	40 PINOS SOLDA
17		CONECTOR 20 PINOS,MACHO 180 GRAUS,FILA DUPLA,SEM POL. CONT. QUADRADO	1	FABRICANTE: AMP DO BRASIL (P/ CABO PLANO)
18	PLACA DO TECLADO	PLACA DE CIRC. IMPRESSO DE FIBRA DE VIDRO DUPLA FACE COM FUROS METALI-ZADOS		

Tabela AP.3 - Lista dos componentes do Módulo do Teclado.

ITEM #	DESIGNAÇÃO NO ESQUEMA	COMPONENTE	QUANT	OBSERVAÇÕES
1	U1,U2	8255-A5	2	PORTA PARALELA
2	U3,U4	8253-5	2	CONT. PROGRAMAVEL
3	U5	74LS138	1	DECODIFICADOR 3-8
4	U6	74LS27	1	3 NOU DE 3 ENTR.
5	U7	74LS04	1	6 INVERSORES
6	XTAL	1 MHz	1	CRISTAL QUARTZO
7		SOQUETE	2	14 PINOS SOLDA
8		SOQUETE	1	16 PINOS SOLDA
9		SOQUETE	2	24 PINOS SOLDA
10		SOQUETE	2	40 PINOS SOLDA
11	R1,R2	1K Ω 5% 1/8W	2	FILME DE CARBONO
12	C1	.001μF/50V	1	CER&MICO/DISCO
13	C2-C5	10μF/15V	4	ELETROL. T&NTALO
14	C6-C12	.1μF/50V	7	CER&MICO/DISCO
15	SW4	DIP	1	8 CHAVES
16	C,D	CONECTORES DE 60 PINOS,MACHO 180 GRAUS,SEM POLARIZACAO, FILA DUPLA, CONTATOS QUAD.	2	FABRICANTE: AMP DO BRASIL (P/ CABO PLANO)
17	PLACA INTERFACE PARALELA	PLACA DE CIRC. IMPRESSO, DE FIBRA DE VIDRO FUROS METALI-ZADOS, CONECTORES COM CONTATOS BA-NHADOS A OURO	1	

Tabela AP.4 - Lista de componentes da Placa de Interface Paralela e Temporização.

ITEM #	DESIGNAÇÃO NO ESQUEMA	COMPONENTE	QUANT	OBSERVAÇÕES
1	U1	74LS139	1	DUPLO DECOD. 2-8
2	U2	7407	1	6 NÃO-INVERSORES DE COLETOR ABERTO
3	U3	7406	1	6 INVERSORES DE COLETOR ABERTO
4	U4	LM555	1	OSCILADOR
5	U5	LM7805	1	REGULADOR 5V/1A
6	U6	LM7812	1	REGULADOR 12V/1A
7		SOQUETE	1	8 PINOS SOLDA
8		SOQUETE	1	14 PINOS SOLDA
9		SOQUETE	1	16 PINOS SOLDA
10		SOQUETE TIPO "ZERO-FORÇA"	1	SOLDA MARCA: TEXTOOL
11	Q5,Q8	2N2369-A	2	
12	Q2,Q3,Q4, Q6,Q7	2N2905-A	5	
13	Q1	2N1711-A	1	
14	Q9	BC 337	1	
15	C1,C8,C10, C12,C14, C15,C17, C18,C19	.1µF/50V	9	CERÂMICO/DISCO
16	C7,C13	.22µF/50V	2	CERÂMICO/DISCO OU POLIESTER METALIZ.
17	C3	.001µF/50V	1	CERÂMICO/DISCO
18	C4	.01µF/50V	1	CERÂMICO/DISCO
19	C2,C9	100µF/25V	2	ELETROL. TÂNTALO
20	C5	47µF/15V	1	ELETROLITICO

Tabela AP.5 - Lista dos componentes do Gravador de EPROMs.

ITEM #	DESIGNAÇÃO NO ESQUEMA	COMPONENTE	QUANT	OBSERVAÇÕES
21	C6	47µF/35V	1	ELETROLITICO
22	C11,C16	10µF/15V	2	ELETROL. TÂNTALO
23	R1	330 Ω 5% 1W	1	FILME METALICO
24	R2	33K Ω 5% 1/8W	1	FILME METALICO
25	R3,R6,R9	680 Ω 5% 1/8W	3	FILME METALICO
26	R4,R5	820 Ω 5% 1/4W	2	FILME METALICO
27	R7	1K2 Ω 5% 1/4W	1	FILME METALICO
28	R8	820 Ω 5% 1/8W	1	FILME METALICO
29	R10	220 Ω 5% 1/4W	1	FILME METALICO
30	R11,R16, R20,R22	1K Ω 5% 1/8W	4	FILME METALICO
31	R12,R19	220 Ω 5% 1/8W	2	FILME METALICO
32	R13,R18	470 Ω 5% 1/8W	2	FILME METALICO
33	R14,R17	3K3 Ω 5% 1/8W	2	FILME METALICO
34	R15,R21	22 Ω 5% 1/8W	2	FILME METALICO
35	R23	4K7 Ω 5% 1/8W	1	FILME METALICO
36	R24-R52	10K Ω 5% 1/8W	29	FILME METALICO
37		CABO PLANO 60 VIAS	1	UM METRO
38	D1-D4	1N914	4	OPÇÃO:1N4148
39		DISSIPADOR DE CALOR PARA OS REG. 78XX	2	TAMANHO MAX. 2X3 CM
40		PARAFUSOS COM PORCA DIAMETRO 1/8" 1/4" DE COMPR.	2	

Tabela AP.5 - Lista dos componentes do Gravador de EPROMs.

ITEM #	DESIGNAÇÃO NO ESQUEMA	COMPONENTE	QUANT	OBSERVAÇÕES
41		CONECTOR PARA CABO PLANO, 60 PINOS,FÊMEA 180 GRAUS, FILA DUPLA, SEM POLARIZA-ÇÃO	2	FABRICANTE: AMP DO BRASIL
42	PLACA GRA-VADOR DE EPROM	PLACA DE CIRC. IMPRESSO DE FIBRA DE VIDRO FUROS METALI-ZADOS	1	

Tabela AP.5 - Lista dos componentes do Gravador de EPROMs.

ITEM #	DESIGNAÇÃO NO ESQUEMA	COMPONENTE	QUANT	OBSERVAÇÕES
1	U1	74LS76	1	FLIP-FLOP JK, SENSIVEL A BORDA DE DESCIDA, COM PRESET E CLEAR
2	U2	74LS14	1	6 INVERSORES SCHMITT-TRIGGER
3	U3	74LS32	1	4 OU DE DUAS ENTR.
4	U4-U10	9368	7	DECOD. HEX/7SEG.
5	U11	7407	1	BUF. COLET. ABERTO
6	M1-M7	MCD 198 K	7	MOSTRADOR 7SEG. CATODO COMUM
7	C1-C11,C13	.1µF/50V	12	CAR&MICO/DISCO
8	D1	1N914	1	OPÇÃO:1N4148
9	LED1-LED4	LED	4	CORES DIFERENTES
10	R1	4K7 Ω 5% 1/8W	1	FILME DE CARBONO
11	R2	330 Ω 5% 1/8W	1	FILME DE CARBONO
12	R3	10K Ω 5% 1/8W	1	FILME DE CARBONO
13	R4-R7	180 Ω 5% 1/8W	4	FILME DE CARBONO
14	C12	10µF/15V	1	ELETROL. T&NTALO
15		SOQUETE	2	14 PINOS SOLDA
16		SOQUETE	8	16 PINOS SOLDA
17		SOQUETE 40 PINOS	3	40 PINOS SOLDA P/ MOSTR. 7 SEG.
18	SW1	MICROCHAVE DE CONTATO MOMEN-T&NEO	1	1 POLO INVERSOR 1A/120V, MARCA JOTO OU MARGIRUS, PINOS PARA SOLDA EM 90 GRAUS

Tabela AP.6 - Lista dos componentes da Placa Rastreadora.

ITEM #	DESIGNAÇÃO NO ESQUEMA	COMPONENTE	QUANT	OBSERVAÇÕES
19	C	CONECTOR DE 60 PINOS,MACHO 180 GRAUS, SEM POLARIZAÇÃO, FILA DUPLA, CONTATOS QUADRADOS	1	FABRICANTE: AMP DO BRASIL (P/ CABO PLANO)
20	PLACA RASTREADORA	PLACA DE CIRC. IMPRESSO, FIBRA DE VIDRO FUROS METALIZADOS, CONECTORES COM CONTATOS BANHADOS A OURO	1	

Tabela AP.6 - Lista dos componentes da Placa Rastreadora.

APÊNDICE C

LISTAGEM DOS PROGRAMAS DESENVOLVIDOS

APÊNDICE C1

PROGRAMA TERMINAL

```
PROGRAM TERMINAL;

USES CRT;
{
 ESTE PROGRAMA TRANSFORMA O MICROCOMPUTADOR IBM-PC NUM TERMINAL PARA O
 KD-16.
}
VAR C,DADO_LIDO_DO_DISCO,
    SIN_NAO,BAUD            :CHAR;

    DATAIN,STATUS           :BYTE;

    NOMEDESTINO,
    NOMEFONTE               :STRING[15];

    POS_INIC,
    POS_FINAL,DESL          :STRING[4];

    DESTINO,
    FONTE,TEX               :TEXT;

    BUFFER                  :ARRAY[1..$E000] OF BYTE;

    I,J,IAUX                :WORD;

{-------------------------------------------------------------------------}

PROCEDURE SAI_HELP;

BEGIN
  WRITELN;
  WRITELN('Menu do KD-16:');
  WRITELN;
  WRITELN(' S -  S[W]<end.>,[[<novo conteudo>],]*<CR>');
  WRITELN(' X -  X[<reg>][[<novo conteudo>],]*<CR>');
  WRITELN(' D -  D[W]<end. inicial>[,<end. final>]<CR>');
  WRITELN(' M -  M<end. inicial>,<end. final>,<end. destino>');
  WRITELN(' I -  I[W]<end. porta>,[,]*<CR>');
  WRITELN(' O -  O[W]<end. porta>,<dado>[,<dado>]*<CR>');
  WRITELN(' G -  G[<end. inicial>][,<end. parada>]<CR>');
  WRITELN(' N -  N[<end. inicial>],[[<end. inicial>],]*<CR>');
  WRITELN;
  WRITELN('Menu do Teminal:');
  WRITELN;
  WRITELN('^R -  Lê um arquivo do disco para o KD-16.');
  WRITELN('^W -  Grava um arquivo com o conteúdo da memoria do KD-16.');
  WRITELN('^S -  Para impressão na tela.');
  WRITELN('^Q -  Continua impressão na tela.');
  WRITELN('^C -  Abandona comando.');
  WRITELN(' ? -  Pede menu.');
  WRITELN(' Q -  Retorna ao DOS.');
  WRITELN(' Se o nome dos arquivos for "ERRO" --> Abandona comando ^W ou ^R.');
  WRITELN('.');
END;
```

```
{-------------------------------------------------------------------}

PROCEDURE MENSSAGEM_INICIAL;
{
 DA OPÇÃO AO USUARIO PARA MUDAR A VELOCIDADE DE TRANSMISSÃO E INICIALIZA
 O CHIP 8250.
}
BEGIN
  CLRSCR;
  SIM_NAO:=' ';
  PORT[$3FB]:=$00; {LINE CONTROL REG. TX BUFFER="00"}
  PORT[$3F9]:=$00; {DESABILITA INTERRUPÇÕES}
  WRITELN('Terminal Versão 1.2');
  WRITE('Modificar Baud-Rate ( ATUAL 4800 ) ? S/N : ');
  REPEAT
     REPEAT UNTIL KEYPRESSED;
     SIM_NAO:=READKEY;
     SIM_NAO:=UPCASE(SIM_NAO);
  UNTIL SIM_NAO IN ['S','N'];
  WRITELN(SIM_NAO);
  CASE SIM_NAO OF
    'S' : BEGIN
          WRITELN('Escolha o Baud-Rate : ');
          WRITELN;
          WRITELN('    1 - 9600 ');
          WRITELN('    2 - 4800 ');
          WRITELN('    3 - 2400 ');
          WRITELN('    4 - 1200 ');
          WRITELN('    5 -  600 ');
          WRITELN('    6 -  300 ');
          WRITELN;
          WRITE('    Entre com a opção : ');
          REPEAT
            REPEAT UNTIL KEYPRESSED;
            BAUD:=READKEY;
          UNTIL BAUD IN ['1','2','3','4','5','6'];
          WRITELN(BAUD);
          CASE BAUD OF
           '1': BEGIN
                 PORT[$3FB]:=$87;{LINE CONTROL REG. -8BITS/2STOPS/PAR.NAO-
                                                    SETA BIT DO DIV. PROG.}
                 PORT[$3F8]:=$0C;{DIV. LATCH LSB}
                 PORT[$3F9]:=$00;{DIV. LATCH MSB}
                END;
           '2': BEGIN
                 PORT[$3FB]:=$87;
                 PORT[$3F8]:=$18;
                 PORT[$3F9]:=$00;
                END;
```

```
            '3': BEGIN
                   PORT[$3FB]:=$87;
                   PORT[$3F8]:=$30;
                   PORT[$3F9]:=$00;
                 END;
            '4': BEGIN
                   PORT[$3FB]:=$87;
                   PORT[$3F8]:=$60;
                   PORT[$3F9]:=$00;
                 END;
            '5': BEGIN
                   PORT[$3FB]:=$87;
                   PORT[$3F8]:=$C0;
                   PORT[$3F9]:=$00;
                 END;
            '6': BEGIN
                   PORT[$3FB]:=$87;
                   PORT[$3F8]:=$80;
                   PORT[$3F9]:=$01;
                 END;
          END;
        END;
    'N' : BEGIN
          PORT[$3FB]:=$87;
          PORT[$3F8]:=$18;
          PORT[$3F9]:=$00;
        END;
  END;
  PORT[$3FC]:=$03;{MODEL CONTROL REG. DTR/=0 RTS/=0}
  PORT[$3FB]:=$07;{8BITS/2STOP/PAR. NAO}
  DELAY(500);
  CLRSCR;
  SAI_HELP;
  WRITELN('Pressione o RESET no SD-16 e aguarde mensagem.');
END;

{-------------------------------------------------------------------------}

FUNCTION CARACTERE_PRONTO : BOOLEAN;
{
 VOLTA VERDADEIRO/FALSO CASO O BUFFER DE RECEPÇAO CONTENHA/NAO-CONTENHA
 UM DADO PRONTO.
}
BEGIN
  STATUS:=PORT[$3FD];{LE "LINE STATUS REG." - "01"-->DADO PRONTO}
  IF (STATUS AND $01) = $00 THEN CARACTERE_PRONTO := FALSE
  {MASC. 0000 0001)         ELSE CARACTERE_PRONTO := TRUE;
END;
```

```
{-------------------------------------------------------------------------}

PROCEDURE LE_BUFFER_DE_RECEPCAO(VAR DATA : BYTE);
{
 Lê CARACTERE DO BUFFER DE RECEPÇÃO MASCARANDO O BIT 7 PARA "0".
}
BEGIN
  DATA := PORT[$3F8]; ( LE RX BUFFER )
  DATAIN := DATA AND $7F; (MASC. 0111 1111)
END;

{-------------------------------------------------------------------------}

FUNCTION BUFFER_TX_VAZIO : BOOLEAN;
{
 VOLTA VERDADEIRO/FALSO CASO O REG. "TX HOLDING" ESTEJA VAZIO/CHEIO.
}
BEGIN
  STATUS:=PORT[$3FD];(LE "LINE STATUS REG." - "20"-->REG. TX VAZIO)
  IF (STATUS AND $20) = $00 THEN BUFFER_TX_VAZIO:=FALSE
  (MASC. 0010 0000)         ELSE BUFFER_TX_VAZIO:=TRUE;
END;

{-------------------------------------------------------------------------}

PROCEDURE SAI_CARACTERE_EM_TX(C : CHAR);
{
 ESCREVE CARACTERE ASCII NO BUFFER TX.
}
BEGIN
  PORT[$3F8]:=ORD(C);
END;

{-------------------------------------------------------------------------}

PROCEDURE LER_ARQ_DO_DISCO;
{
 PROCESSA O COMANDO "R" TRANSFERINDO UM ARQUIVO NO FORMATO INTEL MCS/86
 PARA O KD-16.
}
BEGIN
  WRITELN('^R');WRITELN;

  WRITE('Nome do Arquivo : ');
  READLN(NOMEFONTE);
  WRITELN;

  FOR I:=1 TO LENGTH(NOMEFONTE)
    DO NOMEFONTE[I]:=UPCASE(NOMEFONTE[I]);

  IF NOMEFONTE='ERRO' THEN BEGIN
                             WRITELN('.');
                             EXIT;
                           END;
```

```
DESL:='';
WRITE('Deslocamento ( ATUAL 0000H ) :');
READLN(DESL);
WRITELN;
IF DESL='' THEN DESL:='0000';

FOR I:=1 TO LENGTH(DESL)
  DO DESL[I]:=UPCASE(DESL[I]);

IF DESL='ERRO' THEN BEGIN
                      WRITELN('.');
                      EXIT;
                    END;


WRITELN('Aguarde alguns instantes.');
WRITELN;

ASSIGN(FONTE,NOMEFONTE);
RESET(FONTE);

I:=0;
WRITELN('Carregando ( ',NOMEFONTE,' ) para o buffer.');

REPEAT  ( ENCHE BUFFER )
  BEGIN
    I:=I+1;
    READ(FONTE,DADO_LIDO_DO_DISCO);
    BUFFER[I]:=ORD(DADO_LIDO_DO_DISCO);
    GOTOXY(1,25);
    WRITE(I:6);
  END;
UNTIL EOF(FONTE);

WRITELN;WRITELN;
WRITELN('Conteudo do buffer:');
IAUX:=0;
REPEAT
    IAUX:=IAUX+1;
    IF BUFFER[IAUX] IN [$0D,$0A,$20..$FF] THEN WRITE(CHR(BUFFER[IAUX]));
UNTIL IAUX=I;

   REPEAT UNTIL BUFFER_TX_VAZIO;
   SAI_CARACTERE_EM_TX('R');

   FOR J:=1 TO LENGTH(DESL)
     DO BEGIN
          REPEAT UNTIL BUFFER_TX_VAZIO;
          SAI_CARACTERE_EM_TX(DESL[J]);
        END;

   REPEAT UNTIL BUFFER_TX_VAZIO;
   SAI_CARACTERE_EM_TX(CHR($0D));
```

```
 IAUX:=0;
 WRITELN;WRITELN;
 WRITELN('Inicio da recepção de dados no KD-16.');
 WRITELN;
 REPEAT  { DESCARREGA BUFFER }
   BEGIN
     IAUX:=IAUX+1;
     REPEAT UNTIL ((BUFFER_TX_VAZIO) OR (CARACTERE_PRONTO));
     IF CARACTERE_PRONTO THEN
     BEGIN
       LE_BUFFER_DE_RECEPCAO(DATAIN);
       IF DATAIN=ORD('#') THEN BEGIN
                                 IAUX:=1;
                                 GOTOXY(1,25);
                                 WRITELN(CHR(7),'ERRO NA RECEPCAO DE DADOS !');
                                 EXIT;
                               END;
     END;
     SAI_CARACTERE_EM_TX(CHR(BUFFER[IAUX]));
   END;
 UNTIL IAUX=I-2;     { NAO ENVIA OS DOIS ULTIMOS CARACTERES "CR" E "LF" }
 CLOSE(FONTE);
 WRITELN('Recepção concluida e OK.');
 WRITELN;
END;

{-------------------------------------------------------------------------}

PROCEDURE GRAVA_ARQ_NO_DISCO;
{
 PROCESSA O COMANDO "W" CRIANDO UM ARQUIVO EM DISCO COM O CONTEUDO DE
 UMA FAIXA DE MEMORIA DO KD-16. (FORMATO INTEL MCS/86)
}
BEGIN

{ AQUI SAO OBTIDOS: NOME_DO_ARQ. / END. INICIAL / END. FINAL }

 WRITELN('^W');WRITELN;

 WRITE('Nome do Arquivo  : ');
 READLN(NOMEDESTINO);WRITELN;

 FOR I:=1 TO LENGTH(NOMEDESTINO)
   DO NOMEDESTINO[I]:=UPCASE(NOMEDESTINO[I]);

 IF (NOMEDESTINO='ERRO') THEN BEGIN
                                WRITELN('.');
                                EXIT;
                              END;
```

```
  WRITELN('Abrindo um arquivo com o nome < ',NOMEDESTINO,' >.');
  WRITELN;

  ASSIGN(DESTINO,NOMEDESTINO);
  REWRITE(DESTINO);

  WRITE('Endereço Inicial : ');
  READLN(POS_INIC);

  FOR I:=1 TO LENGTH(POS_INIC)
    DO POS_INIC[I]:=UPCASE(POS_INIC[I]);

  IF (POS_INIC='ERRO') THEN BEGIN
                              WRITELN('.');
                              EXIT;
                            END;

  WRITE('Endereço Final   : ');
  READLN(POS_FINAL);WRITELN;

  FOR I:=1 TO LENGTH(POS_FINAL)
    DO POS_FINAL[I]:=UPCASE(POS_FINAL[I]);

  IF (POS_FINAL='ERRO') THEN BEGIN
                              WRITELN('.');
                              EXIT;
                            END;

  WRITELN('Aguarde Alguns Instantes.');
  WRITELN;

{ AQUI, ENVIA O COMANDO "W" NA FORMA: W iiii,ffff <CR> }

  REPEAT UNTIL BUFFER_TX_VAZIO;
  SAI_CARACTERE_EM_TX('W');

  FOR I:=1 TO LENGTH(POS_INIC)
  DO BEGIN
       REPEAT UNTIL BUFFER_TX_VAZIO;
       SAI_CARACTERE_EM_TX(POS_INIC[I]);
     END;

  REPEAT UNTIL BUFFER_TX_VAZIO;
  SAI_CARACTERE_EM_TX(',');

  FOR I:=1 TO LENGTH(POS_FINAL)
  DO BEGIN
       REPEAT UNTIL BUFFER_TX_VAZIO;
       SAI_CARACTERE_EM_TX(POS_FINAL[I]);
     END;

  REPEAT UNTIL BUFFER_TX_VAZIO;
  SAI_CARACTERE_EM_TX(CHR(13));
```

```
{ FORMA UM BUFFER NA MEMORIA COM OS DADOS ENVIADOS PELO KD-16 }

  WRITELN;
  WRITELN('Carregando buffer com dados do KD-16.');

  I:=1;
  BUFFER[I]:=$3A;  { : }

  REPEAT
    REPEAT UNTIL CARACTERE_PRONTO;
    LE_BUFFER_DE_RECEPCAO(DATAIN);
  UNTIL(DATAIN=$3A);

  REPEAT
    IF CARACTERE_PRONTO THEN
      BEGIN
        LE_BUFFER_DE_RECEPCAO(DATAIN);
        IF (DATAIN<>$00) AND (DATAIN<>$2E) THEN BEGIN
                                                 I:=I+1;
                                                 BUFFER[I]:=DATAIN;
                                                 GOTOXY(1,25);
                                                 WRITE(I:6);
                                                END;
      END;
  UNTIL DATAIN=$2E;

  WRITELN;WRITELN;
  WRITELN('Gravando buffer no disco.');

  FOR IAUX:=1 TO I-2 DO BEGIN
                          WRITE(DESTINO,CHR(BUFFER[IAUX]));
                          WRITE(CHR(BUFFER[IAUX]));
                        END;
  WRITE('.');

  CLOSE(DESTINO);
END;
```

```
{==================================================================}

BEGIN
{
 INICIO DO PROGRAMA PRINCIPAL --> TERMINAL
}
  MENSSAGEM_INICIAL;
  C:=' ';
  ASSIGNCRT(TEX); { USA ROTINAS RAPIDAS DA UNIDADE CRT }
  REPEAT
    IF CARACTERE_PRONTO
    THEN BEGIN
          LE_BUFFER_DE_RECEPCAO(DATAIN);
          IF (DATAIN=$0D) THEN BEGIN
                               PORT[$3FC]:=$00;
                               WRITE(CHR($0D),CHR($0A));
                               PORT[$3FC]:=$03;
                              END;
          IF (DATAIN >= $20) THEN WRITE(CHR(DATAIN));
        END
    ELSE BEGIN
          IF KEYPRESSED
          THEN BEGIN
                C:=READKEY;
                C:=UPCASE(C);
                CASE C OF
                CHR(18) : BEGIN   { CTRL R }
                           LER_ARQ_DO_DISCO;
                          END;
                CHR(23) : BEGIN   { CTRL W }
                           GRAVA_ARQ_NO_DISCO;
                          END;
                '?'     : BEGIN
                           SAI_HELP;
                          END;
               END;

          IF ( C<>CHR(18) ) AND ( C<>CHR(23) ) AND ( C<>'?' )
          THEN BEGIN
                REPEAT UNTIL BUFFER_TX_VAZIO;
                SAI_CARACTERE_EM_TX(C);
               END;
         END;
      END;

  UNTIL C='Q';
  WRITELN;
  WRITELN('Saindo do programa terminal.');
  DELAY(1000);
END.

{XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX}
```

APÊNDICE C2

PROGRAMA GRAVADOR

```
PROGRAM GRAVADOR;

USES CRT,DOS,PRINTER;

{  VARIAVEIS GLOBAIS  }

TYPE  STR2=STRING[2];
      STR3=STRING[3];
      STR4=STRING[4];
      STR12=STRING[12];
      STR80=STRING[80];
      STR34=STRING[34];

CONST TAMANHO_DO_BUFFER=61440; { F000 }

VAR   C_LINHA,C:BYTE;
      TIPO_DE_MEMORIA:STR3;
      BUFFER:ARRAY[0..TAMANHO_DO_BUFFER] OF BYTE;
      NOMEFONTE,NOMEDESTINO:STR12;
      FUNCAO:STR34;
      FONTE,DESTINO:FILE OF BYTE;
      CH:CHAR;
      END_INIC_EP:STRING[4];
      END_INIC_BUF:STRING[4];
      END_FIM_EP:STRING[4];
      END_FIM_BUF:STRING[4];
      END_INICIAL_BUFFER:WORD;
      END_FINAL_BUFFER:WORD;
      END_INICIAL_EPROM:WORD;
      END_FINAL_EPROM:WORD;
      CAMPO:INTEGER;
      NOME_DO_DIRETORIO,DIRETORIO_INICIAL:STRING[20];
      DRIVE_NUMERO:INTEGER;
      PRINTER_ON:BOOLEAN;

CONST H=TRUE;
      L=FALSE;
      U1_A=$300;
      U1_B=$301;
      U1_C=$302;
      U1_CTRL=$303;
      U2_C_LINHA=$306;
      U2_CTRL=$307;
      LE_27XXX=$90;
      ESCREVE_NA_27XXX=$80;
      C_LINHA_SAIDA=$92;
      FUN0='0- Copia eproa   ---> buffer     ';
      FUN1='1- Grava buffer  ---> eproa      ';
      FUN2='2- Copia arquivo ---> buffer     ';
      FUN3='3- Grava buffer  ---> arquivo    ';
      FUN4='4- Compara eproa <--> buffer     ';
      FUN5='5- Verifica eproa apagada        ';
      FUN6='6- Assinatura eproa/buffer       ';
      FUN7='7- Lista conteúdo do buffer      ';
```

```
    FUN8='8- Liga / Desliga impressora      ';
    FUN9='9- Edita conteúdo do buffer       ';
    FUNA='A- Procura bloco no buffer        ';
    FUNB='B- Move bloco no buffer           ';
    FUNC='C- Conversores Mem <---> Intel    ';
    FUND='D- Diretório                      ';
    FUNE='E- Enche buffer com byte          ';
    FUNF='F- Terminal KD-16                 ';
    FUN_SETA=CHR(27)+CHR(26)+' Muda o campo ativo';
    FUN_ESC='ESC-Abandona programa';
    FUN_ENTER='ENTER-Inicia função';
    X:ARRAY[1..10] OF BYTE = (59,4,21,49,49,71,71,45,71,55);
    Y:ARRAY[1..10] OF BYTE = (2,9,9,8,10,8,10,12,14,19);
    { X[1]=59  Y[1]=2      POSIÇÃO DO TIPO DE MEMORIA       }
    { X[2]=4   Y[2]=9      POSIÇÃO DO NOME DO ARQ. FONTE    }
    { X[3]=21  Y[3]=9      POSIÇÃO DO NOME DO ARQ. DESTINO  }
    { X[4]=49  Y[4]=8      POSIÇÃO DO END. INICIAL DA EPROM }
    { X[5]=49  Y[5]=10     POSIÇÃO DO END. FINAL DA EPROM   }
    { X[6]=71  Y[6]=8      POSIÇÃO DO END. INICIAL DO BUFFER }
    { X[7]=71  Y[7]=10     POSIÇÃO DO END. FINAL DO BUFFER  }
    { X[8]=45  Y[8]=12     POSIÇÃO DO CAMPO DE FUNÇÕES      }
    { X[9]=71  Y[9]=14     POSIÇÃO DO STATUS DA IMPRESSORA  }
    { X[10]=55 Y[10]=19    POSIÇÃO DO DIRETORIO             }

{-------------------------------------------------------------------------}

PROCEDURE PRIMEIRA_INICIALIZACAO;

BEGIN
  PORT[U1_CTRL]:=LE_27XXX;       { "A" ENTRADA , "B" E "C" SAIDA  }
  PORT[U1_B]:=$00;
  PORT[U1_C]:=$00;
  PORT[U2_CTRL]:=C_LINHA_SAIDA;  { "A" E "B" ENTRADAS E "C" SAIDA }
  PORT[U2_C_LINHA]:=$A7;
  C:=$00;                        { C'7 --> "1" FIXO       }
  C_LINHA:=$A7;                  { C'6 --> "0" FIXO       }
                                 { C'5 --> "1" DESAB. MEM. }
                                 { C'4 --> "0" DECOD. HAB. }
                                 { C'3 --> "0" VCC=+5V    }
                                 { C'2 --> "1" OE/ DESAB.  }
                                 { C'1 --> "1" TTL1=TTL   }
                                 { C'0 --> "1" TTL22=TTL  }
END;

{-------------------------------------------------------------------------}

PROCEDURE INICIALIZA_PELO_TIPO_DE_MEMORIA;

BEGIN
  IF (TIPO_DE_MEMORIA='64 ') OR (TIPO_DE_MEMORIA='128')
    THEN BEGIN
           C:=$40;                 { 2764/27128 }
           C_LINHA:=$A5;
         END
```

```
   ELSE IF (TIPO_DE_MEMORIA='256')
         THEN BEGIN
                C:=$00;           ( 27256 )
                C_LINHA:=$A5;
              END
         ELSE BEGIN
                C:=$00;           ( 27512 )
                C_LINHA:=$A7;
              END;
  PORT[U1_C]:=C;
  PORT[U2_C_LINHA]:=C_LINHA;
END;

{-------------------------------------------------------------------------}

PROCEDURE SAI_ENDERECO(ENDI:WORD);

BEGIN
  PORT[U1_B]:=LO(ENDI);

  IF TIPO_DE_MEMORIA='64 '
  THEN C:=(( C AND $E0 ) OR ( HI(ENDI) AND $1F ))
  ELSE IF TIPO_DE_MEMORIA='128'
       THEN C:=(( C AND $C0 ) OR ( HI(ENDI) AND $3F ))
       ELSE IF TIPO_DE_MEMORIA='256'
            THEN C:=(( C AND $80) OR ( HI(ENDI) AND $7F))
            ELSE IF TIPO_DE_MEMORIA='512'
                 THEN C:=HI(ENDI);
  PORT[U1_C]:=C;
END;

{    X --> ENDERECO        Y --> CONTROLE                              }
{                                                                      }
{ 2764  000X XXXX    27128  00XX XXXX    27256  0XXX XXXX   27512  XXXX XXXX }
{       YYY0 0000           YY00 0000           Y000 0000                  }
{      ---------           ----------          ----------                  }
{       YYYX XXXX           YYXX XXXX           YXXX XXXX                  }
```

```
{-------------------------------------------------------------------------}

FUNCTION LE_BYTE(END1:WORD):BYTE;  { CHAMA SAI_ENDERECO }

BEGIN
  SAI_ENDERECO(END1);

  C_LINHA:=C_LINHA AND $DF;      { CE/ --> "0" }
  PORTIU2_C_LINHA]:=C_LINHA;

  C_LINHA:=C_LINHA AND $FB;      { OE/ --> "0" }
  PORTIU2_C_LINHA]:=C_LINHA;

  LE_BYTE:=PORTIU1_A];

  C_LINHA:=C_LINHA OR $04;       { OE/ --> "1" }
  PORTIU2_C_LINHA]:=C_LINHA;

  C_LINHA:=C_LINHA OR $20;       { CE/ --> "1" }
  PORTIU2_C_LINHA]:=C_LINHA;
END;

{-------------------------------------------------------------------------}

PROCEDURE PULSAR_PGM_E_VPP(DURACAO:INTEGER);

BEGIN
  IF (TIPO_DE_MEMORIA='64 ') OR (TIPO_DE_MEMORIA='128')
  THEN BEGIN
       C:=C OR $80;                  { VPP1 --> +12,5V }
       PORTIU1_C]:=C;

       C_LINHA:=C_LINHA AND $DF;     { CE/  --> "0" }
       PORTIU2_C_LINHA]:=C_LINHA;

       C:=C AND $BF;                 { PGM/ --> "0" }
       PORTIU1_C]:=C;

       DELAY(DURACAO);

       C:=C OR $40;                  { PGM/ --> "1" }
       PORTIU1_C]:=C;

       C_LINHA:=C_LINHA OR $20;      { CE/ --> "1" }
       PORTIU2_C_LINHA]:=C_LINHA;

       C:=C AND $7F;                 { VPP1 --> +5V }
       PORTIU1_C]:=C;
      END
```

```
ELSE BEGIN
      IF (TIPO_DE_MEMORIA='256')
      THEN BEGIN
            C:=C OR $80;                { VPP1 --> +12,5V }
            PORT[U1_C]:=C;

            C_LINHA:=C_LINHA AND $DF;   { CE/-PGM/ = PGM/ --> "0" }
            PORT[U2_C_LINHA]:=C_LINHA;

            DELAY(DURACAO);

            C_LINHA:=C_LINHA OR $20;    { CE/-PGM/ = PGM/ --> "1" }
            PORT[U2_C_LINHA]:=C_LINHA;

            C:=C AND $7F;               { VPP1 --> +5V }
            PORT[U1_C]:=C;
          END
      ELSE BEGIN
            C_LINHA:=C_LINHA OR $04;       { OE/-VPP22 = OE/ -->"1" DESAB. SAIDAS  }
            PORT[U2_C_LINHA]:=C_LINHA;

            C_LINHA:=C_LINHA AND $FE;      { TTL22 -->"0"  AQUI +5V --> +12,5V     }
            PORT[U2_C_LINHA]:=C_LINHA;     {                     OE/-VPP22=VPP22   }

            C_LINHA:=C_LINHA AND $DF;      { CE/-PGM/ -->"0" AQUI COMEÇA INTERVALO }
            PORT[U2_C_LINHA]:=C_LINHA;     {                 GRAVAÇXO              }

            DELAY(DURACAO);                { CONTROLA DURAÇXO DO PULSO DE GRAVAÇXO }

            C_LINHA:=C_LINHA OR $20;       { CE/-PGM/ -->"1" AQUI TERMINA INTERVALO}
            PORT[U2_C_LINHA]:=C_LINHA;     {                 GRAVAÇXO              }

            C_LINHA:=C_LINHA OR $01;       { TTL22 -->"1" AQUI +12,5V --> +5V      }
            PORT[U2_C_LINHA]:=C_LINHA;
          END;
    END;
END;

{-------------------------------------------------------------------}

FUNCTION BYTE_OK(ENDI:WORD;DADO:BYTE):BOOLEAN;  { CHAMA LE_BYTE }

BEGIN
  IF DADO=LE_BYTE(ENDI) THEN BYTE_OK:=TRUE
                        ELSE BYTE_OK:=FALSE;
END;
```

```
{-------------------------------------------------------------------------}

PROCEDURE GRAVA_BYTE(ENDI:WORD;DADO:BYTE;DURACAO:INTEGER);
 {     CHAMA    SAI_ENDERECO , PULSAR_PGM_E_VPP      }
BEGIN
  PORT[U1_CTRL]:=ESCREVE_NA_27XXX;
  SAI_ENDERECO(ENDI);
  PORT[U1_A]:=DADO;
  PULSAR_PGM_E_VPP(DURACAO);
  PORT[U1_CTRL]:=LE_27XXX;
END;

{-------------------------------------------------------------------------}

PROCEDURE PROGRAMACAO_INTELIGENTE(VAR GRAVACAO_OK:BOOLEAN);
   {         CHAMA   GRAVA_BYTE , BYTE_OK        }
VAR INDICE,END_EP,END_BU:WORD;
   X_CONT:BYTE;

BEGIN
  C_LINHA:=C_LINHA OR $08;       { VCC --> +6V    C'3 --> "1" }
  PORT[U2_C_LINHA]:=C_LINHA;
  DELAY(200);                    {      TEMPO PARA SETAR      }

  GRAVACAO_OK:=TRUE;
  INDICE:=0;
  REPEAT
    BEGIN
      END_EP:=END_INICIAL_EPROM+INDICE;
      END_BU:=END_INICIAL_BUFFER+INDICE;
      X_CONT:=0;

      REPEAT
        BEGIN
          GRAVA_BYTE(END_EP,BUFFER[END_BU],1);
          X_CONT:=X_CONT+1;
        END;
      UNTIL BYTE_OK(END_EP,BUFFER[END_BU]) OR (X_CONT=25);

      IF NOT(BYTE_OK(END_EP,BUFFER[END_BU])) THEN GRAVACAO_OK:=FALSE;

      IF GRAVACAO_OK THEN GRAVA_BYTE(END_EP,BUFFER[END_BU],(X_CONT*3));

      INDICE:=INDICE+1;
    END;
  UNTIL (END_EP=END_FINAL_EPROM) OR NOT(GRAVACAO_OK) OR
        (KEYPRESSED AND (READKEY=CHR(27)) OR (END_BU=END_FINAL_BUFFER));

  C_LINHA:=C_LINHA AND $F7;      { VCC --> +5V     C'3 --> "0" }
  PORT[U2_C_LINHA]:=C_LINHA;
  DELAY(200);                    {      TEMPO PARA SETAR      }
```

```
  { VERIFICAÇÃO FINAL }

  INDICE:=0;
  REPEAT
    BEGIN
      END_EP:=END_INICIAL_EPROM+INDICE;
      END_BU:=END_INICIAL_BUFFER+INDICE;
      IF NOT(BYTE_OK(END_EP,BUFFER[END_BU])) THEN GRAVACAO_OK:=FALSE;
      INDICE:=INDICE+1;
    END;
  UNTIL (END_EP=END_FINAL_EPROM) OR NOT(GRAVACAO_OK) OR
        (KEYPRESSED AND (READKEY=CHR(27)) OR (END_BU=END_FINAL_BUFFER));
END;

{*****************************************************************************}

FUNCTION VALEHEX:BOOLEAN;
{
 RETORNA TRUE/FALSE SE A VARIAVEL "CH" FOR CARACTERE HEX VALIDO OU NÃO
}
BEGIN
  IF( CH IN ['0'..'9','A'..'F']) THEN VALEHEX:=TRUE ELSE VALEHEX:=FALSE;
END;

{---------------------------------------------------------------------------}

FUNCTION VALENOME:BOOLEAN;
{
 RETORNA TRUE/FALSE SE A VARIAVEL "CH" FOR VALIDA
 PARA COMPOR UM NOME DE ARQUIVO.
}
BEGIN
  IF(CH IN[CHR(8),'_',' ','!','#'..')',',','..'.','0'..'9','@'..'Z','-','a'..'{','}','~']
  THEN VALENOME:=TRUE
  ELSE VALENOME:=FALSE;
END;

{---------------------------------------------------------------------------}

PROCEDURE P(CAR,N:BYTE);
{
 IMPRIME NA POSIÇÃO DO CURSOR "N" BYTES ASCII INDICADOS POR "CAR".
}
VAR I:BYTE;

BEGIN
  FOR I:=1 TO N DO WRITE(CHR(CAR));
END;
```

```
{-------------------------------------------------------------------------}

PROCEDURE IMPRIME(XP,YP:BYTE;FRASE:STR80;BRILHO:BOOLEAN);
{
 IMPRIME UMA STRING DE ATÉ 80 CARACTERES COM O ATRIBUTO
 BRILHO, NA POSIÇÃO X,Y.
}
BEGIN
  IF (XP IN [1..80]) AND (YP IN [1..25]) THEN GOTOXY(XP,YP);
  IF BRILHO THEN HIGHVIDEO ELSE LOWVIDEO;
  WRITE(FRASE);
END;

{-------------------------------------------------------------------------}

FUNCTION STRING_ATUAL:STR34;
{
 SAI NA VARIAVEL STRING_ATUAL A STRING DO CAMPO VALIDO.
}
BEGIN
  CASE CAMPO OF
    1: STRING_ATUAL:=TIPO_DE_MEMORIA;
    2: STRING_ATUAL:=NOMEFONTE;
    3: STRING_ATUAL:=NOMEDESTINO;
    4: STRING_ATUAL:=END_INIC_EP;
    5: STRING_ATUAL:=END_FIM_EP;
    6: STRING_ATUAL:=END_INIC_BUF;
    7: STRING_ATUAL:=END_FIM_BUF;
    8: STRING_ATUAL:=FUNCAO;
  END;
END;

{-------------------------------------------------------------------------}

PROCEDURE RODA_CAMPO;
{
 MUDA O CAMPO ATIVO NA TELA PRINCIPAL.
}
BEGIN
  GOTOXY(XICAMPO[],YICAMPO[]);TEXTCOLOR(13);WRITE(STRING_ATUAL);
  IF CAMPO=8 THEN CAMPO:=1 ELSE CAMPO:=CAMPO+1;
  GOTOXY(XICAMPO[],YICAMPO[]);TEXTCOLOR(15);WRITE(STRING_ATUAL);
END;

{-------------------------------------------------------------------------}

FUNCTION NUMHEX(STR_HEX:STR2):BYTE;
{
 RECEBE UMA STRING DE DOIS CARACTERES HEX VALIDOS E DEVOLVE SEU
 VALOR NUMERICO EM UM BYTE.
}
VAR HH:BYTE;
    ERRO:INTEGER;
```

```
BEGIN
  VAL(('$'+STR_HEX),HH,ERRO);
  IF ERRO<>0 THEN HH:=0;
  NUMHEX:=HH
END;

{-------------------------------------------------------------------}

FUNCTION NUMHEX2(STR_HEX2:STR4):WORD;
{
 RECEBE UMA STRING DE 4 CARACTERES E DEVOLVE INTEIRO CORRESPONDENTE.
}
VAR I:WORD;
    ERRO:INTEGER;

BEGIN
  VAL(('$'+STR_HEX2),I,ERRO);
  IF ERRO<>0 THEN I:=0;
  NUMHEX2:=I
END;

{-------------------------------------------------------------------}

FUNCTION CARACTHEX(BAI:BYTE):STR2;
{
 RECEBE UM BYTE E DEVOLVE SEU VALOR NUMA STRING DE DOIS CARACTERES HEX.
}
VAR CHD,CHU:CHAR;
    D,U:BYTE;

BEGIN
  D:=BAI DIV 16;
  U:=BAI MOD 16;
  IF D<=9 THEN CHD:=CHR(48+D) ELSE CHD:=CHR(55+D);
  IF U<=9 THEN CHU:=CHR(48+U) ELSE CHU:=CHR(55+U);
  CARACTHEX:=CONCAT(CHD,CHU);
END;

{-------------------------------------------------------------------}

FUNCTION CARACTHEX2(PALAVRA:WORD):STR4;
{
 RECEBE UMA PALAVRA E CONVERTE PARA UMA STRING DE 4 CARACTERES HEX.
}
BEGIN
  CARACTHEX2:=CONCAT(CARACTHEX(HI(PALAVRA)),CARACTHEX(LO(PALAVRA)));
END;
```

```
{-------------------------------------------------------------------}

PROCEDURE ALTERA_CAMPO;
{
 MODIFICA O CONTEUDO DO CAMPO VALIDO SE O CARACTERE NA VARIAVEL
 "CH" FOR UM CARACTERE VALIDO PARA AQUELE CAMPO.
}
BEGIN
  IF ((CAMPO IN[2,3]) AND VALENOME)
  THEN BEGIN
        CASE CAMPO OF
          2: IF CH=' ' THEN NOMEFONTE:=''
                       ELSE IF CH=CHR(8) THEN NOMEFONTE:=COPY(NOMEFONTE,1,LENGTH(NOMEFONTE)-1)
                                         ELSE NOMEFONTE:=NOMEFONTE+CH;
          3: IF CH=' ' THEN NOMEDESTINO:=''
                       ELSE IF CH=CHR(8) THEN NOMEDESTINO:=COPY(NOMEDESTINO,1,LENGTH(NOMEDESTINO)-1)
                                         ELSE NOMEDESTINO:=NOMEDESTINO+CH;
        END;
       END
  ELSE IF ((CAMPO IN [4..7]) AND VALEHEX)
       THEN BEGIN
             CASE CAMPO OF
               4: END_INIC_EP:=COPY(STRING_ATUAL,2,LENGTH(STRING_ATUAL)-1)+CH;
               5: END_FIM_EP:=COPY(STRING_ATUAL,2,LENGTH(STRING_ATUAL)-1)+CH;
               6: END_INIC_BUF:=COPY(STRING_ATUAL,2,LENGTH(STRING_ATUAL)-1)+CH;
               7: END_FIM_BUF:=COPY(STRING_ATUAL,2,LENGTH(STRING_ATUAL)-1)+CH;
             END;
            END
       ELSE BEGIN
             IF (CAMPO=1)
             THEN BEGIN
                   CASE CH OF
                     '1': BEGIN
                            TIPO_DE_MEMORIA:='128';
                            END_FIM_EP:='3FFF';IMPRIME(X[5],Y[5],'3FFF',H);
                            END_FIM_BUF:='3FFF';IMPRIME(X[7],Y[7],'3FFF',H);
                          END;
                     '2': BEGIN
                            TIPO_DE_MEMORIA:='256';
                            END_FIM_EP:='7FFF';IMPRIME(X[5],Y[5],'7FFF',H);
                            END_FIM_BUF:='7FFF';IMPRIME(X[7],Y[7],'7FFF',H);
                          END;
                     '5': BEGIN
                            TIPO_DE_MEMORIA:='512';
                            END_FIM_EP:='F000';IMPRIME(X[5],Y[5],'F000',H);
                            END_FIM_BUF:='F000';IMPRIME(X[7],Y[7],'F000',H);
                          END;
                     '6': BEGIN
                            TIPO_DE_MEMORIA:='64 ';
                            END_FIM_EP:='1FFF';IMPRIME(X[5],Y[5],'1FFF',H);
                            END_FIM_BUF:='1FFF';IMPRIME(X[7],Y[7],'1FFF',H);
                          END;
                     ELSE EXIT;
                   END;
```

```
                END_INIC_EP:='0000';IMPRIME(X[4],Y[4],'0000',H);
                END_INIC_BUF:='0000';IMPRIME(X[6],Y[6],'0000',H);
              INICIALIZA_PELO_TIPO_DE_MEMORIA;
              END
            ELSE BEGIN
                IF ((CAMPO=8) AND VALEHEX)
                THEN BEGIN
                     CASE CH OF
                       '0': FUNCAO:=FUN0;
                       '1': FUNCAO:=FUN1;
                       '2': FUNCAO:=FUN2;
                       '3': FUNCAO:=FUN3;
                       '4': FUNCAO:=FUN4;
                       '5': FUNCAO:=FUN5;
                       '6': FUNCAO:=FUN6;
                       '7': FUNCAO:=FUN7;
                       '8': FUNCAO:=FUN8;
                       '9': FUNCAO:=FUN9;
                       'A': FUNCAO:=FUNA;
                       'B': FUNCAO:=FUNB;
                       'C': FUNCAO:=FUNC;
                       'D': FUNCAO:=FUND;
                       'E': FUNCAO:=FUNE;
                       'F': FUNCAO:=FUNF;
                     END;
                   END
                ELSE EXIT;
              END;
          END;
  GOTOXY(X[CAMPO],Y[CAMPO]);
  CASE CAMPO OF
    2,3: IF STRING_ATUAL=''
         THEN BEGIN
              WRITE('          ');
              GOTOXY(X[CAMPO],Y[CAMPO]);
             END
         ELSE IF CH=CHR(8)
              THEN WRITE(STRING_ATUAL+' '+CHR(8))
              ELSE WRITE(STRING_ATUAL);
    ELSE WRITE(STRING_ATUAL);
  END;
END;

{-------------------------------------------------------------------------}

PROCEDURE LIMPA_QUADRO;
{
 APAGA O QUADRO DA TELA PRINCIPAL
}
BEGIN
  WINDOW(2,14,79,23);
  CLRSCR;
  WINDOW(1,1,80,25);
END;
```

```
{-------------------------------------------------------------------}

PROCEDURE MENU;
{
 IMPRIME O MENU NO QUADRO DA TELA PRINCIPAL.
}
BEGIN
  LIMPA_QUADRO;
  IMPRIME(6,14,FUN0,H);
  IMPRIME(6,15,FUN1,H);
  IMPRIME(6,16,FUN2,H);
  IMPRIME(6,17,FUN3,H);
  IMPRIME(6,18,FUN4,H);
  IMPRIME(6,19,FUN5,H);
  IMPRIME(6,20,FUN6,H);
  IMPRIME(6,21,FUN7,H);
  IMPRIME(42,14,FUN8,H);
  IMPRIME(42,15,FUN9,H);
  IMPRIME(42,16,FUNA,H);
  IMPRIME(42,17,FUNB,H);
  IMPRIME(42,18,FUNC,H);
  IMPRIME(42,19,FUND,H);
  IMPRIME(42,20,FUNE,H);
  IMPRIME(42,21,FUNF,H);
  IMPRIME(3,23,FUN_SETA,H);
  IMPRIME(28,23,FUN_ESC,H);
  IMPRIME(56,23,FUN_ENTER,H);
  GOTOXY(X[10],Y[10]);WRITE('( ');TEXTCOLOR(13);WRITE(NOME_DO_DIRETORIO);
  TEXTCOLOR(15);WRITE(' )');
  TEXTCOLOR(13);
  IF PRINTER_ON
     THEN IMPRIME(X[9],Y[9],'( LIG ) ',H)
     ELSE IMPRIME(X[9],Y[9],'( DESL )',H);
  TEXTCOLOR(15);
  GOTOXY(X[CAMPO],Y[CAMPO]);
END;

{-------------------------------------------------------------------}

PROCEDURE INICIALIZA_PROGRAMA_GRAVADOR;

VAR I:WORD;

BEGIN
  PRIMEIRA_INICIALIZACAO;
  TEXTCOLOR(15);
  TIPO_DE_MEMORIA:='64 ';
  NOMEFONTE:='????????.???';
  NOMEDESTINO:='????????.???';
  END_INIC_EP:='0000';
  END_FIM_EP:='1FFF';
  END_INIC_BUF:='0000';
  END_FIM_BUF:='1FFF';
```

```
  FUNCAO:='';
  CAMPO:=0;
  FOR I:=0 TO TAMANHO_DO_BUFFER DO BUFFER[I]:=0;
  NOME_DO_DIRETORIO:='';
  DRIVE_NUMERO:=0;
  GETDIR(DRIVE_NUMERO,NOME_DO_DIRETORIO);
  PRINTER_ON:=FALSE;
  INICIALIZA_PELO_TIPO_DE_MEMORIA;
END;

{-------------------------------------------------------------------------}

PROCEDURE IMPRIME_TELA_PRINCIPAL;

BEGIN
  CLRSCR;

  { LINHA 1 }
  P(201,1);P(205,27);P(209,1);P(205,5);P(203,1);P(205,44);P(187,1);

  { LINHA 2 }
  P(186,1);P(32,5);WRITE('Gravador de Eprom');P(32,5);P(179,1);P(32,1);
  WRITE('1.0');P(32,1);P(186,1);P(32,4);WRITE('Tipo de Memória:');P(32,1);
  WRITE('27');TEXTCOLOR(13);WRITE(TIPO_DE_MEMORIA);TEXTCOLOR(15);P(32,1);
  WRITE('(64/128/256/512)');P(32,1);P(186,1);

  { LINHA 3 }
  P(204,1);P(205,27);P(207,1);P(205,5);P(206,1);P(205,44);P(185,1);

  { LINHA 4 }
  P(186,1);P(32,6);WRITE('Arquivos de Trabalho');P(32,7);P(186,1);P(32,10);
  WRITE('Endereços de Trabalho');P(32,13);P(186,1);

  { LINHA 5 }
  P(199,1);P(196,16);P(194,1);P(196,16);P(215,1);P(196,21);P(194,1);
  P(196,22);P(182,1);

  { LINHA 6 }
  P(186,1);P(32,5);WRITE('Fonte');P(32,6);P(179,1);P(32,4);WRITE('Destino');
  P(32,5);P(186,1);P(32,8);WRITE('Eprom');P(32,8);P(179,1);P(32,7);
  WRITE('Buffer');P(32,9);P(186,1);

  { LINHA 7 }
  P(199,1);P(196,16);P(197,1);P(196,16);P(215,1);P(196,21);P(197,1);
  P(196,22);P(182,1);

  { LINHA 8 }
  P(186,1);P(32,16);P(179,1);P(32,16);P(186,1);P(32,4);WRITE('Inicial:');
  P(32,1);TEXTCOLOR(13);WRITE(END_INIC_EP);TEXTCOLOR(15);P(32,4);P(179,1);
  P(32,4);WRITE('Inicial:');P(32,1);TEXTCOLOR(13);WRITE(END_INIC_BUF);
  TEXTCOLOR(15);P(32,5);P(186,1);

  NOMEFONTE:=NOMEFONTE+'            ';NOMEDESTINO:=NOMEDESTINO+'            ';
```

```
 { LINHA 9 }
 P(186,1);P(32,2);TEXTCOLOR(13);WRITE(NOMEFONTE:12);TEXTCOLOR(15);P(32,2);
 P(179,1);P(32,2);TEXTCOLOR(13);WRITE(NOMEDESTINO:12);TEXTCOLOR(15);P(32,2);
 P(186,1);P(32,21);P(179,1);P(32,22);P(186,1);

 { LINHA 10 }
 P(186,1);P(32,16);P(179,1);P(32,16);P(186,1);P(32,6);WRITE('Final:');
 P(32,1);TEXTCOLOR(13);WRITE(END_FIM_EP);TEXTCOLOR(15);P(32,4);P(179,1);
 P(32,6);WRITE('Final:');P(32,1);TEXTCOLOR(13);WRITE(END_FIM_BUF);
 TEXTCOLOR(15);P(32,5);P(186,1);

 { LINHA 11 }
 P(204,1);P(205,16);P(207,1);P(205,16);P(206,1);P(205,21);P(207,1);
 P(205,22);P(185,1);

 { LINHA 12 }
 P(186,1);P(32,33);;P(186,1);P(32,1);WRITE('Função:');
 P(32,1);TEXTCOLOR(13);WRITE(FUNCAO:34);TEXTCOLOR(15);P(32,1);P(186,1);

 { LINHA 13 }
 P(204,1);P(205,33);P(202,1);P(205,44);P(185,1);

 { LINHA 14 - 23 }
 P(186,1);P(32,78);P(186,1);
 P(186,1);P(32,78);P(186,1);
 P(186,1);P(32,78);P(186,1);
 P(186,1);P(32,78);P(186,1);
 P(186,1);P(32,78);P(186,1);
 P(186,1);P(32,78);P(186,1);
 P(186,1);P(32,78);P(186,1);
 P(186,1);P(32,78);P(186,1);
 P(186,1);P(32,78);P(186,1);
 P(186,1);P(32,78);P(186,1);

 { LINHA 24 }
 P(200,1);P(205,78);P(188,1);

 MENU;
 GOTOXY(XI8],YI8]); { POSIÇÃO DA ENTRADA DE FUNÇÃO }
END;

{-------------------------------------------------------------------------}

PROCEDURE COPIA_EPROM_NO_BUFFER;
{
 COPIA UM BLOCO DE BYTES = "END_FINAL_EPROM - END_INICIAL_EPROM" PARA O BUFFER.
 O ENDEREÇO DO INICIO DO BLOCO NA EPROM É ESPECIFICADO EM "END_INICIAL_EPROM".
 O ENDEREÇO DO INICIO DO BLOCO NO BUFFER É ESPECIFICADO EM "END_INICIAL_BUFFER".
}
VAR INDICE,END_EP,END_BU:WORD;
```

```
BEGIN
  GOTOXY(3,12);
  WRITE('BYTES 0000 ... ',CARACTHEX2(END_FINAL_EPROM-END_INICIAL_EPROM),
  ' ( ',(END_FINAL_EPROM-END_INICIAL_EPROM):5,' )   ');

  IF ((END_FINAL_BUFFER-END_INICIAL_BUFFER) < (END_FINAL_EPROM-END_INICIAL_EPROM))
  THEN BEGIN
        LIMPA_QUADRO;
        IMPRIME(10,18,'ERRO:   BLOCO DA EPROM  >  BLOCO DE BUFFER.',H);
        IMPRIME(10,22,'TECLE < ESPACO > PARA VOLTAR AO MENU.',H);
        GOTOXY(X[8],Y[8]);
        WRITE(CHR(7));
        REPEAT UNTIL READKEY=' ';
        IMPRIME(3,12,'                                ',H);
        GOTOXY(X[CAMPO],Y[CAMPO]);
        EXIT;
       END
  ELSE BEGIN
        INDICE:=0;CH:=' ';
        REPEAT
          BEGIN
            END_EP:=END_INICIAL_EPROM+INDICE;
            END_BU:=END_INICIAL_BUFFER+INDICE;
            BUFFER[END_BU]:=LE_BYTE(END_EP);
            INDICE:=INDICE+1;
            IF KEYPRESSED THEN CH:=READKEY;
          END;
        UNTIL (END_EP=END_FINAL_EPROM) OR (CH=CHR(27));
       END;
  IF CH=CHR(27) THEN BEGIN
                      IMPRIME(3,12,'                                ',H);
                      CH:=' ';
                      EXIT;
                     END;
  LIMPA_QUADRO;
  IMPRIME(10,18,'BLOCO DA EPROM COPIADO NO BUFFER.',H);
  IMPRIME(10,22,'TECLE < ESPACO > PARA VOLTAR AO MENU.',H);
  GOTOXY(X[8],Y[8]);
  REPEAT UNTIL READKEY=' ';
  IMPRIME(3,12,'                                ',H);
  GOTOXY(X[CAMPO],Y[CAMPO]);
END;

{-------------------------------------------------------------------------}

PROCEDURE GRAVA_BUFFER_NA_EPROM;
{
 GRAVA NA EPROM, A PARTIR DO ENDEREÇO INICIAL ESPECIFICADO POR
 "END_INICIAL_EPROM", UM BLOCO DE DADOS COM TAMANHO ESPECIFICADO
 POR "END_FINAL_BUFFER - END_INICIAL_BUFFER", ORIGINADOS NO BUFFER,
 APARTIR DO ENDEREÇO ESPECIFICADO POR "END_INICIAL_BUFFER".
}

VAR GRAVACAO_OK:BOOLEAN;
```

```
BEGIN
  GOTOXY(3,12);
  WRITE('BYTES 0000 ... ',CARACTHEX2(END_FINAL_BUFFER-END_INICIAL_BUFFER),
  ' ( ',(END_FINAL_BUFFER-END_INICIAL_BUFFER):5,' )   ');

  IF ((END_FINAL_EPROM-END_INICIAL_EPROM) < (END_FINAL_BUFFER-END_INICIAL_BUFFER))
  THEN BEGIN
        LIMPA_QUADRO;
        IMPRIME(10,18,'ERRO:  BLOCO NO BUFFER  >  BLOCO NA EPROM.',H);
        IMPRIME(10,22,'TECLE < ESPACO > PARA VOLTAR AO MENU.',H);
        GOTOXY(X[8],Y[8]);
        WRITE(CHR(7));
        REPEAT UNTIL READKEY=' ';
        IMPRIME(3,12,'                                ',H);
        GOTOXY(X[CAMPO],Y[CAMPO]);
       END
  ELSE BEGIN
        GRAVACAO_OK:=TRUE;
        PROGRAMACAO_INTELIGENTE(GRAVACAO_OK);
        IF GRAVACAO_OK THEN BEGIN
                             LIMPA_QUADRO;
                             IMPRIME(10,18,'GRAVAÇÃO CONCLUIDA E OK !',H);
                             IMPRIME(10,22,'TECLE < ESPACO > PARA VOLTAR AO MENU.',H);
                             GOTOXY(X[8],Y[8]);
                             REPEAT UNTIL READKEY=' ';
                             IMPRIME(3,12,'                                    ',H);
                             GOTOXY(X[CAMPO],Y[CAMPO]);
                            END
                       ELSE BEGIN
                             LIMPA_QUADRO;
                             IMPRIME(10,18,'ERRO:  GRAVAÇÃO NÃO OK !',H);
                             IMPRIME(10,22,'TECLE < ESPACO > PARA VOLTAR AO MENU.',H);
                             GOTOXY(X[8],Y[8]);
                             WRITE(CHR(7));
                             REPEAT UNTIL READKEY=' ';
                             IMPRIME(3,12,'                                    ',H);
                             GOTOXY(X[CAMPO],Y[CAMPO]);
                            END;
       END;
END;

{-------------------------------------------------------------------------}

PROCEDURE COPIA_ARQUIVO_NO_BUFFER;
{
 SE O TAMANHO ESPECIFICADO PARA O BUFFER FOR SUFICIENTE PARA CONTER
 O ARQUIVO "NOMEFONTE", ENTÃO A CARGA OCORRERA A PARTIR DO ENDEREÇO
 END_INICIAL_BUFFER.
}
VAR INDICE:WORD;
    APONTA:LONGINT;
    DESL_ARQ:STR4;
```

```
BEGIN
  IF (NOMEFONTE='') OR (NOMEFONTE='????????.???') THEN
                     BEGIN
                       LIMPA_QUADRO;
                       IMPRIME(10,18,'ERRO: NOME DO ARQUIVO INVALIDO !',H);
                       IMPRIME(10,22,'TECLE < ESPACO > PARA VOLTAR AO MENU.',H);
                       WRITE(CHR(7));
                       GOTOXY(X[CAMPO],Y[CAMPO]);
                       REPEAT UNTIL READKEY=' ';
                       EXIT;
                     END;
  LIMPA_QUADRO;
  IMPRIME(10,18,'ENTRE COM O DESLOCAMENTO ( HEX ) : ',H);
  TEXTCOLOR(13);
  CH:='0';
  DESL_ARQ:='0000';
  WRITE(DESL_ARQ);
  REPEAT
    CH:=READKEY;
    CH:=UPCASE(CH);
    IF VALEHEX THEN DESL_ARQ:=COPY(DESL_ARQ,2,3)+CH;
    GOTOXY(45,18);WRITE(DESL_ARQ);
  UNTIL CH=CHR(13);
  TEXTCOLOR(15);
  ASSIGN(FONTE,NOMEFONTE);
  {$I-} RESET(FONTE);{$I+}
  IF IORESULT <> 0 THEN BEGIN
                         LIMPA_QUADRO;
                         IMPRIME(10,18,'ERRO NA ABERTURA DO ARQUIVO !',H);
                         IMPRIME(10,20,'VERIFIQUE SE O ARQ. EXISTE NO DIRETORIO CORRENTE.',H);
                         IMPRIME(10,22,'TECLE < ESPACO > PARA VOLTAR AO MENU.',H);
                         WRITE(CHR(7));
                         GOTOXY(X[CAMPO],Y[CAMPO]);
                         REPEAT UNTIL READKEY=' ';
                         EXIT;
                       END;

  IF ( FILESIZE(FONTE)-1 < NUMHEX2(DESL_ARQ) )
  THEN BEGIN
         CLOSE(FONTE);
         LIMPA_QUADRO;
         IMPRIME(10,18,'ERRO: DESLOCAMENTO INCORRETO !',H);
         IMPRIME(10,22,'TECLE < ESPACO > PARA VOLTAR AO MENU.',H);
         GOTOXY(X[8],Y[8]);
         WRITE(CHR(7));
         REPEAT UNTIL READKEY=' ';
         IMPRIME(3,12,'                                ',H);
         GOTOXY(X[CAMPO],Y[CAMPO]);
         EXIT;
       END;
```

```
IF ((END_FINAL_BUFFER-END_INICIAL_BUFFER) < FILESIZE(FONTE)-1-NUMHEX2(DESL_ARQ))
THEN BEGIN
       CLOSE(FONTE);
       LIMPA_QUADRO;
       IMPRIME(10,18,'ERRO:   ARQUIVO  >  BLOCO DE BUFFER !',H);
       IMPRIME(10,22,'TECLE < ESPACO > PARA VOLTAR AO MENU.',H);
       GOTOXY(X[8],Y[8]);
       WRITE(CHR(7));
       REPEAT UNTIL READKEY=' ';
       IMPRIME(3,12,'                                ',H);
       GOTOXY(X[CAMPO],Y[CAMPO]);
       EXIT;
     END
 ELSE BEGIN
       APONTA:=NUMHEX2(DESL_ARQ);
       SEEK(FONTE,APONTA);
       FOR INDICE:=0 TO FILESIZE(FONTE)-1-NUMHEX2(DESL_ARQ) DO
       BEGIN
         READ(FONTE,BUFFER[END_INICIAL_BUFFER + INDICE]);
         IF (KEYPRESSED AND (READKEY=CHR(27))) THEN BEGIN
                                                     CLOSE(FONTE);
                                                     DELAY(500);
                                                     EXIT;
                                                    END;
       END;
     END;
 CLOSE(FONTE);
 LIMPA_QUADRO;
 IMPRIME(10,18,'CARGA DO ARQUIVO <  '+NOMEFONTE+' >  FOI CONCLUIDA E OK !',H);
 IMPRIME(10,22,'TECLE < ESPACO > PARA VOLTAR AO MENU.',H);
 GOTOXY(3,12);
 WRITE('Num. bytes: HEX:',CARACTHEX2(INDICE),' DEC:',INDICE:5);
 GOTOXY(X[8],Y[9]);
 REPEAT UNTIL READKEY=' ';
 GOTOXY(X[CAMPO],Y[CAMPO]);
END;

{-------------------------------------------------------------------}

PROCEDURE GRAVA_BUFFER_NO_ARQUIVO;
{
 GRAVA UM BLOCO DE DADOS DE TAMANHO ESPECIFICADO POR
 "END_FINAL_BUFFER-END_INICIAL_BUFFER" NO ARQUIVO "NOMEDESTINO",
  COM INICIO ESPECIFICADO POR "END_INICIAL_BUFFER".
}
VAR INDICE:WORD;
```

```
BEGIN
  IF (NOMEDESTINO='') OR (NOMEDESTINO='????????.???') THEN
                      BEGIN
                        LIMPA_QUADRO;
                        IMPRIME(10,18,'ERRO: NOME DO ARQUIVO INVALIDO !',H);
                        IMPRIME(10,22,'TECLE < ESPACO > PARA VOLTAR AO MENU.',H);
                        WRITE(CHR(7));
                        GOTOXY(X[CAMPO],Y[CAMPO]);
                        REPEAT UNTIL READKEY=' ';
                        EXIT;
                      END;

  ASSIGN(DESTINO,NOMEDESTINO);
  {$I-} REWRITE(DESTINO);{$I+}
  IF IORESULT <> 0 THEN BEGIN
                         LIMPA_QUADRO;
                         IMPRIME(10,18,'ERRO NA ABERTURA DO ARQUIVO ! ',H);
                         IMPRIME(10,20,'VERIFIQUE SE O ARQ. ESTA PROTEGIDO OU SE O DISCO ESTA CHEIO.',H);
                         IMPRIME(10,22,'TECLE < ESPACO > PARA VOLTAR AO MENU.',H);
                         WRITE(CHR(7));
                         GOTOXY(X[CAMPO],Y[CAMPO]);
                         REPEAT UNTIL READKEY=' ';
                         EXIT;
                        END;

  FOR INDICE:=0 TO (END_FINAL_BUFFER-END_INICIAL_BUFFER) DO
  BEGIN
    WRITE(DESTINO,BUFFER[END_INICIAL_BUFFER + INDICE]);
    IF (KEYPRESSED AND (READKEY=CHR(27))) THEN BEGIN
                                                 CLOSE(DESTINO);
                                                 DELAY(500);
                                                 EXIT;
                                               END;
  END;
  CLOSE(DESTINO);
  LIMPA_QUADRO;
  IMPRIME(5,18,'GRAVAÇÃO DO BLOCO DE BUFFER PARA O ARQUIVO <  '+
  NOMEDESTINO+'  >  OK !',H);
  IMPRIME(5,22,'TECLA < ESPACO > PARA VOLTAR AO MENU.',H);
  GOTOXY(3,12);
  WRITE('Num. bytes: HEX:',CARACTHEX2(INDICE),' DEC:',INDICE:5);
  GOTOXY(X[8],Y[8]);
  REPEAT UNTIL READKEY=' ';
  GOTOXY(X[CAMPO],Y[CAMPO]);
END;
```

```
{-------------------------------------------------------------------}

PROCEDURE LIGA_DESLIGA_IMPRESSORA;
{
 LIGA OU DESLIGA A IMPRESSORA PELA MUDANÇA DA VARIAVEL PRINTER_ON.
 A IMPRESSÃO SO SERA ATIVA NAS FUNÇÕES ABAIXO:
     - COMPARA_EPROM_BUFFER;
     - LISTA_BUFFER;
     - CKECHSUM;
}
BEGIN
  IF PRINTER_ON THEN PRINTER_ON:=FALSE ELSE PRINTER_ON:=TRUE;
END;

{-------------------------------------------------------------------}

PROCEDURE COMPARA_EPROM_BUFFER;
{
 COMPARA EPROM COM BUFFER E LISTA AS DIFERENÇAS NO QUADRO DA TELA PRINCIPAL.
}
VAR INDICE,END_EP,END_BU:WORD;

BEGIN
  IF ((END_FINAL_EPROM-END_INICIAL_EPROM) <> (END_FINAL_BUFFER-END_INICIAL_BUFFER))
  THEN BEGIN
       LIMPA_QUADRO;
       IMPRIME(10,10,'ERRO:  TAMANHO DIFERENTE DOS BLOCOS !',H);
       IMPRIME(10,22,'TECLE ( ESPACO ) PARA VOLTAR AO MENU.',H);
       GOTOXY(X[8],Y[8]);
       WRITE(CHR(7));
       REPEAT UNTIL READKEY=' ';
       IMPRIME(3,12,'                                ',H);
       GOTOXY(X[CAMPO],Y[CAMPO]);
       EXIT;
      END;

  IMPRIME(3,12,'( ESC ) P/ PARAR              ',H);
  INDICE:=0;
  WINDOW(3,14,78,23);
  CLRSCR;
```

```
  REPEAT
    BEGIN
      END_EP:=END_INICIAL_EPROM+INDICE;
      END_BU:=END_INICIAL_BUFFER+INDICE;
      IF NOT(BYTE_OK(END_EP,BUFFER[END_BU]))
      THEN BEGIN
            WRITE('      ',CARACTHEX2(END_EP),'( ',CARACTHEX(LE_BYTE(END_EP)),
            ' ) <> ',CARACTHEX2(END_BU),'( ',CARACTHEX(BUFFER[END_BU]),
            ' )       ');
            IF PRINTER_ON
            THEN BEGIN
                  WRITELN(LST,'DIFERENCA: ',CARACTHEX2(END_EP),'( ',CARACTHEX(LE_BYTE(END_EP)),
                  ' ) <> ',CARACTHEX2(END_BU),'( ',CARACTHEX(BUFFER[END_BU]),
                  ' )');
                 END;
            REPEAT UNTIL KEYPRESSED;
           END;
      INDICE:=INDICE+1;
    END;
  UNTIL (END_EP=END_FINAL_EPROM) OR (KEYPRESSED AND (READKEY=CHR(27)));

  WINDOW(1,1,80,25);
  IMPRIME(3,12,'< ESPACO > P/ VOLTAR AO MENU  ',H);
  GOTOXY(X[8],Y[8]);
  REPEAT UNTIL READKEY=' ';
  IMPRIME(3,12,'                              ',H);
  GOTOXY(X[CAMPO],Y[CAMPO]);
END;

{-------------------------------------------------------------------}

PROCEDURE VERIFICA_EPROM_APAGADA;
{
 VERIFICA SE TODOS OS CONTEUDOS DOS ENDEREÇOS DENTRO DO BLOCO ESPECIFICADO
 POSSUEM VALOR FF.
}
VAR INDICE,END_EP:WORD;
    DIFERENCA:BOOLEAN;

BEGIN
  LIMPA_QUADRO;
  INDICE:=0;
  DIFERENCA:=FALSE;
  IMPRIME(3,12,'< ESC > PARA PARAR           ',H);
  REPEAT
    BEGIN
      END_EP:=END_INICIAL_EPROM+INDICE;
      IMPRIME(10,18,CARACTHEX2(END_EP),H);
      IF LE_BYTE(END_EP) <> $FF THEN DIFERENCA:=TRUE;
      INDICE:=INDICE+1;
    END;
  UNTIL (END_EP = END_FINAL_EPROM) OR (DIFERENCA = TRUE)
                                OR (KEYPRESSED AND (READKEY=CHR(27)));
```

```
  IMPRIME(3,12,'                              ',N);
  LIMPA_QUADRO;
  IF DIFERENCA = TRUE THEN BEGIN
                             IMPRIME(10,18,'ESTA EPROM NAO ESTA TOTALMENTE APAGADA.',N);
                             IMPRIME(10,22,'TECLE < ESPACO > PARA VOLTAR AO MENU.',N);
                             WRITE(CHR(7));
                           END
                      ELSE BEGIN
                             IMPRIME(10,18,'ESTA EPROM ESTA APAGADA NO BLOCO ESPECIFICADO.',N);
                             IMPRIME(10,22,'TECLE < ESPACO > PARA VOLTAR AO MENU.',N);
                           END;
  REPEAT UNTIL READKEY=' ';
  GOTOXY(X[CAMPO],Y[CAMPO]);
END;

{-------------------------------------------------------------------}

PROCEDURE CHECKSUM;
{
 CALCULA O CHECKSUM DO BLOCO DA EPROM E DO BUFFER.
}
VAR INDICE,END_EP,END_BU:WORD;
    CHECK_EP,CHECK_BU:BYTE;

BEGIN
  LIMPA_QUADRO;
  IMPRIME(3,12,'          AGUARDE             ',N);

  { CALCULA CHECKSUM DA EPROM }

  INDICE:=0;
  CHECK_EP:=0;
  REPEAT
    BEGIN
      END_EP:=END_INICIAL_EPROM+INDICE;
      IMPRIME(10,18,CARACTHEX2(END_EP),N);
      CHECK_EP:=CHECK_EP+LE_BYTE(END_EP);
      INDICE:=INDICE+1;
    END;
  UNTIL (END_EP = END_FINAL_EPROM) OR (KEYPRESSED AND (READKEY=CHR(27)));
  CHECK_EP:=0 - CHECK_EP;
  IMPRIME(10,18,'CHECKSUM DA EPROM = '+CARACTHEX(CHECK_EP),N);
  IF PRINTER_ON THEN WRITELN(LST,'CHECKSUM DA EPROM = ',CARACTHEX(CHECK_EP));
```

```
  { CALCULA CHECKSUM DO BUFFER }

  INDICE:=0;
  CHECK_BU:=0;
  REPEAT
    BEGIN
      END_BU:=END_INICIAL_BUFFER+INDICE;
      IMPRIME(10,20,CARACTHEX2(END_BU),H);
      CHECK_BU:=CHECK_BU+BUFFER[END_BU];
      INDICE:=INDICE+1;
    END;
  UNTIL (END_BU = END_FINAL_BUFFER) OR (KEYPRESSED AND (READKEY=CHR(27)));
  CHECK_BU:=0 - CHECK_BU;
  IMPRIME(10,20,'CHECKSUM DO BUFFER = '+CARACTHEX(CHECK_BU),H);
  IF PRINTER_ON THEN WRITELN(LST,'CHECKSUM DO BUFFER = ',CARACTHEX(CHECK_BU));
  IMPRIME(3,12,'< ESPACO > P/ VOLTAR AO MENU   ',H);
  GOTOXY(X[CAMPO],Y[CAMPO]);
  REPEAT UNTIL READKEY=' ';
  IMPRIME(3,12,'                               ',H);
END;

{-------------------------------------------------------------------}

PROCEDURE LISTA_CONTEUDO_DO_BUFFER;
{
 LISTA CONTEUDO DO BUFFER ENTRE OS LIMITES ESPECIFICADOS POR
 "END_INICIAL_BUFFER" E "END_FINAL_BUFFER".
 A TECLA <ESC> PROVOCA O FIM DO COMANDO.
 A LISTAGEM CONTINUARA ENQUANTO HOUVER UMA TECLA ACIONADA.
}
VAR I,J,END_BU:LONGINT;
    LINHA:BYTE;
    K:INTEGER;

{.....................................................}

PROCEDURE LISTA_LINHA;
{
 LISTA UMA LINHA ASSIM COMPOSTA:
 <END> < ... ATÉ 16 BYTES HEX ... > < ... ATÉ 16 CARACTERES ASCII ... >
 O PARAMETRO "K" CONTROLA O PERFEITO POSICIONAMENTO DOS BYTES E CARACTERES
 DE ACORDO COM O CABEÇALHO IMPRESSO NA PRIMEIRA LINHA DO QUADRO.
}
VAR N:LONGINT;

BEGIN
  IMPRIME(1,LINHA,CARACTHEX2(I)+'  ',L);
  IF K > 0 THEN P(32,K*3);
  FOR N:=I TO J DO WRITE(CARACTHEX(BUFFER[N]),' '); WRITE(' ');
  IF K < 0 THEN P(32,-K*3);
  GOTOXY(56,LINHA);
  IF K > 0 THEN P(32,K);
  FOR N:=I TO J DO IF BUFFER[N] IN [32..255] THEN WRITE(CHR(BUFFER[N]))
                                             ELSE WRITE('.');
```

```
  IF PRINTER_ON THEN
  BEGIN
    WRITE(LST,CARACTHEX2(I),'  ');
    IF K > 0 THEN FOR N:=1 TO (K*3) DO WRITE(LST,' ');
    FOR N:=I TO J DO WRITE(LST,CARACTHEX(BUFFER[N]),' '); WRITE(LST,' ');
    IF K < 0 THEN FOR N:=1 TO (-K*3) DO WRITE(LST,' ');
    IF K > 0 THEN FOR N:=1 TO K DO WRITE(LST,' ');
    FOR N:=I TO J DO IF BUFFER[N] IN [32..255] THEN WRITE(LST,CHR(BUFFER[N]))
                                              ELSE WRITE(LST,'.');
    WRITE(LST,CHR(13),CHR(10));
  END;
  IF LINHA < 11 THEN LINHA:=LINHA+1;
  I:=J+1;  K:=0;
END;

{.....................................................}

BEGIN
  IMPRIME(3,12,'(ESC) ABANDONA                ',H);  LIMPA_QUADRO;
  IMPRIME(3,14,'END.  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  0123456789ABCDEF',H);
  IF PRINTER_ON THEN
  WRITELN(LST,'END.  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  0123456789ABCDEF');
  WINDOW(3,15,79,23);  LINHA:=1;  K:=0;  I:=END_INICIAL_BUFFER;
  IF (15 - I MOD 16) <> 0 THEN BEGIN
                               J:=I + (15 - I MOD 16);
                               K:=(I MOD 16) ;
                               LISTA_LINHA;
                              END;
  WHILE (I < END_FINAL_BUFFER+1) DO
     BEGIN
       IF ((I+16) <= END_FINAL_BUFFER) THEN J:=I+15
                                       ELSE BEGIN
                                              J:=END_FINAL_BUFFER;
                                              K:=(J-I)-15;
                                            END;
       LISTA_LINHA;  REPEAT UNTIL KEYPRESSED;  WRITELN;
     IF ( KEYPRESSED AND ( READKEY=CHR(27)) ) THEN I:=END_FINAL_BUFFER+1;
     END;
  WINDOW(1,1,80,25);
  IMPRIME(3,12,'< ESPACO > P/ VOLTAR AO MENU  ',H);
  GOTOXY(X[8],Y[8]);
  REPEAT UNTIL READKEY=' ';
  IMPRIME(3,12,'                              ',H);
  GOTOXY(X[CAMPO],Y[CAMPO]);
END;
```

```
{-------------------------------------------------------------------------}

PROCEDURE ENCHE_BUFFER_COM_BYTE;
{
 ENCHE BLOCO DE BUFFER COM BYTE ESPECIFICADO.
}
VAR I:WORD;
    NOVOHEX:STRING[2];
    CAR:CHAR;
    BYTE_PARA_ENCHER:BYTE;

BEGIN
  LIMPA_QUADRO;
  IMPRIME(10,18,'ENTRE COM O BYTE ( HEX ): ',H);
  CH:='0';
  NOVOHEX:='00';
  REPEAT
    CH:=READKEY;
    CH:=UPCASE(CH);
    IF VALEHEX THEN NOVOHEX:=COPY(NOVOHEX,2,1)+CH;
    IF NUMHEX(NOVOHEX) IN [32..255] THEN CAR:=CHR(NUMHEX(NOVOHEX))
                                    ELSE CAR:='.';
    GOTOXY(36,18);
    WRITE(NOVOHEX,'    ',CAR);
  UNTIL CH=CHR(13);
  BYTE_PARA_ENCHER:=NUMHEX(NOVOHEX);
  FOR I:=END_INICIAL_BUFFER TO END_FINAL_BUFFER DO
  BEGIN
    BUFFER[I]:=BYTE_PARA_ENCHER;
    IF (KEYPRESSED AND(READKEY=CHR(27))) THEN BEGIN
                                              DELAY(500);
                                              EXIT;
                                             END;
  END;
  LIMPA_QUADRO;
  GOTOXY(10,18);
  WRITE('BLOCO DO BUFFER CHEIO COM < ',NOVOHEX,' > ');
  IMPRIME(10,22,'TECLE < ESPACO > PARA VOLTAR AO MENU.',H);
  GOTOXY(X[8],Y[8]);
  REPEAT UNTIL READKEY=' ';
  GOTOXY(X[CAMPO],Y[CAMPO]);
END;

{-------------------------------------------------------------------------}

PROCEDURE LISTA_DIRETORIO;
{
 APAGA A JANELA DO QUADRO PRINCIPAL.
 IMPRIME 3 COLUNAS FORMADAS PELO NOME DO ARQUIVO E SEUS TAMANHOS.
}
VAR DIRINFO:SEARCHREC;
```

```
BEGIN
  WINDOW(2,14,79,23);
  CLRSCR;
  WINDOW(3,14,77,23);
  FINDFIRST('*.*',ARCHIVE,DIRINFO);
  WHILE DOSERROR=0 DO
  BEGIN
    DIRINFO.NAME:=DIRINFO.NAME+'           ';
    WRITE(DIRINFO.NAME:12,DIRINFO.SIZE:8,'':5);
    FINDNEXT(DIRINFO);
  END;
  WINDOW(1,1,80,25);
END;

{-------------------------------------------------------------}

PROCEDURE DIRETORIO;
{
 POSSIBILITA TROCAR O DIRETORIO CORRENTE.
 IMPRIME O DIRETORIO CORRENTE.
}
VAR NOVO_NOME_DO_DIRETORIO:STRING[30];

BEGIN
  IMPRIME(3,12,'TROCAR DIR ATIVO ( S , N ) ?  ',H);
  REPEAT UNTIL KEYPRESSED;
  IF READKEY IN ['S','s'] THEN BEGIN
                                IMPRIME(3,12,'                                ',H);
                                IMPRIME(3,12,'NOVO DIR: ',H);
                                NOVO_NOME_DO_DIRETORIO:='';
                                READLN(NOVO_NOME_DO_DIRETORIO);
                                GOTOXY(X[8],Y[8]);
                                CHDIR(NOVO_NOME_DO_DIRETORIO);
                                NOME_DO_DIRETORIO:='';
                                DRIVE_NUMERO:=0;
                                GETDIR(DRIVE_NUMERO,NOME_DO_DIRETORIO);
                               END;
  LISTA_DIRETORIO;
  IMPRIME(3,12,'TECLE ( ESPACO ) P/ CONTINUAR ',H);
  REPEAT UNTIL READKEY=' ';
  IMPRIME(3,12,'                              ',H);
  GOTOXY(X[CAMPO],Y[CAMPO]);
END;

{-------------------------------------------------------------}

PROCEDURE EDITA_CONTEUDO_DO_BUFFER;
{
 EDITA CONTEUDO DO BUFFER BYTE A BYTE.
 ( ENTER ) DEIXA INTACTO O CONTEUDO DO BYTE E INCREMENTA O ENDEREÇO.
 ( ESC ) ABANDONA O COMANDO.
 ( CARACTERE HEX VALIDO ) MODIFICA O BYTE.
 O CONTEUDO DO BYTE E MOSTRADO EM HEX E ASCII.
}
```

```
VAR CAR:CHAR;
    I:LONGINT;
    NOVOHEX:STRING[2];

BEGIN
  IMPRIME(3,12,'<ESC> ABANDONA                ',H);  LIMPA_QUADRO;
  IMPRIME(3,14,'ENDERECO HEX ASCII   HEX ASCII',H);
  WINDOW(3,15,78,23); I:=END_INICIAL_BUFFER;
  WHILE (I <= END_FINAL_BUFFER) DO
    BEGIN
      IF BUFFER[I] IN [32..255] THEN CAR:=CHR(BUFFER[I])
                                ELSE CAR:='.';
      WRITE('  ',CARACTHEX2(I),'   ',CARACTHEX(BUFFER[I]),'    ',CAR,'     ');
      REPEAT
        NOVOHEX:=CARACTHEX(BUFFER[I]);
        CH:=READKEY;
        CH:=UPCASE(CH);
        IF VALEHEX THEN
          BEGIN
            NOVOHEX:=COPY(NOVOHEX,2,1)+CH;
            BUFFER[I]:=NUMHEX(NOVOHEX);
          END;
        IF BUFFER[I] IN [32..255] THEN CAR:=CHR(BUFFER[I])
                                  ELSE CAR:='.';
        GOTOXY(22,WHEREY);
        WRITE(NOVOHEX,'    ',CAR);
      UNTIL NOT(VALEHEX);
      WRITELN;
      IF CH=CHR(27) THEN I:=END_FINAL_BUFFER+1   ( ESC )
                    ELSE I:=I+1;
      CH:=' '; ( "CH" DEVE CONTER UM CARACTERE NEUTRO ANTES DE VOLTAR DA ROTINA )
    END;
  WINDOW(1,1,80,25);
  IMPRIME(3,12,'< ESPACO > P/ VOLTAR AO MENU   ',H);
  GOTOXY(X[8],Y[8]);
  REPEAT UNTIL READKEY=' ';
  IMPRIME(3,12,'                               ',H);
  GOTOXY(X[CAMPO],Y[CAMPO]);
END;

{-------------------------------------------------------------------------}

PROCEDURE PROCURA_BLOCO_NO_BUFFER;
{
 PROCURA DENTRO DO BLOCO DO BUFFER, POR UMA SEQüêNCIA ESPECIFICADA
 DE CARACTERES ASCII OU UMA SEQüêNCIA DE BYTES HEX.
}
VAR ESCOLHA:CHAR;
    I,J,END_ACHA:LONGINT;
    NDB:BYTE;
    N,ERRO1:INTEGER;
    ENCONTROU:BOOLEAN;
    LINHA_DE_ENTRADA:STRING[48];
    LINHA_DO_BUFFER:STRING[48];
```

```
BEGIN
  REPEAT
    LIMPA_QUADRO;
    WINDOW(3,14,78,23);
    WRITELN('ESCOLHA O TIPO DE PROCURA:');
    WRITELN;
    WRITELN('H - SEQUENCIA HEX');
    WRITELN('A - SEQUENCIA ASCII');
    ESCOLHA:=READKEY;
    ESCOLHA:=UPCASE(ESCOLHA);
  UNTIL (ESCOLHA='H') OR (ESCOLHA='A');
  CASE ESCOLHA OF
    'H': BEGIN
          CLRSCR;
          WRITELN('ENTRE COM A SEQUÊNCIA DE BYTES HEX:  ( NO MAXIMO 16 BYTES )');
          WRITELN;
          WRITELN('SE QUISER USE < ESPACO > COMO SEPARADOR ENTRE CADA BYTE HEX.');
          WRITELN;
          WRITELN('TERMINE COM < ENTER >');
          WRITELN;
          LINHA_DE_ENTRADA:='';
          READLN(LINHA_DE_ENTRADA);
          IF LINHA_DE_ENTRADA='' THEN BEGIN
                                      CLRSCR;
                                      IMPRIME(10,18,'LINHA DE ENTRADA VAZIA !',H);
                                      WRITE(CHR(7));
                                      WINDOW(1,1,80,25);
                                      GOTOXY(X[CAMPO1],Y[CAMPO1]);
                                      DELAY(2000);
                                      EXIT;
                                     END;
          J:=1;
          REPEAT
            IF COPY(LINHA_DE_ENTRADA,J,1)=' ' THEN DELETE(LINHA_DE_ENTRADA,J,1)
                                              ELSE IF LENGTH(LINHA_DE_ENTRADA)=1 THEN J:=1
                                                                                 ELSE J:=J+1;
          UNTIL J=LENGTH(LINHA_DE_ENTRADA);
          FOR J:=1 TO LENGTH(LINHA_DE_ENTRADA)
            DO LINHA_DE_ENTRADA[J]:=UPCASE(LINHA_DE_ENTRADA[J]);
          WRITELN(LINHA_DE_ENTRADA);
          FOR J:=1 TO LENGTH(LINHA_DE_ENTRADA) DO
          BEGIN
            ERRO1:=0;
            VAL('$'+COPY(LINHA_DE_ENTRADA,J,1),N,ERRO1);
            IF ERRO1 <> 0 THEN BEGIN
                                CLRSCR;
                                IMPRIME(2,2,'SEQUÊNCIA INVALIDA ! ',H);
                                WRITE(CHR(7));
                                WINDOW(1,1,80,25);
                                GOTOXY(X[CAMPO1],Y[CAMPO1]);
                                DELAY(2000);
                                EXIT;
                               END;
          END;
```

```
        IMPRIME(60,10,'PROCURANDO...',H);
        I:=END_INICIAL_BUFFER;
        END_ACHA:=END_INICIAL_BUFFER;
        ENCONTROU:=FALSE;
        NDB:=LENGTH(LINHA_DE_ENTRADA) DIV 2;
        WHILE (LINHA_DE_ENTRADA <> LINHA_DO_BUFFER) AND
              NOT(KEYPRESSED AND (READKEY=CHR(27))) AND
             ( I <= (END_FINAL_BUFFER - NDB + 1 )) DO
        BEGIN
          LINHA_DO_BUFFER:='';
          FOR J:=I TO I+NDB-1 DO LINHA_DO_BUFFER:=LINHA_DO_BUFFER+CARACTHEX(BUFFER[J]);
          IF (LINHA_DE_ENTRADA = LINHA_DO_BUFFER) THEN BEGIN
                                                         ENCONTROU:=TRUE;
                                                         END_ACHA:=I;
                                                       END;
          I:=I+1;
        END;
        IF ENCONTROU
        THEN BEGIN
               CLRSCR;
               WINDOW(1,1,80,25);
               GOTOXY(10,18);
               WRITE('A SEQÜÊNCIA PROCURADA FOI ENCONTRADA A PARTIR DO ENDEREÇO :',CARACTHEX2(END_ACHA));
               IMPRIME(10,22,'TECLE < ESPACO > PARA VOLTAR AO MENU.',H);
               GOTOXY(3,12);
               WRITE('END. DA SEQ.: ',CARACTHEX2(END_ACHA),'           ');
               GOTOXY(X[8],Y[8]);
               REPEAT UNTIL READKEY=' ';
               GOTOXY(X[CAMPO],Y[CAMPO]);
             END
        ELSE BEGIN
               CLRSCR;
               WINDOW(1,1,80,25);
               IMPRIME(10,18,'SEQÜÊNCIA  < N A O > ENCONTRADA ! ',H);
               IMPRIME(10,22,'TECLE < ESPACO > PARA VOLTAR AO MENU.',H);
               WRITE(CHR(7));
               GOTOXY(X[8],Y[8]);
               REPEAT UNTIL READKEY=' ';
               GOTOXY(X[CAMPO],Y[CAMPO]);
             END;
      END;
'A': BEGIN
       CLRSCR;
       WRITELN('ENTRE COM A SEQÜÊNCIA DE CARACTERES ASCII:  ( NO MAXIMO 16 CARACTERES )');
       WRITELN;
       WRITELN('TERMINE COM < ENTER >');
       WRITELN;
       LINHA_DE_ENTRADA:='                ';
       READLN(LINHA_DE_ENTRADA);
```

```
IF LINHA_DE_ENTRADA='' THEN BEGIN
                      CLRSCR;
                      IMPRIME(10,10,'LINHA DE ENTRADA VAZIA !',H);
                      WRITE(CHR(7));
                      WINDOW(1,1,80,25);
                      GOTOXY(X[CAMPO],Y[CAMPO]);
                      DELAY(2000);
                      EXIT;
                     END;
FOR J:=1 TO LENGTH(LINHA_DE_ENTRADA) DO
BEGIN
  IF NOT( ORD( LINHA_DE_ENTRADA[J] ) IN [32..255] )
  THEN BEGIN
         CLRSCR;
         IMPRIME(2,2,'SEQÜÊNCIA INVALIDA ! ',H);
         WRITE(CHR(7));
         WINDOW(1,1,80,25);
         GOTOXY(X[CAMPO],Y[CAMPO]);
         DELAY(2000);
         EXIT;
       END;
END;
IMPRIME(60,10,'PROCURANDO...',H);
I:=END_INICIAL_BUFFER;
END_ACHA:=END_INICIAL_BUFFER;
ENCONTROU:=FALSE;
NDB:=LENGTH(LINHA_DE_ENTRADA);
WHILE (LINHA_DE_ENTRADA <> LINHA_DO_BUFFER) AND
      NOT(KEYPRESSED AND (READKEY=CHR(27))) AND
     ( I <= (END_FINAL_BUFFER - NDB + 1 )) DO
BEGIN
  LINHA_DO_BUFFER:='';
  FOR J:=I TO I+NDB-1 DO LINHA_DO_BUFFER:=LINHA_DO_BUFFER+CHR(BUFFER[J]);
  IF (LINHA_DE_ENTRADA = LINHA_DO_BUFFER) THEN BEGIN
                                                ENCONTROU:=TRUE;
                                                END_ACHA:=I;
                                               END;
  I:=I+1;
END;
IF ENCONTROU
THEN BEGIN
       CLRSCR;
       WINDOW(1,1,80,25);
       GOTOXY(10,10);
       WRITE('A SEQÜÊNCIA PROCURADA FOI ENCONTRADA A PARTIR DO ENDEREÇO :',CARACTHEX2(END_ACHA));
       IMPRIME(10,22,'TECLE < ESPACO > PARA VOLTAR AO MENU.',H);
       GOTOXY(3,12);
       WRITE('END. DA SEQ.: ',CARACTHEX2(END_ACHA),'          ');
       GOTOXY(X[8],Y[8]);
       REPEAT UNTIL READKEY=' ';
       GOTOXY(X[CAMPO],Y[CAMPO]);
     END
```

```
          ELSE BEGIN
                CLRSCR;
                WINDOW(1,1,80,25);
                IMPRIME(10,18,'SEQUENCIA  ( N A O ) ENCONTRADA ! ',H);
                IMPRIME(10,22,'TECLE ( ESPACO ) PARA VOLTAR AO MENU.',H);
                WRITE(CHR(7));
                GOTOXY(X[8],Y[8]);
                REPEAT UNTIL READKEY=' ';
                GOTOXY(X[CAMPO],Y[CAMPO]);
               END;

        END;
  END; ( FIM DO CASE )
  WINDOW(1,1,80,25);
END;

{-------------------------------------------------------------------}

PROCEDURE MOVE_BLOCO_NO_BUFFER;
(
 MOVE UM BLOCO DA ORIGEM PARA O DESTINO COM UM NUMERO DE BYTES ESPECIFICADO.
)
VAR LINHA_DE_ENTRADA:STRING[30];
    ORIG,DEST,QUANT,I:WORD;
    ORIG_STR,DEST_STR,QUANT_STR:STRING[4];

BEGIN
  LIMPA_QUADRO;
  IMPRIME(3,14,'ENTRE COM: <END. INIC. DA ORIGEM>,<END. INIC. DO DESTINO>,<NUM. DE BYTES>',H);
  IMPRIME(3,16,'VALORES EM HEX.     TERMINE ENTRADA DE PARAMETROS COM  ( ENTER )',H);
  GOTOXY(3,18);

  READLN(LINHA_DE_ENTRADA);
  IF LINHA_DE_ENTRADA='' THEN EXIT;
  FOR I:=1 TO LENGTH(LINHA_DE_ENTRADA)
    DO LINHA_DE_ENTRADA[I]:=UPCASE(LINHA_DE_ENTRADA[I]);
  FOR I:=1 TO LENGTH(LINHA_DE_ENTRADA)
    DO BEGIN
       IF NOT(LINHA_DE_ENTRADA[I] IN ['0'..'9','A'..'F',',']) THEN
         BEGIN
           WINDOW(3,14,78,23);
           CLRSCR;
           IMPRIME(2,2,'CARACTERE INVALIDO ! ',H);
           WRITE(CHR(7));
           WINDOW(1,1,80,25);
           GOTOXY(X[CAMPO],Y[CAMPO]);
           DELAY(2000);
           EXIT;
         END;
     END;
```

```
I:=1;     ORIG_STR:='';
WHILE COPY(LINHA_DE_ENTRADA,I,1) <> ',' DO
BEGIN
  IF LENGTH(ORIG_STR)=4 THEN DELETE(ORIG_STR,1,1);
  ORIG_STR:= ORIG_STR + COPY(LINHA_DE_ENTRADA,I,1);
  I:=I+1;
END;
ORIG:=NUMHEX2(ORIG_STR);

I:=I+1;   DEST_STR:='';
WHILE COPY(LINHA_DE_ENTRADA,I,1) <> ',' DO
BEGIN
  IF LENGTH(DEST_STR)=4 THEN DELETE(DEST_STR,1,1);
  DEST_STR:= DEST_STR + COPY(LINHA_DE_ENTRADA,I,1);
  I:=I+1;
END;
DEST:=NUMHEX2(DEST_STR);

I:=I+1;   QUANT_STR:='';
WHILE I <= LENGTH(LINHA_DE_ENTRADA) DO
BEGIN
  IF LENGTH(QUANT_STR)=4 THEN DELETE(QUANT_STR,1,1);
  QUANT_STR:= QUANT_STR + COPY(LINHA_DE_ENTRADA,I,1);
  I:=I+1;
END;
QUANT:=NUMHEX2(QUANT_STR);

IF ((DEST+QUANT > END_FINAL_BUFFER) OR (ORIG+QUANT > END_FINAL_BUFFER) OR
    (DEST < END_INICIAL_BUFFER) OR (ORIG < END_INICIAL_BUFFER)) THEN
                    BEGIN
                      LIMPA_QUADRO;
                      IMPRIME(10,18,'ERRO: PARAMETROS INVALIDOS !',H);
                      IMPRIME(10,22,'TECLA < ESPACO > PARA VOLTAR AO MENU.',H);
                      WRITE(CHR(7));
                      GOTOXY(X[0],Y[0]);
                      REPEAT UNTIL READKEY=' ';
                      GOTOXY(X[CAMPO],Y[CAMPO]);
                      EXIT;
                    END;
IF DEST > ORIG THEN FOR I:=0 TO QUANT DO
                  BEGIN
                    BUFFER[DEST+QUANT-I]:=BUFFER[ORIG+QUANT-I];
                    IF KEYPRESSED AND (READKEY=CHR(27)) THEN EXIT;
                  END;
IF DEST < ORIG THEN FOR I:=0 TO QUANT DO
                  BEGIN
                    BUFFER[DEST+I]:=BUFFER[ORIG+I];
                    IF KEYPRESSED AND (READKEY=CHR(27)) THEN EXIT;
                  END;
LIMPA_QUADRO;
IMPRIME(10,18,'TRANSFERENCIA REALIZADA !',H);
IMPRIME(10,22,'TECLE < ESPACO > PARA VOLTAR AO MENU.',H);
GOTOXY(X[0],Y[0]);
REPEAT UNTIL READKEY=' ';
```

```
  GOTOXY(X[CAMPO],Y[CAMPO]);
END;

{-------------------------------------------------------------------------}

PROCEDURE CONVERSOR_INTELMEM;
{
 CONVERTE UM ARQUIVO EM FORMATO INTEL MCS/86
 PARA OUTRO EM FORMATO IMAGEM DE MEMORIA.
}
{$M $4000,$0,$0}

VAR STR_OFFSET:STRING[4];

BEGIN
  IF ((NOMEFONTE='') OR (NOMEFONTE='????????.???')) THEN
  BEGIN
    LIMPA_QUADRO;
    IMPRIME(10,18,'ENTRE ANTES COM O NOME DO ARQUIVO DE TRABALHO !',H);
    IMPRIME(10,22,'TECLE < ESPACO > PARA VOLTAR AO MENU.',H);
    WRITE(CHR(7));
    GOTOXY(X[8],Y[8]);
    REPEAT UNTIL READKEY=' ';
    GOTOXY(X[CAMPO],Y[CAMPO]);
    EXIT;
  END;
  LIMPA_QUADRO;
  IMPRIME(10,18,'ENTRE COM O DESLOCAMENTO DESEJADO: ',H);
  STR_OFFSET:='0100'; GOTOXY(45,18); WRITE(STR_OFFSET);
  REPEAT
    CH:=READKEY;
    CH:=UPCASE(CH);
    IF CH=CHR(27) THEN BEGIN
                         CH:=' ';
                         DELAY(500);
                         EXIT;
                       END;
    IF CH=' ' THEN STR_OFFSET:='0100';
    IF VALEHEX THEN STR_OFFSET:=COPY(STR_OFFSET,2,3)+CH;
    GOTOXY(45,18); WRITE(STR_OFFSET);
  UNTIL CH=CHR(13);
  IF ((NOMEDESTINO='????????.???') OR (NOMEDESTINO=''))
  THEN NOMEDESTINO:=COPY(NOMEFONTE,1,POS('.',NOMEFONTE))+'MEM';
  CLRSCR;
  SWAPVECTORS;
  EXEC(DIRETORIO_INICIAL+'\INTELMEM.EXE',NOMEFONTE+' '+NOMEDESTINO+' '+STR_OFFSET);
  SWAPVECTORS;
  DELAY(2000);
  IMPRIME_TELA_PRINCIPAL;
  GOTOXY(X[CAMPO],Y[CAMPO]);
END;
```

```
{-------------------------------------------------------------------}

PROCEDURE CONVERSOR_MEMINTEL;
{
 CONVERTE UM ARQUIVO IMAGEM DE MEMORIA
 PARA OUTRO EM FORMATO INTEL MCS/86.
}
{$M $4000,$0,$0}

BEGIN
  IF ((NOMEFONTE='') OR (NOMEFONTE='????????.???')) THEN
  BEGIN
    LIMPA_QUADRO;
    IMPRIME(10,18,'ENTRE ANTES COM O NOME DO ARQUIVO DE TRABALHO !',H);
    IMPRIME(10,22,'TECLE < ESPACO > PARA VOLTAR AO MENU.',H);
    WRITE(CHR(7));
    GOTOXY(X[8],Y[8]);
    REPEAT UNTIL READKEY=' ';
    GOTOXY(X[CAMPO],Y[CAMPO]);
    EXIT;
  END;
  IF ((NOMEDESTINO='????????.???') OR (NOMEDESTINO=''))
  THEN NOMEDESTINO:=COPY(NOMEFONTE,1,POS('.',NOMEFONTE))+'INT';
  CLRSCR;
  SWAPVECTORS;
  EXEC(DIRETORIO_INICIAL+'\MEMINTEL.EXE',NOMEFONTE+' '+NOMEDESTINO);
  SWAPVECTORS;
  DELAY(2000);
  IMPRIME_TELA_PRINCIPAL;
  GOTOXY(X[CAMPO],Y[CAMPO]);
END;

{-------------------------------------------------------------------}

PROCEDURE CONVERSORES;
{
 SELECIONA TIPO DE CONVERSOR, CONFORME A ESCOLHA.
}
VAR TECLA:CHAR;
BEGIN
  LIMPA_QUADRO;
  IMPRIME(3,14,'OS PROGRAMAS CONVERSORES SERÃO CARREGADOS DO DIRETORIO '+DIRETORIO_INICIAL,H);
  IMPRIME(10,16,'ESCOLHA O TIPO DE CONVERSOR: ',H);
  IMPRIME(10,18,'I - Intel ---> Mem',H);
  IMPRIME(10,20,'M - Mem   ---> Intel',H);
  IMPRIME(10,22,'< ESPACO > Abandona comando',H);
  REPEAT
    TECLA:=READKEY;
    TECLA:=UPCASE(TECLA);
  UNTIL (TECLA='I') OR (TECLA='M') OR (TECLA=' ');
```

```
  CASE TECLA OF
    'I': CONVERSOR_INTELMEM;
    'M': CONVERSOR_MEMINTEL;
    ' ': EXIT;
  END;
END;

{-------------------------------------------------------------------}

PROCEDURE TERMINAL_KD_16;
{
 EXECUTA O PROGRAMA "TERMINAL", PARA COMUNICAÇÃO COM O KD-16.
}
BEGIN
  CLRSCR;
  SWAPVECTORS;
  EXEC(DIRETORIO_INICIAL+'\TERMINAL.EXE','');
  SWAPVECTORS;
  DELAY(2000);
  IMPRIME_TELA_PRINCIPAL;
  GOTOXY(X[CAMPO],Y[CAMPO]);
END;

{-------------------------------------------------------------------}

PROCEDURE EXECUTA;
{
 CHAMA A ROTINA SELECIONADA CONFORME A FUNÇÃO.
 TRANSFORMA OS VALORES EM FORMA DE STRINGS PARA VALORES NUMERICOS.
 TESTA VALIDADE DOS ENDEREÇOS LIMITES DOS BLOCOS DE MEMORIA E EPROM.
}
VAR SELETOR:BYTE;
    ERRO:INTEGER;

BEGIN
  END_INICIAL_BUFFER:=NUMHEX2(END_INIC_BUF);
  END_FINAL_BUFFER:=NUMHEX2(END_FIM_BUF);
  END_INICIAL_EPROM:=NUMHEX2(END_INIC_EP);
  END_FINAL_EPROM:=NUMHEX2(END_FIM_EP);

  IF (END_FINAL_EPROM < END_INICIAL_EPROM) THEN
  BEGIN
    LIMPA_QUADRO;
    IMPRIME(10,10,'ERRO: END. FINAL DA EPROM  <  END. INICIAL DA EPROM.',H);
    IMPRIME(10,22,'TECLE < ESPACO > PARA VOLTAR AO MENU.',H);
    WRITE(CHR(7));
    GOTOXY(X[CAMPO],Y[CAMPO]);
    REPEAT UNTIL READKEY = ' ';
    EXIT;
  END;
```

```
IF (END_FINAL_BUFFER > 61440) THEN
BEGIN
  LIMPA_QUADRO;
  IMPRIME(10,18,'ERRO: END. FINAL DO BUFFER  > F000 ( 61440 )',H);
  IMPRIME(10,22,'TECLE < ESPACO > PARA VOLTAR AO MENU.',H);
  WRITE(CHR(7));
  GOTOXY(X[CAMPO],Y[CAMPO]);
  REPEAT UNTIL READKEY = ' ';
  EXIT;
END;

IF (END_FINAL_BUFFER < END_INICIAL_BUFFER) THEN
BEGIN
  LIMPA_QUADRO;
  IMPRIME(10,18,'ERRO: END. FINAL DO BUFFER  <  END. INICIAL DO BUFFER.',H);
  IMPRIME(10,22,'TECLE < ESPACO > PARA VOLTAR AO MENU.',H);
  WRITE(CHR(7));
  GOTOXY(X[CAMPO],Y[CAMPO]);
  REPEAT UNTIL READKEY = ' ';
  EXIT;
END;

ERRO:=0;  SELETOR:=0;
VAL('$'+COPY(FUNCAO,1,1),SELETOR,ERRO);
IF ERRO <> 0 THEN EXIT;

CASE SELETOR OF
  $0: COPIA_EPROM_NO_BUFFER;
  $1: GRAVA_BUFFER_NA_EPROM;
  $2: COPIA_ARQUIVO_NO_BUFFER;
  $3: GRAVA_BUFFER_NO_ARQUIVO;
  $4: COMPARA_EPROM_BUFFER;
  $5: VERIFICA_EPROM_APAGADA;
  $6: CHECKSUM;
  $7: LISTA_CONTEUDO_DO_BUFFER;
  $8: LIGA_DESLIGA_IMPRESSORA;
  $9: EDITA_CONTEUDO_DO_BUFFER;
  $A: PROCURA_BLOCO_NO_BUFFER;
  $B: MOVE_BLOCO_NO_BUFFER;
  $C: CONVERSORES;
  $D: DIRETORIO;
  $E: ENCHE_BUFFER_COM_BYTE;
  $F: TERMINAL_KD_16;
 END;
END;
```

```
{---------------------------------------------------------------------}

BEGIN { INICIO DO PROGRAMA GRAVADOR }
  DIRETORIO_INICIAL:='';
  DRIVE_NUMERO:=0;
  GETDIR(DRIVE_NUMERO,DIRETORIO_INICIAL);
  INICIALIZA_PROGRAMA_GRAVADOR;
  IMPRIME_TELA_PRINCIPAL;
  REPEAT
    CH:=READKEY;
    CH:=UPCASE(CH);
    IF CH=CHR(13) THEN BEGIN
                         EXECUTA;
                         MENU;
                        END;
    IF CH=#0 THEN BEGIN { FOI ACIONADA UMA TECLA ESPECIAL }
                    CH:=READKEY;
                    IF CH='M' THEN RODA_CAMPO;
                    IF CH='K' THEN FOR C:=1 TO 7 DO RODA_CAMPO;
                    CH:=#0;
                  END;
    IF CH<>CHR(27)THEN ALTERA_CAMPO;
  IF CH=CHR(27) THEN BEGIN
                       DELAY(200);
                       WRITE(CHR(7));
                       IMPRIME(3,12,' ESC -> SAI    N -> NAO SAI',H);
                       REPEAT
                       CH:=READKEY;
                       CH:=UPCASE(CH);
                       UNTIL ((CH=CHR(27)) OR (CH='N'));
                       IF CH='N' THEN BEGIN
                                        IMPRIME(3,12,'                          ',H);
                                        CH:=CHR(00);
                                        GOTOXY(X[CAMPO],Y[CAMPO]);
                                      END;
                    END;
  UNTIL CH=CHR(27);
  CLRSCR;
END.

{XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX}
```

APÊNDICE C3

PROGRAMA MEMINTEL

```
PROGRAM MEMINTEL;

USES CRT;

{
 ESTE PROGRAMA CONVERTE UM ARQUIVO BINARIO (COMPOSTO DE BYTES) PARA
 UM ARQUIVO TIPO TEXT NO PADRÃO INTEL MCS/86.

 APÓS A ENTRADA DO NOME DO ARQUIVO FONTE E OPCIONALMENTE DOS  ENDE-
 REÇOS DE EXECUÇÃO E DE CARGA, INICIA-SE A CONVESÃO PELO ACIONAMENTO
 DA TECLA <RETURN>.

 PRIMEIRAMENTE, O  ARQUIVO  SERA  CARREGADO  NA MEMORIA, POSTERIOR-
 MENTE, REALIZA-SE A CONVERSÃO E SIMULTANEAMENTE GERA-SE O ARQUIVO
 DESTINO.
}

VAR SAIDA:STRING[60];
    NOMEFONTE,NOMEDESTINO:STRING[12];
    CS_EXE,IP_EXE,CS_CARGA,IP_CARGA,DESCONTO:STRING[4];
    CH:CHAR;
    FLAG_ERRO:BOOLEAN;
    FONTE:FILE OF BYTE;
    X:ARRAY[1..7] OF INTEGER;
    Y:ARRAY[1..7] OF INTEGER;
    CAMPO:INTEGER;
    DESC,TAMANHO:WORD;
    C:BYTE;

{-------------------------------------------------------------------------}

TYPE CARAC=STRING[12];
FUNCTION STRING_ATUAL:CARAC;
{
 RETORNA NA VARIAVEL 'STRING_ATUAL', A STRING DO CAMPO ATIVO.
}
BEGIN
 CASE CAMPO OF
  1:STRING_ATUAL:=NOMEFONTE;
  2:STRING_ATUAL:=CS_EXE;
  3:STRING_ATUAL:=IP_EXE;
  4:STRING_ATUAL:=CS_CARGA;
  5:STRING_ATUAL:=IP_CARGA;
  6:STRING_ATUAL:=DESCONTO;
 END;{FIM DO CASE}
END;
```

```
{-------------------------------------------------------------------}

FUNCTION VALEHEX:BOOLEAN;
{
 RETORNA TRUE/FALSE SE A VARIAVEL 'CH' FOR CARACTERE HEX VALIDO OU NAO.
}
BEGIN
 IF (CH IN ['0'..'9','A'..'F']) THEN VALEHEX := TRUE
                                ELSE VALEHEX := FALSE;
END;

{-------------------------------------------------------------------}

FUNCTION VALENOME:BOOLEAN;
{
 RETORNA TRUE/FALSE SE A VARIAVEL 'CH' FOR VALIDA PARA COMPOR
 UM NOME DE ARQUIVO.
}
BEGIN
 IF (CH IN[CHR(8),' ','!','#'..')',',','.','0'..'9','@'..'Z','_','a'..'{','}'])
 THEN VALENOME := TRUE
 ELSE VALENOME := FALSE;
END;

{-------------------------------------------------------------------}

PROCEDURE RODA_CAMPO;
{
 MUDA O CAMPO ATIVO NA TELA DE ENTRADA.
}
BEGIN
 GOTOXY( X[CAMPO],Y[CAMPO] );
 TEXTCOLOR(13);
 WRITE( STRING_ATUAL );
 IF CAMPO=6 THEN CAMPO := 1
            ELSE CAMPO := CAMPO + 1;
 GOTOXY( X[CAMPO],Y[CAMPO] );
 TEXTCOLOR(15);
 WRITE( STRING_ATUAL );
END;

{-------------------------------------------------------------------}

PROCEDURE ALTERA_CAMPO;
{
 MODEDIFICA O CONTEUDO DO CAMPO VALIDO SE O CARACTERE NA VARIAVEL 'CH'
 FOR UM CARACTERE VALIDO PARA AQUELE CAMPO.
}
VAR MEIO:STRING[12];
```

```
BEGIN
  IF (( CAMPO = 1 ) AND VALENOME ) THEN
  BEGIN
    GOTOXY(X[CAMPO],Y[CAMPO]);
    IF ( CH = ' ' )
      THEN NOMEFONTE := ''
      ELSE IF ( CH = CHR(8) )
            THEN NOMEFONTE := COPY(NOMEFONTE,1,LENGTH(NOMEFONTE)-1)
            ELSE NOMEFONTE := NOMEFONTE + CH;
    IF NOMEFONTE='' THEN BEGIN
                           WRITE('           ');
                           GOTOXY(X[CAMPO],Y[CAMPO]);
                         END
                    ELSE IF CH=CHR(8) THEN WRITE(NOMEFONTE+' '+CHR(8))
                                      ELSE WRITE(NOMEFONTE);
    EXIT;
  END;
  IF  ((CAMPO IN [2..6]) AND VALEHEX )
  THEN BEGIN
         GOTOXY(X[CAMPO],Y[CAMPO]);
         MEIO := COPY(STRING_ATUAL,2,3)+CH;
         CASE CAMPO OF
           2: CS_EXE    := MEIO;
           3: IP_EXE    := MEIO;
           4: CS_CARGA  := MEIO;
           5: IP_CARGA  := MEIO;
           6: DESCONTO  := MEIO;
         END; ( FIM DO CASE )
       WRITE(MEIO);
       END;
END;

{-------------------------------------------------------------------------}

TYPE STR2=STRING[2];
FUNCTION NUMHEX(STR_HEX:STR2):BYTE;
{
 RECEBE UMA STRING DE DOIS CARACTERES HEX VALIDOS E DEVOLVE SEU
 VALOR NUMERICO EM UM BYTE.
}
VAR HH   : BYTE;
    ERRO : INTEGER;

BEGIN
  VAL( ( '$' + STR_HEX ) , HH , ERRO );
  IF ERRO <> 0 THEN HH := 0;
  NUMHEX := HH
END;
```

```
{-------------------------------------------------------------------------}

TYPE STR4 = STRING[4];
FUNCTION NUMHEX2( STR_HEX2 : STR4) : WORD;
{
 RECEBE UMA STRING DE 4 CARACTERES E DEVOLVE O INTEIRO CORRESPONDENTE.
}
VAR I    : WORD;
    ERRO : INTEGER;

BEGIN
  VAL( ('$' + STR_HEX2 ) , I , ERRO );
  IF ERRO <> 0 THEN I := 0;
  NUMHEX2 := I
END;

{-------------------------------------------------------------------------}

FUNCTION CARACTHEX( BAI : BYTE ) : STR2;
{
 RECEBE UM BYTE E DEVOLVE SEU VALOR NUMA STRING DE DOIS CARACTERES HEX.
}
VAR CHD,CHU : CHAR;
    D,U     : BYTE;

BEGIN
 D := BAI DIV 16;
 U := BAI MOD 16;
 IF D <= 9 THEN CHD := CHR( 48 + D ) ELSE CHD := CHR( 55 + D );
 IF U <= 9 THEN CHU := CHR( 48 + U ) ELSE CHU := CHR( 55 + U );
 CARACTHEX := CHD + CHU;
END;

{-------------------------------------------------------------------------}

FUNCTION CARACTHEX2( PALAVRA : WORD ) : STR4;
{
 RECEBE UM INTEIRO E CONVERTE PARA UMA STRING DE 4 CARACTERES HEX.
}
BEGIN
 CARACTHEX2:=CARACTHEX( HI( PALAVRA )) + CARACTHEX( LO( PALAVRA ));
END;

{-------------------------------------------------------------------------}

TYPE LINHA = STRING[80];
FUNCTION SAIDA_CHECKSUM:LINHA;
{
 RECEBE A STRING 'SAIDA' E FORNECE A MESMA STRING ACRESCIDA DO
 BYTE DE CHECAGEM DE ERRO, 'CHECKSUM'.
}
VAR CHECKSUM:BYTE;
    II:INTEGER;
```

```
BEGIN ( COMEÇO DA FUNCTION SAIDA_CHECKSUM )
  CHECKSUM := 0;    II := 2;
  WHILE (II < LENGTH( SAIDA )) DO
  BEGIN
    CHECKSUM := CHECKSUM + NUMHEX( COPY( SAIDA , II , 2 ));
    II := II + 2;
  END;
  SAIDA_CHECKSUM := SAIDA + CARACTHEX( 0 - CHECKSUM );
END;

{-------------------------------------------------------------------------}

PROCEDURE CONVERTE;
{
 RELIZA A CONVERSÃO DO ARQUIVO FONTE(BINARIO) PARA O ARQUIVO
 DESTINO(TEXTO) NO FORMATO INTEL MCS/86.
}
{$M $F000,0,$FFFF}
CONST TAMANHO=$E000;
VAR BUFFER:ARRAY[1..TAMANHO] OF BYTE;
    I,J,K,ILOOP:WORD;
    FONTE:FILE OF BYTE;
    DESTINO:TEXT;
    NOME:STRING[12];

BEGIN

  IF (NOMEFONTE = '') OR (NOMEFONTE='????????.???') THEN
                     BEGIN
                       WRITE( CHR( 7 ) );
                       GOTOXY(X[7],Y[7]);
                       WRITE( 'ERRO: NOME INVALIDO       ' );
                       FLAG_ERRO := TRUE;
                       EXIT;
                     END;

  ASSIGN( FONTE , NOMEFONTE );
  {$I-} RESET( FONTE ); {$I+}

  IF IORESULT <> 0 THEN BEGIN
                         WRITE( CHR( 7 ) );
                         GOTOXY( X[7],Y[7] );
                         WRITE( 'ERRO: ARQ. INEXISTENTE   ' );
                         FLAG_ERRO := TRUE;
                         EXIT;
                        END;
```

```
IF FILESIZE( FONTE ) > TAMANHO THEN BEGIN
                                    WRITE( CHR( 7 ) );
                                    GOTOXY( X[7],Y[7] );
                                    WRITE( 'ERRO: ARQ. GRANDE        ' );
                                    CLOSE(FONTE);
                                    FLAG_ERRO := TRUE;
                                    EXIT;
                                   END;

GOTOXY( X[7],Y[7] );
WRITE( 'LENDO ARQ. P/ A MEMORIA  ' );
DELAY( 2000 );
DESC:= HUNHEX2(DESCONTO);
FOR I := 1 TO FILESIZE(FONTE) DO READ(FONTE,BUFFER[I]);

GOTOXY( X[7],Y[7] );
WRITE( 'ARQ. CARREGADO NA MEMORIA' );
DELAY( 2000 );

GOTOXY( X[7],Y[7] );
WRITE( '                         ' );
GOTOXY( X[7],Y[7] );
WRITE( 'NUMERO DE BYTES = ' , FILESIZE( FONTE ) - ( DESC ));
DELAY( 2000 );
IF NOMEDESTINO='' THEN
NOMEDESTINO := COPY( NOMEFONTE , 1 , POS( '.' , NOMEFONTE )) + 'INT';
GOTOXY( X[7],Y[7] );
WRITE( '                         ' );
GOTOXY( X[7],Y[7] );
WRITE( 'DESTINO ', NOMEDESTINO );
DELAY( 2000 );

ASSIGN( DESTINO , NOMEDESTINO );
{$I-} REWRITE( DESTINO ); {$I+}

CASE IORESULT OF
  $F1:BEGIN
        WRITE( CHR(7) );
        GOTOXY( X[7],Y[7] );
        WRITE( 'ERRO: DIR. CHEIO OU INEX.' );
        DELAY( 2000 );
        FLAG_ERRO := TRUE;
        EXIT;
      END;
  $F0:BEGIN
        WRITE( CHR(7) );
        GOTOXY( X[7],Y[7] );
        WRITE( 'ERRO: ERRO DE GRAVACAO   ' );
        DELAY( 2000 );
        FLAG_ERRO := TRUE;
        EXIT;
      END;
END;  { FIM DO CASE }
```

```
GOTOXY( X[7],Y[7] );
WRITE( 'CONVERTENDO ARQUIVO      ' );
DELAY( 2000 );

{ PRIMEIRA LINHA }
SAIDA := ':04000003' + CS_EXE + IP_EXE;
WRITELN( DESTINO , SAIDA_CHECKSUM );

{ SEGUNDA LINHA }
SAIDA := ':02000002' + CS_CARGA;
WRITELN( DESTINO , SAIDA_CHECKSUM );

{ NUCLEO }

ILOOP := 1+DESC ; SAIDA := '';

WHILE ILOOP < FILESIZE( FONTE ) DO
BEGIN
  IF (( ILOOP+16 ) <= FILESIZE( FONTE ))
        THEN J := ILOOP + 15
        ELSE J:= FILESIZE( FONTE );

  FOR K := ILOOP TO J DO SAIDA:=SAIDA+CARACTHEX(BUFFER[K]);

  SAIDA := '00'+ SAIDA;
  { CODIGO DO REGISTRO DE DADOS }

  SAIDA := CARACTHEX2( NUMHEX2( IP_CARGA )+( ILOOP-1 ) -DESC)+SAIDA;
  { ENDEREÇO DE CARGA DO PRIMEIRO DADO DESSE REGISTRO DE DADO }

  SAIDA := ';' + CARACTHEX(( J - ILOOP ) + 1 ) + SAIDA;
  { NUMERO DE BYTES PRESENTES NESSE REGISTRO DE DADOS }

  WRITELN( DESTINO , SAIDA_CHECKSUM );

  SAIDA := '';
  ILOOP := J + 1;
END;

{ ULTIMA LINHA }
SAIDA := ':00000001';
WRITELN( DESTINO , SAIDA_CHECKSUM );

CLOSE( FONTE );
CLOSE( DESTINO );

GOTOXY( X[7],Y[7] );
WRITE( 'CONVERSÃO REALIZADA      ' );
DELAY( 2000 );

GOTOXY( X[7],Y[7] );
WRITE( 'ENTRADA DE PARAMETROS    ' );
END;
```

```
{-------------------------------------------------------------------------}

BEGIN ( INICIO DO PROGRAMA MEMINTEL )
  IF PARAMCOUNT > 0 THEN
  BEGIN
    IF PARAMCOUNT = 1 THEN
    BEGIN
      NOMEFONTE:=PARAMSTR(1)+'            ';
      NOMEDESTINO := COPY( NOMEFONTE , 1 , POS( '.' , NOMEFONTE )) + 'INT';
    END;
    IF PARAMCOUNT = 2 THEN
    BEGIN
      NOMEFONTE:=PARAMSTR(1)+'            ';
      NOMEDESTINO:=PARAMSTR(2);
    END;
    IF PARAMCOUNT > 2 THEN BEGIN
                             WRITE(CHR(7));
                             CLRSCR;
                             WRITE('ERRO: NUMERO DE PARAMETROS INVALIDO ! ');
                             DELAY(4000);
                             EXIT;
                           END;
  END ELSE BEGIN
            NOMEFONTE := '????????.???';
            NOMEDESTINO:='';
          END;
  CS_EXE    := '0000';
  IP_EXE    := '0100';
  CS_CARGA  := '0000';
  IP_CARGA  := '0100';
  DESCONTO  := '0000';
  CAMPO     := 1;
  X[1] := 21;  Y[1] := 3;
  X[2] :=  4;  Y[2] := 11;
  X[3] :=  9;  Y[3] := 11;
  X[4] := 18;  Y[4] := 11;
  X[5] := 23;  Y[5] := 11;
  X[6] := 55;  Y[6] := 3;
  X[7] := 45;  Y[7] := 11;
  TEXTCOLOR(15);
  GOTOXY(X[CAMPO],Y[CAMPO]);

  { TELA PRINCIPAL }

  CLRSCR;
  WRITELN( ' PROGRAMA CONVERSOR MEMINTEL 1.0                DESCONTO');

  WRITELN('                                                  ┌──────┐');
  WRITE('┌ ARQUIVO BINARIO ┐ '); WRITE( NOMEFONTE ); WRITE(' │      │');
  TEXTCOLOR(13); WRITE('   0000   '); TEXTCOLOR(15);WRITELN('└──────┘');
  WRITELN('└─────────────────┘');
  WRITELN;
  WRITELN;
  WRITELN( ' END. EXECUCAO   END. CARGA' );
```

```
WRITELN( '┌───────────┬───────────┐');
WRITELN( '│  CS : IP  │  CS : IP  │                    STATUS');
WRITELN( '├───────────┼───────────┤          ┌──────────────────┐');
WRITE( '│ ' ); TEXTCOLOR(13); WRITE( CS_EXE,':',IP_EXE ); TEXTCOLOR(15);
WRITE( ' │ ' );   TEXTCOLOR(13); WRITE( CS_CARGA,':',IP_CARGA );TEXTCOLOR(15);
WRITE( ' │');
WRITELN( '            │ ENTRADA DE PARAMETROS   │');
WRITELN( '└───────────┴───────────┘          └──────────────────┘' );
GOTOXY(1,23);
WRITELN('<',CHR(27),' ',CHR(26),'> MUDA CAMPO ATIVO   < RETURN > CONVERTE ARQUIVO   < ESC > ABANDONA PROGRAMA');
GOTOXY(X[CAMPO],Y[CAMPO]);
REPEAT
  FLAG_ERRO := FALSE;
  CH := READKEY;
  CH := UPCASE(CH);
  IF CH = CHR(13)  THEN CONVERTE;
  IF CH=#0 THEN BEGIN { FOI ACIONADA UMA TECLA ESPECIAL }
                CH:=READKEY;
                IF CH='M' THEN RODA_CAMPO;
                IF CH='K' THEN FOR C:=1 TO 5 DO RODA_CAMPO;
                CH:=#0;
               END;

  IF CH <> CHR(27) THEN ALTERA_CAMPO;

  IF   FLAG_ERRO   THEN BEGIN
                        DELAY( 2000 );
                        GOTOXY( X[7],Y[7] );
                        HIGHVIDEO;
                        WRITE( 'ENTRADA DE PARAMETROS    ' );
                        FLAG_ERRO := FALSE;
                       END;
 IF CH=CHR(27) THEN BEGIN
                    DELAY(200);
                    GOTOXY(X[7],Y[7]);
                    WRITE('ESC -> SAI  N -> NAO SAI');
                    REPEAT
                      CH:=READKEY;
                      CH:=UPCASE(CH);
                    UNTIL ((CH='N') OR (CH=CHR(27)));
                    IF CH='N' THEN BEGIN
                                    CH:=CHR(00);
                                    GOTOXY(X[7],Y[7]);
                                    WRITE('ENTRADA DE PARAMETROS    ');
                                    GOTOXY(X[CAMPO],Y[CAMPO]);
                                   END;
                   END;
 UNTIL ( CH = CHR(27) );
CLRSCR;
END.
{XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX}
```

APÊNDICE C4

PROGRAMA RASTRO

```
PROGRAM RASTRO;

USES CRT,PRINTER;

{  VARIAVEIS GLOBAIS  }

TYPE  STR2=STRING[2];
      STR4=STRING[4];
      STR5=STRING[5];
      STR80=STRING[80];

VAR
      C            :BYTE;    { VARIAVEL AUXILIAR DO PROG. PRINCIPAL}
      CH           :CHAR;    { USADA PARA ENTRADAS PELO TECLADO }
      OPER         :STR5;    { GUARDA A ULTIMA OPERÇÃO NO BARRAMENTO }
      CAMPO        :INTEGER; { GUARDA O NUMERO DO CAMPO ATIVO NA TELA }
      DENTRO       :BOOLEAN; { SE TRUE=> O END. NO BARRAMENTO CONTIDO NUMA DAS FAIXAS ESPECIFICADAS }
      PRINTER_ON   :BOOLEAN; { SE TRUE=> INDICA QUE A IMPRESSORA SERA USADA }
      IMP_COM      :STR5;    { PODE ASSUMIR AS MENSAGENS "LIGA" OU "DESL" P/INDICAR AÇÃO SOBRE A IMPRESSORA }
      INIC_FAIXA_1 :STR5;    { LIMITE INICIAL DA FAIXA 1           }
      FIM_FAIXA_1  :STR5;    { LIMITE FINAL DA FAIXA 1             }
      IORD_FAIXA_1 :STR5;    {    PODEM ASSUMIR:                   }
      IOWR_FAIXA_1 :STR5;    {        DESAB.     OCORREU    HABIL. }
      MRD_FAIXA_1  :STR5;    {    -> "    " , "    " , "      "   }
      MWR_FAIXA_1  :STR5;    {                                     }
      INIC_FAIXA_2 :STR5;    { LIMITE INICIAL DA FAIXA 2           }
      FIM_FAIXA_2  :STR5;    { LIMITE FINAL DA FAIXA 2             }
      IORD_FAIXA_2 :STR5;    {    PODEM ASSUMIR:                   }
      IOWR_FAIXA_2 :STR5;    {        DESAB.     OCORREU    HABIL. }
      MRD_FAIXA_2  :STR5;    {    -> "    " , "    " , "      "   }
      MWR_FAIXA_2  :STR5;    {                                     }
      INIC_FAIXA_3 :STR5;    { LIMITE INICIAL DA FAIXA 3           }
      FIM_FAIXA_3  :STR5;    { LIMITE FINAL DA FAIXA 3             }
      IORD_FAIXA_3 :STR5;    {    PODEM ASSUMIR:                   }
      IOWR_FAIXA_3 :STR5;    {        DESAB.     OCORREU    HABIL. }
      MRD_FAIXA_3  :STR5;    {    -> "    " , "    " , "      "   }
      MWR_FAIXA_3  :STR5;    {                                     }
      END_FAIXA_1  :STR5;    { ULTIMO END. OCORRIDO NO CAMPO 1     }
      END_FAIXA_2  :STR5;    { ULTIMO END. OCORRIDO NO CAMPO 2     }
      END_FAIXA_3  :STR5;    { ULTIMO END. OCORRIDO NO CAMPO 3     }
      DADO_FAIXA_1 :STR2;    { ULTIMO DADO OCORRIDO NO CAMPO 1     }
      DADO_FAIXA_2 :STR2;    { ULTIMO DADO OCORRIDO NO CAMPO 2     }
      DADO_FAIXA_3 :STR2;    { ULTIMO DADO OCORRIDO NO CAMPO 3     }
      END_ENTR     :STR5;    { ULTIMO END. LIDO DO BARRAMENTO      }
      DADO_ENTR    :STR2;    { ULTIMO DADO LIDO DO BARRAMENTO      }
      INIC_NUM_1   :LONGINT; { INIC_FAIXA_1 CONVERTIDO P/ NUMERO   }
      INIC_NUM_2   :LONGINT; { INIC_FAIXA_2 CONVERTIDO P/ NUMERO   }
      INIC_NUM_3   :LONGINT; { INIC_FAIXA_3 CONVERTIDO P/ NUMERO   }
      FIM_NUM_1    :LONGINT; { FIM_FAIXA_1  CONVERTIDO P/ NUMERO   }
      FIM_NUM_2    :LONGINT; { FIM_FAIXA_2  CONVERTIDO P/ NUMERO   }
      FIM_NUM_3    :LONGINT; { FIM_FAIXA_3  CONVERTIDO P/ NUMERO   }
      END_ENTR_NUM :LONGINT; { END_ENTR     CONVERTIDO P/ NUMERO   }
```

```
CONST H=TRUE;
      L=FALSE;
      BLOCO_BRANCO='▓▓▓';          { SINALIZA OCORRÊNCIA DE UM COMANDO      }
      NULO_5='▒▒▒▒▒';              { SINALIZA CAMPO DE ENDEREÇO DESABILITADO }
      NULO_4='▒▒▒▒';               { SINALIZA CAMPO DE COMANDO DESABILITADO  }
      ESPACO_2='  ';               { LIMPA CAMPO DE DADOS   }
      ESPACO_4='    ';             { LIMPA CAMPO DE COMANDO }
      ESPACO_5='     ';            { LIMPA CAMPO DE END.    }
      CTRL_8255_1=$303;
      MODO_0_ABC_ENT=$9B;
      CTRL_8255_2=$307;
      MODO_0_A_SAI_BC_ENT=$8B;
      PA1_8255_END_0_7=$300;
      PB1_8255_END_8_15=$301;
      PC1_8255_DADO_0_7=$302;
      PA2_8255_LIBERA_RDY=$304;
      PC2_8255_END_16_19_OPER=$306;
      { MATRIZ DE POSIÇÕES DOS CAMPOS NA TELA PRINCIPAL }
      X:ARRAY[1..25] OF BYTE = (4,15,41,46,51,56,4,15,41,46,51,56,
                                4,15,41,46,51,56,26,26,26,7,36,36,36);
      Y:ARRAY[1..25] OF BYTE = (8,8,8,8,8,8,10,10,10,10,10,10,12,12,
                                12,12,12,12,8,10,12,17,8,10,12);
      { X[1]=4    Y[1]=8    POSIÇÃO DO INICIO 1ª FAIXA       }
      { X[2]=15   Y[2]=8    POSIÇÃO DO FIM    1ª FAIXA       }
      { X[3]=41   Y[3]=8    POSIÇÃO DO IORD   1ª FAIXA       }
      { X[4]=46   Y[4]=8    POSIÇÃO DO IOWR   1ª FAIXA       }
      { X[5]=51   Y[5]=8    POSIÇÃO DO MRD    1ª FAIXA       }
      { X[6]=56   Y[6]=8    POSIÇÃO DO MWR    1ª FAIXA       }
      { X[7]=4    Y[7]=10   POSIÇÃO DO INICIO 2ª FAIXA       }
      { X[8]=15   Y[8]=10   POSIÇÃO DO FIM    2ª FAIXA       }
      { X[9]=41   Y[9]=10   POSIÇÃO DO IORD   2ª FAIXA       }
      { X[10]=46  Y[10]=10  POSIÇÃO DO IOWR   2ª FAIXA       }
      { X[11]=51  Y[11]=10  POSIÇÃO DO MRD    2ª FAIXA       }
      { X[12]=56  Y[12]=10  POSIÇÃO DO MWR    2ª FAIXA       }
      { X[13]=4   Y[13]=12  POSIÇÃO DO INICIO 3ª FAIXA       }
      { X[14]=15  Y[14]=12  POSIÇÃO DO FIM    3ª FAIXA       }
      { X[15]=41  Y[15]=12  POSIÇÃO DO IORD   3ª FAIXA       }
      { X[16]=46  Y[16]=12  POSIÇÃO DO IOWR   3ª FAIXA       }
      { X[17]=51  Y[17]=12  POSIÇÃO DO MRD    3ª FAIXA       }
      { X[18]=56  Y[18]=12  POSIÇÃO DO MWR    3ª FAIXA       }
      { X[19]=26  Y[19]=8   POSIÇÃO DO ENDEREÇO ACES. 1ª     }
      { X[20]=26  Y[20]=10  POSIÇÃO DO ENDEREÇO ACES. 2ª     }
      { X[21]=26  Y[21]=12  POSIÇÃO DO ENDEREÇO ACES. 3ª     }
      { X[22]=7   Y[22]=17  POSIÇÃO DO STATUS DA IMPRESSORA  }
      { X[23]=36  Y[23]=8   POSIÇÃO DO DADO ACES. 1ª         }
      { X[24]=36  Y[24]=10  POSIÇÃO DO DADO ACES. 2ª         }
      { X[25]=36  Y[25]=12  POSIÇÃO DO DADO ACES. 3ª         }
```

```
{-------------------------------------------------------------------------}

FUNCTION VALEHEX:BOOLEAN;
{
 RETORNA TRUE/FALSE SE A VARIAVEL "CH" FOR CARACTERE HEX VALIDO OU NAO
}
BEGIN
  IF( CH IN ['0'..'9','A'..'F']) THEN VALEHEX:=TRUE ELSE VALEHEX:=FALSE;
END;

{-------------------------------------------------------------------------}

PROCEDURE P(CAR,N:BYTE);
{
 IMPRIME NA POSIÇAO DO CURSOR "N" BYTES ASCII INDICADOS POR "CAR".
}
VAR I:BYTE;

BEGIN
  FOR I:=1 TO N DO WRITE(CHR(CAR));
END;

{-------------------------------------------------------------------------}

PROCEDURE IMPRIME(XP,YP:BYTE;FRASE:STR80;BRILHO:BOOLEAN);
{
 IMPRIME UMA STRING DE ATé 80 CARACTERES COM O ATRIBUTO
 BRILHO, NA POSIÇXO X,Y.
}
BEGIN
  IF (XP IN [1..80]) AND (YP IN [1..25]) THEN GOTOXY(XP,YP);
  IF BRILHO THEN TEXTCOLOR(15) ELSE TEXTCOLOR(13);
  WRITE(FRASE);
END;

{-------------------------------------------------------------------------}

PROCEDURE ATUALIZA_BARRAMENTO;

{
  SE:
  O END_ENTR PERTENCER A FAIXA "n";
  A FAIXA "n" ESTIVER HABILITADA;
  O CAMPO DENTRO DA FAIXA "n" ESTIVER HABILITADO;
  ENTXO ----> ATUALIZA FAIXA "n"
  SE A IMPRESSORA ESTIVER HABILITADA => SAIDA TAMBéM NA IMPRESSORA
}
VAR SALTA_1,SALTA_2:BOOLEAN;
```

```
BEGIN
SALTA_1:=TRUE; { SE O END. LIDO NO BARRAMENTO NAO ESTIVER CONTIDO NAS FAIXAS 1 OU 2          }
SALTA_2:=TRUE; { OU SE APESAR DE CONTIDOS, O COMANDO LIDO NO BARRAMENTO ESTIVER             }
               { DESABILITADO, ESTAS VARIAVEIS FAZEM COM QUE SEJAM DEIXADAS COLUNAS EM BRANCO }

  { ATUALIZA FAIXA 1 }
  IF ((INIC_NUM_1<=END_ENTR_NUM)AND(FIM_NUM_1>=END_ENTR_NUM)AND
     (INIC_FAIXA_1<>NULO_5))THEN        { VERIFICA SE DENTRO DA FAIXA 1 }
     BEGIN
       END_FAIXA_1:=END_ENTR;
       DADO_FAIXA_1:=DADO_ENTR;
       IMPRIME(XI[19],Y[19],END_FAIXA_1,L);
       IMPRIME(XI[23],Y[23],DADO_FAIXA_1,L);

{  LIMPA TODAS AS INDICAÇÕES "    " DA FAIXA 1, SE HABILITADA }
       IF IORD_FAIXA_1<>NULO_4 THEN
       BEGIN
         IORD_FAIXA_1:=ESPACO_4;IMPRIME(XI[3],Y[3],ESPACO_4,L);
       END;
       IF IOWR_FAIXA_1<>NULO_4 THEN
       BEGIN
         IOWR_FAIXA_1:=ESPACO_4;IMPRIME(XI[4],Y[4],ESPACO_4,L);
       END;
       IF MRD_FAIXA_1<>NULO_4 THEN
       BEGIN
         MRD_FAIXA_1:=ESPACO_4;IMPRIME(XI[5],Y[5],ESPACO_4,L);
       END;
       IF MWR_FAIXA_1<>NULO_4 THEN
       BEGIN
         MWR_FAIXA_1:=ESPACO_4;IMPRIME(XI[6],Y[6],ESPACO_4,L);
     END;

{ ATUALIZA AS FAIXAS HABILITADAS CONFORME A VARIAVEL "OPER" }
       IF ((OPER='IORD')AND(IORD_FAIXA_1<>NULO_4))THEN
         BEGIN
           IORD_FAIXA_1:=BLOCO_BRANCO;
           IMPRIME(XI[3],Y[3],IORD_FAIXA_1,L);
           IF PRINTER_ON THEN
           BEGIN
             SALTA_1:=FALSE;
             WRITE(LST,'1-( ',END_FAIXA_1,' - ',DADO_FAIXA_1,' - ',OPER:4,' )    ');
           END;
         END;
       IF ((OPER='IOWR')AND(IOWR_FAIXA_1<>NULO_4))THEN
         BEGIN
           IOWR_FAIXA_1:=BLOCO_BRANCO;
           IMPRIME(XI[4],Y[4],IOWR_FAIXA_1,L);
           IF PRINTER_ON THEN
           BEGIN
             SALTA_1:=FALSE;
             WRITE(LST,'1-( ',END_FAIXA_1,' - ',DADO_FAIXA_1,' - ',OPER:4,' )    ');
           END;
         END;
```

```
      IF ((OPER='MRD')AND(MRD_FAIXA_1<>NULO_4))THEN
        BEGIN
          MRD_FAIXA_1:=BLOCO_BRANCO;
          IMPRIME(X[5],Y[5],MRD_FAIXA_1,L);
          IF PRINTER_ON THEN
          BEGIN
            SALTA_1:=FALSE;
            WRITE(LST,'1-( ',END_FAIXA_1,' - ',DADO_FAIXA_1,' - ',OPER:4,' )    ');
          END;
        END;
      IF ((OPER='MWR')AND(MWR_FAIXA_1<>NULO_4))THEN
        BEGIN
          MWR_FAIXA_1:=BLOCO_BRANCO;
          IMPRIME(X[6],Y[6],MWR_FAIXA_1,L);
          IF PRINTER_ON THEN
          BEGIN
            SALTA_1:=FALSE;
            WRITE(LST,'1-( ',END_FAIXA_1,' - ',DADO_FAIXA_1,' - ',OPER:4,' )    ');
          END;
        END;
    END;

{ SE NENHUM COMANDO HABILITADO NA FAIXA 1 ACONTECEU E   }
{ A IMPRESSORA ESTA HABILITADA  =>  IMPRIME ESPAÇOS     }
    IF (PRINTER_ON AND (SALTA_1=TRUE)) THEN
    WRITE(LST,'                              ');


  { ATUALIZA FAIXA 2}
  IF ((INIC_NUM_2<=END_ENTR_NUM)AND(FIM_NUM_2>=END_ENTR_NUM)AND
      (INIC_FAIXA_2<>NULO_5))THEN          { VERIFICA SE DENTRO DA FAIXA 2 }
    BEGIN
      END_FAIXA_2:=END_ENTR;
      DADO_FAIXA_2:=DADO_ENTR;
      IMPRIME(X[20],Y[20],END_FAIXA_2,L);
      IMPRIME(X[24],Y[24],DADO_FAIXA_2,L);

{  LIMPA TODAS AS INDICAÇÕES " [redacted] " DA FAIXA 2, SE HABILITADA }
      IF IORD_FAIXA_2<>NULO_4 THEN
       BEGIN
         IORD_FAIXA_2:=ESPACO_4;IMPRIME(X[9],Y[9],ESPACO_4,L);
       END;
      IF IOWR_FAIXA_2<>NULO_4 THEN
       BEGIN
         IOWR_FAIXA_2:=ESPACO_4;IMPRIME(X[10],Y[10],ESPACO_4,L);
       END;
      IF MRD_FAIXA_2<>NULO_4 THEN
       BEGIN
         MRD_FAIXA_2:=ESPACO_4;IMPRIME(X[11],Y[11],ESPACO_4,L);
       END;
      IF MWR_FAIXA_2<>NULO_4 THEN
       BEGIN
         MWR_FAIXA_2:=ESPACO_4;IMPRIME(X[12],Y[12],ESPACO_4,L);
       END;
```

```
{ ATUALIZA AS FAIXAS HABILITADAS CONFORME A VARIAVEL "OPER" }
      IF ((OPER='IORD')AND(IORD_FAIXA_2<>NULO_4))THEN
        BEGIN
          IORD_FAIXA_2:=BLOCO_BRANCO;
          IMPRIME(X[9],Y[9],IORD_FAIXA_2,L);
          IF PRINTER_ON THEN
          BEGIN
            SALTA_2:=FALSE;
            WRITE(LST,'2-( ',END_FAIXA_2,' - ',DADO_FAIXA_2,' - ',OPER:4,' )    ');
          END;
        END;
      IF ((OPER='IOWR')AND(IOWR_FAIXA_2<>NULO_4))THEN
        BEGIN
          IOWR_FAIXA_2:=BLOCO_BRANCO;
          IMPRIME(X[10],Y[10],IOWR_FAIXA_2,L);
          IF PRINTER_ON THEN
          BEGIN
            SALTA_2:=FALSE;
            WRITE(LST,'2-( ',END_FAIXA_2,' - ',DADO_FAIXA_2,' - ',OPER:4,' )    ');
          END;
        END;
       IF ((OPER='MRD')AND(MRD_FAIXA_2<>NULO_4))THEN
        BEGIN
          MRD_FAIXA_2:=BLOCO_BRANCO;
          IMPRIME(X[11],Y[11],MRD_FAIXA_2,L);
          IF PRINTER_ON THEN
          BEGIN
            SALTA_2:=FALSE;
            WRITE(LST,'2-( ',END_FAIXA_2,' - ',DADO_FAIXA_2,' - ',OPER:4,' )    ');
          END;
        END;
       IF ((OPER='MWR')AND(MWR_FAIXA_2<>NULO_4))THEN
        BEGIN
          MWR_FAIXA_2:=BLOCO_BRANCO;
          IMPRIME(X[12],Y[12],MWR_FAIXA_2,L);
          IF PRINTER_ON THEN
          BEGIN
            SALTA_2:=FALSE;
            WRITE(LST,'2-( ',END_FAIXA_2,' - ',DADO_FAIXA_2,' - ',OPER:4,' )    ');
          END;
        END;
      END;

 { SE NENHUM COMANDO HABILITADO NA FAIXA 2 ACONTECEU E   }
 { A IMPRESSORA ESTA HABILITADA  =>  IMPRIME ESPAÇOS     }
      IF (PRINTER_ON AND (SALTA_2=TRUE)) THEN
        WRITE(LST,'                              ');
```

```
 { ATUALIZA FAIXA 3}
 IF ((INIC_NUM_3<=END_ENTR_NUM)AND(FIM_NUM_3>=END_ENTR_NUM)AND
     (INIC_FAIXA_3<>NULO_5))THEN        { VERIFICA SE DENTRO DA FAIXA 3 }
    BEGIN
      END_FAIXA_3:=END_ENTR;
      DADO_FAIXA_3:=DADO_ENTR;
      IMPRIME(X[21],Y[21],END_FAIXA_3,L);
      IMPRIME(X[25],Y[25],DADO_FAIXA_3,L);

{ LIMPA TODAS AS INDICAÇÕES "    " DA FAIXA 3, SE HABILITADA }
      IF IORD_FAIXA_3<>NULO_4 THEN
      BEGIN
      IORD_FAIXA_3:=ESPACO_4;IMPRIME(X[15],Y[15],ESPACO_4,L);
      END;
      IF IOWR_FAIXA_3<>NULO_4 THEN
      BEGIN
      IOWR_FAIXA_3:=ESPACO_4;IMPRIME(X[16],Y[16],ESPACO_4,L);
      END;
      IF MRD_FAIXA_3<>NULO_4 THEN
      BEGIN
      MRD_FAIXA_3:=ESPACO_4;IMPRIME(X[17],Y[17],ESPACO_4,L);
      END;
      IF MWR_FAIXA_3<>NULO_4 THEN
      BEGIN
      MWR_FAIXA_3:=ESPACO_4;IMPRIME(X[18],Y[18],ESPACO_4,L);
      END;

 { ATUALIZA AS FAIXAS HABILITADAS CONFORME A VARIAVEL "OPER" }
       IF ((OPER='IORD')AND(IORD_FAIXA_3<>NULO_4))THEN
         BEGIN
           IORD_FAIXA_3:=BLOCO_BRANCO;
           IMPRIME(X[15],Y[15],IORD_FAIXA_3,L);
           IF PRINTER_ON THEN
           WRITE(LST,'3-( ',END_FAIXA_3,' - ',DADO_FAIXA_3,' - ',OPER:4,' )    ');
         END;
       IF ((OPER='IOWR')AND(IOWR_FAIXA_3<>NULO_4))THEN
         BEGIN
           IOWR_FAIXA_3:=BLOCO_BRANCO;
           IMPRIME(X[16],Y[16],IOWR_FAIXA_3,L);
           IF PRINTER_ON THEN
           WRITE(LST,'3-( ',END_FAIXA_3,' - ',DADO_FAIXA_3,' - ',OPER:4,' )    ');
         END;
       IF ((OPER='MRD')AND(MRD_FAIXA_3<>NULO_4))THEN
         BEGIN
           MRD_FAIXA_3:=BLOCO_BRANCO;
           IMPRIME(X[17],Y[17],MRD_FAIXA_3,L);
           IF PRINTER_ON THEN
           WRITE(LST,'3-( ',END_FAIXA_3,' - ',DADO_FAIXA_3,' - ',OPER:4,' )    ');
         END;
```

```
      IF ((OPER='MWR')AND(MWR_FAIXA_3<>NULO_4))THEN
        BEGIN
          MWR_FAIXA_3:=BLOCO_BRANCO;
          IMPRIME(X[18],Y[18],MWR_FAIXA_3,L);
          IF PRINTER_ON THEN
          WRITE(LST,'3-( ',END_FAIXA_3,' - ',DADO_FAIXA_3,' - ',OPER:4,' )     ');
        END;
    END;

{ AO FINAL DA TERCEIRA COLUNA MUDA DE LINHA NA IMPRESSORA }
  IF PRINTER_ON THEN WRITE(LST,CHR(13),CHR(10));
  GOTOXY(X[CAMPO],Y[CAMPO]);
END;

{-------------------------------------------------------------------}

FUNCTION STRING_ATUAL:STR5;
{
 SAI NA VARIAVEL STRING_ATUAL A STRING DO CAMPO VALIDO.
}
BEGIN
  CASE CAMPO OF
    1:  STRING_ATUAL:=INIC_FAIXA_1;
    2:  STRING_ATUAL:=FIM_FAIXA_1;
    3:  STRING_ATUAL:=IORD_FAIXA_1;
    4:  STRING_ATUAL:=IOWR_FAIXA_1;
    5:  STRING_ATUAL:=MRD_FAIXA_1;
    6:  STRING_ATUAL:=MWR_FAIXA_1;
    7:  STRING_ATUAL:=INIC_FAIXA_2;
    8:  STRING_ATUAL:=FIM_FAIXA_2;
    9:  STRING_ATUAL:=IORD_FAIXA_2;
    10: STRING_ATUAL:=IOWR_FAIXA_2;
    11: STRING_ATUAL:=MRD_FAIXA_2;
    12: STRING_ATUAL:=MWR_FAIXA_2;
    13: STRING_ATUAL:=INIC_FAIXA_3;
    14: STRING_ATUAL:=FIM_FAIXA_3;
    15: STRING_ATUAL:=IORD_FAIXA_3;
    16: STRING_ATUAL:=IOWR_FAIXA_3;
    17: STRING_ATUAL:=MRD_FAIXA_3;
    18: STRING_ATUAL:=MWR_FAIXA_3;
  END;
END;

{-------------------------------------------------------------------}

FUNCTION NUMHEX(STR_HEX:STR2):BYTE;
{
 RECEBE UMA STRING DE DOIS CARACTERES HEX VALIDOS E DEVOLVE SEU
 VALOR NUMERICO EM UM BYTE.
}
VAR HH:BYTE;
    ERRO:INTEGER;
```

```
BEGIN
  VAL(('$'+STR_HEX),HH,ERRO);
  IF ERRO<>0 THEN HH:=0;
  NUMHEX:=HH
END;

{-------------------------------------------------------------------}

FUNCTION NUMHEX2(STR_HEX2:STR4):WORD;
{
 RECEBE UMA STRING DE 4 CARACTERES E DEVOLVE O INTEIRO CORRESPONDENTE.
}
VAR I:WORD;
    ERRO:INTEGER;

BEGIN
  VAL(('$'+STR_HEX2),I,ERRO);
  IF ERRO<>0 THEN I:=0;
  NUMHEX2:=I
END;

{-------------------------------------------------------------------}

FUNCTION NUMHEX5(STR_HEX5:STR5):LONGINT;
{
 RECEBE UMA STRING DE 5 CARACTERES E DEVOLVE O INTEIRO CORRESPONDENTE.
}
VAR L1:LONGINT;
    L2:BYTE;

BEGIN
  L1:=NUMHEX2(COPY(STR_HEX5,2,4));
  L2:=NUMHEX(COPY(STR_HEX5,1,1));
  NUMHEX5:=L1+L2*65536;
END;

{-------------------------------------------------------------------}

FUNCTION CARACTHEX(BAI:BYTE):STR2;
{
 RECEBE UM BYTE E DEVOLVE SEU VALOR NUMA STRING DE DOIS CARACTERES HEX.
}
VAR CHD,CHU:CHAR;
    D,U:BYTE;

BEGIN
  D:=BAI DIV 16;
  U:=BAI MOD 16;
  IF D<=9 THEN CHD:=CHR(48+D) ELSE CHD:=CHR(55+D);
  IF U<=9 THEN CHU:=CHR(48+U) ELSE CHU:=CHR(55+U);
  CARACTHEX:=CONCAT(CHD,CHU);
END;
```

```
{-------------------------------------------------------------------}

FUNCTION CARACTHEX2(PALAVRA:WORD):STR4;
{
 RECEBE UMA PALAVRA E CONVERTE PARA UMA STRING DE 4 CARACTERES HEX.
}
BEGIN
  CARACTHEX2:=CONCAT(CARACTHEX(HI(PALAVRA)),CARACTHEX(LO(PALAVRA)));
END;

{-------------------------------------------------------------------}

PROCEDURE LE_BARRAMENTO;
{
  APÓS A PARALISAÇÃO DO KD-16 PELA PLACA RASTREADORA, LÊ O BARRAMENTO ATRAVÉS
  DAS PORTAS 8255 P1 E P2 DA PLACA DE INTERFACE PARALELA E TEMPORIZAÇÃO.
  OS BYTES SÃO LIDOS E MONTADOS PARA FORMAR O ENDEREÇO DE ENTRADA "END_ENTR".
  OS COMANDOS LIDOS PERMITEM A DETERMINAÇÃO DA VARIAVEL "OPER".
}

VAR END_0_7,END_8_15,END_16_19,DADO_0_7,COMANDO : BYTE;
    END_0_7_H,END_8_15_H,END_16_19_H           : STR2;
    TECLA:CHAR;

BEGIN
  REPEAT     { ESPERA SINAL READY IR PARA "0" }
    COMANDO:=(PORT[PC2_8255_END_16_19_OPER] AND $08);
    IF KEYPRESSED THEN TECLA:=READKEY; { TECLA "ESC" PERMITE SAIDA FORÇADA DESTE "LOOP" }
  UNTIL ((COMANDO=$00) OR (TECLA=CHR(27)));

          { LE ENDEREÇO E DADO }
  END_0_7:= PORT[PA1_8255_END_0_7];
  END_0_7_H:=CARACTHEX(END_0_7);

  END_8_15:= PORT[PB1_8255_END_8_15];
  END_8_15_H:=CARACTHEX(END_8_15);

  END_16_19:= ((PORT[PC2_8255_END_16_19_OPER] SHR 4) AND $0F );
  END_16_19_H:=CARACTHEX(END_16_19);

  END_ENTR:=COPY(END_16_19_H,2,1)+END_8_15_H+END_0_7_H;
  END_ENTR_NUM:=NUMHEX5(END_ENTR); { "END_ENTR_NUM" É O ENDEREÇO }
                                   {  DE ENTRADA TRANSFORMADO P/ NUMERO }

  DADO_0_7:= PORT[PC1_8255_DADO_0_7];
  DADO_ENTR:=CARACTHEX(DADO_0_7);
```

```
COMANDO:= (PORTIPC2_8255_END_16_19_OPER] AND $07);
CASE COMANDO OF
  $07: OPER:='IORD';
  $06: OPER:='MWR';    ( DECODIFICAçÃO DOS COMANDOS )
  $05: OPER:='MRD';
  $03: OPER:='IOWR';
  ELSE OPER:= '    ';
END;

{
 OS DIVERSOS TESTES ABAIXO MODIFICAM OU NÃO A VARIAVEL "DENTRO" DE MODO QUE,
 ESTA INFORME SOBRE A OCORRÊNCIA NO BARRAMENTO DE UM ENDEREÇO E UM COMANDO
 HABILITADOS
}
 IF (  (INIC_NUM_1<=END_ENTR_NUM) AND (FIM_NUM_1>=END_ENTR_NUM) AND
       (   ((OPER='IORD') AND (IORD_FAIXA_1<>NULO_4))  OR
           ((OPER='IOWR') AND (IOWR_FAIXA_1<>NULO_4))  OR
           ((OPER='MRD')  AND (MRD_FAIXA_1<>NULO_4))   OR
           ((OPER='MWR')  AND (MWR_FAIXA_1<>NULO_4))         ) )
  THEN BEGIN
        DENTRO:=TRUE;
        EXIT;
       END
  ELSE DENTRO:=FALSE;

 IF (  (INIC_NUM_2<=END_ENTR_NUM) AND (FIM_NUM_2>=END_ENTR_NUM) AND
       (   ((OPER='IORD') AND (IORD_FAIXA_2<>NULO_4))  OR
           ((OPER='IOWR') AND (IOWR_FAIXA_2<>NULO_4))  OR
           ((OPER='MRD')  AND (MRD_FAIXA_2<>NULO_4))   OR
           ((OPER='MWR')  AND (MWR_FAIXA_2<>NULO_4))         ) )
  THEN BEGIN
        DENTRO:=TRUE;
        EXIT;
       END
  ELSE DENTRO:=FALSE;

 IF (  (INIC_NUM_3<=END_ENTR_NUM) AND (FIM_NUM_3>=END_ENTR_NUM) AND
       (   ((OPER='IORD') AND (IORD_FAIXA_3<>NULO_4))  OR
           ((OPER='IOWR') AND (IOWR_FAIXA_3<>NULO_4))  OR
           ((OPER='MRD')  AND (MRD_FAIXA_3<>NULO_4))   OR
           ((OPER='MWR')  AND (MWR_FAIXA_3<>NULO_4))         ) )
  THEN BEGIN
        DENTRO:=TRUE;
        EXIT;
       END
  ELSE DENTRO:=FALSE;

END;
```

```
{-------------------------------------------------------------------}

PROCEDURE RODA_CAMPO;
{
 MUDA O CAMPO ATIVO NA TELA PRINCIPAL.
}
BEGIN
  IF CAMPO<1 THEN BEGIN
                   CAMPO:=CAMPO+1;
                   GOTOXY(X[CAMPO],Y[CAMPO]);TEXTCOLOR(13);WRITE(STRING_ATUAL);
                   GOTOXY(X[CAMPO],Y[CAMPO]);TEXTCOLOR(15);WRITE(STRING_ATUAL);
                   EXIT;
                  END;
  GOTOXY(X[CAMPO],Y[CAMPO]);TEXTCOLOR(13);WRITE(STRING_ATUAL);
  IF CAMPO=18 THEN CAMPO:=1 ELSE CAMPO:=CAMPO+1;
  GOTOXY(X[CAMPO],Y[CAMPO]);TEXTCOLOR(15);WRITE(STRING_ATUAL);
END;

{-------------------------------------------------------------------}

PROCEDURE IMPRIME_TELA_PRINCIPAL;

BEGIN
  CLRSCR;
  TEXTCOLOR(15);

  { LINHA 1 }
  P(201,1);P(205,52);P(209,1);P(205,5);P(187,1);WRITELN;

  { LINHA 2 }
  P(186,1);P(32,4);WRITE('Rastreador passo a passo para o Kit KD-16');
  P(32,7);P(179,1);P(32,1);WRITE('1.0');P(32,1);P(186,1);WRITELN;

  { LINHA 3 }
  P(204,1);P(205,21);P(209,1);P(205,30);P(207,1);
  P(205,5);P(185,1);WRITELN;

  { LINHA 4 }
  P(186,1);P(32,3);WRITE('Faixa rastreada');P(32,3);P(179,1);P(32,13);
  WRITE('Barramento');P(32,13);P(186,1);WRITELN;

  { LINHA 5 }
  P(199,1);P(196,10);P(194,1);P(196,10);P(197,1);P(196,9);P(194,1);
  P(196,6);P(194,1);P(196,4);P(194,1);P(196,4);
  P(194,1);P(196,4);P(194,1);P(196,4);P(182,1);WRITELN;

  { LINHA 6 }
  P(186,1);P(32,2);WRITE('Inicio');P(32,2);P(179,1);P(32,3);WRITE('Fim');
  P(32,4);P(179,1);P(32,1);WRITE('Endereço');P(179,1);P(32,1);WRITE('Dado');
  P(32,1);P(179,1);WRITE('IORD');P(179,1);WRITE('IOWR');P(179,1);
  WRITE('MRD ');P(179,1);WRITE('MWR ');P(186,1);WRITELN;
```

```
{ LINHA 7 }
P(204,1);P(205,10);P(216,1);P(205,10);P(216,1);P(205,9);P(216,1);
P(205,6);P(216,1);P(205,4);P(216,1);P(205,4);P(216,1);P(205,4);
P(216,1);P(205,4);P(185,1);WRITELN;

{ LINHA 8 }
P(186,1);P(32,2);TEXTCOLOR(13);WRITE(INIC_FAIXA_1);TEXTCOLOR(15);
P(32,3);P(179,1);P(32,2);TEXTCOLOR(13);WRITE(FIM_FAIXA_1);
TEXTCOLOR(15);P(32,3);P(179,1);P(32,2);TEXTCOLOR(13);WRITE(END_FAIXA_1);
TEXTCOLOR(15);P(32,2);P(179,1);P(32,2);TEXTCOLOR(13);WRITE(DADO_FAIXA_1);
TEXTCOLOR(15);P(32,2);P(179,1);TEXTCOLOR(13);WRITE(IORD_FAIXA_1);
TEXTCOLOR(15);P(179,1);TEXTCOLOR(13);WRITE(IOWR_FAIXA_1);
TEXTCOLOR(15);P(179,1);TEXTCOLOR(13);WRITE(MRD_FAIXA_1);TEXTCOLOR(15);
P(179,1);TEXTCOLOR(13);WRITE(MWR_FAIXA_1);TEXTCOLOR(15);P(186,1);
WRITELN;

{ LINHA 9 }
P(199,1);P(196,10);P(197,1);P(196,10);P(197,1);P(196,9);P(197,1);
P(196,6);P(197,1);P(196,4);P(197,1);P(196,4);P(197,1);P(196,4);
P(197,1);P(196,4);P(182,1);WRITELN;

{ LINHA 10 }
P(186,1);P(32,2);TEXTCOLOR(13);WRITE(INIC_FAIXA_2);TEXTCOLOR(15);
P(32,3);P(179,1);P(32,2);TEXTCOLOR(13);WRITE(FIM_FAIXA_2);TEXTCOLOR(15);
P(32,3);P(179,1);P(32,2);TEXTCOLOR(13);WRITE(END_FAIXA_2);TEXTCOLOR(15);
;P(32,2);P(179,1);P(32,2);TEXTCOLOR(13);WRITE(DADO_FAIXA_2);TEXTCOLOR(15);
P(32,2);P(179,1);TEXTCOLOR(13);WRITE(IORD_FAIXA_2);TEXTCOLOR(15);P(179,1);
TEXTCOLOR(13);WRITE(IOWR_FAIXA_2);TEXTCOLOR(15);P(179,1);TEXTCOLOR(13);
WRITE(MRD_FAIXA_2);TEXTCOLOR(15);P(179,1);TEXTCOLOR(13);WRITE(MWR_FAIXA_2);
TEXTCOLOR(15);P(186,1);WRITELN;

{ LINHA 11 }
P(199,1);P(196,10);P(197,1);P(196,10);P(197,1);P(196,9);P(197,1);
P(196,6);P(197,1);P(196,4);P(197,1);P(196,4);P(197,1);P(196,4);
P(197,1);P(196,4);P(182,1);WRITELN;

{ LINHA 12 }
P(186,1);P(32,2);TEXTCOLOR(13);WRITE(INIC_FAIXA_3);TEXTCOLOR(15);
P(32,3);P(179,1);P(32,2);TEXTCOLOR(13);WRITE(FIM_FAIXA_3);TEXTCOLOR(15);
P(32,3);P(179,1);P(32,2);TEXTCOLOR(13);WRITE(END_FAIXA_3);TEXTCOLOR(15);
;P(32,2);P(179,1);P(32,2);TEXTCOLOR(13);WRITE(DADO_FAIXA_3);TEXTCOLOR(15);
P(32,2);P(179,1);TEXTCOLOR(13);WRITE(IORD_FAIXA_3);TEXTCOLOR(15);P(179,1);
TEXTCOLOR(13);WRITE(IOWR_FAIXA_3);TEXTCOLOR(15);P(179,1);
TEXTCOLOR(13);WRITE(MRD_FAIXA_3);TEXTCOLOR(15);P(179,1);
TEXTCOLOR(13);WRITE(MWR_FAIXA_3);TEXTCOLOR(15);P(186,1);WRITELN;


{ LINHA 13 }
P(204,1);P(205,10);P(207,1);P(205,10);P(207,1);P(205,9);P(207,1);
P(205,6);P(207,1);P(205,4);P(207,1);P(205,4);P(207,1);P(205,4);
P(207,1);P(205,4);P(185,1);WRITELN;

{ LINHA 14 }
P(186,1);P(32,58);P(186,1);WRITELN;
```

```
 { LINHA 15 }
 P(186,1);P(32,2);WRITE('P- Próximo passo  ');P(32,13);
 WRITE(NULO_4,' -> Desabilitado');P(32,4);
 P(186,1);WRITELN;

 { LINHA 16 }
 P(186,1);P(32,58);P(186,1);WRITELN;

 { LINHA 17 }
 P(186,1);P(32,2);WRITE('I- ');TEXTCOLOR(13);WRITE(IMP_COM);
 TEXTCOLOR(15);WRITE(' impressora');
 P(32,13);WRITE ('ESC- Abandona programa');P(32,2);P(186,1);WRITELN;

 { LINHA 18 }
 P(186,1);P(32,58);P(186,1);WRITELN;

 { LINHA 19 }
 P(186,1);P(32,2);WRITE('H- Habilita/Desabilita');P(32,11);
 WRITE(CHR(27),' ',CHR(26),' Muda campo ativo');P(32,3);P(186,1);WRITELN;

 { LINHA 20 }
 P(186,1);P(32,58);P(186,1);WRITELN;

 { LINHA 21 }
 P(200,1);P(205,58);P(188,1);WRITELN;

 GOTOXY(XICAMPO],Y[CAMPO]);

END;

{-------------------------------------------------------------------------}

PROCEDURE CALC_END;
{
 TRANSFORMA AS STRINGS RELATIVAS AOS ENDEREÇOS INICIAIS E FINAIS EM NUMEROS
 QUE POSSIBILITEM COMPARAÇÕES DE MAGNITUDE COM A VARIAVEL "END_ENTR_NUM" PARA
 DEFINIÇÃO DE PERTINENCIA A CADA UMA DAS FAIXAS.
}
BEGIN
  INIC_NUM_1:=NUMHEX5(INIC_FAIXA_1);
  INIC_NUM_2:=NUMHEX5(INIC_FAIXA_2);
  INIC_NUM_3:=NUMHEX5(INIC_FAIXA_3);
  FIM_NUM_1:=NUMHEX5(FIM_FAIXA_1);
  FIM_NUM_2:=NUMHEX5(FIM_FAIXA_2);
  FIM_NUM_3:=NUMHEX5(FIM_FAIXA_3);
```

```
( IMPEDE QUE EXISTAM ENDEREÇOS FINAIS MENORES QUE OS ENDEREÇOS INICIAIS )
 IF CAMPO IN [1,7,13] THEN
 BEGIN
   IF INIC_NUM_1>FIM_NUM_1 THEN BEGIN
                                 FIM_NUM_1:=INIC_NUM_1;
                                 FIM_FAIXA_1:=INIC_FAIXA_1;
                                 IMPRIME_TELA_PRINCIPAL;
                                END;
   IF INIC_NUM_2>FIM_NUM_2 THEN BEGIN
                                 FIM_NUM_2:=INIC_NUM_2;
                                 FIM_FAIXA_2:=INIC_FAIXA_2;
                                 IMPRIME_TELA_PRINCIPAL;
                                END;
   IF INIC_NUM_3>FIM_NUM_3 THEN BEGIN
                                 FIM_NUM_3:=INIC_NUM_3;
                                 FIM_FAIXA_3:=INIC_FAIXA_3;
                                 IMPRIME_TELA_PRINCIPAL;
                                END;
 END;
 IF CAMPO IN [2,8,14] THEN
 BEGIN
   IF INIC_NUM_1>FIM_NUM_1 THEN BEGIN
                                 INIC_NUM_1:=FIM_NUM_1;
                                 INIC_FAIXA_1:=FIM_FAIXA_1;
                                 IMPRIME_TELA_PRINCIPAL;
                                END;
   IF INIC_NUM_2>FIM_NUM_2 THEN BEGIN
                                 INIC_NUM_2:=FIM_NUM_2;
                                 INIC_FAIXA_2:=FIM_FAIXA_2;
                                 IMPRIME_TELA_PRINCIPAL;
                                END;
   IF INIC_NUM_3>FIM_NUM_3 THEN BEGIN
                                 INIC_NUM_3:=FIM_NUM_3;
                                 INIC_FAIXA_3:=FIM_FAIXA_3;
                                 IMPRIME_TELA_PRINCIPAL;
                                END;
 END;
END;

{-------------------------------------------------------------------------}

PROCEDURE ALTERA_CAMPO;
{
 MODIFICA O CONTEUDO DO CAMPO VALIDO SE O CARACTERE NA VARIAVEL
 "CH" FOR UM CARACTERE VALIDO PARA AQUELE CAMPO.
}
BEGIN
  IF VALEHEX THEN
  BEGIN
    CASE CAMPO OF
      1: IF ((INIC_FAIXA_1<>NULO_5)AND(FIM_FAIXA_1<>NULO_5))
         THEN INIC_FAIXA_1 :=COPY(STRING_ATUAL,2,LENGTH(STRING_ATUAL)-1)+CH;
      2: IF ((INIC_FAIXA_1<>NULO_5)AND(FIM_FAIXA_1<>NULO_5))
         THEN FIM_FAIXA_1  :=COPY(STRING_ATUAL,2,LENGTH(STRING_ATUAL)-1)+CH;
```

```
     7: IF ((INIC_FAIXA_2<>NULO_5)AND(FIM_FAIXA_2<>NULO_5))
        THEN INIC_FAIXA_2 :=COPY(STRING_ATUAL,2,LENGTH(STRING_ATUAL)-1)+CH;
     8: IF ((INIC_FAIXA_2<>NULO_5)AND(FIM_FAIXA_2<>NULO_5))
        THEN FIM_FAIXA_2  :=COPY(STRING_ATUAL,2,LENGTH(STRING_ATUAL)-1)+CH;
    13:IF ((INIC_FAIXA_3<>NULO_5)AND(FIM_FAIXA_3<>NULO_5))
        THEN INIC_FAIXA_3 :=COPY(STRING_ATUAL,2,LENGTH(STRING_ATUAL)-1)+CH;
    14:IF ((INIC_FAIXA_3<>NULO_5)AND(FIM_FAIXA_3<>NULO_5))
        THEN FIM_FAIXA_3  :=COPY(STRING_ATUAL,2,LENGTH(STRING_ATUAL)-1)+CH;
   END; ( FIM DO CASE )
   GOTOXY(X[CAMPO],Y[CAMPO]);
   TEXTCOLOR(15);
   CALC_END;
   WRITE(STRING_ATUAL);
  END;   ( FIM DO IF   )
END;

{-------------------------------------------------------------------}

PROCEDURE LIGA_DESLIGA_IMPRESSORA;
{
 LIGA OU DESLIGA A IMPRESSORA PELA MUDANÇA DA VARIAVEL PRINTER_ON.
}
BEGIN
  IF PRINTER_ON THEN PRINTER_ON:=FALSE ELSE PRINTER_ON:=TRUE;
  IF PRINTER_ON THEN IMP_COM:='Desl.' ELSE IMP_COM:='Liga ';
  IMPRIME(X[22],Y[22],IMP_COM,L);
  GOTOXY(X[CAMPO],Y[CAMPO]);
END;

{-------------------------------------------------------------------}

PROCEDURE INICIALIZA;

BEGIN
  TEXTCOLOR(15); ( DE BRILHO MAIS FORTE )
  CH:=' ';
  CAMPO:=18;
  OPER:='';
  END_ENTR:='00000';
  DADO_ENTR:='00';
  PORT[CTRL_8255_1]:=MODO_0_ABC_ENT;
  PORT[CTRL_8255_2]:=MODO_0_A_SAI_BC_ENT;
  INIC_FAIXA_1 :='00000';
  FIM_FAIXA_1  :='FFFFF';
  IORD_FAIXA_1 :=ESPACO_4;
  IOWR_FAIXA_1 :=ESPACO_4;
  MRD_FAIXA_1  :=ESPACO_4;
  MWR_FAIXA_1  :=ESPACO_4;
  INIC_FAIXA_2 :=NULO_5;
  FIM_FAIXA_2  :=NULO_5;
  IORD_FAIXA_2 :=NULO_4;
  IOWR_FAIXA_2 :=NULO_4;
  MRD_FAIXA_2  :=NULO_4;
  MWR_FAIXA_2  :=NULO_4;
```

```
 INIC_FAIXA_3 :=NULO_5;
 FIM_FAIXA_3  :=NULO_5;
 IORD_FAIXA_3 :=NULO_4;
 IOWR_FAIXA_3 :=NULO_4;
 MRD_FAIXA_3  :=NULO_4;
 MWR_FAIXA_3  :=NULO_4;
 END_FAIXA_1  :=ESPACO_5;
 END_FAIXA_2  :=ESPACO_5;
 END_FAIXA_3  :=ESPACO_5;
 DADO_FAIXA_1 :=ESPACO_2;
 DADO_FAIXA_2 :=ESPACO_2;
 DADO_FAIXA_3 :=ESPACO_2;
 END_ENTR_NUM :=0;
 PRINTER_ON:=FALSE;
 DENTRO:=TRUE;
 IMP_COM:='Liga ';
 CALC_END;
END;

{-------------------------------------------------------------------}

PROCEDURE CHAVEIA_CAMPO;
{
 HABILITA OU DESABILITA O CAMPO ATIVO
}

BEGIN
  CASE CAMPO OF
    1 , 2 :BEGIN
           IF INIC_FAIXA_1=NULO_5
           THEN BEGIN
                 INIC_FAIXA_1:='00000';
                 FIM_FAIXA_1:='FFFFF';
                 END_FAIXA_1:=ESPACO_5;
                 DADO_FAIXA_1:=ESPACO_4;
                 IORD_FAIXA_1:=ESPACO_4;
                 IOWR_FAIXA_1:=ESPACO_4;
                 MRD_FAIXA_1:=ESPACO_4;
                 MWR_FAIXA_1:=ESPACO_4;
                END
           ELSE BEGIN
                 INIC_FAIXA_1:=NULO_5;
                 FIM_FAIXA_1:=NULO_5;
                 END_FAIXA_1:=ESPACO_5;
                 DADO_FAIXA_1:=ESPACO_4;
                 IORD_FAIXA_1:=NULO_4;
                 IOWR_FAIXA_1:=NULO_4;
                 MRD_FAIXA_1:=NULO_4;
                 MWR_FAIXA_1:=NULO_4;
                END;
          END;
     3    :IF ((IORD_FAIXA_1=NULO_4)AND(INIC_FAIXA_1<>NULO_5))
                                    THEN IORD_FAIXA_1:=ESPACO_4
                                    ELSE IORD_FAIXA_1:=NULO_4;
```

```
4    :IF ((IOWR_FAIXA_1=NULO_4)AND(INIC_FAIXA_1<>NULO_5))
                                THEN IOWR_FAIXA_1:=ESPACO_4
                                ELSE IOWR_FAIXA_1:=NULO_4;
5    :IF ((MRD_FAIXA_1=NULO_4)AND(INIC_FAIXA_1<>NULO_5))
                               THEN MRD_FAIXA_1:=ESPACO_4
                               ELSE MRD_FAIXA_1:=NULO_4;
6    :IF ((MWR_FAIXA_1=NULO_4)AND(INIC_FAIXA_1<>NULO_5))
                               THEN MWR_FAIXA_1:=ESPACO_4
                               ELSE MWR_FAIXA_1:=NULO_4;

7 , 8 :BEGIN
       IF INIC_FAIXA_2=NULO_5
       THEN BEGIN
             INIC_FAIXA_2:='00000';
             FIM_FAIXA_2:='FFFFF';
             END_FAIXA_2:=ESPACO_5;
             DADO_FAIXA_2:=ESPACO_4;
             IORD_FAIXA_2:=ESPACO_4;
             IOWR_FAIXA_2:=ESPACO_4;
             MRD_FAIXA_2:=ESPACO_4;
             MWR_FAIXA_2:=ESPACO_4;
            END
        ELSE BEGIN
             INIC_FAIXA_2:=NULO_5;
             FIM_FAIXA_2:=NULO_5;
             END_FAIXA_2:=ESPACO_5;
             DADO_FAIXA_2:=ESPACO_4;
             IORD_FAIXA_2:=NULO_4;
             IOWR_FAIXA_2:=NULO_4;
             MRD_FAIXA_2:=NULO_4;
             MWR_FAIXA_2:=NULO_4;
            END;
       END;
 9    :IF ((IORD_FAIXA_2=NULO_4)AND(INIC_FAIXA_2<>NULO_5))
                               THEN IORD_FAIXA_2:=ESPACO_4
                               ELSE IORD_FAIXA_2:=NULO_4;
 10   :IF ((IOWR_FAIXA_2=NULO_4)AND(INIC_FAIXA_2<>NULO_5))
                               THEN IOWR_FAIXA_2:=ESPACO_4
                               ELSE IOWR_FAIXA_2:=NULO_4;
 11   :IF ((MRD_FAIXA_2=NULO_4)AND(INIC_FAIXA_2<>NULO_5))
                               THEN MRD_FAIXA_2:=ESPACO_4
                               ELSE MRD_FAIXA_2:=NULO_4;
 12   :IF ((MWR_FAIXA_2=NULO_4)AND(INIC_FAIXA_2<>NULO_5))
                               THEN MWR_FAIXA_2:=ESPACO_4
                               ELSE MWR_FAIXA_2:=NULO_4;
```

```
13,14 :BEGIN
        IF INIC_FAIXA_3=NULO_5
        THEN BEGIN
              INIC_FAIXA_3:='00000';
              FIN_FAIXA_3:='FFFFF';
              END_FAIXA_3:=ESPACO_5;
              DADO_FAIXA_3:=ESPACO_4;
              IORD_FAIXA_3:=ESPACO_4;
              IOWR_FAIXA_3:=ESPACO_4;
              MRD_FAIXA_3:=ESPACO_4;
              MWR_FAIXA_3:=ESPACO_4;
             END
        ELSE BEGIN
              INIC_FAIXA_3:=NULO_5;
              FIN_FAIXA_3:=NULO_5;
              END_FAIXA_3:=ESPACO_5;
              DADO_FAIXA_3:=ESPACO_4;
              IORD_FAIXA_3:=NULO_4;
              IOWR_FAIXA_3:=NULO_4;
              MRD_FAIXA_3:=NULO_4;
              MWR_FAIXA_3:=NULO_4;
             END;
       END;
 15    :IF ((IORD_FAIXA_3=NULO_4)AND(INIC_FAIXA_3<>NULO_5))
                                  THEN IORD_FAIXA_3:=ESPACO_4
                                  ELSE IORD_FAIXA_3:=NULO_4;
 16    :IF ((IOWR_FAIXA_3=NULO_4)AND(INIC_FAIXA_3<>NULO_5))
                                  THEN IOWR_FAIXA_3:=ESPACO_4
                                  ELSE IOWR_FAIXA_3:=NULO_4;
 17    :IF ((MRD_FAIXA_3=NULO_4)AND(INIC_FAIXA_3<>NULO_5))
                                 THEN MRD_FAIXA_3:=ESPACO_4
                                 ELSE MRD_FAIXA_3:=NULO_4;
 18    :IF ((MWR_FAIXA_3=NULO_4)AND(INIC_FAIXA_3<>NULO_5))
                                 THEN MWR_FAIXA_3:=ESPACO_4
                                 ELSE MWR_FAIXA_3:=NULO_4;
END;
CALC_END;
IMPRIME_TELA_PRINCIPAL;

{
AS DUAS LINHAS ABAIXO EVITAM QUE O CAMPO ATIVO FIQUE COM BRILHO MAIS FRACO
APOS A REIMPRESSAO DA TELA PRINCIPAL }
CAMPO:=CAMPO-1;
RODA_CAMPO;
END;
```

```
{-------------------------------------------------------------------}

PROCEDURE AVANCA_PROGRAMA;
{
 ENVIA PULSO PELA PORTA P2, BIT A0, SOLICITANDO A LIBERAÇÃO DO RDY.
 O PROCESSO DE PULSAR O BIT A0 CONTINUA AUTOMATICAMENTE ATÉ QUE A
 VARIAVEL "DENTRO", DEFINIDA NA ROTINA "LE_BARRAMENTO" SINALIZE A
 PETINENCIA A ALGUMA DAS FAIXAS DE TRABALHO.
 A ROTINA ATUALIZA_BARRAMENTO FAZ A ATUALIZAÇÃO DA TELA PRINCIPAL
 CONFORME O ENDEREÇO, DADO E COMANDO IDENTIFICADO.
}
 VAR TECLA:CHAR;

 BEGIN
   REPEAT
     PORTIPA2_8255_LIBERA_RDY]:= $01;
     PORTIPA2_8255_LIBERA_RDY]:= $00;
     LE_BARRAMENTO;
     IF KEYPRESSED THEN TECLA:=READKEY;
   UNTIL (DENTRO OR (TECLA=CHR(27)));
 {
   CASO NUNCA OCORRA UMA IDENTIFICAÇÃO POSITIVA DAS FAIXAS E COMANDOS
   HABILITADOS, PODE-SE SAIR DESTE "LOOP" SEM FIM, ATRAVÉS DA TECLA "ESC" }
   TECLA:=' ';
   ATUALIZA_BARRAMENTO;
 END;

 {-------------------------------------------------------------------}

 BEGIN { INICIO DO PROGRAMA RASTRO }
   INICIALIZA;
   IMPRIME_TELA_PRINCIPAL;
   RODA_CAMPO; { "INICIALIZA COLOCA "CAMPO"=18, "RODA_CAMPO" FAZ "CAMPO"=1 COM BRILHO + FORTE }
   LE_BARRAMENTO;
   ATUALIZA_BARRAMENTO;
   REPEAT
     CH:=READKEY;
     CH:=UPCASE(CH);
     CASE CH OF
       'P': AVANCA_PROGRAMA;
       'I': LIGA_DESLIGA_IMPRESSORA;
       'H': CHAVEIA_CAMPO;
       #0: BEGIN { FOI ACIONADA UMA TECLA ESPECIAL }
             CH:=READKEY;
             IF CH='H' THEN RODA_CAMPO;
             IF CH='K' THEN FOR C:=1 TO 17 DO RODA_CAMPO;
             CH:=#0;
           END;
       ELSE IF CH<>CHR(27) THEN ALTERA_CAMPO;
     END;
```

```
  IF CH=CHR(27) THEN BEGIN
                       DELAY(200);
                       WRITE(CHR(7));
                       IMPRIME(10,22,' ESC -> SAI    N -> NAO SAI',H);
                       REPEAT
                         CH:=READKEY;
                         CH:=UPCASE(CH);
                       UNTIL ((CH=CHR(27)) OR (CH='N'));
                       IF CH='N' THEN BEGIN
                                        IMPRIME(10,22,'                             ',H);
                                        CH:=CHR(00);
                                        GOTOXY(X[CAMPO],Y[CAMPO]);
                                      END;
                     END;
 UNTIL CH=CHR(27);
 WRITELN;WRITE('          RETONANDO AO DOS...');
 DELAY(1000);
 CLRSCR;
END.

{XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX}
```

REFERÊNCIAS BIBLIOGRAFICAS

1) ADLER, Robin, BAKER, Mark, MARSHALL, Howard D. The Logic Analyser: A New Instrument For Observing Logic Signals. *Hewlett-Packard Journal*, Palo Alto, v. 25, n. 2, p. 2-12, Oct. 1973.

2) BENCHMARK report: Intel iAPX 88 vs Zilog Z80. IN: iAPX 88 book. Santa Clara, California: Intel Corporation, 1981. 186 p. (Appendix).

3) BRISTOW, Geoff, VINSON, Peter, WOLLEN, Dave. Software to write software. In: TSENG, Vincent (Org.). *Microprocessor Development and Development Systems*. London: Granada, 1982. p. 102-121.

4) BYTE. Peterborough: McGraw-Hill Inc., v. 15, n. 3, Mar. 1990. (Prices table, p. 328-329).

5) CIARCIA, Steve. Build the circuit cellar MPX-16 computer system. *Byte*. Peterborough, p. 78-114, Nov. 1982; p. 42-78, Dec. 1982; p. 54-82, Jan. 1983. (Part 1-3).

6) ELEKTOR ELECTRONICS. London: Elektor Publishers Ltd., v. 16, n. 175, Feb. 1990. (prices table, p. 5-7).

7) FARNBACH, Willian A. The Logic State Analyser - Displaing Complex Digital Process in Understandable Form. *Hewlett-Packard Journal*, Palo Alto, v. 25, n. 5, p. 2-9, Jan. 1974.

8) HORTON, Eric p. Build a low-cost 8048-family emulator. *Electronic Design*, [s.l.], p. 179-210, May. 1985.

9) HP 64000-UX microprocessor development environment. Palo Alto: Hewlett-Packard Company, July 1986. 22 p. (Technical Data).

10) IBM-PC hardware reference. Boca Ralon, Florida: International Business Machines Corporation, 1983. 322 p. (I/O channel p. 16-22). (Catálogo de equipamento).

11) INTEL Component data catalog. Santa Clara, California: Intel Corporation, 1980. 1085 p.

12) INTEL iAPX 88 book. Santa Clara, California: Intel Corporation, 1981. 186 p.

13) INTEL iAPX 86,88 user's manual. Santa Clara, California: Intel Corporation, 1981. 856 p.

14) INTEL MCS-86 product description. Santa Clara, California: Intel Corporation, Feb. 1979. 113 p.

15) INTEL Memory components handbook. Santa Clara, California: Intel Corporation, 1980. 440 p.

16) INTEL Peripheral design handbook. Santa Clara, California: Intel Corporation, 1981. 834 p.

17) INTEL SDK-86 MCS-86 system design kit. Monitor listins. Santa Clara, California: Intel Corporation, 1978. 127 p. (Manual order number: 9800699-03 Rev. C).

18) INTEL SDK-86 MCS-86 system design kit. User's guide. Santa Clara , California: Intel Corporation, 1978. 143 p. (Manual order number: 9800698A)

19) INTEL The 8086 family user's manual. Santa Clara, California: Intel Corporation, Oct. 1979. 725p.

20) KISTER, Jack. Test and diagnostics. In: TSENG, Vincent (Org.). *Microprocessor Development and Development Systems*. London: Granada, 1982. p. 56-71.

21) KRUMMEL, Larry, SCHULTZ, Gaymond. Advances in micro-computer development systems. *Computer*, [s.l.], p. 363-369, Feb. 1977.

22) LEJEUNE, Bernard. In-circuit emulator I. In: TSENG, Vincent (Org.). *Microprocessor Development and Development Systems*. London: Granada, 1982. p. 124-128.

23) LIU, Yu-Cheng, GIBSON, Glenn A. *Microcomputer Systems: The 8086/8088 Family. Architecture, Programming, and Design*. 2. ed. Engle Wood Cliffs: Prentice-Hall, 1986. 630 p.

24) MAGERS, Celeste S. Managing software development in microprocessor projects. In: The Institute of Electrical and Electronics Engineers Inc. *Microcomputer systems design and techniques*. Antology; 53 different papers. New York: IEEE, 1980. p. 349-357.

25) MICHELIM, Peter. Cross Debug Systems. Support Multiple Design Environments. *Design Center*, Palo Alto, n. 3, p. 24-27, 1988.

26) MIHALIK, Mike. What does an emulator provide? In: TSENG, Vincent (Org.). *Microprocessor Development and Development Systems*. London: Granada, 1982. p. 146.

27) MIHALIK, Mike. Special emulation processor. In: TSENG, Vincent (Org.). *Microprocessor Development and Development Systems*. London: Granada, 1982. p. 164.

28) MILLER, Alan R. O depurador. In:___. *Assembly IBM PC*; técnicas de programação. São Paulo: EBRAS, 1987. p. 101-105. Tradução de: Assembly Language Techniques for the IBM PC.

29) NATIONAL Semiconductor Microcontroller development suport. Santa Clara, California: National Semiconductor Corporation, Jan. 1989. 13 p. (Lit. # 100456).

30) NATIONAL Semiconductors NS 32CG16 ISE development tool. Santa Clara, California: National Semiconductor Corporation, Jan. 1989. 7 p. (Lit. # 114297).

31) NORTON, Peter. Formatos de arquivos e o que eles significam para você. In:___. *PC-DOS*. Como usar o DOS com inteligência. Rio de Janeiro: Ed. Campos, 1988. p. 113-114. Tradução de: PC-DOS: Introduction to High-Performance Computing.

32) NORTON, Peter, SOCHA, John. Programas .COM versus .EXE. In:___. *Peter Norton, linguagem Assembly para IBM PC*. Rio de Janeiro: Ed. Campos, 1988. p. 235-236. Tradução de: Peter Norton's Assembly Language Book for the IBM PC.

33) NORTON, Peter. The Program Segment Prefix (PSP). In:___. *The Peter Norton Programmer's Guide to the IBM PC*. Washington: Microsoft Press, 1985. p. 260-266.

34) OGDIN, Carol A. The logic analyzer. *Mini-Micro Systems*, [s.l.], p. 370-374, Sept. 1977.

35) OGDIN, Carol A. Project control. *Mini-Micro Systems*, [s.l.], p. 344-348, Nov./Dec. 1977.

36) OGDIN, Carol A. Microcomputer suport aids. *Mini-Micro Systems*, [s.l.], p.358-362, Feb. 1978.

37) OSBORNE, Adam. Sinais de controle de Modem. In:___. *Microprocessadores; conceitos básicos*. São Paulo: McGraw-Hill do Brasil, 1983. v. 1, p. 314-320. Tradução de: Introduction to Microcomputer; v.1 Basic concetps.

38) PANNACH, Arndt, KAPPLER, Wolfgang. A Multichannel Word Generator for Testing Digital Componentes and Systems. *Henlett-Packard Journal*, Palo Alto, v. 26, n. 12, p. 17-24, Aug. 1975.

39) PROGRAMAS Conversores. Manual de operação MS-DOS/PC-DOS. São Paulo: XPTO, 1987. 31 p.

40) QUADROS, Daniel G. A. Prefixo de Segmento de Programa (PSP). In:___. *PC Assembler usando DOS*. Rio de Janeiro: ED. Campos, 1988. p. 64-70.

41) REAL-TIME emulators for 8 and 16-bit microprocessors. Palo Alto: Hewlett-Packard Company, Dec. 1986. 48 p. (Technical Data).

42) ROMANO, John. Quest for better software solutions. *Design Center*, Palo Alto, n. 3, p. 4-56, 1988.

43) SAPONAS, Thomas A. Firmware for a Microprocessor Analyzer. *Hewlett-Packard Journal*, Palo Alto, v. 28, n. 5, p. 12-15, Jan. 1977.

44) SMITH,Jeffrey. A Logic State Analyzer for Microprocessor Systems. *Hewlett-Packard Journal*, Palo Alto, v. 28, n. 5, p. 2-11, Jan. 1977.

45) SMALL, Charles T., MORRILL JR., Justin S. The Logic State Analyser, a Viewing Port For the Data Domain. *Hewlett-Packard Journal*, Palo Alto, v. 26, n. 12, p. 2-11, Aug. 1975.

46) TEXAS INSTRUMENTS. TTL data book for design engineers. 2. ed. [s.l.]: Texas Instruments Incorporated, 1976. 828 p.

47) TEXAS INTRUMENTS. The TTL data book. [s.l.]: Texas Instruments Incorporated, 1987. 1222 p. 2 v.

48) TSENG, Vincent (Org.). What is a development system? In:___. *Microprocessor Development and Development Systems*. London: Granada, 1982. p. 1-5.

49) TSENG, Vincent (Org.). Introduction. In:___. *Microprocessor Development and Development Systems*. London: Granada, 1982. p. XIV-XV.

50) TSENG, Vincent (Org.). Edictor's Conclusions. In:___. *Microprocessor Development and Development Systems*. London: Granada, 1982. p. 122.

51) USATEGUI, José Mª Angulo. Instrumentos de medida para el analisis de problemas con micro-procesadores. In:___. *Microprocessadores*. Curso sobre aplicaciones en sistemas industriales. 4. ed. Madrid: Paraninfo, 1985. p. 224-234.

52) USATEGUI, José Mª Angulo. Avance paso a paso. In:___. *Microprocessadores*. Curso sobre aplicaciones en sistemas industriales. 4. ed. Madrid: Paraninfo, 1985. p. 178-180.

53) WAGNER, William E. Unravelling Problems in the Desing of Microprocessor-Based Systems. *Hewlett-Packard Journal*, Palo Alto, v. 26, n. 12, p. 12-16, Aug. 1975.