UNIVERSIDADE DE SAO PAULO
INSTITUTO DE FISICA E QUIMICA DE SAO CARLOS
DEPARTAMENTO DE FISICA E CIENCIA DOS MATERIAIS

**"Automação de um Espectrômetro por
Ressonância Magnética Nuclear
Pulsada"**

Marco Aurélio de Oliveira Pagnano

**Dissertação apresentada ao IFQSC/USP
para obtenção do título de Mestre em
Ciencias "Fisica Aplicada"**

Orientador: Prof.Dr.Claudio Jose Magon

**Sao Carlos - Sao Paulo
1993**

SERVIÇO DE BIBLIOTECA E INFORMAÇÃO - IFQSC
FÍSICA

210

**IFQSC UNIVERSIDADE
DE SÃO PAULO**
Instituto de Física e Química de São Carlos

Fone (0162) 72-6222
Fax (0162) 72-2218

Av. Dr. Carlos Botelho, 1465
Caixa Postal 369
CEP 13560.970 - São Carlos - SP
Brasil

MEMBROS DA COMISSÃO JULGADORA DA DISSERTAÇÃO DE MESTRADO DE MARCO AURELIO DE OLIVEIRA PAGNANO
APRESENTADA AO INSTITUTO DE FÍSICA E QUÍMICA DE SÃO CARLOS, DA UNIVERSIDADE DE SÃO PAULO, EM
27/05/1992

COMISSÃO JULGADORA:

Claudio Jose Magon

Prof. Dr. Claudio Jose Magon

Said R. Rabbani

Prof. Dr. Said R. Rabbani

Maria Stela Veludo de Paiva

Profa. Dra. Maria Stela Veludo de Paiva

**Dedico este trabalho a DEUS, criador de tudo e
aos meus pais, Paulo Múcio e Neuza.**

AGRADECIMENTOS

A realização desta tese não se deve exclusivamente a mim, mas sim a um trabalho de diversas pessoas. Quero expressar a todos o meu reconhecimento e gratidão, pelo desenvolvimento profissional e humano que me proporcionaram.

Agradeço profundamente:

ao meu orientador, Prof. Dr. Cláudio J. Magon, pelo constante incentivo, pela confiança, pelo exemplo de persistência e dedicação às atividades de pesquisa, e principalmente por demonstrar a todo momento seu lado humano e compreensível perante a todos;

ao Prof. Dr. Tito J. Bonagamba, pelo incansável apoio e pelas ótimas sugestões;

aos Engenheiros, Mateus José Martins, Edson Vidoto, Walter Soto Ensinas Jr. e Elídio de Carvalho Lobão, pelos valiosos conhecimentos transmitidos e pelas interessantes discussões;

aos Técnicos, Odir A. Canevarollo, João Gomes da Silva e J. Carlos Gazziro, pela disponibilidade e pelo enorme interesse em colaborar;

aos alunos de Iniciação Científica, Rodrigo Villares Portugal, Cleber Renato Mendonça e Luís A. Gonçalves de Campos, pelas colaborações e sugestões;

ao amigo, Ruberley Rodrigues de Souza, pelas valiosas contribuições, dentre elas as dicas sobre o editor de texto (WP);

aos colegas de laboratório, Luiz Telmo da Silva Auler e Pedro Luís Frare Jr. , pelos conhecimentos transmitidos, pelas inúmeras contribuições, e pelos agradáveis momentos de descontração passados no mesmo;

à secretária, Leila Lamon, por ser tão prestativa e eficiente;

ao técnico da gráfica, Italo Celestini, pela encardenação desta tese;

ao IFQSC-USP e CAPES, por me fornecerem subsídios para a realização deste trabalho;

à minha família, pelo apoio confiante e pelo carinho sempre demonstrado;

aos meus avós, Edith e Edgard Pagnano, que embora não entendessem o meu trabalho, sempre perguntaram a respeito do andamento do mesmo;

à minha companheira, Nívea Fernandes dos Santos, pela sua paciência, apoio, compreensão e carinho dedicado durante a elaboração deste trabalho;

Enfim, a todas pessoas que me ajudaram a concretizar este trabalho e que sem querer esqueci de mencionar seus nomes.

Muito obrigado a todos !!!

SERVIÇO DE BIBLIOTECA E INFORMAÇÃO - IFQSC
FÍSICA

Resumo

Este trabalho consistiu na automação do espectrômetro de Ressonância Magnética Nuclear Pulsada existente no laboratório dos Professores José Pedro Donoso e Cláudio José Magon. Foram efetuados circuitos adicionais e melhorias no equipamento que eles possuíam, à nível de "hardware" e "software". Foram construídas e programadas interfaces paralelas que ligam o microcomputador a um gerador de pulsos, fabricado pela Tecmag Inc., e a um digitalizador rápido (10 ns) Nicolet 430. O programador de pulsos pode acionar independentemente 75 linhas durante o intervalo de tempo subdividido em 2048 eventos, também independentes. O "software" desenvolvido permite a programação de todas as linhas e eventos de forma eficiente e prática. Foi todo escrito em linguagem C, é modular, portátil e permite a geração de seqüências de pulsos sofisticadas e também o controle da aquisição e transferência de dados.

Abstract

This work describes the project developed to automatize the Pulsed Nuclear Magnetic Resonance Spectrometer housed in the laboratory of the Professors José Pedro Donoso e Cláudio José Magon. To satisfy our needs we have improved their old equipment, at the level of hardware and software. Were built and programmed the parallel interface between the microcomputer and a pulse programmer made by TECMAG INC. and with a fast digitizer (10 ns) Nicolet 430. The pulse programmer can control 75 independent output channels during 2048 time intervals. The software we developed allow us to control the whole pulse sequence on a very efficient and practical way. It was written in C-Language, and provides the timing signals necessary to generate sophisticated pulse sequences, the data acquisition and data transfer to the host computer.

Índice

Introdução.....	1
-----------------	---

Capítulo I - Os Princípios do Espectrômetro por RMN

1.0	Introdução.....	3
1.1	O momento magnético do núcleo (spin).....	3
1.2	O campo magnético estático e o Teorema de Larmor.....	4
1.3	A magnetização do sistema e o seu equilíbrio termodinâmico.....	4
1.4	O campo magnético girante e a retirada do sistema do equilíbrio.....	5
1.5	A frequência de ressonância do sistema.....	6
1.6	O campo magnético linearmente polarizado.....	7
1.7	O pulso de radiofrequência de β graus.....	8
1.8	O sinal de RMN e a sua recepção.....	9
1.9	A relaxação do sistema de spins.....	10
1.10	A relaxação transversal.....	11
1.11	O campo magnético não homogêneo.....	12
1.12	A relaxação longitudinal.....	14
1.13	O Espectrômetro de RMN.....	15

Capítulo II - O Receptor de RF

2.0	Introdução.....	17
2.1	O sistema heterodino.....	18
2.2	Deteção em fase e quadratura.....	20

Capítulo III - O transmissor de RF

3.0	Introdução.....	22
3.1	Gerador da frequência intermediária.....	23
3.2	O controlador de pulsos.....	24

3.3 A modulação SSB.....	26
--------------------------	----

Capítulo IV - O amplificador engatilhado e o detector de "duty cycle"

4.0 Introdução.....	27
4.1 Os sinais de entrada do amplificador engatilhado.....	27
4.2 Detector de "duty cycle".....	28

Capítulo V - O circuito de RF

5.0 Introdução.....	33
5.1 O circuito ressonante.....	34
5.2 O cabo $\lambda/4$ e os diodos cruzados.....	35

Capítulo VI - O Gerador de Pulsos

6.0 Introdução.....	36
6.1 A sequência de eventos.....	36
6.2 O intervalo de tempo.....	38
6.3 Instrução para o contador de programa.....	39
6.4 "Loop" e "Branch Address".....	41
6.5 Contadores de "Loop".....	41
6.6 "Trigger" externo.....	42
6.7 A sequência de eventos em hexadecimal.....	42
6.8 Transferência de dados.....	43
6.8 Instruções para o Pulskit.....	45

Capítulo VII - A interface para o Pulskit

7.0 Introdução.....	46
7.1 A interface paralela 8255.....	46
7.2 Circuito decodificador e "buffers".....	49
7.3 Programação da interface.....	51

Capítulo VIII - O Digitalizador de sinais

8.0 Introdução.....	52
8.1 A interface GPIB.....	52
8.2 O software NI-488-MS-DOS.....	53

Capítulo IX - Descrição do Programa de controle e aquisição de dados - "Pulse Sequence Program"

9.0 Introdução.....	56
9.1 Carregar ou criar uma seqüência de pulsos.....	57
9.2 Nomeação dos canais ("Line Name").....	58
9.3 Programação dos Pulsos ("Pulse").....	59
9.4 Tipos de Eventos ("Event Type").....	59
9.5 Programação de "Loops" ("Loops").....	62
9.6 A compilação da seqüência de pulsos ("Compile").....	62
9.7 A execução da seqüência de pulsos ("Execute").....	63
9.8 Seqüência para a aquisição de um Fid.....	65
9.9 O fluxograma do programa PSP.....	66
9.10 As variáveis globais do programa PSP.....	66
9.11 As funções do "software" PSP.....	68

Conclusão.....	70
----------------	----

Projeto I - Dobrador de freqüência.....	72
---	----

Projeto II - Detector de "duty cycle".....	73
--	----

Projeto III - Interface para o Pulskit.....	74
---	----

Apêndice - Programa para o controle do

Gerador de pulsos (Pulskit).....	75
----------------------------------	----

Introdução

As bases experimentais da espectroscopia por ressonância magnética foram estabelecidas há quase 40 anos por F. Block e E.M. Purcell. Desde então, ela é utilizada pelos físicos, químicos, biofísicos e bioquímicos em estudos relativos à dinâmica dos átomos e às conformações e estruturas moleculares. Hoje em dia as técnicas de RMN, estão muito avançadas, em parte devido à Tomografia por RMN, que atraiu muitos interesses, devido à sua grande utilidade na Medicina.

O espectrômetro por Ressonância Magnética Nuclear Pulsada é constituído por duas partes distintas, uma responsável pela transmissão e recepção de radiofrequência, e outra responsável pelo controle e aquisição de dados.

Inicializamos este trabalho de mestrado quando recebemos um gerador de pulsos, Pulskit da empresa Tecmag, e um digitalizador, Nicolet 430. Aparelhos responsáveis pela parte de controle e aquisição de dados, ambos programáveis, que necessitavam de um interfaceamento com um microcomputador, a fim de serem automatizados.

O Pulskit necessita de uma interface própria, já o Nicolet pode ser programado via interface RS232 ou GPIB.

O nosso trabalho foi desenvolvido nas seguintes partes:

- (a) - Projeto e construção de um dobrador de frequências, necessário pois a temporização do gerador de pulsos é de 10 MHz e a saída do gerador de frequências existente é de 5 MHz.
- (b) - Projeto e construção de uma interface paralela, para possibilitar a comunicação entre o microcomputador e o gerador de pulsos .
- (c) - Elaboração de um software de controle do espectrômetro, para

o controle, através do microcomputador, de todos os eventos envolvidos em um experimento de RMN. Por exemplo, pode-se citar a conformação dos pulsos de RF, ou mesmo a aquisição dos dados processados pelo digitalizador. Este software deve programar o gerador de pulsos, através da interface acima mencionada e programar o digitalizador, através da interface GPIB. A manipulação deste software pelo usuário, pode ser feita via teclado ou mouse, e é extremamente simples, por ser constituído por imagens gráficas que representam bem os tipos de eventos.

(d) - Projeto e construção de um detector de "duty cycle", circuito que não permite a passagem de uma seqüência de pulsos que tenha um "duty cycle" maior ou igual a 0.05 ou tenha um pulso maior que 100pseg.

No capítulo I desta dissertação, são apresentados os princípios de RMN, o significado dos tempos de relaxação, e o espectrômetro de RMN. Nos capítulos II e III é descrita a parte responsável pela transmissão e recepção de RF. E nos seguintes é explicado o trabalho que foi desenvolvido .

I Os Princípios do Espectrômetro por RMN

1.0 Introdução

A espectroscopia por ressonância magnética nuclear (RMN) é utilizada pelos físicos, químicos, biofísicos e bioquímicos em estudos relativos à dinâmica dos átomos e às conformações e estruturas moleculares. Ela nos informa a densidade local dos núcleos atômicos pertencentes à amostra e os tempos de relaxação que caracterizam os elementos da amostra. Será visto no decorrer deste capítulo os princípios da RMN, os tempos de relaxação e um resumo do funcionamento de um espectrômetro por RMN. Estes assuntos podem ser encontrados nas referências [1,2,3,4,5], onde são abordados com mais profundidade.

1.1 O momento magnético do núcleo (spin)

O núcleo atômico gira em torno de si mesmo com um momento angular $\vec{L}$. Como ele possui cargas elétricas, sua rotação produz um momento magnético $\vec{\mu}$, que também é chamado de spin, (Fig 1).

Figura 1-Momento angular e Momento magnético

1.2 O campo magnético estático e o Teorema de Larmor

Quando este núcleo é colocado na presença de um campo magnético estático $\vec{B}_0$, ele precessiona em torno do eixo definido por $\vec{B}_0$, que definiremos como eixo z, (Fig 2). A frequência desta precessão é diretamente proporcional à este campo (Teorema de Larmor). Ou seja :

$$\vec{\omega}_0 = \gamma * \vec{B}_0 \quad \langle 1 \rangle$$

onde γ é uma constante, denominada de fator giromagnético, cujo valor difere não só para cada elemento químico, como também para cada isótopo de um mesmo elemento.

Figura 2 - Precessão do spin em torno de $\vec{B}_0$

1.3 A magnetização do sistema e o seu equilíbrio termodinâmico

A fim de atingir o equilíbrio termodinâmico, o spin orientar-se-á preferencialmente ao longo do campo $\vec{B}_0$, ficando com o momento magnético $\vec{\mu}$ paralelo ao campo $\vec{B}_0$, (Fig 3). Teremos então dois movimentos simultâneos, o de precessão em torno de $\vec{B}_0$ e o de alinhamento a $\vec{B}_0$.

Figura 3 - Spin em equilíbrio termodinâmico

Uma amostra é composta por milhares de núcleos e consequentemente por milhares de spins. Quando não há campo magnético sobre esta amostra, seus spins estão todos desorientados, mas ao aplicarmos um campo magnético $\vec{B}_0$ neste sistema de spins, todos os núcleos precessionarão em torno dele, até atingirem a posição de equilíbrio.

Chamamos de magnetização do sistema $\vec{M}$, à soma vetorial de todos os momentos magnéticos (spins). No estado de equilíbrio ($\vec{M}_0$) estará ao longo do eixo z.

1.4 O campo magnético girante e a retirada do sistema do equilíbrio

Para removermos os spins, presentes na amostra, da condição de equilíbrio, aplica-se durante um curto intervalo de tempo, um segundo campo magnético $\vec{B}_1$ no plano transversal a $\vec{B}_0$, plano xy. Este campo é do tipo girante, (Fig 4), e sua orientação muda com uma frequência w_g , sendo o sentido de rotação o mesmo da precessão dos spins em torno de $\vec{B}_0$, sendo dado por :

$$\vec{B}_1 = A * \cos (w_g * t) \vec{i} + A * \sin (w_g * t) \vec{j} \quad <2>$$

Figura 4 - Campo Girante

1.5 A frequência de ressonância do sistema

Mudando-se o sistema de referência, para um sistema que gire com frequência w_g , (Fig 5), o campo $\vec{B}_1$ neste sistema ficará parado e o campo resultante na direção do eixo z será dado por:

$$B_z = B_0 - w_g/\gamma \quad \langle 3 \rangle$$

A frequência w_g que anula o campo B_z é chamada de frequência de ressonância do sistema, que é igual a frequência de Larmor w_c .

Figura 5 - Sistema Referencial Girante

Usando-se então um referencial que gira na frequência de ressonância, tem-se apenas o campo constante $\vec{B}_1$ agindo sobre a amostra. Desta maneira, os spins precessionarão em torno de $\vec{B}_1$, (Fig 6).

Para que eles precessionem um determinado ângulo β , devemos aplicar este campo por um determinado tempo T , que é dado por:

$$T = \beta / (\gamma * B_1) \quad \langle 4 \rangle$$

Figura 6 - Precessão do spin em torno de B_1

1.6 O campo magnético linearmente polarizado

Na prática este campo $\vec{B}_1$ é produzido por uma bobina alimentada por um gerador de frequências, (Fig 7). O campo gerado por esta bobina é linearmente polarizado, e do tipo :

$$\vec{B}_1 = A * \cos(W_g * t) \vec{i} \quad W_g = 2 * \pi * F_g \quad \langle 5 \rangle$$

onde F_g é a frequência do gerador.

Este campo produzido pela bobina, pode ser visto como composto por duas componentes circularmente polarizadas, agindo no plano transversal xy :

$$\vec{B}_1 = a.\cos(W_g.t)\vec{i} + a.\sin(W_g.t)\vec{j} + a.\cos(-W_g.t)\vec{i} + a.\sin(-W_g.t)\vec{j} \quad \langle 6 \rangle$$

onde $a = A/2$.

Figura 7 - Bobina e gerador de radio frequência

Teremos uma componente que gira no mesmo sentido de precessão dos spins em torno do campo $\vec{B}_0$, e outra no sentido contrário. Observando a equação 3, percebe-se que a componente que gira ao contrário nunca anulará o campo resultante $\vec{B}_r$, ou seja, não provocará ressonância, podendo então ser desprezada. A componente que gira ao contrário só não será desprezada se o campo $\vec{B}_0$ for invertido e desta maneira a componente desprezada será a positiva. Concluimos então que o campo linearmente polarizado equivale ao campo girante.

1.7 O pulso de radiofrequência de β graus .

Para a maioria dos núcleos e para campos magnéticos $\vec{B}_0$ acima de mil Gauss, a frequência de Larmor ω_0 fica na faixa de radiofrequência (RF). Como a frequência na qual o campo $\vec{B}_1$ gira no plano transversal, tem que ser igual à frequência de Larmor, para produzir ressonância e retirar os spins da condição de equilíbrio, concluimos então que esta frequência será da ordem de $M\text{H}_z$ (RF).

Vimos que, para retirarmos os spins da condição de equilíbrio e os precessionarmos β graus em torno do campo $\vec{B}_0$, devemos aplicar este mesmo campo sobre a amostra, por um determinado tempo T , dado pela equação 4. Para isso alimentamos a bobina, durante o tempo T , pelo gerador de RF, ou seja, aplicamos um pulso de RF de duração T , (Fig 8). Quando este pulso desloca a magnetização de β graus em relação ao eixo z , ele é chamado de pulso de β graus.

Figura 8 - Pulso β de duração T

1.8 O sinal de RMN e a sua recepção

Supondo que aplicamos na bobina um pulso de RF de 90 graus ($\pi/2$), a magnetização resultante precessionará em torno do campo linearmente polarizado $\vec{B}_1$ até o plano transversal xy . Após o pulso, não havendo mais o campo $\vec{B}_1$, a magnetização precessionará em torno do campo estático $\vec{B}_0$ com a frequência de Larmor ω_0 , até voltar ao estado de equilíbrio $\vec{M}_0$.

Para detectarmos o sinal produzido pela magnetização em estado de não equilíbrio, utilizamos a mesma bobina que foi utilizada para excitar a amostra. A variação de fluxo magnético devido a precessão dos núcleos em torno do eixo z , induz nesta bobina, uma voltagem com a frequência de Larmor ω_0 , cuja amplitude é

proporcional à magnetização transversal M_{xy} , segundo a lei de Faraday.

1.9 A relaxação do sistema de spins

A relaxação ocorre logo após o pulso de RF de β graus. Tal pulso retira a magnetização do equilíbrio ao longo do eixo z, paralela ao campo estático $\vec{B}_0$, e a leva para a condição de não equilíbrio, com uma precessão de β graus, em torno do campo gerado pela bobina $\vec{B}_1$. Observando a figura 9, percebe-se que nesta condição, teremos uma componente da magnetização $\vec{M}$ no plano transversal xy, chamada de componente transversal ($\vec{M}_{xy}$) e uma componente no eixo z, chamada de componente longitudinal ($\vec{M}_z$). O retorno ao equilíbrio provoca a eliminação da componente transversal, que é chamada de relaxação transversal e a volta da componente longitudinal, chamada de relaxação longitudinal. Pela lei de Faraday, a componente longitudinal $\vec{M}_z$ e a componente no eixo y ($\vec{M}_y$) da componente transversal $\vec{M}_{xy}$, não induzem corrente na bobina, portanto apenas a componente no eixo x ($\vec{M}_x$) da componente transversal $\vec{M}_{xy}$, induz corrente.

Figura 9 - A magnetização transversal e longitudinal

1.10 A relaxação transversal

Como já vimos, uma amostra é composta por um grande número de núcleos (10^{23}). Ao ser colocada na presença de um campo magnético estático $\vec{B}_0$, cada núcleo deverá sentir um campo magnético ligeiramente diferente dos outros, devido ao campo magnético aleatório gerado pelos núcleos vizinhos. Isto produz uma não homogeneidade intrínseca do sistema devido à interação spin-spin, fazendo com que cada núcleo precesse com uma frequência ligeiramente diferente de seus vizinhos.

Após o pulso de RF (usaremos como exemplo o pulso de $\pi/2$) todos os spins estarão no plano transversal com a mesma orientação. Como eles precessionam com frequências ligeiramente diferentes, logo eles começam a se separar, (Fig 10). Com esta separação, a magnetização $\vec{M}_{xy}$ resultante diminui. Após um certo tempo característico T_2 , os spins ter-se-ão espalhado totalmente no plano transversal e a magnetização $\vec{M}_{xy}$ será nula.

Figura 10 - Relaxação Transversal

Observando a magnetização resultante no plano transversal $\vec{M}_{xy}$ e sabendo que apenas a componente no eixo x, a magnetização $\vec{M}_x$, induz corrente na bobina, concluímos que o sinal de RMN decai, na maioria dos casos, de forma exponencial :

$$S(t) = M_0 * \exp(-t/T2) * \cos(W_0.t) \quad \langle 7 \rangle$$

onde M_0 é a magnetização de equilíbrio e $T2$ é o tempo característico do decaimento da magnetização transversal $\vec{M}_{xy}$ e é denominado de tempo de relaxação transversal.

Este decaimento $T2$ é intrínseco do sistema de spins nucleares e o sinal de RMN, que surge na bobina, é denominado de FID (free induction decay), decaimento livre de indução, (Fig 11).

Figura 11 - Sinal FID

1.11 Campo Magnético não homogêneo

Na maioria das vezes o decaimento livre de indução ocorre num tempo ainda menor que $T2$, chamado de $T2^*$, pois geralmente o campo magnético estático $\vec{B}_0$ não é homogêneo, ou seja, o campo possui um valor diferente para cada posição da amostra, provocando precessões com frequências diferentes. A inhomogeneidade do campo estático é,

na maioria das vezes, maior que a inomogeneidade intrínseca do sistema devido a interação spin-spin. Concluimos então, que geralmente T_2^* é menor que T_2 .

O sinal de RMN resultante decai da mesma forma que o mostrado na equação 6, onde T_2 será agora T_2^* .

Podemos retirar o efeito da inomogeneidade do campo estático sobre o sistema, através da técnica spin-eco. Esta técnica consiste em aplicar um pulso de π assim que os spins estiverem completamente defasados, ou seja, quando não houver mais magnetização resultante $\vec{M}_{xy}$. Este pulso provoca uma rotação de 180 graus dos spins em torno do eixo x, e consequentemente um refasamento e a regeneração do sinal, (Fig 12). Tal técnica entretanto, não retira o efeito da inomogeneidade intrínseca do sistema devido a interação spin-spin.

Figura 12 - Técnica de spin-eco

O efeito desta técnica pode ser explicado em analogia à uma corrida de fundo, onde os atletas mais rápidos tomam a dianteira e o conjunto de participantes vai se espalhando ao longo do trajeto, de acordo com o ritmo de cada um; se ao toque de um sinal, os corredores dão meia volta e começam a refazer o percurso (cada qual com a velocidade que desenvolveu até então), todos chegarão juntos ao mesmo ponto de partida.

A seqüência de pulsos da técnica spin-eco é mostrada na figura 13. A amplitude destes ecos decai exponencialmente ao variarmos T , com o tempo característico T_2 intrínstico do sistema.

Figura 13 - Seqüência Spin-Eco

1.12 A relaxação longitudinal

O retorno ao equilíbrio termodinâmico faz a componente longitudinal $\vec{M}_z$ voltar ao valor inicial $\vec{M}_0$, tal volta provoca uma liberação de energia dos spins para a rede. Este retorno também é, na maioria dos casos, de forma exponencial:

$$M_z(t) = M_0 * \exp(-t/T_1)$$

<8>

onde T_1 é o tempo de relaxação longitudinal, também chamado de tempo de relaxação spin-rede.

1.13 O Espectrômetro de RMN

No espectrômetro de RMN , (Fig 14), existe um gerador de radiofrequência altamente estável, um magneto que gera o campo magnético estático; um gerador de pulsos, que é programado por um computador, onde são conformados os diferentes intervalos de tempo requeridos em um experimento de RMN ; e um transmissor que produz os pulsos de RF, chaveando o sinal do gerador de radiofrequência em intervalos de tempo determinados pelo gerador de pulsos.

Depois de amplificados, tais pulsos são aplicados à bobina encarregada de excitar os núcleos presentes na amostra, através do campo magnético linearmente polarizado por ela produzido. A volta dos spins à condição de equilíbrio, induz nesta mesma bobina, sinais fracos de RMN. Estes por sua vez, são amplificados e detectados por um receptor. O sinal resultante, é enviado a um digitalizador, cuja função é armazená-lo digitalmente e posteriormente processá-lo.

Os sinais digitalizados e processados são transferidos para um computador, que também irá processá-los, mostrando os resultados no vídeo. Todo o sistema de geração e aquisição de dados é controlado por este computador, que programa o gerador de pulsos com os intervalos de tempos requeridos para as seqüências específicas de medidas.

Maiores detalhes de cada componente do espectrômetro serão vistos no decorrer dos outros capítulos.

Figura 14 - Espectrômetro de RMN

II O receptor de RF

2.0 Introdução

Os spins excitados por um pulso de RF induzem um sinal na bobina, cuja frequência é igual à frequência de ressonância dos mesmos. A amplitude deste sinal é muito baixa, necessitando de amplificação. Como cada elemento químico possui uma frequência de ressonância, são analisados sinais nas mais variadas frequências.

A amplificação é feita inicialmente por um pré amplificador e depois pelo receptor, que retira a radiofrequência do sinal, detectando a informação contida na envolvente. O nosso receptor, (Fig 15), pertence ao conjunto NMRkit^[6], produzido pela empresa Tecmag.

Figura 15 - O receptor de RF

Para aumentar a relação sinal/ruído na detecção, os últimos estágios de amplificação de RF são sintonizados. Isto é uma dificuldade quando se necessita operar o espectrômetro em frequência variável. Para contornar este problema utilizamos o sistema heterodino explicado no próximo item.

2.1 O sistema heterodino

Na ressonância, o sinal emitido pelo circuito que contém a amostra, após a aplicação de um pulso $\pi/2$, é dado por :

$$S(t) = A * \exp(-t/T_2) * \cos(2*\pi*f_0*t)$$

Ou seja, os spins respondem na mesma frequência f_0 de excitação, produzindo um transiente com certo tempo característico T_2 .

A frequência f_0 não é igual à frequência do gerador f_g . Estas duas frequências, f_0 e f_g , sempre diferem entre si por uma quantidade fixa f_i , chamada de frequência intermediária :

$$f_0 = f_g - f_i \quad <9>$$

Se o campo estático $\vec{B}_0$ não obedece exatamente a condição de Larmor, os spins excitados por um pulso na frequência f_0 responderão em uma frequência ligeiramente diferente de f_0 . Em outras palavras, se $\vec{B}_0$ é campo de ressonância na frequência f_0 e $\vec{B}_0 + \delta\vec{B}_0$ é o campo aplicado então, os spins responderão em uma frequência $f_0 + \delta f$, onde $\delta f = \gamma * \delta B_0$. Neste caso, o sinal emitido pelo circuito que contém a amostra é dado por:

$$S(t) = A * \exp(-t/T_2) * \cos(2*\pi*(f_0 + \delta f)*t) \quad <10>$$

Este sinal, após pré amplificado é enviado ao receptor, onde também será novamente amplificado e decomposto em duas fases ortogonais 0° e 90° .

O pré-amplificador utilizado^[7] (não mostrado na figura 15) é fabricado pela empresa Doty Scientific, modelo LN-2 e possui as seguintes características:

- Ganho de 31 dB
- Figura de ruído de 1.0 dB
- Tempo de recuperação (tempo morto) de 4 μ seg.

O transiente observado no sinal é em geral muito lento comparado com f_0 ; normalmente nas aplicações de interesse T2 é muito maior que 1 μ seg. A envolvente do sinal emitido pela amostra é quem traz a informação sobre o tempo de relaxação dos spins da amostra. Por isso, o receptor heterodino possui vários misturadores que permitem remover a portadora do sinal deixando os últimos estágios de amplificação (de audio) apenas o sinal da envolvente. Este método permite que parte do ganho de RF seja fornecido por amplificadores sintonizados na frequência f_i , permitindo que a amostra seja excitada em uma frequência qualquer de livre escolha do usuário.

A multiplicação de dois sinais senoidais resulta num sinal composto por outras duas frequências distintas, sendo que uma terá frequência igual à diferença das frequências dos sinais originais e outra terá frequência igual à soma.^[8,9]

$$\cos(W_a.t) * \cos(W_b.t) = (\cos(W_a + W_b) + \cos(W_a - W_b)) / 2 \quad <11>$$

No domínio da frequência podemos representar esta expressão por :

$$f_a * f_b = (f_a + f_b) + (f_a - f_b) \quad <12>$$

No nosso caso, o sinal já amplificado pelo 1º estágio do receptor (denotado na figura 15 pelo amplificador de RF), é multiplicado pelo sinal estacionário de frequência f_g resultando num sinal de frequência $f_i + \delta f$, que será novamente amplificado por um amplificador sintonizado na frequência intermediária f_i , (denotado na figura 15 por FI).

No domínio da frequência o que ocorre é o seguinte:

$$(f_c + \delta f) * f_i = (f_i - f_i + \delta f) * f_i = (f_i - \delta f) + (2f_i - f_i + \delta f) \quad <13>$$

O último termo desta equação, de frequência alta é filtrado eletronicamente.

Desta maneira, todo o sinal emitido pela amostra terá sua frequência convertida na frequência $f_i - \delta f$, possibilitando o uso de um amplificador centrado em f_i com largura de banda maior que δf . Este sistema é conhecido como sistema heterodino^[8,9].

2.2 Detecção em fase e quadratura

Uma vez amplificado, este sinal de frequência $f_i - \delta f$ é misturado com a referência f_i . Filtrando os sinais de alta frequência o resultado obtido é o seguinte^[3] :

$$F(t) = A \cdot \exp(-t/t_2) \cdot \cos(2\pi \cdot (f_i + \delta f) \cdot t) * \cos(2\pi \cdot f_i \cdot t + \Theta) = \\ [(A/2) \cdot \exp(-t/T_2) \cdot \cos(2\pi \cdot \delta f \cdot t + \Theta)] \quad <14>$$

O ângulo Θ representa a defasagem entre o sinal de referência f_i e o sinal proveniente da amostra.

Observe que se a defasagem Θ for de 90° ou 270° não haverá o sinal de fase. A fim de resolver este problema, multiplicamos também o sinal de frequência f_i por uma portadora de mesma frequência, mas defasado de 90° além da defasagem Θ . Teremos assim, outro sinal, chamado de sinal de quadratura :

$$G(t) = A \cdot \exp(-t/t_2) \cdot \cos(2\pi \cdot f_i) * \cos(2\pi \cdot f_i + \Theta + 90^\circ) = \\ (A/2) \cdot \exp(-t/T_2) \cdot \cos(\Theta + 90^\circ) \quad <15>$$

Portanto se a defasagem for de 90° ou 270° , o sinal de fase será zero, mas em compensação o sinal de quadratura será máximo. Estes sinais são depois digitalizados e armazenados para uma futura análise e processamento.

No caso em que $\theta=0$ ou $\pi/2$ obtemos:

$$S(t) = (A/2) * \exp(-t/T2) * \cos(2.\pi.\delta f.t) \quad (\theta=0) \quad <16>$$

$$S(t) = (A/2) * \exp(-t/T2) * \sin(2.\pi.\delta f.t) \quad (\theta=0) \quad <17>$$

Estes são denominados de sinais em fase e quadratura, (Fig 16).

O nosso receptor detecta estes dois sinais simultaneamente e os amplifica, através de 2 amplificadores de audio seguidos de um filtro passa-baixa. Após amplificados os sinais de audio são enviados ao digitalizador.

Figura 16 - Detecção em fase e quadratura

III O transmissor de RF

3.0 Introdução

Para retirarmos os spins da condição de equilíbrio, devemos aplicar sobre a bobina, dentro da qual está colocada a amostra, um sinal de RF de alta potência com a frequência f_0 , igual à frequência de ressonância do elemento analisado, por um determinado tempo.

Este pulso de RF, que tem a duração determinada pelo gerador de pulsos, é produzido por um transmissor, (Fig 17), e depois amplificado por um amplificador engatilhado.

Figura 17 - O transmissor e o amplificador engatilhado de RF

O transmissor^[8] que no nosso espectrômetro também pertence ao conjunto NMRkit é composto por três módulos:

- Gerador da frequência intermediária
- Controlador de pulsos
- Modulador SSB

3.1 Gerador da frequência intermediária

A frequência intermediária produzida possui o valor de 11,25 MHz. Ela é derivada da frequência de 10 MHz, que é muito estável, e é produzida pelo nosso gerador de RF^[10], 8640B Signal Generator, da empresa Hewlett-Packard. Na realidade, a base de tempo deste gerador é de frequência 5 MHz, que acoplada à um dobrador de frequência, que foi projetado e construído por nós (ver projeto 1), gera a base de tempo de 10 MHz para o NMRkit e PULSkit.

Este dobrador é composto principalmente por dois circuitos integrados 74ls86^[24], possuindo cada um, 4 portas lógicas do tipo ou-exclusivo. As sete portas em cascata provocam um atraso do sinal de 5 MHz e a porta 2d fará a lógica ou-exclusivo do sinal de 5 MHz com o sinal de 5 MHz atrasado, produzindo um sinal de 10 MHz.^[11,12]

No gerador de FI o sinal de 10 MHz produzido é aplicado à dois dobradores de frequência, resultando num sinal de 40 MHz. Ele também é aplicado num flip-flop^[11,12], que divide a frequência por dois, resultando num sinal de 5MHz. Estes dois sinais são misturados (multiplicados)^[8,9], e o sinal produzido será composto por uma componente de 45 MHz e por uma de 35 MHz, que será eliminada por um filtro.

Dois flip-flops em série dividirão a frequência do sinal de 45 MHz por quatro, resultando em quatro sinais de 11.25 MHz, 90° defasados entre si, (Fig 18).

Figura 18 - Fases de 11.25 MHz

3.2 O controlador de pulsos

Com o gerador de pulsos, que vai ser visto com mais detalhe no próximo capítulo, programamos seqüências de pulsos, à nível TTL^[11,12], em 75 canais independentes. Destes, quatro são enviados ao sistema NMRkit. São os canais, denominados pela empresa Tecmag : PH0, PH1, SP0 e SP5.

Os pulsos que são fornecidos pelos canais PH0 e PH1 entram no controlador de pulsos, o qual determina a fase da frequência intermediária, de acordo com a tabela 1.

PH1	PH0	FASE
0	0	0°
0	1	90°
1	0	180°
1	1	270°

Tabela 1 - Fases de f_i

Os pulsos fornecidos pelos canais SP0 e SP5 determinam, através do mesmo controlador de pulsos, o pulso TX e o pulso RX.

TX é o pulso que estabelece a duração do pulso de RF e RX é o pulso que bloqueia o receptor durante a transmissão do pulso de RF. Este bloqueio é necessário para evitar que o pulso de RF amplificado sature ou provoque danos no receptor.

No diagrama de tempo, mostrado na figura 19, podemos ver a formação destes pulsos. A subida de SP0 bloqueia o receptor através do pulso RX e dispara um pulso (chamado "PRED"), cuja duração é ajustável (geralmente 10 μ seg). O pulso TX só será disparado com o término de "PRED" e ficará ativo até SP0 descer, desta forma o pulso de RF só é liberado após o bloqueio do receptor.

O pulso SP5 é programado para ser ativado na descida de SP0 e sua duração, que é em geral alguns microsegundos, determinará por quanto tempo ainda o receptor ficará bloqueado, mesmo após o término do pulso de RF. Isto é necessário pois, após a aplicação do pulso de RF ainda haverá corrente circulando na bobina, que vai sendo amortecida com o decorrer do tempo ("ringing"), e que pode também saturar o receptor.

Figura 19 - Diagrama de Tempo

3.3 A modulação SSB

Para que a frequência do sinal emitido pela amostra f_0 seja igual à diferença entre f_g e f_i , ver equação 1, devemos aplicar à bobina um pulso de RF de mesma frequência.

O pulso com esta frequência é criado no modulador SSB^[3,8,9] ("Single Side Band") através da multiplicação de f_g pela frequência intermediária f_i :

$$f_0 = f_g * f_i = (f_g - f_i) + (f_g + f_i) \quad <18>$$

A portadora de frequência $(f_g + f_i)$ é eliminada por um filtro.

O pulso TX entrará no modulador determinando a duração do pulso com esta frequência f_0 .

Portanto no nosso sistema devemos ajustar no gerador de frequência, uma frequência igual à soma da frequência de ressonância do elemento químico em análise com a frequência intermediária (11.25 MHz) :

$$f_g = f_0 + f_i \quad <19>$$

IV O amplificador engatilhado e o detector de "duty cycle"

4.0 Introdução

O amplificador de potência para o pulso de RF, por nós utilizado, é um amplificador engatilhado, fabricado pela empresa Matec, modelo 515 A^[13]. Ele possui as seguintes características:

- Nível de entrada ≈ 0 dBm
- Potência máxima de 1Kw para cargas de 50Ω
- Faixa de frequência de 0.5 a 150 MHz
- Menor pulso de entrada : 1 μ seg
- Maior pulso de entrada : 100 μ seg (quando no modo "High Power")
- "Duty cycle" máximo : 0.005

Como o amplificador não suporta um pulso maior que 100 μ seg e um "duty cycle" maior que 0.005, contruímos um aparelho que detecta estas características da seqüência de pulsos emitida, bloqueando sua entrada no amplificador, caso elas atinjam o limite. Este aparelho pode ser visto no projeto 2.

4.1 Os sinais de entrada do amplificador engatilhado

O amplificador engatilhado, (Fig 17), amplifica um sinal de RF, de baixo nível (0 dBm), apenas no período de tempo, determinado por um pulso de entrada, à nível TTL.^[11,12] Este pulso deve ser disparado no mínimo 10 μ seg após a entrada do sinal de RF de baixo nível.

Um dos canais do gerador de pulsos (CP00) fornece o pulso TTL para o gatilho do amplificador, enquanto que o sinal de RF de baixo nível é fornecido pelo modulador SSB, (Fig 20).

Figura 20 - Sinais de entrada e saída do amplificador engatilhado (Matec)

4.2 Detector de "duty cycle"

Considere o circuito RC mostrado na figura 21, onde são aplicados os pulsos periódicos de largura Δ e separação δ ($\Delta \ll \delta$). O diodo D faz com que o capacitor se carregue durante o tempo Δ através do resistor r e se descarregue através de R . As constante de tempo de carga e descarga valem respectivamente $T_c = r.C$ e $T_d = R.C$. [14,15]

Figura 21 - Sequência de pulsos carrega e descarrega o capacitor

A tensão de carga do capacitor, após $n+1$ pulsos, é dada por:
(ver figura 22)

$$V' = V_{CC} - (V_{CC} - V_n) \cdot e^{-\frac{\Delta}{\tau_c}} \quad \langle 20 \rangle$$

E a tensão de descarga do capacitor, é dada por:

$$V_{n+1} = V' \cdot e^{-\frac{\delta}{\tau_d}} \quad \langle 21 \rangle$$

Figura 22 - Tensão no capacitor após $n+1$ pulsos

Substituindo a equação 20 na equação 21 obteremos :

$$V_{n+1} = V_{CC} \cdot e^{-\frac{\delta}{\tau_d}} (1 - e^{-\frac{\Delta}{\tau_c}}) + V_n \cdot e^{-\left(\frac{\Delta}{\tau_c} + \frac{\delta}{\tau_d}\right)} \quad \langle 22 \rangle$$

Que pode ser expresso por :

$$V_{n+1} = K + V_n \cdot Q \quad \langle 23 \rangle$$

onde K e Q são constantes dadas por:

$$K = V_{CC} \cdot e^{-\frac{\delta}{\tau_d}} \cdot (1 - e^{-\frac{\Delta}{\tau_c}}) \quad \langle 24 \rangle$$

e

$$Q = e^{-\left(\frac{\Delta}{\tau_c} + \frac{\delta}{\tau_d}\right)} \quad \langle 25 \rangle$$

Supondo que o capacitor no início estava descarregado teremos:

$$V_0 = 0$$

e pela equação 23 :

$$V_1 = K + V_0 \cdot Q = K$$

$$V_2 = K + V_1 \cdot Q = K + K \cdot Q + K(1+Q)$$

$$V_3 = K + V_2 \cdot Q = K + K(1+Q)Q = K(1+Q+Q^2)$$

$$V_n = K(1+Q+Q^2+Q^3+\dots+Q^{n-1}) \text{ que é equivalente a :}$$

$$V_n = K \cdot \sum_{m=0}^{n-1} Q^m \quad \langle 26 \rangle$$

Para n tendendo ao infinito teremos :

$$V_{\infty} = K \cdot \sum_{m=0}^{\infty} Q^m = K \cdot \frac{1}{1-Q} \quad \langle 27 \rangle$$

E agora substituindo Q e K, obtemos o valor assintótico da tensão no capacitor após a aplicação de um número muito grande de pulsos:

$$V_{\infty} = V_{CC} \cdot \frac{e^{-\frac{\delta}{Td}}(1 - e^{-\frac{\Delta}{Tc}})}{1 - e^{-\left(\frac{\Delta}{Tc} + \frac{\delta}{Td}\right)}} \quad \langle 28 \rangle$$

Utilizaremos no circuito valores tais que :

$$r.C \gg \Delta \text{ e } R.C \gg \delta$$

E sabendo que, $e^{-x} \approx 1 - x$, para $x \ll 1$, usaremos esta propriedade na equação acima, e obteremos :

$$V_{\infty} \approx V_{CC} \cdot \frac{\left(1 - \frac{\delta}{Td}\right) \frac{\Delta}{Tc}}{\frac{\Delta}{Tc} + \frac{\delta}{Td}} \approx V_{CC} \cdot \frac{\frac{\Delta}{Tc}}{\frac{\Delta}{Tc} + \frac{\delta}{Td}} \quad \langle 29 \rangle$$

Que resulta finalmente em :

$$V_{\infty} \approx V_{CC} \cdot \frac{\frac{\Delta}{\delta}}{\frac{\Delta}{\delta} + \frac{T_C}{T_D}} \quad <30>$$

O "Duty cycle" (DC) é uma relação, usada geralmente para seqüências de pulsos digitais, dada por :

$$DC = \frac{T_a}{T_a + T_b} \quad <31>$$

onde T_a é o tempo em que o sinal fica em nível alto numa seqüência e T_b é o tempo em que o sinal fica em nível baixo.

Para a seqüência exemplificada na figura 21, poderemos transformar esta relação em :

$$DC = \frac{\Delta}{\delta} \quad <32>$$

Para concluir, reescrevemos a equação 30 em termos de "duty cycle" :

$$V_{\infty} = V_{CC} \cdot \frac{DC}{DC + \frac{T_C}{T_D}} \quad <33>$$

Como conhecemos os valores de T_C e T_D , saberemos qual a tensão de saída, V_{∞} , do circuito, após vários pulsos, para um determinado "duty cycle".

Se $T_C/T_D \gg DC$, V_{∞} será aproximadamente proporcional ao "duty cycle" DC , isto é :

$$V_{\infty} = V_{CC} \cdot \frac{T_D}{T_C} \cdot DC \quad <34>$$

A composição do nosso projeto (projeto 2) satisfaz esta aproximação.

A tensão no capacitor é amplificada [16.17] e depois comparada [16.17] com uma tensão de referência que corresponde ao valor limite do "duty cycle". Quando ela ultrapassa este limite, a passagem da seqüência de pulsos é bloqueada.

O amplificador engatilhado não pode receber pulsos maiores do que 100 μ seg. Em nosso projeto utilizamos um monoestável [11.12], que produz um pulso de 100 μ seg, sendo disparado pelo pulso de entrada. Este pulso é comparado, através de flip-flops [11.12], com o pulso de entrada, permitindo o bloqueio da seqüência de pulsos.

5.1 O circuito ressonante

Vamos primeiro considerar o comportamento do circuito tanque L-Cp^[14,15] (ver figura 23). A sua impedância, parte resistiva e reativa, estão graficadas na figura 24. A única frequência em que ele é puramente resistivo corresponde à frequência de ressonância. Entretanto, nesta frequência, a sua resistência equivalente é muito maior do que 50 Ω , o que não permite o seu acoplamento com o resto do sistema.^[14,15]

Se o circuito L-Cp operar em uma frequência f_0 (diferente da sua frequência de ressonância) na qual a sua resistência é de 50 Ω (ver figura 24), a sua parte reativa (no caso positiva) precisa ser cancelada a fim de permitir o acoplamento do circuito. Isto é feito adicionando um capacitor em serie Cs, de valor tal que cancele exatamente a componente indutiva do circuito L-Cp na frequência f_0 .

O capacitor Cp é chamado de capacitor de sintonia e Cs de capacitor de acoplamento.

Figura 24 - Impedância do circuito tanque ressonante em função da frequência

5.2 O cabo $\lambda/4$ e os diodos cruzados

O cabo $\lambda/4$, onde $\lambda = (\text{velocidade da luz})/f_c$, de impedância $Z_c = 50\Omega$, possui a seguinte propriedade:

$$Z_e * Z_s = Z_c^2 \quad \langle 35 \rangle$$

sendo que Z_e é a impedância de entrada do cabo e Z_s é a impedância de saída do cabo.^[3,9]

Através dos diodos cruzados D1 e D2^[16,17], estas impedâncias podem ser controladas dinamicamente pela seqüência de pulsos. Vejamos a seguir como isto é feito.

Quando o pulso de RF amplificado for aplicado, a impedância Z_s será igual à zero, pois o grupo de diodos D2 conduzirão, e de acordo com a equação 35, Z_e será muito grande, fazendo o pulso ir totalmente para a bobina.

Agora, quando a bobina emitir sinal, que será de baixo nível, a impedância Z_s será igual a 50Ω , pois o grupo de diodos D2 não conduzirão e segundo a equação 35, Z_e será igual a 50Ω , e o sinal será transmitido totalmente para o pré amplificador. Este sinal não irá para o amplificador engatilhado, pois é barrado pelo grupo de diodos D1.

VI O Gerador de pulsos

6.0 Introdução

O Gerador de Pulsos que é utilizado em nosso sistema é o Pulskit^[18] da empresa Tecmag, sendo este um gerador de pulsos programável designado especialmente para aplicações em espectroscopia e imagens por NMR. Com ele podemos programar, através de palavras de controle de 128 bits^[19], seqüências de pulsos com até 2048 eventos em 75 canais independentes. Seus pulsos são produzidos por componentes do tipo TTL^[11,12], possuindo amplitudes de 5 volts .

Este aparelho pode ser interfaceado com os computadores VAX-11 ou PDP-11 via interface DR-11/W, ou qualquer outro utilizando a interface apropriada. Em nosso sistema este gerador de pulsos foi interfaceado com um PC AT-386-SX^[20,21], sendo a interface projetada e construída por nós.

6.1 A seqüência de eventos

Podemos dizer que evento é um intervalo de tempo em que o canal ficará ativo ou inativo. Observando a figura 25, podemos constatar que:

Evento N00: nenhum canal ativo no intervalo de tempo de 1 seg.

Evento N01: Apenas o canal SP0 está ativo no intervalo de tempo de 10 useg.

Evento N04: Os canais SIG e ADC estão ativos no intervalo de tempo de 1 useg.

Evento Nº7: Os canais SIG, HLD e CHA estão ativos no intervalo de tempo de 24 useg. (descrevi apenas alguns dos eventos da figura)

Figura 25 - Sequência de Eventos

Cada evento corresponde à uma palavra de controle do Pulskit, sendo que esta possui 128 bits. Destes, 32 bits determinam o intervalo de tempo do evento, 4 bits contêm a instrução para o contador de programa, 5 bits determinam o contador de "loop" utilizado, 11 bits determinam o "branch adress", 1 bit é usado para habilitar ou desabilitar o "trigger" externo e os 75 restantes correspondem ao canais de saída do Pulskit. O nome e a representação destes 128 bits podem ser vistos na figura 26.

Cada um destes grupos de bits será descrito a seguir.

Figura 26 - A palavra de controle (128 bits)

6.2 Intervalo de tempo

O intervalo de tempo do evento é definido por 32 bits. Para determiná-lo, devemos primeiramente representa-lo em multiplos de 100 nseg, depois converter este múltiplo, que está em decimal, para hexadecimal e finalmente fazer o complemento dois desse número. O

número em hexadecimal resultante corresponderá aos 32 bits, que fazem parte da palavra de controle do Pulskit.

Como a resolução do intervalo de tempo de um evento é 100 nseg e apenas 32 bits determinam um intervalo, concluimos que o maior intervalo de tempo é o de 430 seg ($2^{32} = 4.294 * 10^9 * 100 \text{nseg}$).

Se necessitarmos de um intervalo de tempo maior, dois ou mais eventos podem ser concatenados.

Vejamos por exemplo um intervalo de tempo de 24 useg :

$$24 \text{ useg} = 240 * 100 \text{ nseg}$$

(converteu o número em múltiplo de 100 nseg)

$$240_{\text{dec}} = 0000 \text{ } 00\text{F0}_{\text{hexa}}$$

(converteu o número decimal para hexadecimal)

$$\text{Complemento dois } (0000 \text{ } 00\text{F0})_{\text{hexa}} = (\text{FFFF } \text{FF10})_{\text{hexa}}$$

Portanto FFFF FF10 são os 32 bits que determinam um intervalo de tempo de 24 useg.

6.3 Instrução para o contador de programa

Cada palavra de controle é guardada numa posição de memória.^[19,20] A primeira palavra é guardada na posição de memória zero, a segunda é guardada na posição de memória um e assim sucessivamente, (Fig 27). Cada palavra de controle será uma linha do programa. Notamos então que o número da posição de memória corresponde ao número do evento.

Chamamos de ponteiro^[19,22,23], uma variável que contém uma determinada posição de memória. Ao dizermos que o ponteiro aponta para o evento n, significa que esta variável contém a posição de memória n.

Figura 27 - O contador de programa

O microprocessador^[19,20] do Pulskit possui um ponteiro, chamado de contador de programa. Quando começamos a executar o programa, este ponteiro aponta para a posição de memória zero. O microprocessador lê a palavra de controle desta posição de memória e a guarda em um registrador, chamado de "pipeline".

Os 32 bits que definem o intervalo de tempo são carregados em um contador^[11,12], este é incrementado a cada pulso de "clock" e quando termina a contagem, habilita o registrador "pipeline" para receber outra palavra. O "clock" utilizado possui uma frequência de 10 MHz, que é fornecida pelo dobrador de frequência esquematizado no projeto 1.

Nesta palavra de controle, 4 bits irão definir qual a sequência do programa, ou seja, eles definem a instrução para o contador de programas. Se estes 4 bits forem $(E)_{\text{hexa}}$, o microprocessador irá incrementar o contador de programa, que irá apontar para o próximo evento. Se forem $(2)_{\text{hexa}}$ ou $(3)_{\text{hexa}}$ significa que teremos um "loop".

6.4 "Loop" e "Branch Address"

O "loop" é um desvio na seqüência de execução do programa.^[22,23] O microprocessador ao ler que existe um "loop", faz o contador de programa apontar para o evento definido por 11 bits que são chamados de "branch adress".

Podemos ter em uma seqüência de pulsos, no máximo 2048 eventos. Isto significa que o programa do Pulskit terá no máximo 2048 palavras de controle. Consegue-se representa-las com a combinação de 11 bits ($2^{11}=2048$) ou seja o "branch adress".

Se a instrução para o contador de programas for $(2)_{\text{hexa}}$, o número de vezes que o programa executará o "loop" será infinito, o contador de programa irá sempre para o evento definido pelo "branch adress". Mas se a instrução for $(3)_{\text{hexa}}$, o número de vezes que o programa executará o "loop" será determinado pelo contador de "loop".

6.5 Contadores de "Loop"

O Pulskit possui 5 contadores de "loop" independentes. A palavra de controle possui 5 bits que definirão qual dos contadores de "loop" será utilizado.

Cada contador de "loop" possui um registrador de 16 bits, onde é carregado o número de vezes que o "loop" será realizado em um determinado evento. O maior valor que o contador de "loop" pode ser carregado é 65536.

Na seqüência da figura 25, o contador de "loop" é carregado com o valor 1000, ou seja, o contador de programa voltará 1000 vezes para o evento n04 após o término do evento n07.

Pode haver apenas uma instrução de "loop" por evento, pois só podemos programar um "branch adress" por palavra de controle.

Podemos usar várias vezes o mesmo contador de "loop", em uma seqüência de pulsos, se o valor deste contador permanecer o mesmo.

O microprocessador do Pulskit não aceita "loops" entrelaçados.

6.6 "Trigger" externo

Existe um bit na palavra de controle que habilita o Pulskit a receber um "trigger" externo, sincronizando o gerador de pulsos com eventos externos.

Ao colocarmos o bit 117, da palavra de controle, em nível lógico um, o gerador realizará aquele evento, mas ficará esperando chegar o "trigger" externo para realizar o próximo.

6.7 A seqüência de eventos em hexadecimal

Para programarmos a seqüência de eventos ilustrada na figura 25, devemos transferir para o PULSkit, oito palavras de controle, representadas a seguir em hexadecimal, de acordo com o esquema da figura 26 :

0000 E000 0000 FF67 6980 0000 0000 0000	(1s)
0000 E000 0001 FFFF FF38 0000 0000 0000	(10µs)
0000 E000 0002 FFFF FF38 0000 0000 0000	(10µs)
0000 E000 0000 FFFF FECA 0000 0000 0000	(29µs)
0000 E000 2800 FFFF FFF6 0000 0000 0000	(1µs)
0000 E000 1800 FFFF FF74 0000 0000 0000	(14µs)
0000 E000 3C00 FFFF FFF6 0000 0000 0000	(1µs)
0001 3004 1C00 FFFF FF10 0000 0000 0000	(24µs)

Devemos também transferir os valores dos 5 contadores de "loop". Sendo cada contador de 16 bits, representaremos estes valores, em hexadecimal :

03E8 0000 0000 0000 0000 ($1000_{dec} = 03E8_{hex}$)

Observe que, quatro contadores não são utilizados, mas devem ser carregados por zero.

6.8 Transferência de dados

O gerador de pulsos é conectado ao microcomputador por 19 linhas, sendo que, 16 linhas são destinadas à transferência de dados e 3 são linhas de controle. Desta forma são transferidos simultaneamente para o Pulskit, 16 bits de dados e 3 bits de controle.

As combinações destes 3 bits com os 16 bits de dados, determinarão as instruções para o Pulskit. Podemos ver estas instruções na tabela 2.

Quando os 16 bits de dados e os 3 bits de controle estiverem estáveis na saída da interface, um destes três bits, o bit READ, deve ir para o nível lógico zero, por no mínimo 200 nseg. Quando isto ocorre os dados e os bits de controle são transferidos para o Pulskit, sendo que este tempo de transferência leva aproximadamente 700 nseg. Após este tempo poderemos zerar novamente o bit READ, para transferir outros dados. Estes intervalos de tempos são representados na figura 28.

NOME	BITS DE CONTROLE 2-1-0	BITS DE DADOS		FUNÇÃO
		15-12	11-00	
INPP	1 1 1	xxxx	x111 1111 1111	Inicializar o pro- gramador de pulsos
LDMD	0 0 1	1001	xxxx xxxx xxxx	Modo Carregar
LDLC	0 0 1	1000	x01 dados 08-00	Carregar contadores
LMEM	1 0 1	dados 15-00		Carregar memória
EXMD	0 0 1	1010	xxxx xxxx xxxx	Modo executar
START	0 0 1	1011	x111 0111 1111	Começa executar a seqüência de pulsos
STOP	0 0 1	1100	x111 1101 1111	Para a execução da seqüência de pulsos

Tabela 2 - Instruções para o Pulskit

Figura 28 - A temporização para a transferência de dados

6.10 Instruções para o PULSkit

O microprocessador do gerador de pulsos lê, decodifica e executa cada uma das oito diferentes instruções. Cada instrução é executada após oito ciclos de "clock".

Descreveremos a seguir as instruções que são enviadas do computador para o gerador de pulsos :

INPP : Inicializa o programador de pulso, ou seja, habilita o Pulskit para a comunicação.

LDMD : Coloca o Pulskit no modo "load". Neste modo o Pulskit fica habilitado para ter seus contadores ou memória carregados.

LDLC : Através de cinco palavras de 16 bits, que são transferidas uma a uma, carregamos os contadores de "loop".

LDMEM : A memória do Pulskit é dividida em 8 bancos de 2k por 16 bits. Após cada transferência de uma palavra de 16 bits, o banco seletor é incrementado de um. Com oito transferências teremos uma palavra de controle na memória do Pulskit e o contador de programa passa a apontar para a próxima posição de memória. Este processo é feito até que todos eventos sejam programados.

EXMD : Recebendo esta instrução, o banco seletor habilita todos os bancos de memória, formando a palavra de controle de 128 bits.

START : Começa execução de uma seqüência de eventos programada. Esta instrução sempre coloca o contador de programas apontando para o evento zero, portanto sempre que pararmos a seqüência e a reiniciarmos, ela começará novamente à partir do evento zero.

STOP : Para a execução da seqüência de eventos programada.

VII A interface para o PULSkit

7.0 Introdução

O gerador de pulsos Pulskit é normalmente interfaceado com o computador VAX-11 ou PDP-11, via interface DR11-W DMA. Para interfaceá-lo com outros computadores, são requeridas 16 linhas de dados e 3 linhas de controle.

Resolvemos inicialmente interfacear o Pulskit com um microcomputador do tipo IBM-PC-XT^[20,21], que possui 8 linhas de dados, ou seja, transfere 8 bits de cada vez. Mas agora estamos utilizando um microcomputador do tipo IBM-PC-AT-386 SX^[20,21], que possui 16 linhas de dados.

Caso este microcomputador apresente algum defeito, deve ser substituído rapidamente, pois controla todo o espectrômetro, que é utilizado por muitas pessoas. Devemos então empregar uma interface que sirva tanto para este microcomputador como para outros mais simples, portanto utilizamos a interface paralela 8255^[25,26], que recebe 8 bits de dados do microcomputador e os armazena para posterior leitura do Pulskit.

7.1 A interface paralela 8255

A interface 8255 possui 3 registradores, chamados de portas de entrada ou saída. Cada uma destas portas forma um grupo de 8 bits para a transferência de dados entre os dispositivos de entrada/saída e a interface. Esta interface está esquematizada na figura 29.

O sentido de entrada ou saída de cada um destes grupos é determinado por programação.

Figura 29 - A interface paralela 8255

Qualquer instrução que é mandada do microcomputador para o Pulskit, necessita de 3 bits de controle e 16 bits de dados. Podemos então programar a interface para que a porta A e B sejam de saída.

O microcomputador transfere, os 16 bits de dados, enviando 8 bits para a porta A e 8 bits para a porta B.

Os 8 bits da porta C são divididos em 2 grupos de 4 bits, 4 bits mais significativos e 4 bits menos significativos. O sentido de entrada ou saída de cada grupo também pode ser programado.

Como são utilizados apenas 3 bits de controle, programamos a porta C, para que os 4 bits menos significativos sejam de saída e os 4 bits mais significativos sejam de entrada. Desta forma

sobrarão ainda 4 bits de entrada e 1 bit de saída, que poderão ser utilizados para outras finalidades.

Como mandamos dados para cada porta, podemos mandar dados para o controle da interface. Esses dados são chamados de palavras de controle da interface e são armazenados em um outro registrador, que é chamado de registrador da palavra de controle.^[25,26]

São essas palavras de controle que definem como se comportará cada porta. Para o nosso uso, devemos enviar ao registrador da palavra de controle, a palavra de controle (88)_{1:18}. (ver figura 30)

Figura 30 - Fluxo de dados na 8255

O microcomputador seleciona uma das três portas ou o registrador da palavra de controle, para enviar ou receber seus dados, através das duas linhas de endereço (A_0 e A_1)^[20,21] que estão conectadas à interface. Os registradores, selecionados conforme as combinações destas duas linhas de endereço, podem ser vistas na tabela 3.

A1	A0	registrador
0	0	Porta A
0	1	Porta B
1	0	Porta C
1	1	Controle

Tabela 3 - Seleção dos registradores

As linhas de controle do 8255, RD e WR, definirão se a operação é de leitura ou escrita respectivamente, na interface.

7.2 Circuito decodificador e "buffers"

Podemos ter várias interfaces 8255 ou mesmo outras, conectadas ao microcomputador, que são ligadas em paralelo ao barramento de dados e outras linhas. Estas ficam em "tri-state" e apenas uma, de cada vez, pode ser habilitada.

Cada interface é habilitada pelo microprocessador, através das linhas de endereço, que conectadas ao circuito decodificador, determinam qual interface será utilizada. A entrada que habilita a comunicação entre o microcomputador e a determinada interface é chamada de "chip select". [19,25,26]

O circuito decodificador é utilizado para que um endereço habilite apenas uma interface. Para que elas não sejam ligadas diretamente ao barramento do microprocessador, são utilizados "buffers" [11,12], assim este fica protegido contra qualquer eventual dano no circuito das mesmas. Este circuito está representado no projeto 3 e seus circuitos integrados serão descritos a seguir.

O circuito integrado 74LS245 [24] (CI1) é um transmissor bidirecional para barramento ("tri-state"). Em outras palavras ele

permite que dados trafeguem entre o barramento e a interface 8255.

A direção do fluxo de dados do CI1 é determinado pelo nível lógico aplicado no pino "dir" do mesmo. O pino "output enable" determina quando estes dados passam pelo CI, ou seja, quando este pino está desabilitado, o CI entra no modo "tri-state".

O CI2 e CI3 são os 74LS244^[24], eles são "buffers" de 8 bits que estão sempre ativados neste circuito. Desta maneira estes sempre passam os sinais do barramento, em apenas uma direção, para o circuito da interface.

O CI6 (74LS85)^[24] é um comparador utilizado para decodificar o endereço de I/O do circuito. Este possui duas entradas de 4 bits, uma conectada às linhas de endereço e outra a níveis lógicos de tensão por nós escolhidos. Quando estas duas entradas possuem o mesmo valor, um sinal de nível lógico 1 aparece no pino 6. Também utilizamos a linha de endereço A_3 para ativar/desativar este CI, desta maneira, aumenta-se a faixa de decodificação do circuito.

As linhas de endereço A_5 , A_6 , A_7 e A_8 são comparadas respectivamente com os níveis lógicos 0, 0, 0, 1, (níveis escolhidos por nós). Com isso a faixa de endereço de I/O será entre $(300)_{\text{hexa}}$ e $(31F)_{\text{hexa}}$.

Limitamos ainda mais a faixa de endereço, utilizando o CI5 (74LS138)^[24] que decodifica a faixa de endereços gerados por A_4 , A_5 e A_6 .

Este CI decodifica 3 entradas binárias em 8 saídas independentes (de S_0 a S_7). Como utilizamos a saída S_1 , as linhas de endereço devem ser : $A_4=0$, $A_5=0$, $A_6=1$.

7.3 Programação da interface

Para que a interface utilizada para o Pulskit seja habilitada, as linhas de endereço do microcomputador devem estar nos seguintes níveis lógicos :

$A_9=1$, $A_8=1$, $A_7=0$, $A_6=0$, $A_5=0$, $A_4=0$, $A_3=0$, $A_2=1$.

Supondo que as linhas A_{11} e A_{10} estejam em nível lógico zero, a interface será habilitada pelos endereços, $(304)_{\text{hexa}}$, $(305)_{\text{hexa}}$, $(306)_{\text{hexa}}$ e $(307)_{\text{hexa}}$.

Comparando estes endereços com a tabela 3, concluimos que : $(304)_{\text{hexa}}$ habilita a porta A, $(305)_{\text{hexa}}$ habilita a porta B, $(306)_{\text{hexa}}$ habilita a porta C e $(307)_{\text{hexa}}$ habilita o registrador da palavra de controle. Como já vimos, o microprocessador define se irá enviar/receber dados para os registradores, através das linhas RD e WR.

VIII O Digitalizador de sinais

8.0 Introdução

Utilizamos em nosso sistema o digitalizador Nicolet System 400, modelo 430^[27], que nos permite digitalizar os sinais analógicos de entrada provenientes do receptor, guardar estes sinais na sua memória interna, e mostra-los em seu display. Neste display podemos medir o tempo e a voltagem do sinal, além de outras características do mesmo. O modelo 430 é caracterizado por possuir 2 canais, com conversores de 12 bits, que fazem 10 mega aquisições por segundo.

Os sinais digitalizados, que são guardados na memória, podem ser manipulados por várias funções matemáticas internas e depois são transferidos para o microcomputador, via GPIB.^[29]

Das várias funções matemáticas do Nicolet^[28] utilizamos principalmente a função SUMAVERAGE e a função FFT.

A função SUMAVERAGE realiza a média entre os sinais adquiridos e a FFT calcula a transformada de Fourier do sinal.

8.1 A interface GPIB

Para comunicarmos o microcomputador AT-386 SX com o digitalizador de sinais por nós utilizado, Nicolet, utilizamos a interface GPIB (GENERAL PURPOSE INTERFACE BUS), que é também denominada de IEEE-48. Esta interface permite a comunicação do microcomputador com até 16 instrumentos compatíveis.

A GPIB foi originalmente projetada pela HP (Hewlett-Packard), com o intuito de interfacear os computadores com os instrumentos por

ela fabricados. Devido a sua alta taxa de transferência, de até 1 megabyte/segundos, a GPIB logo ganhou popularidade, levando outras empresas a projetar instrumentos compatíveis com a mesma.

Muitas interfaces similares foram fabricadas. Nós utilizamos a placa fabricada pela National Instruments, que vem com o software NI-488 MS-DOS, que nos permite controlar facilmente os instrumentos à ela conectados.

8.2 O software NI-488 MS-DOS

Toda interface necessita de um software, que permitirá a comunicação entre o microcomputador e o aparelho à ele conectado.

Este software possui várias funções próprias, escritas para as linguagens C , BASICA, e QuickBASICA, que vêm facilitar esta comunicação.

Utilizamos as funções escritas em C^[22,23], dentre as quais, aquelas que enviam dados do microcomputador para os instrumentos e outras que recebem os dados. Descreveremos abaixo as funções por nós utilizadas.

`ibfind("string"):`

Esta função é usada para habilitar umas das placas GPIB conectadas. No nosso caso, possuímos apenas uma placa, que é denominada de GPIB0. Se tivéssemos outra placa conectada, seria denominada de GPIB1. Usamos então "GPIB0" como "string".

Escrevemos :

```
board0 = ibfind("GPIB0")
```

Desta maneira habilitamos a placa GPIB0 e a associamos à variável do tipo inteira, `board0` . Esta variável será utilizada como argumento de outras funções.

Caso a placa GPIB não possa ser habilitada, a função `ibfind` retornará um valor menor do que zero.

Esta função é também usada para habilitar um dos 16 instrumentos conectados à placa GPIB. O nosso digitalizador Nicolet corresponde ao instrumento de número 14, sendo denominado de "DEV14", portanto utilizamos "DEV14" como "string".

Como no exemplo acima, escrevemos:

```
nic = ibfind("DEV14").
```

Desta forma habilitamos o digitalizador Nicolet e o associamos à variável inteira `nic`, que é utilizada no decorrer do programa, também como argumento de outras funções.

`ibwrt(nic,mnm,nº):`

Esta função é utilizada para enviar dados para o Nicolet.

Argumentos:

`nic` é a variável associada ao Nicolet.

`mnm` é o mneumônico que será enviado para o Nicolet. Cada mneumônico representa uma instrução do Nicolet; para maiores detalhes, ver Manual do Nicolet, "Remote Command Sets".

`nº` é o número de caracteres do mneumônico.

`ibrd(nic,str,nº):`

Esta função é utilizada para receber dados do Nicolet.

Argumentos:

`nic` é a variável associada ao Nicolet.

`str` é uma variável do tipo "string", que receberá dados do Nicolet.

`nº` é a dimensão da "string".

Vejamos um exemplo que utiliza estas funções de escrita e leitura:

Perguntamos a data para o Nicolet através de, `ibwrt(nic,":system:date?",13)`. E para recebermos a resposta, `ibrd(nic,date,10)`. Esta função irá atribuir a data à variável "date".

`ibsre(board0,0):`

Função usada para desabilitar a GPIB0.

Argumentos:

`board0` é a variável associada a placa conectada.

`0` desabilita a placa, qualquer outro número habilita a placa.

`ibcnt`: É uma variável interna da placa GPIB, que nos informa quantos bytes foram transferidos na ultima operação de leitura ou escrita.

IX Descrição do Programa de Controle e Aquisição de Dados - "Pulse Sequence Program"

9.0 Introdução

Como seria muito desgastante, demorado e tedioso para o usuário do espectrômetro, que fosse programar uma seqüência de pulsos, ter que manipular cada um dos 128 bits de uma palavra de controle, a fim de especificar as características de um evento, resolvemos criar um "software" onde o usuário pudesse programar a seqüência de pulsos, com maior facilidade e sem se preocupar com as palavras de controle, deixando ao próprio "software" a interpretação da seqüência, transformando seus eventos nas palavras de controle correspondentes.

Este programa foi denominado de "Pulse Sequence Program" (PSP), que além de programar o Pulskit, também programa o digitalizador Nicolet, adquire seus dados e os grafica no vídeo.

Este programa pode ser manipulado tanto pelo teclado como por um mouse.

O "software" PSP possui um "Help" que pode ser acessado pela tecla F1, ele além de descrever o programa, também menciona os arquivos de suporte utilizados. Possui além disso, o recurso (acionado pela tecla F3) que transforma a tela colorida numa tela em branco e preto, pois assim o usuário pode imprimir as seqüências através da tecla "Print Scrn". Para que o vídeo volte a ficar colorido deve-se digitar a tecla F4.

Através da figura 31, podemos ver como ele é apresentado ao usuário.

Figura 31 - Programa PSP

9.1 Carregar ou criar uma seqüência de Pulsos

Com o PSP podemos criar uma nova seqüência ("Open a new file"), ou carregar alguma já existente ("Open an existing file"), ver a figura 31, que poderá estar não necessariamente no sub-diretório do programa PSP, mas em qualquer outro sub-diretório, (Fig 32).

Figura 32 - Entrada de uma seqüência

Carregaremos, como exemplo, uma seqüência para a observação do eco, (Fig 33). Podemos também carregar a seqüência diretamente via "prompt" do DOS, escrevendo apenas o nome do programa (PSP) e em seguida o nome da seqüência :

```
c:\nmr\psp>psp eco.seq
```

A figura 33 também mostra o menu principal, com os comandos utilizados na elaboração e manipulação de uma seqüência, que serão explicados nos próximos itens.

Figura 33 - Seqüência de eco e o menu principal

9.2 Nomeação dos canais ("Line Name")

No PSP podemos associar cada canal do Pulskit a uma palavra mneumônica correspondente, como por exemplo, associamos o canal CP00 ao mneumônico "gatematec". Podemos ter um canal associado a vários mneumônicos, mas não podemos ter dois ou mais canais associados ao mesmo mneumônico.

Estas associações são gravadas em um arquivo (psp.chn), para que depois escolhamos os canais de interesse, (Fig 34).

Figura 34 - Nomeação e escolha dos canais

9.3 Programação dos Pulsos ("Pulse")

Ativamos ou desativamos o evento para um determinado canal através do comando "Toggle"; ou utilizando o mouse, posicionando o cursor através do 1º botão do mesmo e ativando/desativando o pulso através do 2º botão.

9.4 Tipos de Eventos ("Event Type")

Vimos anteriormente, que todo evento deve ser definido com um intervalo de tempo que determina a sua duração em cada canal escolhido. No PSP entramos com este intervalo, primeiramente com o comando "Time", para definir o tipo de evento, e depois com o

comando "Time value", para entrarmos com o valor. Podemos especificar este intervalo de tempo ("Specification") dando-lhe um nome correspondente, (Fig 35).

Figura 35 - Tipos de eventos

Temos também a opção de definir o evento como uma tabela ("Table"), composta por vários intervalos de tempo. Quando a sequência de pulsos for executada e um dos eventos for definido como tabela, ela será executada primeiro com o primeiro elemento da tabela, depois com o segundo elemento e assim sucessivamente.

As tabelas são criadas e especificadas pelo usuário no próprio programa PSP, depois são gravadas num arquivo (`psp.tbl`), para que depois possamos acessa-las e se quisermos modifica-las.

Podemos definir numa sequência vários eventos como tabelas, desde que as tabelas tenham o mesmo número de elementos.

O comando "Table Index" mostra o valor de cada elemento da tabela, sem precisarmos edita-la ("Table Edit").

Podemos também definir o evento como uma função da duração de outros eventos, para isso cada evento utilizado numa função, deve ser associado à uma variável, através do comando "Tag", que nos dá inicialmente 25 letras (de a até z) como opção de escolha, número que vai diminuindo à medida que as letras vão sendo escolhidas. Depois basta especificarmos como será a função, como por exemplo a função eco-delay mostrada na figura 36, sendo representada por b-a-c-50 .

Podemos associar, além do evento tipo "time", um evento do tipo "table" como variavel de uma função, neste caso, quando a seqüência for executada, ela será primeiro executada com o valor do evento sendo calculado como função do 1º elemento da tabela, depois será executada com o valor do evento como função do 2º elemento e assim sucessivamente até o ultimo elemento da tabela.

Figura 36 - Evento como função

9.5 Programação de "Loops" ("Loops")

Temos no PSP o recurso de programarmos "loops", definindo o início e o fim de cada "loop" envolvido, como também o número de vezes que ele será realizado, observe a figura 37. Não podemos esquecer que não são permitidos "loops" entrelaçados.

Figura 37 - Programação de "loops"

9.6 A compilação da seqüência de pulsos ("Compile")

Utilizamos este comando, caso desejarmos ver como seriam as palavras de controle de uma seqüência gerada, pois este transforma a seqüência nas palavras de controle correspondentes (em hexadecimal) e as grava no arquivo `psp.out`:

```
Number of events      : 11
Size of table file    : 8
Table index           : 1
  0 : software delay : 1.0000000 sg :
  1 : scope_trigger  : 1.0 us : CP02 SP00
  2 : delay          : 19.0 us : SP00
  3 : pulso_de_pi/2  : 1.3 us : SP00 CP00
```

```

4 :    pi/2_pi_sep :          50.0 us : SP00
5 :    pulso_de_pi :          2.6 us : SP00 CP00
6 :    dead_time :          10.0 us : SP05
7 :    eco_delay :          13.7 us :
8 :    Nicolet_trigg :          10.0 us : CP01
9 :    Recovery_time :    1.0000000 sg :
10 : interrupt_loop :          1.0000 ms : CP17 CP18 CP19
    0000 0000 0000 0000 0005
0 : 0000 E000 0000 FF67 6980 0000 0000 0000
1 : 0000 E000 0041 FFFF FFF6 0000 0000 0000
2 : 0000 E000 0001 FFFF FF42 0000 0000 0000
3 : 0000 E000 0011 FFFF FFF3 0000 0000 0000
4 : 0000 E000 0001 FFFF FE0C 0000 0000 0000
5 : 0000 E000 0011 FFFF FFE6 0000 0000 0000
6 : 0000 E000 0002 FFFF FF9C 0000 0000 0000
7 : 0000 E000 0000 FFFF FF77 0000 0000 0000
8 : 0000 E000 0020 FFFF FF9C 0000 0000 0000
9 : 0001 3001 0000 FF67 6980 0000 0000 0000
10 : E000 200A 0000 FFFE 7960 0000 0000 0000

```

0 Normal end of execution

O evento nº0 e o ultimo evento são criados automaticamente pelo próprio programa, pois devem constar em todas as seqüências. O evento nº0 é um "delay" de 1 segundo exigido pelo Pulskit, e o ultimo evento ativa os canais CP17, CP18, CP19, e fica num "loop" infinito.

Os canais CP17, CP18, CP19, estão conectados à interface do Pulskit e são utilizados para informar ao programa PSP, que a seqüência terminou, e que ele pode adquirir os dados do Nicolet. O "loop" infinito é necessário para que o canal CP06 fique ativo por um tempo suficiente para ser detectado pelo programa PSP.

Caso haja algum erro na manipulação do programa, ele será mancionado no arquivo psp.out .

9.7 A execução da seqüência de pulsos ("Execute")

Através deste comando, compilamos e executamos a seqüência, além de programarmos o Nicolet, com uma configuração "default", para que ele receba os sinais de RMN.

Durante a execução da sequência podemos incrementar ou decrementar de 10% , qualquer evento do tipo "time", (Fig 38).

Figura 38 - A execução de uma sequência

Com o comando "transfer" informamos que queremos a transferência dos sinais de RMN, digitalizados no Nicolet, para o microcomputador, onde serão guardados em arquivos de dados, com os nomes previamente escolhidos. Os dados adquiridos são graficados no próprio PSP, (Fig 39).

Figura 39 - Aquisição de dados

9.8 Sequência para a aquisição de um Fid

Na figura 40 poderemos observar uma sequência, criada no PSP, para a obtenção de um Fid.

Figura 40 - Sequência para a aquisição de um Fid

A seguir, (Fig 41), poderemos ver o Fid adquirido, que é guardado num arquivo, cujo nome foi previamente estipulado.

Figura 41 - O fid adquirido pelo programa PSP

9.9 O fluxograma do programa PSP

Apresentaremos a seguir, o fluxograma do "software" PSP, com o intuito de facilitar o seu entendimento :

9.10 As variáveis globais do programa PSP

As variáveis globais, também chamadas de variáveis externas, possuem o escopo global e não local. Elas existem fora de qualquer função, disponíveis para aquela que necessitar.^[22] Mencionaremos abaixo, as variáveis globais utilizadas por nós :

BUF é o buffer de parâmetros da seqüência, nele são guardados :

Nome da seqüência - SEQ

Tempo dos eventos - TIME

Tabela de pulsos - PULSE

Tipos de eventos - TYPE

Loops - LOOP

Contadores de loops - COUNT

Especificação dos eventos - SPEC

Rótulo dos eventos - TAG

Nome (usuário) das linhas - NAME

Nome (Tecmag) das linhas - LINE

Código das linhas - CODE

KIT buffer de dados que serão transferidos para o Pulskit:

- Palavras de Controle

- Valor dos contadores

char FILE_NAME[80] - string que indica o drive, sub-diretório, nome e extensão do arquivo de entrada.

int CHG - flag que informa se a seqüência foi alterada.

FILE *OUT - ponteiro para arquivo de saída geral (psp.out), contém as mensagens de erro.

Mensagens de erro :

int IER - número da mensagem de erro

char ERS[80] - string que contém a mensagem de erro

Parâmetros do Mouse :

int M1, M2, M3, M4, MH, MV;

Parâmetros de janelas e mensagens :

int X1, Y1, X2, Y2;

9.11 As funções do "software" PSP

Uma função em C é um bloco codificado independente, que executa alguma tarefa coerente. Imaginemo-la como uma "caixa-preta", põe-se dados e retiram-se dados. Os trabalhos internos são invisíveis, desconhecidos do resto do programa. Tudo o que o mundo exterior necessita saber sobre uma função é : a entrada; a saída; os efeitos colaterais durante sua execução.^[22]

O programa PSP é um conjunto de várias funções, veremos a seguir as funções mais importantes :

GETIMAGE/PUTIMAGE : grava/restaura a imagem definida pelo retângulo X1, Y1, X2, Y2, num arquivo temporário (c:\psp\temp).

GETSETTINGS/PUTSETTINGS : guarda/restaura os parâmetros gráficos (cor, cor de fundo, tipo de texto, espessura da linha).

FREEBUF/LOADBUF : libera/carrega o buffer que contém a sequência (c:\psp\temp\psp.buf).

FREEKIT/LOADKIT : libera/carrega o buffer que contém os dados que serão transferidos para o Pulskit (c:\psp\temp\psp.kit).

Funções do mouse :

- status do mouse
- posição xy
- janela

Funções de edição :

- tabela TBL_EDT
- pulso PULSE_EDT
- canal LINE_EDT
- função FCT_EDT
- loops LOOP_EDT

WINDOW : indica a janela de atuação (VIEWPORT) e atualiza as variáveis globais, que determinam esta janela, X1, Y1, X2, Y2;

MESSAGE : escreve ou apaga mensagens na janela correspondente. Coloca as coordenadas do retângulo que limita a mensagem, nas variáveis X1, Y1, X2, Y2;

PUTOUT : coloca dados na porta A, B, C da interface 8255 e dispara o pulso READ .

Conclusão

O principal objetivo deste trabalho foi alcançado. Conseguimos efetuar todas as implementações necessárias, tanto a nível de "hardware" como também de "software", a fim de automatizar o nosso espectrômetro de Ressonância Magnética Nuclear Pulsada. Atualmente ele se encontra completamente operacional onde, toda a rotina do experimento baseada em sequências de pulsos relativamente sofisticadas, pode ser programada com facilidade até por um estudante que não possua qualquer experiência prévia em computação.

O equipamento tem sido utilizado durante os últimos três meses por todos os estudantes e professores do laboratório na obtenção de resultados para as suas pesquisas. Durante este tempo pudemos acatar novas sugestões para implementar o software e identificar os seus erros. Para concluir, tanto a nossa opinião quanto a dos usuários do equipamento, é que o estágio alcançado é bastante satisfatório e cumpre as necessidades de rotina do laboratório.

O custo econômico do projeto foi muito baixo, sendo que a maior parte do trabalho consistiu no desenvolvimento de "software" para automatizar o espectrômetro já existente e não na compra de novos equipamentos.

Tecnicamente falando, todos os programas (escritos em linguagem C) são bastante portáteis, podendo ser utilizados em qualquer computador da linha PC. Eles não requerem grande quantidade de memória e podem operar em vídeo VGA branco e preto ou colorido. Além disso, o programa é modular, o que facilitará as modificações e ampliações futuras.

Escrever o "software" foi para nós um trabalho excitante,

porém ao mesmo tempo tedioso e frustrante. A sensação é que o programa nunca está terminado. Há sempre uma nuance qualquer, uma nova opção sugerida por usuário e também, há sempre algum novo truque capaz de torna-lo mais veloz, mais prático e mais eficiente. Este é um típico trabalho que não tem final.

Por este motivo, temos plena consciência de que este programa será, com o passar do tempo modificado para atender tanto as necessidades dos usuários como também aos avanços da técnica de RMN.

Sugerimos como continuação deste trabalho, a parte de apoio computacional para a análise dos resultados experimentais; pois além do controle do equipamento e aquisição de dados, que foi ênfase do nosso trabalho, há necessidade de manusear matematicamente os resultados obtidos a fim de, deles extrair os parâmetros físicos de interesse experimental. Também, existe a necessidade de implementar rotinas que facilitem a comparação dos resultados obtidos com os modelos teóricos existentes.

Seria também interessante o aperfeiçoamento do nosso sistema de recepção e transmissão dos sinais de radiofrequência, o núcleo central do espectrômetro de RMN. O sistema atual é ideal para o estudo de amostras líquidas, que possuem tempo de relaxação relativamente longos. Entretanto, os temas de pesquisas atuais que estão sendo encaminhados pelo nosso Grupo envolve o estudo de materiais sólidos.

Title	Detector de "Duty Cycle"	
Size	Document Number	
Date	APR 26, 1978	BY
	2	
	REV

APÊNDICE - "PULSE SEQUENCE PROGRAM"

```

/* ***** PSP_DCL.H ***** */

#include <stdio.h>
#include <stdlib.h>
#include <graphics.h>
#include <conio.h>
#include <process.h>
#include <math.h>
#include <time.h>
#include <io.h>
#include <alloc.h>
#include <fcntl.h>
#include <sys\stat.h>
#include <dir.h>
#include <dos.h>
#include <ctype.h>
#include <string.h>
#include "c:\at-gpib\decl.h" /* declarations for "C" NI-Software */

/* Constantes: */

#define LEFT 75
#define RIGHT 77
#define UP 72
#define DOWN 80
#define ESC 27
#define F1 59
#define F2 60
#define F3 61
#define F4 62
#define F5 63
#define F6 64
#define F7 65
#define F8 66
#define F9 67
#define F10 68
#define DELETE 83
#define INSERT 82
#define BACK 8
#define HOME 71
#define DELIMITADOR 1
#define VARIABEL 2
#define NUMERO 3
#define PI 3.141516
#define TWOPI 6.2831853072

#ifndef NOEXTERN
#define NOEXTERN extern
#endif

/* Funcoes: */

void WINDOW (int);
void PSP_COVER (void);
void HELP (void);
void COLOR (int);
void MESSAGE (int,int);

```

```

void DATA_WINDOW (int);
void REM_BLANK (char *);
int PULSE_EDT (void);
int GET_PULSE (int,int);
void SET_PULSE (int,int,int);
int EVENT (void);
void SET_EV (int,int);
int GET_EV (int);
void PUTCHAR (int);
int GETCHAR (unsigned int,unsigned int);
void TOGGLE (unsigned int,unsigned int);
void CURSOR (char);
void KURSOR (char);
int DIR_LIST (int,int,char *);
void BIOSKEY (char *);
void INPUT (int,int,char *,int,int,int,int);
int LOAD_BUF (void);
int FREE_BUF (void);
int GET_DATA (int);
int SAVE_DATA (int);
int SAVE_ASK (char *);
void SAVE_AS (char *);
void SAVE_LABELS (int,char *,char *,char *);
void WRITE_LABELS (int,int,int,int,char *,char *);
void PULSE_UPDATE (int,int,int,int);
int TIME_UPDATE (void);
void STRTIME (char *,unsigned long int);
void INTTIME (char *,unsigned long int *);
void MOLDURA (void);
void INPUTIME (char *,int,int,int);
int GETIMAGE (int,int,int,int);
int PUTIMAGE (int,int,int,int);
void GETSETTINGS (void);
void PUTSETTINGS (void);
int SIZEARQ (FILE *,int *,int *);
int LINE_EDT (void);
int CHK_CHN (void);
int TBL_EDT (char *);
int CHK_TBL (void);
int SAVE_TITUL (int,char *);
int SAVE_TABLE (char *,int *,unsigned long int *);
void WRITE_TITUL (int,int,int,int,char *,char *);
void WRITE_TABLE (int,int,int,int,unsigned long int *);
int TBL_EVAL (char *,unsigned long int *,int,int *);
int FCT_EDT (char *);
int FCT_EVAL (char *,unsigned long int *);
int FCT_EXPR (char *,char *);
int CHK_FCT (void);
int LOOP_EDT (void);
void LOOP_DRAW (void);
void LOOP_LIMITS (int *,int *);
int COMPILER (void);
void MOUSE (int *,int *,int *,int *);
void MOUSE_HAND (void);
void MOUSE_ARROW (void);
int CHK_MOUSE (void);
void RESET_MOUSE (void);
void SHOW_MOUSE (void);
void HIDE_MOUSE (void);
void GET_MOUSE_XY (void);

```

```

void SET_MOUSE_XY      (int,int);
void SET_MOUSE_WINDOW (int,int,int,int);
void GET_MOUSE_STATUS  (void);
void GET_MOUSE_SENS    (void);
void SET_MOUSE_SENS    (int,int,int);
int  GET_DEF           (int);
void EVAL_EXP          (double *);
void EVAL_EXP1         (double *);
void EVAL_EXP2         (double *);
void EVAL_EXP3         (double *);
void EVAL_EXP4         (double *);
void EVAL_EXP5         (double *);
void EVAL_EXP6         (double *);
void ATOM              (double *);
void GET_TOKEN         (void);
void PUTBACK           (void);
void UNARY             (char,double *);
void SERROR            (int);
double FIND_VAR        (char *);
int  ISDELIM           (char);
int  LDPPG             (void);
void PUTOUT            (unsigned char,unsigned char,unsigned char);
int  FREE_KIT          (void);
int  LOAD_KIT          (void);
int  SET_ADC           (void);
int  CHK_ADC           (void);
int  TRANSFER          (char *);
void ITR_BREAK         (void);
void SAVE_KIT          (void);
int  EXEC              (void);
void START_PULSKIT     (void);
void STOP_PULSKIT      (void);
void TRIG_RESET        (void);
void PLOT_DATA         (void);
void PLOT_FFT          (void);
void FFT               (void);
void CHGDATA           (int *,int *);
int  BITREV            (int,int);
int  LOG2              (int);
int  FFT_ON            (void);
void FFT_OFF           (void);
int  GET_PHASE         (float *);
void PHASE_SHIFT       (int *,int *,float);

```

/* sequencia: */

```

NOEXTERN void          *BUF;    /* ponteiro para buffer de dados */
NOEXTERN char          *SEQ;    /* ponteiro para nome da sequencia */
NOEXTERN unsigned long int *TIME; /* ponteiro para tempo dos eventos */
NOEXTERN unsigned long int *PULSE; /* ponteiro para tabela de pulsos */
NOEXTERN unsigned long int *TYPE; /* ponteiro para tipo dos eventos */
NOEXTERN unsigned long int *LOOP; /* ponteiro para loops */
NOEXTERN unsigned int    *COUNT; /* ponteiro para loop counters */
NOEXTERN char          *SPEC;    /* ptr. para especificacao dos eventos */
NOEXTERN char          *TAG;     /* ponteiro para rotulo dos eventos */
NOEXTERN char          *NAME;    /* ptr. para nome (usuario) das linhas */
NOEXTERN char          *LINE;    /* ptr. para nome (Tecmag) das linhas */
NOEXTERN char          *CODE;    /* ponteiro para codigo das linhas */
NOEXTERN unsigned char  *KIT;    /* pointer for pulskit transfer buffer */
NOEXTERN int           *TRF;    /* pointer for transfer Nicolet data */

```

```

NOEXTERN long int      *DAT; /* pointer to cummlate Nicolet data */

/* Arquivos de dados: */

NOEXTERN char FILE_NAME[80]; /* drive,dir,nome,ext do arquivo de entrada */
NOEXTERN int  CHG;           /* 0: sequencia nao foi alterada 1: foi alterada */
NOEXTERN FILE *OUT;          /* ptr. p/ arquivo de saida geral (psp.out) */
NOEXTERN char DEF[80];       /* string de default */

/* Mensagens de erro: */

NOEXTERN int  IER;
NOEXTERN char ERS[80];

/* Mouse parameters */

NOEXTERN int M1,M2,M3,M4,MW,MH,MV;

/* Window and messages parameters */

NOEXTERN int X1,Y1,X2,Y2;

/* Indices associados a tabela de pulsos: */

NOEXTERN int      NL,NC; /* no. de linhas e colunas da tela de pulsos */
NOEXTERN int      N,M;  /* no. de linhas e colunas da matriz de pulsos */
NOEXTERN int      I,J;  /* posicao do cursor na tela de pulsos */
NOEXTERN int      II,JJ; /* indices para deslocamento da tela de pulsos */
NOEXTERN unsigned int NE; /* numero de eventos compilados */
NOEXTERN unsigned int NT; /* numero de elementos nas tabelas compiladas */
NOEXTERN int      TB;   /* indices do elemento da tabela */

/* Graphics settings (10 levels): */

NOEXTERN struct viewporttype OLDVIEW[10];
NOEXTERN struct fillsettingstype OLDFILL[10];
NOEXTERN struct textsettingstype OLDTEXT[10];
NOEXTERN struct linesettingstype OLDLINE[10];
NOEXTERN char OLDCOLOR[10];
NOEXTERN char OLDBKCOLOR[10];
NOEXTERN int  SETLEVEL;
NOEXTERN int  IMGLEVEL; /* (to save image) */

/* interpretador de funcoes */

char *PROG;
char TOKEN[80];
char TOK_TYPE;
double VARS[26];

/* new definition of backcolor (troca branco por preto) */

#undef BLACK
#define BLACK 15
#undef WHITE
#define WHITE 0

/* fourier transform */

NOEXTERN int *DAT_REAL; /* parte real da sinal */

```

```

NOEXTERN int    *DAT_IMAG;    /* parte imag da sinal */
NOEXTERN int    *FFT_REAL;    /* parte real da transformada */
NOEXTERN int    *FFT_IMAG;    /* parte imag da transformada */
NOEXTERN double *CO;          /* tabela de cossenos para fft */
NOEXTERN double *SI;          /* tabela de senos para fft */
NOEXTERN float  DWT;          /* dwell time */
NOEXTERN float  PHASE;        /* phase shift */

/* Pulskit and ADC interface */

#define CTRL_PORTO 0x307
#define DADOS_A    0x304
#define DADOS_B    0x305
#define DADOS_C    0x306
NOEXTERN int ADC;    /* ADC address */
NOEXTERN int ITR;    /* Sequence execution interruption flag */
NOEXTERN int CHS;    /* Size (or number of points) of Nicolet channels */
NOEXTERN unsigned int NM;    /* numero de medias */
NOEXTERN int FTR;    /* Data transfer control flag */

/* ***** */

```



```

/* ***** MAIN.C ***** */

#define NOEXTERN
#include "psp_dcl.h"
#undef NOEXTERN

/* ***** */

int main(argc,argv)
int argc;
char *argv[ ];
{
    int gdriver=VGA,gmode=VGAHI,errorcode;      /* 640x480-16 cores-1 pagina */
    char ch[2],str[80];
    FILE *dat;
    int i,flag;

    BUF=NULL; SEQ=NULL; PULSE=NULL; TYPE=NULL; NAME=NULL; LINE=NULL;
    CODE=NULL; TIME=NULL; SPEC=NULL; TAG=NULL; LOOP=NULL; COUNT=NULL;
    KIT=NULL; CHG=0; OUT=NULL; IER=0; strcpy(ERS,"Normal end of execution");
    NL=10; NC=32; N=25; M=256; I=0; J=0; II=0; JJ=0; NE=0; NT=0; TB=0; ADC=-1;
    ITR=-1; SETLEVEL=-1; IMGLEVEL=-1; DAT=NULL; TRF=NULL;
    DAT_REAL=NULL; DAT_IMAG=NULL; FFT_REAL=NULL; FFT_IMAG=NULL;
    CO=NULL; SI=NULL; DWT=0.0000001; PHASE=0.0;

    getcwd(str,68); strcat(str,"\\psp.out");
    if((OUT=fopen(str,"w"))==NULL)
        { sprintf(ERS,"Output file '%s' cannot be open (main)",str);
          IER=1; printf("\n%d %s",IER,ERS); exit(IER); }

    IER=CHK_ADC(); if(IER>0) { fclose(OUT); exit(IER); }
    IER=CHK_CHN(); if(IER>0) { fclose(OUT); exit(IER); }
    IER=CHK_TBL(); if(IER>0) { fclose(OUT); exit(IER); }
    IER=CHK_FCT(); if(IER>0) { fclose(OUT); exit(IER); }

    initgraph(&gdriver,&gmode,""); errorcode = graphresult();
    if(errorcode != 0)
    { strcpy(ERS,"Graphics error (main): ");
      strcat(ERS,grapherrormsg(errorcode));
      IER=2; fprintf(OUT,"\n%d %s",IER,ERS); fclose(OUT); exit(IER); }

    setrgbpalette(63,0,0,0); /* troca branco por preto */
    setrgbpalette(0,63,63,63);
    setrgbpalette(7,60,60,60); /* set LIGHTGRAY very light */
    setrgbpalette(56,48,48,48); /* set DARKGRAY less dark */

    IER=CHK_MOUSE(); if(IER>0) { fclose(OUT); closegraph(); exit(IER); }
    M1=0; MOUSE(&M1,&M2,&M3,&M4); /* initialize mouse */
    if(M1==0)
    { IER=3; strcpy(ERS,"Microsoft Mouse not found");
      fprintf(OUT,"\n\n%d %s",IER,ERS); fclose(OUT); closegraph(); exit(IER); }
    MOUSE_ARROW(); /* define arrow mouse mask */
    SET_MOUSE_XY(320,240); /* set mouse position at the center of the screen */
    SET_MOUSE_SENS(40,40,50); /* set mouse sensitivity */

    begin:

    IER=0; strcpy(ERS,"Normal end of execution"); WINDOW(0); clearviewport();
    if((argc==1)|| (argc>2))

```

```

{ PSP_COVER(); WINDOW(2); MOLDURA();
MESSAGE(110,1); MESSAGE(111,1); MESSAGE(112,1);
for(;;)
{ BIOSKEY(ch);
  if((M2==1)&&(MW==2))
  { MESSAGE(110,-1); if((M3>X1)&&(M3<X2)&&(M4>Y1)&&(M4<Y2)) ch[0]='q';
    MESSAGE(111,-1); if((M3>X1)&&(M3<X2)&&(M4>Y1)&&(M4<Y2)) ch[0]='o';
    MESSAGE(112,-1); if((M3>X1)&&(M3<X2)&&(M4>Y1)&&(M4<Y2)) ch[0]='n'; }
  switch(ch[0])
  { case 'q': /* quit */
    if(CHG==1) IER=SAVE_DATA(1);
    fprintf(OUT,"\n\n%d %s",IER,ERS); fclose(OUT);
    RESET_MOUSE(); closegraph(); exit(IER);
  case 'o': /* open file */
    if(CHG==1) IER=SAVE_DATA(1);
    MESSAGE(110,0); MESSAGE(111,0); MESSAGE(112,0);
    strcpy(str,"Read a sequency from"); IER=DIR_LIST(35,8,str);
    if(IER>0) { fprintf(OUT,"\n%d %s",IER,ERS); goto begin; }
    if(strlen(str)==0) goto begin;
    else { strcpy(FILE_NAME,str); flag=1; goto cont; }
  case 'n': /* new file */
    if(CHG==1) IER=SAVE_DATA(1); flag=0; goto cont;
  }
}
}
if(argc==2)
{ IER=GET_DEF(4); if(IER>0) goto begin;
  sprintf(FILE_NAME,"%s%s%s",DEF,"\\",argv[1]); flag=1; }
cont:
argc=1; WINDOW(0); clearviewport(); WINDOW(2); MOLDURA(); MESSAGE(1,1);
MESSAGE(2,1); MESSAGE(3,1); MESSAGE(4,1); MESSAGE(5,1); MESSAGE(6,1);
MESSAGE(7,1); MESSAGE(8,1); MESSAGE(9,1); MESSAGE(10,1); MESSAGE(11,1);
MESSAGE(12,1); IER=GET_DATA(flag);
if(IER>0) { fprintf(OUT,"\n%d %s",IER,ERS); goto begin; }
TIME_UPDATE(); for(i=0;i<10;i++) DATA_WINDOW(i);
for(;;)
{ BIOSKEY(ch);
  if(M2==1)
  { if(MW==2)
    { MESSAGE(1,-1); if((M3>X1)&&(M3<X2)&&(M4>Y1)&&(M4<Y2)) ch[0]='r';
      MESSAGE(2,-1); if((M3>X1)&&(M3<X2)&&(M4>Y1)&&(M4<Y2)) ch[0]='p';
      MESSAGE(3,-1); if((M3>X1)&&(M3<X2)&&(M4>Y1)&&(M4<Y2)) ch[0]='l';
      MESSAGE(4,-1); if((M3>X1)&&(M3<X2)&&(M4>Y1)&&(M4<Y2)) ch[0]='t';
      MESSAGE(5,-1); if((M3>X1)&&(M3<X2)&&(M4>Y1)&&(M4<Y2)) ch[0]='e';
      MESSAGE(6,-1); if((M3>X1)&&(M3<X2)&&(M4>Y1)&&(M4<Y2)) ch[0]='f';
      MESSAGE(7,-1); if((M3>X1)&&(M3<X2)&&(M4>Y1)&&(M4<Y2)) ch[0]='o';
      MESSAGE(8,-1); if((M3>X1)&&(M3<X2)&&(M4>Y1)&&(M4<Y2)) ch[0]='s';
      MESSAGE(9,-1); if((M3>X1)&&(M3<X2)&&(M4>Y1)&&(M4<Y2)) ch[0]='a';
      MESSAGE(10,-1); if((M3>X1)&&(M3<X2)&&(M4>Y1)&&(M4<Y2)) ch[0]='q';
      MESSAGE(11,-1); if((M3>X1)&&(M3<X2)&&(M4>Y1)&&(M4<Y2)) ch[0]='c';
      MESSAGE(12,-1); if((M3>X1)&&(M3<X2)&&(M4>Y1)&&(M4<Y2)) ch[0]='x'; }
    if(MW==7)
    { WINDOW(7); if((M3>X1)&&(M3<X2)&&(M4>Y1)&&(M4<Y2))
      { J=JJ+(M3-X1)/16; I=II+(M4-Y1)/16; ch[0]='p'; } }
    if(MW==6)
    { WINDOW(6); if((M3>X1)&&(M3<X2)&&(M4>Y1)&&(M4<Y2))
      { I=II+(M4-Y1)/16; ch[0]='l'; } }
    if((MW==5)||!(MW==4)) ch[0]='o';
    if(MW==3)
    { WINDOW(3);

```

```

if((M3>X1+6)&&(M3<X1+511)&&(M4>Y1+5)&&(M4<Y1+19))    ch[0]='q';
if((M3>X1+6)&&(M3<X1+180)&&(M4>Y1+44)&&(M4<Y1+100))    ch[0]='e';
if((M3>X1+180)&&(M3<X1+300)&&(M4>Y1+20)&&(M4<Y1+107))    ch[0]='o';
if((M3>X1+6)&&(M3<X2-5)&&(M4>Y1+108)&&(M4<Y2-5))    ch[0]='e'; }
if(MW==9)
{ WINDOW(9); if((M3>X1)&&(M3<X2)&&(M4>Y1)&&(M4<Y2))
  { J=(M3-X1)/16+JJ; ch[0]='e'; } }
if(IER== -1) ch[0]='p'; if(IER== -2) ch[0]='l'; if(IER== -3) ch[0]='e';
if(IER== -4) ch[0]='q'; if(IER== -5) ch[0]='o'; if(IER<0) IER=0;
switch(ch[0])
{ case 'r':
  goto begin;
  case 'p':
    IER = PULSE_EDT(); if(IER>0) goto begin;
    DATA_WINDOW(2); break;
  case 'l':
    IER = LINE_EDT(); if(IER>0) goto begin;
    DATA_WINDOW(2); break;
  case 't':
    IER = TBL_EDT(str); if(IER>0) goto begin; break;
  case 'e':
    IER = EVENT(); if(IER>0) goto begin;
    DATA_WINDOW(2); break;
  case 'f':
    IER=FCT_EDT(str); if(IER>0) goto begin;
    DATA_WINDOW(2); break;
  case 's':
    IER=SAVE_DATA(0);
    if(IER>0) { fprintf(OUT,"\\n%d %s",IER,ERS); goto begin; };
    DATA_WINDOW(2); break;
  case 'a':
    IER=SAVE_DATA(2);
    if(IER>0) { fprintf(OUT,"\\n%d %s",IER,ERS); goto begin; };
    DATA_WINDOW(2); DATA_WINDOW(3); break;
  case 'q':
    if(BUF==NULL) flag=0; else flag=1;
    if(flag==0) IER=LOAD_BUF(); if(IER>0) goto begin;
    WINDOW(3); INPUT(144,8,SEQ,45,45,0,1);
    if(flag==0) IER=FREE_BUF(); if(IER>0) goto begin;
    CHG=1; DATA_WINDOW(2); DATA_WINDOW(1); break;
  case 'o':
    IER=LOOP_EDT(); if(IER>0) goto begin;
    DATA_WINDOW(2); break;
  case 'c':
    IER=COMPILER(); if(IER>0) goto begin; break;
  case 'x':
    if(ADC<0) break;
    IER=COMPILER(); if(IER>0) goto begin;
    IER=EXEC(); if(IER>0) goto begin; break;
}
}
}

/* ***** */

```

```

/* ***** CHAR.C ***** */

#include "psp_dcl.h"

/* ***** */

void PUTCHAR(n)          /* n : numero do caracter */
int n;
{
    unsigned int t;
    register i,j,k,q,c,cc;
    static unsigned int p[24][16] = /* [caracteres][linhas] */
    { 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, // 00-LLL
      0x0000, 0x0000, 0x0000, 0xffff, 0xffff, 0x0000, 0x0000, 0x0000,
      0x0000, 0x0000, 0x0000, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, // 01-LLH
      0x0001, 0x0001, 0x0001, 0xffff, 0xffff, 0x0000, 0x0000, 0x0000,
      0x0000, 0x0000, 0x0000, 0x8000, 0x8000, 0x8000, 0x8000, 0x8000, // 02-HLL
      0x8000, 0x8000, 0x8000, 0xffff, 0xffff, 0x0000, 0x0000, 0x0000,
      0x0000, 0x0000, 0x0000, 0x8001, 0x8001, 0x8001, 0x8001, 0x8001, // 03-HLH
      0x8001, 0x8001, 0x8001, 0xffff, 0xffff, 0x0000, 0x0000, 0x0000,
      0x0000, 0x0000, 0x0000, 0xffff, 0xffff, 0x0000, 0x0000, 0x0000, // 04-HHH
      0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
      0x0000, 0x0000, 0x0000, 0xffff, 0xffff, 0x0001, 0x0001, 0x0001, // 05-HHL
      0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0000, 0x0000, 0x0000,
      0x0000, 0x0000, 0x0000, 0xffff, 0xffff, 0x8000, 0x8000, 0x8000, // 06-LHH
      0x8000, 0x8000, 0x8000, 0x8000, 0x8000, 0x0000, 0x0000, 0x0000,
      0x0000, 0x0000, 0x0000, 0xffff, 0xffff, 0x8001, 0x8001, 0x8001, // 07-LHL
      0x8001, 0x8001, 0x8001, 0x8001, 0x8001, 0x0000, 0x0000, 0x0000,
      0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, // 08-blk
      0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
      0xffff, 0xf99f, 0xe187, 0xc183, 0xc183, 0x8181, 0x8181, 0xe187, // 09-tim
      0xe187, 0x81c1, 0x80e1, 0xc073, 0xc033, 0xe187, 0xf99f, 0xffff,
      0xffff, 0xffff, 0xf000, 0xe000, 0xc3fb, 0xc003, 0xc003, 0xcff3, // 10-tbl
      0xc003, 0xcff3, 0xc003, 0xc003, 0x1fc7, 0x0007, 0xffff, 0xffff,
      0xffff, 0xffff, 0xc003, 0xc1fb, 0xc303, 0xc203, 0xc203, 0xc203, // 11-fct
      0xc7c3, 0xc203, 0xc203, 0xc603, 0xde03, 0xc003, 0xffff, 0xffff,
      0xffff, 0xffff, 0xc003, 0xc003, 0xc003, 0xc003, 0xc003, 0xc003, // 12-sqr
      0xc003, 0xc003, 0xc003, 0xc003, 0xc003, 0xc003, 0xffff, 0xffff,
      0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, // 13-blk
      0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, // 14-blk
      0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
      0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, // 15-blk
      0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
      0xffff, 0xfe7f, 0xfe7f, 0xfc3f, 0xfc3f, 0xf81f, 0xf81f, 0xf00f, // 16-trg
      0xf00f, 0xe007, 0xe007, 0xc003, 0xc003, 0x8001, 0xffff, 0xffff,
      0xffff, 0xffff, 0xfc3f, 0xf00f, 0xe007, 0xc003, 0xc003, 0xc003, // 17-crc
      0xc003, 0xc003, 0xc003, 0xe007, 0xf00f, 0xfc3f, 0xffff, 0xffff,
      0xffff, 0xffff, 0xffff, 0xffff, 0xffff, 0xffff, 0xffff, 0xffff, // 18-red
      0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, // 19-blk
      0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
      0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, // 20-blk
      0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
      0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, // 21-blk
      0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
      0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, // 22-blk
      0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
      0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, // 23-blk
      0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000 };

```

```

if((n<0)|| (n>23)) return; GETSETTINGS(); i=I-II; j=J-JJ;
if(j%2==0) c=LIGHTGRAY; else c=WHITE; if(n<16) cc=RED; else cc=BLUE;
setfillstyle(SOLID_FILL,c); bar(16*j,16*i,16*j+15,16*i+15);
for(k=0;k<16;k++)
    if(p[n][k]!=0x0000)
    { for(q=0;q<16;q++)
        { t=0x8000; if(p[n][k]&(t>=q)) putpixel(16*j+q,16*i+k,cc);
          else putpixel(16*j+q,16*i+k,c); } }
PUTSETTINGS(); return;
}

/* ***** */

int GETCHAR(i,j) /* Devolve o no. do caracter do tipo pulso (0-7) */
unsigned int i,j; /* i = 0...74 = 0...N-1 : linha */
{ /* j = 0...2047 = 0...M-1 : coluna */
    static int type[2][2][2] = { 0,1,7,6,2,3,5,4 };
    unsigned int coluna,bit;
    int last,this,next;
    unsigned long int one=0x80000000;;

    this=j; last=this-1; next=this+1;
    if(last>=0)
    { coluna=last/NC; bit=last%NC;
      if((PULSE[i*(M/NC)+coluna])&(one>>bit)) last=1; else last=0; }
    else last=0;
    coluna=this/NC; bit=this%NC;
    if((PULSE[i*(M/NC)+coluna])&(one>>bit)) this=1; else this=0;
    if(next<M)
    { coluna=next/NC; bit=next%NC;
      if((PULSE[i*(M/NC)+coluna])&(one>>bit)) next=1; else next=0; }
    else next=0;
    return(type[last][this][next]);
}

/* ***** */
/* ***** MOUSE.C ***** */

#include "psp_dcl.h"

/* ***** */

void MOUSE(m1,m2,m3,m4)
int *m1,*m2,*m3,*m4;
{
    union REGS reg;

    reg.x.ax = *m1; reg.x.bx = *m2; reg.x.cx = *m3; reg.x.dx = *m4;
    int86(0x33,&reg,&reg);
    *m1 = reg.x.ax; *m2 = reg.x.bx; *m3 = reg.x.cx; *m4 = reg.x.dx; return;
}

/* ***** */

void MOUSE_HAND(void)
{
    struct REGPACK reg;

    static int mask_hand[] =

```

```

    { 0xE1FF, 0xE1FF, 0xE1FF, 0xE1FF, 0xE1FF, 0xE000, 0xE000, 0xE000,
      0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
      0x1E00, 0x1200, 0x1200, 0x1200, 0x1200, 0x13FF, 0x1249, 0x1249,
      0xF249, 0x9001, 0x9001, 0x9001, 0x8001, 0x8001, 0x8001, 0xFFFF };
    reg.r_ax = 9; reg.r_bx = 5; reg.r_cx = 0;
    reg.r_dx = FP_OFF(mask_hand); reg.r_es = FP_SEG(mask_hand);
    intr(0x33,&reg); return;
}

/* ***** */

void MOUSE_ARROW(void)
{
    struct REGPACK reg;

    static int mask_arrow[] =
        { 0x9FFF, 0x8FFF, 0x87FF, 0x83FF, 0x81FF, 0x80FF, 0x807F, 0x803F,
          0x801F, 0x800F, 0x80FF, 0x887F, 0x98FF, 0xFC3F, 0xFC3F, 0xFE3F,
          0x0000, 0x2000, 0x3000, 0x3800, 0x3C00, 0x3E00, 0x3F00, 0x3F80,
          0x3FC0, 0x3E00, 0x3600, 0x2300, 0x0300, 0x0180, 0x0180, 0x0000 };
    reg.r_ax = 9; reg.r_bx = 0; reg.r_cx = 0;
    reg.r_dx = FP_OFF(mask_arrow); reg.r_es = FP_SEG(mask_arrow);
    intr(0x33,&reg); return;
}

/* ***** */

int CHK_MOUSE(void) /* checks the mouse driver to see if it was */
{ /* properly installed. */
    unsigned long address;
    unsigned char first_byte;
    union REGS inregs,outregs; /* structures to contain register values for */
    struct SREGS segregs; /* intdosx . */

    inregs.x.ax = 0x3533; /* get interrupt vector for 0x33 */
    intdosx(&inregs,&outregs,&segregs);
    address = ( (long) segregs.es) << 16 ) + (long) outregs.x.bx;
    first_byte = (unsigned char) * (long far *) address;
    /* be sure vector isn't 0 and first instruction isn't iret */
    if((address==0L)||!(first_byte==0xCF))
    { strcpy(ERS,"The mouse driver must be installed to use this program");
      IER=1; fprintf(OUT,"\n\n%d %s",IER,ERS); return(IER); }
    return(IER);
}

/* ***** */

void RESET_MOUSE(void)
{
    M1=0; MOUSE(&M1,&M2,&M3,&M4); return;
}

/* ***** */

void SHOW_MOUSE(void)
{
    M1=1; MOUSE(&M1,&M2,&M3,&M4); return;
}

/* ***** */

```

```

void HIDE_MOUSE(void)
{
    M1=2; MOUSE(&M1,&M2,&M3,&M4); return;
}

/* ***** */

void GET_MOUSE_XY(void)
{
    M1=3; MOUSE(&M1,&M2,&M3,&M4); return;
}

/* ***** */

void SET_MOUSE_XY(x,y)
int x,y;
{
    M1=4; M3=x; M4=y; MOUSE(&M1,&M2,&M3,&M4); return;
}

/* ***** */

void GET_MOUSE_SENS(void)
{
    M1=27; MOUSE(&M1,&M2,&M3,&M4); return;
}

/* ***** */

void SET_MOUSE_SENS(sx,sy,sm)
int sx,sy,sm;
{
    M1=26; M2=sx; M3=sy; M4=sm; MOUSE(&M1,&M2,&M3,&M4); return;
}

/* ***** */

void SET_MOUSE_WINDOW(x1,y1,x2,y2)
int x1,y1,x2,y2;
{
    M1=7; M3=x1; M4=x2; MOUSE(&M1,&M2,&M3,&M4);
    M1=8; M3=y1; M4=y2; MOUSE(&M1,&M2,&M3,&M4); return;
}

/* ***** */

void GET_MOUSE_STATUS(void)
{
    int n;

    /* button status : M2 : 0-none 1-left 2-right 3-both */
    /* x,y position : M3, M4 : (in pixels) */
    /* window : MW : (as defined in function WINDOW()) */
    /* movement : MH : horizontal MV : vertical */

    GETSETTINGS(); M1=11; MOUSE(&M1,&M2,&M3,&M4);
    if(M3==0) MH=0; else MH=M3/abs(M3); if(M4==0) MV=0; else MV=M4/abs(M4);
    M1=3; MOUSE(&M1,&M2,&M3,&M4); MW=0;
    for(n=1;n<50;n++)
    { WINDOW(n); if((M3>=X1)&&(M3<=X2)&&(M4>=Y1)&&(M4<=Y2)) { MW=n; break; } }
}

```

```

    PUTSETTINGS(); return;
}

/* ***** */

/* ***** MENUS.C ***** */

#include "psp_dcl.h"

/* ***** */

void WINDOW(i)
int i;
{
    int x1,y1,x2,y2;

    switch (i)
    { case 0: x1= 0; y1= 0; x2=639; y2=479; break; /* tela toda */
      case 1: x1= 1; y1=462; x2=638; y2=479; break; /* warnings */
      case 2: x1= 1; y1=389; x2=638; y2=461; break; /* menu */
      case 3: x1= 1; y1=262; x2=638; y2=388; break; /* dados */
      case 4: x1= 1; y1=180; x2=100; y2=261; break; /* nome dos loops */
      case 5: x1=110; y1=180; x2=628; y2=261; break; /* loops */
      case 6: x1= 1; y1= 20; x2=100; y2=179; break; /* nome das linhas */
      case 7: x1=110; y1= 20; x2=628; y2=179; break; /* pulsos */
      case 8: x1= 1; y1= 1; x2=109; y2= 19; break; /* titulo */
      case 9: x1=110; y1= 1; x2=628; y2= 19; break; /* nome dos eventos */
      case 51: x1= 1; y1= 1; x2=638; y2=261; break; /* table edit */
      case 54: x1=160; y1=120; x2=480; y2=360; break; /* Help */
      case 55: x1=160; y1=160; x2=480; y2=320; break; /* file edt */
      case 56: x1= 1; y1= 1; x2=638; y2=261; break; /* plot area */
      case 57: x1=321; y1= 1; x2=638; y2=133; break; /* data real plot area */
      case 58: x1=321; y1=129; x2=638; y2=261; break; /* data imag plot area */
      case 59: x1= 1; y1= 1; x2=318; y2=133; break; /* fft real plot area */
      case 60: x1= 1; y1=129; x2=318; y2=261; break; /* fft imag plot area */
    }
    /* ----- */
    setviewport(x1,y1,x2,y2,1); X1=x1; Y1=y1; X2=x2; Y2=y2; return;
}

/* ***** */

void MESSAGE(n,z)
int n,z;
{
    int x,y,w;
    char m,t,i,j,k,str[80];

    /* n : numero da mensagem */
    /* z : z>0 : escreve, : z=0 apaga, z<0 : devolve retangulo X1,Y1,X2,Y2 */
    /* m : 0,1,.. caracter diferenciado (se m<0 nenhum e' diferenciado */
    /* t : tamanho dos caracteres, em multiplos de 8*8 pixels */
    /* x, y : posicao da mensagem no viewport */
    /* i : cor da mensagem */
    /* j : cor do caracter diferenciado */
    /* k : cor do fundo */

    GETSETTINGS(); i=BLUE; j=RED; k=WHITE; m=0; t=1; w=2;

```



```

switch(n)
{ /* ----- Open menu ----- */
  case 110: strcpy(str,"Quit");          t=2; x=500; y=16; break;
  case 111: strcpy(str,"Open an existing file"); t=2; x= 16; y=16; break;
  case 112: strcpy(str,"Open a new file");  m=7; t=2; x= 16; y=48; break;
  /* ----- Main menu ----- */
  case 1 : strcpy(str,"Return");          x= 15; y=8; break;
  case 2 : strcpy(str,"Pulse");            x=137; y=8; break;
  case 3 : strcpy(str,"Line name");        x=259; y=8; break;
  case 4 : strcpy(str,"Tabel edit");        x=381; y=8; break;
  case 5 : strcpy(str,"Event type");        x=503; y=8; break;
  case 6 : strcpy(str,"Function edit");     x= 15; y=24; break;
  case 7 : strcpy(str,"Loops");            m=1; x=137; y=24; break;
  case 8 : strcpy(str,"Save");              x=259; y=24; break;
  case 9 : strcpy(str,"Save as..");         m=1; x=381; y=24; break;
  case 10: strcpy(str,"Sequence name");     m=2; x=503; y=24; break;
  case 11: strcpy(str,"Compile");           x= 15; y=40; break;
  case 12: strcpy(str,"Execute");           m=1; x=137; y=40; break;
  /* ----- Pulse menu ----- */
  case 20: strcpy(str,"Return");           x= 15; y=24; break;
  case 21: strcpy(str,"Toggle");           x=137; y=24; break;
  /* ----- Line menu ----- */
  case 35: strcpy(str," <<-- ");           x= 15; y= 8; break;
  case 36: strcpy(str,"Return");           x=137; y= 8; break;
  case 37: strcpy(str,"Copy (red -> blue)"); x=259; y= 8; break;
  case 38: strcpy(str,"Delete");           x= 15; y=40; break;
  case 39: strcpy(str," -->> ");           x=503; y= 8; break;
  case 40: strcpy(str,"Insert");            x= 15; y=24; break;
  case 41: strcpy(str,"Save");              x=137; y=24; break;
  case 42: strcpy(str,"Append");            x=259; y=24; break;
  case 43: strcpy(str,"Write");             x=381; y=24; break;
  case 44: strcpy(str,"Move");              x=503; y=24; break;
  /* ----- Table menu ----- */
  case 80: strcpy(str," <<-- ");           x= 15; y= 8; break;
  case 81: strcpy(str,"Return");           x=137; y= 8; break;
  case 82: strcpy(str,"Delete");           x=259; y= 8; break;
  case 83: strcpy(str,"Insert");            x=381; y= 8; break;
  case 84: strcpy(str," -->> ");           x=503; y= 8; break;
  case 85: strcpy(str,"Save");              x= 15; y=24; break;
  case 86: strcpy(str,"Append");            x=137; y=24; break;
  case 87: strcpy(str,"Write");             x=259; y=24; break;
  case 88: strcpy(str,"Select");            m=2; x=381; y=24; break;
  case 89: strcpy(str,"Edit");             x=503; y=24; break;
  /* ----- Event menu ----- */
  case 60: strcpy(str,"Return");           x= 15; y= 8; break;
  case 61: strcpy(str,"Time");             x=137; y= 8; break;
  case 62: strcpy(str,"Table");            m=2; x=259; y= 8; break;
  case 63: strcpy(str,"Function");          x=381; y= 8; break;
  case 64: strcpy(str,"Cancel");            x=503; y= 8; break;
  case 65: strcpy(str,"Time value");        m=5; x= 15; y=24; break;
  case 66: strcpy(str,"Specification");     x=137; y=24; break;
  case 67: strcpy(str,"Tag");              m=2; x=259; y=24; break;
  case 68: strcpy(str,"Table index");       m=10; x=381; y=24; break;
  case 69: strcpy(str,"ADC +");            m=4; x= 15; y=40; break;
  case 70: strcpy(str,"ADC -");            m=4; x=137; y=40; break;
  case 71: strcpy(str,"Delete");           x=259; y=40; break;
  case 72: strcpy(str,"Insert");           x=381; y=40; break;
  /* ----- function menu ----- */
  case 50: strcpy(str,"Return");           x= 15; y=40; break;
  case 51: strcpy(str,"Delete");           x=137; y=40; break;

```

```

case 52 : strcpy(str,"Append");           x=259; y=40; break;
case 53 : strcpy(str,"Write");            x=381; y=40; break;
/* ----- loops menu ----- */
case 120: strcpy(str,"Return");           x= 15; y=24; break;
case 121: strcpy(str,"Begin");            x=137; y=24; break;
case 122: strcpy(str,"End");              x=259; y=24; break;
case 123: strcpy(str,"Off");              x=381; y=24; break;
case 124: strcpy(str,"Counter");          x=503; y=24; break;
/* ----- Execute menu ----- */
case 150: strcpy(str,"Return");           x= 15; y=24; break;
case 151: strcpy(str,"Start");            m=2; x=137; y=24; break;
case 152: strcpy(str,"Stop");             x=259; y=24; break;
case 153: strcpy(str,"Increment");        x=381; y=24; break;
case 154: strcpy(str,"Decrement");        x=503; y=24; break;
case 155: strcpy(str,"Transfer");         x= 15; y=40; break;
case 156: strcpy(str,"Data file");        m=5; x=137; y=40; break;
/* ----- Transfer menu ----- */
case 160: strcpy(str,"Stop");             x= 15; y=24; break;
/* ----- Warnings ----- */
case 100: sprintf(str,"Saving sequence in : %s ",FILE_NAME);
                                m=-1; w=1; x=16; y=4; break;
case 101: sprintf(str,"Reading sequence from : %s ",FILE_NAME);
                                m=-1; w=1; x=16; y=4; break;
case 102: sprintf(str,"Creating a new sequence : %s ",FILE_NAME);
                                m=-1; w=1; x=16; y=4; break;
/* ----- */
}
WINDOW(w);
if(z<0) { X1=X1+x-4; X2=X1+8*t*strlen(str)+4; Y1=Y1+y-4; Y2=Y1+8*t+4;
          PUTSETTINGS(); return; }
setfillstyle(SOLID_FILL,k); bar(x,y,x+8*t*strlen(str)-1,y+8*t-1);
if(z>0) { setcolor(i); settextstyle(0,0,t); outtextxy(x,y,str);
          if(m>=0)
            { str[0] = str[m]; str[1] = '\0';
              setcolor(j); outtextxy(x+m*8*t,y,str); } }
PUTSETTINGS(); return;
}

/* ***** */

void PSP_COVER(void)
{
  char name[80];

  GETSETTINGS(); WINDOW(3); MOLDURA();
  setcolor(BLUE); strcpy(name,"IFQSC - DFCM - USP");
  settextstyle(0,0,1); outtextxy(32,36,name);
  setcolor(RED); strcpy(name,"PSP");
  settextstyle(0,0,4); outtextxy(32,52,name);
  setcolor(BLUE); strcpy(name,"Pulse Sequence Programming");
  settextstyle(0,0,1); outtextxy(32,92,name);
  setcolor(BLUE); strcpy(name,"Copyrights 1992 by:");
  settextstyle(0,0,1); outtextxy(426,16,name);
  setcolor(BLUE); strcpy(name,"Marco A. Pagnano");
  settextstyle(0,0,1); outtextxy(426,32,name);
  setcolor(BLUE); strcpy(name,"Claudio J. Magon");
  settextstyle(0,0,1); outtextxy(426,48,name);
  setcolor(BLUE); strcpy(name,"Luiz Telmo da Silva Auler");
  settextstyle(0,0,1); outtextxy(426,64,name);
  setcolor(BLUE); strcpy(name,"Mateus Jose Martins");

```

```

    settextstyle(0,0,1); outtextxy(426,80,name);
    setcolor(BLUE); strcpy(name,"All rights reserved worldwide");
    settextstyle(0,0,1); outtextxy(400,108,name);
    PUTSETTINGS(); return;
}

/* ***** */

void HELP(void)
{
    FILE *hel;
    int x1,y1,x2,y2;
    char ch[2],str[80];
    int i,j;

    if((hel=fopen("c:\\psp\\psp.hel","r"))==NULL)
    { fprintf(OUT,
        "WARNING: Output file 'c:\\psp\\psp.hel' cannot be open (HELP).");
        return; }
    GETSETTINGS(); x1=35; y1=8; x2=35+570; y2=8+240; IER=GETIMAGE(x1,y1,x2,y2);
    if(IER>0) { PUTSETTINGS(); fclose(hel); return; }
    setviewport(x1,y1,x2,y2,1); setfillstyle(SOLID_FILL,LIGHTGRAY);
    bar(0,0,(x2-x1),(y2-y1)); setlinestyle(SOLID_LINE,0,THICK_WIDTH);
    setcolor(YELLOW); rectangle(5,5,(x2-x1-5),(y2-y1-5));
    line(5,212,x2-x1-5,212); SET_MOUSE_XY((x1+x2)/2,(y1+y2)/2);
    setcolor(BLUE); outtextxy(108,221,"More"); rectangle(100,217,148,231);
    setcolor(RED); outtextxy(108,221,"M"); setcolor(BLUE);
    outtextxy(388,221,"Cancel"); rectangle(380,217,444,231); setcolor(RED);
    outtextxy(388,221,"C"); setcolor(BLUE); settextstyle(0,0,1);
    if(fgets(str,79,hel)==NULL) return;
    for(i=0;i<24;i++)
    { for(j=0;j<strlen(str)-1;j++) if(isprint(str[j])==0) str[j]=' ';
        str[strlen(str)-1]='\0'; outtextxy(8,16+8*i,str);
        if(fgets(str,79,hel)==NULL) break;}
    for(;;)
    { BIOSKEY(ch);
        if(M2==1)
        { if((M3>x1+100)&&(M3<x1+148)&&(M4>y1+217)&&(M4<y1+231)) ch[0]='m';
            if((M3>x1+380)&&(M3<x1+444)&&(M4>y1+217)&&(M4<y1+231)) ch[0]='c'; }
        switch(ch[0])
        { case 'm':
            if(fgets(str,79,hel)==NULL) { rewind(hel); fgets(str,79,hel); }
            bar(8,8,x2-x1-5,211);
            for(i=0;i<24;i++)
            { for(j=0;j<strlen(str)-1;j++) if(isprint(str[j])==0) str[j]=' ';
                str[strlen(str)-1]='\0'; outtextxy(8,16+8*i,str);
                if(fgets(str,79,hel)==NULL) break; }
            break;
            case 'c':
                fclose(hel); PUTIMAGE(x1,y1,x2,y2); PUTSETTINGS(); return;
            }
        }
    }
}

/* ***** */

/* ***** EVENT.C ***** */

#include "psp_dcl.h"

```

```

/* ***** */

int EVENT(void)
{
    char ch[2],str[80];
    unsigned char c;
    int n,m,isave,jsave;

    GETSETTINGS(); WINDOW(2); IER=GETIMAGE(X1,Y1,X2,Y2);
    if(IER>0) { PUTSETTINGS(); return(IER); }
    MOLDURA(); MESSAGE(60,1); MESSAGE(61,1); MESSAGE(62,1); MESSAGE(63,1);
    MESSAGE(64,1); MESSAGE(65,1); MESSAGE(66,1); MESSAGE(67,1); MESSAGE(68,1);
    MESSAGE(69,1); MESSAGE(70,1); MESSAGE(71,1); MESSAGE(72,1);
    setfillstyle(SOLID_FILL,WHITE); setlinestyle(SOLID_LINE,0,NORM_WIDTH);
    settxtstyle(0,0,1); WINDOW(9); isave=I; jsave=J; I=II;
    KURSOR(1); LOAD_BUF(); TIME_UPDATE();
    for(;;)
    { TIME_UPDATE(); DATA_WINDOW(4); BIOSKEY(ch);
      if(M2==1)
      { if(MW==2)
        { MESSAGE(60,-1); if((M3>X1)&&(M3<X2)&&(M4>Y1)&&(M4<Y2)) ch[0]='r';
          MESSAGE(61,-1); if((M3>X1)&&(M3<X2)&&(M4>Y1)&&(M4<Y2)) ch[0]='t';
          MESSAGE(62,-1); if((M3>X1)&&(M3<X2)&&(M4>Y1)&&(M4<Y2)) ch[0]='b';
          MESSAGE(63,-1); if((M3>X1)&&(M3<X2)&&(M4>Y1)&&(M4<Y2)) ch[0]='f';
          MESSAGE(64,-1); if((M3>X1)&&(M3<X2)&&(M4>Y1)&&(M4<Y2)) ch[0]='c';
          MESSAGE(65,-1); if((M3>X1)&&(M3<X2)&&(M4>Y1)&&(M4<Y2)) ch[0]='v';
          MESSAGE(66,-1); if((M3>X1)&&(M3<X2)&&(M4>Y1)&&(M4<Y2)) ch[0]='s';
          MESSAGE(67,-1); if((M3>X1)&&(M3<X2)&&(M4>Y1)&&(M4<Y2)) ch[0]='g';
          MESSAGE(68,-1); if((M3>X1)&&(M3<X2)&&(M4>Y1)&&(M4<Y2)) ch[0]='x';
          MESSAGE(69,-1); if((M3>X1)&&(M3<X2)&&(M4>Y1)&&(M4<Y2)) ch[0]='+';
          MESSAGE(70,-1); if((M3>X1)&&(M3<X2)&&(M4>Y1)&&(M4<Y2)) ch[0]='-';
          MESSAGE(71,-1); if((M3>X1)&&(M3<X2)&&(M4>Y1)&&(M4<Y2)) ch[0]='d';
          MESSAGE(72,-1); if((M3>X1)&&(M3<X2)&&(M4>Y1)&&(M4<Y2)) ch[0]='i'; }
        if(MW==9)
        { WINDOW(9); KURSOR(0); J=(M3-X1)/16+JJ; KURSOR(1); }
        if(MW==7)
        { WINDOW(9); KURSOR(0); WINDOW(7);
          J=JJ+(M3-X1)/16; I=II+(M4-Y1)/16; IER=-1; ch[0]='r'; }
        if(MW==6)
        { WINDOW(9); KURSOR(0); WINDOW(6);
          I=II+(M4-Y1)/16; IER=-2; ch[0]='r'; }
        if((MW==4)|| (MW==5)) { IER=-5; ch[0]='r'; }
        if(MW==3)
        { WINDOW(3);
          if((M3>X1+6)&&(M3<X1+511)&&(M4>Y1+5)&&(M4<Y1+19))
          { IER=-4; ch[0]='r'; }
          if((M3>X1+180)&&(M3<X1+300)&&(M4>Y1+20)&&(M4<Y1+107))
          { IER=-5; ch[0]='r'; } }
        switch(ch[0])
        { case 'r':
          WINDOW(9); KURSOR(0); J=jsave; I=isave; IER=FREE_BUF();
          WINDOW(2); IER=PUTIMAGE(X1,Y1,X2,Y2);
          PUTSETTINGS(); return(IER);
          case 'c':
          CHG=1; SET_EV(0,J); PUTCHAR(8); KURSOR(1); break;
          case 't':
          CHG=1; SET_EV(1,J); PUTCHAR(9); KURSOR(1); break;
          case 'b':
          CHG=1; SET_EV(2,J); PUTCHAR(10); KURSOR(1); break;
          case 'f':
        }
    }
}

```

```

    CHG=1; SET_EV(3,J); PUTCHAR(11); KURS(1); break;
case '+':
    CHG=1; SET_EV(8,J); PUTCHAR(16); KURS(1); break;
case '-':
    CHG=1; SET_EV(9,J); PUTCHAR(17); KURS(1); break;
case 'v':
    if(GET_EV(J)==1)
    { CHG=1; WINDOW(3); setcolor(RED); bar(62,42,178,54);
      rectangle(62,42,154,54); rectangle(158,42,178,54);
      INPUTIME(str,0,64,44); INTIME(str,&TIME[J]); setcolor(WHITE);
      rectangle(62,42,154,54); rectangle(158,42,178,54); WINDOW(9); }
    break;
case 's':
    n=GET_EV(J);
    if(n==1)
    { CHG=1; WINDOW(3); bar(62,58,178,68); setcolor(RED);
      rectangle(62,58,178,68); INPUT(64,60,&SPEC[15*J],14,14,0,1);
      setcolor(WHITE); rectangle(62,58,178,68); WINDOW(9); }
    if(n==2) { CHG=1; IER=TBL_EDT(str); strcpy(&SPEC[15*J],str); }
    if(n==3) { CHG=1; FCT_EDT(str); strcpy(&SPEC[15*J],str); }
    break;
case 'g':
    n=GET_EV(J);
    if((n==1)|| (n==2))
    { CHG=1; WINDOW(3); bar(62,74,74,86); setcolor(RED); HIDE_MOUSE();
      rectangle(62,74,74,86); c=0; str[1]='\0'; setcolor(BLUE);
      for(;;)
      { if(c==0) str[0]='*'; else str[0]=96+c;
        setcolor(BLUE); outtextxy(64,76,str);
        BIOSKEY(ch); if(ch[0]=='\r') break;
        switch(ch[1])
        { case UP: case RIGHT:
            setcolor(WHITE); outtextxy(64,76,str);
            for(;;)
            { if(c<26) c=c+1; else c=0; m=0;
              for(n=0;n<M;n++) if(TAG[n]==96+c) { m=1; break; }
              if(m==0) break; }
            break;
          case DOWN: case LEFT:
            setcolor(WHITE); outtextxy(64,76,str);
            for(;;)
            { if(c>0) c=c-1; else c=26; m=0;
              for(n=0;n<M;n++) if(TAG[n]==96+c) { m=1; break; }
              if(m==0) break; }
            break; } }
      if(c==0) TAG[J]='*'; else TAG[J]=96+c; ch[0]=0; ch[1]=0;
      SHOW_MOUSE(); setcolor(WHITE); rectangle(62,74,74,86); WINDOW(9); }
    break;
case 'x':
    n=GET_EV(J);
    if(n==2)
    { TBL_EVAL(&SPEC[15*J],&TIME[J],TB,&n);
      if(n==0) break;
      HIDE_MOUSE(); WINDOW(3); bar(64,92,88,100); setcolor(RED);
      rectangle(62,90,90,102); m=TB+1; setcolor(BLUE);
      for(;;)
      { sprintf(str,"%3d",m); setcolor(BLUE); outtextxy(64,92,str);
        BIOSKEY(ch); if(ch[0]=='\r') break;
        switch(ch[1])
        { case UP: case RIGHT:

```

```

        if(m>=n) break; bar(64,92,88,100); m=m+1; break;
        case DOWN: case LEFT:
            if(m<=1) break; bar(64,92,88,100); m=m-1; break; } }
        TB=m-1; ch[0]=0; ch[1]=0; SHOW_MOUSE(); setcolor(WHITE);
        rectangle(62,90,90,102); WINDOW(9); break; }
        break;
        case 'd':
            break; /* delete */
        case 'i':
            break; /* insert */
    }
    switch(ch[1])
    { case RIGHT:
        if(J==M-1) break;
        if(J<JJ+NC-1) { KURSOR(0); J=J+1; KURSOR(1); break; }
        PULSE_UPDATE(1,0,0,0); JJ=JJ+1; J=J+1; KURSOR(1); break;
        case LEFT:
            if(J==0) break;
            if(J>JJ) { KURSOR(0); J=J-1; KURSOR(1); break; }
            PULSE_UPDATE(0,1,0,0); JJ=JJ-1; J=J-1; KURSOR(1); break;
        }
    }
}

/* ***** */

void SET_EV(n,j) /* set event type n to sequence column j */
int j,n;
{
    unsigned int i,k,coluna;
    unsigned long int bit,one=0x80000000;

    if((n>15)||!(n<0)||!(j<0)||!(j>M-1)) return;
    coluna=j/NC; bit=one>>(j%NC);
    for(i=0;i<4;i++)
    { k=n&(0x0001<<i);
        if(k>0) TYPE[i*(M/NC)+coluna] = TYPE[i*(M/NC)+coluna]|bit;
        else TYPE[i*(M/NC)+coluna] = (TYPE[i*(M/NC)+coluna]^bit)^bit; }
    return;
}

/* ***** */

int GET_EV(j) /* retorna o tipo de evento (0 -> 15) associado */
int j; /* a coluna j da sequencia (j= 0 -> M-1) */
{
    int i,k,n;
    unsigned int coluna;
    unsigned long int bit,one=0x80000000;

    if((j<0)||!(j>M-1)) return (0);
    n=0; k=1; coluna=j/NC; bit=one>>(j%NC);
    for(i=0;i<4;i++) { if((TYPE[i*(M/NC)+coluna]&bit)!=0) n=n+k; k=2*k; }
    return (n);
}

/* ***** */

void DATA_WINDOW(w)
int w;

```

```

{
    unsigned int coluna;
    unsigned long int t,bit,one=0x80000000;
    char str[80],drive[3],dir[66],name[9],ext[5];
    int i,j,n,m,flag;

    GETSETTINGS(); WINDOW(3); setfillstyle(SOLID_FILL,WHITE);
    settextstyle(0,0,1); setcolor(BLUE);
    switch(w)
    { case 0:
        MOLDURA(); setlinestyle(SOLID_LINE,1,THICK_WIDTH); setcolor(RED);
        rectangle(6,5,511,19); rectangle(516,5,631,19); setcolor(BLUE);
        outtextxy(16,8,"Sequence name :"); outtextxy(520,8,"Saved ? :");
        outtextxy(8,28,"File :"); outtextxy(8,44,"Time :");
        outtextxy(8,60,"Spec :"); outtextxy(8,76,"Tag :");
        outtextxy(8,92,"Index:"); outtextxy(8,114,"Fct :");
        outtextxy(188,28,"Loop 1:"); outtextxy(188,44,"Loop 2:");
        outtextxy(188,60,"Loop 3:"); outtextxy(188,76,"Loop 4:");
        outtextxy(188,92,"Loop 5:"); outtextxy(308,28,"Aver :");
        outtextxy(308,44,"Data :"); outtextxy(308,60,"File :");
        outtextxy(308,76,"Total:"); outtextxy(308,92,"Run :");
        outtextxy(476,28,"Pts :"); outtextxy(476,44,"Dwt :");
        outtextxy(476,60,"Time :"); outtextxy(476,76,"Freq :");
        outtextxy(476,92,"Phase:"); setlinestyle(DASHED_LINE,1,NORM_WIDTH);
        setcolor(RED); line(3,108,635,108); line(180,20,180,107);
        line(300,20,300,107); line(468,20,468,107); break;
    case 1:
        /* sequence name */
        if(BUF==NULL) flag=0; else flag=1;
        if(flag==0) IER=LOAD_BUF(); if(IER>0) { PUTSETTINGS(); return; }
        bar(144,8,504,16); strcpy(str,SEQ); outtextxy(144,8,str);
        if(flag==0) IER=FREE_BUF(); if(IER>0) { PUTSETTINGS(); return; }
        break;
    case 2:
        /* change: yes/no */
        if(CHG==0) strcpy(str,"yes"); else strcpy(str,"no ");
        bar(600,8,624,16); outtextxy(600,8,str); break;
    case 3:
        /* filename */
        fnsplit(FILE_NAME,drive,dir,name,ext);
        sprintf(str,"%s%s",name,ext);
        bar(64,28,160,36); outtextxy(64,28,str); break;
    case 4:
        /* sequence data */
        if(BUF==NULL) flag=0; else flag=1;
        if(flag==0) IER=LOAD_BUF(); if(IER>0) { PUTSETTINGS(); return; }
        if(GET_EV(J)==3)
        { FCT_EXPR(&SPEC[15*J],str); bar(64,114,632,122);
          outtextxy(64,114,str); }
        else bar(64,114,632,122);
        if(GET_EV(J)==2)
        { TBL_EVAL(&SPEC[15*J],&TIME[J],TB,&n);
          if(n==0) strcpy(str," * ( * )");
          else sprintf(str,"%3d (%3d)",TB+1,n);
          bar(64,92,176,100); outtextxy(64,92,str); }
        else bar(64,92,176,100);
        if(TIME[J]!=0) STRTIME(str,TIME[J]); else strcpy(str,"*-----*");
        while(str[0]==' ') { for(m=0;m<strlen(str);m++) str[m]=str[m+1]; }
        bar(64,44,176,52); outtextxy(64,44,str); strcpy(str,&SPEC[15*J]);
        while(str[0]==' ') { for(m=0;m<strlen(str);m++) str[m]=str[m+1]; }
        bar(64,60,176,68); outtextxy(64,60,str);
        str[0]=TAG[J]; str[1]='\0'; bar(64,76,72,84); outtextxy(64,76,str);
        goto nm;
    case 5:
        /* loop and average counters */

```

```

if(BUF==NULL) flag=0; else flag=1;
if(flag==0) IER=LOAD_BUF(); if(IER>0) { PUTSETTINGS(); return; }
for(m=0;m<5;m++)
{ if(COUNT[m]!=0) sprintf(str,"%-u",COUNT[m]); else strcpy(str,"*---*");
  bar(252,28+16*m,292,100); outtextxy(252,28+16*m,str); }
if(NM!=0) sprintf(str,"%-u",NM); else strcpy(str,"*---*");
bar(364,28,460,36); outtextxy(364,28,str);
nm:
NM=0; j=0;
for(n=0;n<N;n++)
{ if(strcmp(&CODE[4*n],"085")==0)
  { if(j!=0)
    { WINDOW(1); setcolor(RED);
      outtextxy(16,4,"Error: duplicate nicolet trigger line");
      outtextxy(320,4,"(press any key to return)"); getch();
      bar(16,4,637,12); if(flag==0) IER=FREE_BUF();
      PUTSETTINGS(); return; }
    for(m=0;m<M;m++)
    { coluna=m/NC; bit=one>>(m%NC);
      if(GETCHAR(n,m)>5) /* low -> high transition at trig_nic (CP01) */
      { t=1;
        for(i=0;i<5;i++)
        { if((((LOOP[i*(M/NC)+coluna])&(bit))!=0) t=t*COUNT[i]; }
          if(j==0) { NM=t; j=1; } else NM=NM+t; } } }
      if(NM!=0) sprintf(str,"%-u",NM); else strcpy(str,"*-----*");
      bar(364,28,460,36); outtextxy(364,28,str);
      if(flag==0) IER=FREE_BUF(); if(IER>0) { PUTSETTINGS(); return; }
      break;
    }
  case 6:
    if(ADC<0) break;
    ibwrt(ADC,":STORAGE HOLDLAST",17);
    ibwrt(ADC,":WAVEFORM:LENGTH?",17); /* Ask for number of points */
    ibrd(ADC,str,80); CHS=atoi(str);
    sprintf(str,"%-d",CHS); bar(532,28,630,36); outtextxy(532,28,str);
    ibwrt(ADC,":CHANNEL1:RATE?",15); /* Ask for dwell time */
    ibrd(ADC,str,80); DWT=atof(str);
    sprintf(str,"%-9.4f us",DWT*1000000.0); /* dwell time */
    bar(532,44,630,52); outtextxy(532,44,str);
    sprintf(str,"%-9.4f ms",DWT*((float)CHS)*1000.0); /* time */
    bar(532,60,630,68); outtextxy(532,60,str);
    sprintf(str,"%-9.4f KHz",0.001/(2.0*DWT)); /* frequency */
    bar(532,76,630,84); outtextxy(532,76,str);
    sprintf(str,"%-9.4f",PHASE); /* phase */
    bar(532,92,630,100); outtextxy(532,92,str);
    ibwrt(ADC,":storage live",13); ibsre(ibfind("gpib0"),0); break;
  }
  PUTSETTINGS(); return;
}

/* ***** */

void KURSOR(n)
char n;
{
  register int j,c;

  GETSETTINGS(); WINDOW(9); setviewport(X1,Y1,X2,Y2,0);
  j=J-JJ; if(j%2==0) c=LIGHTGRAY; else c=WHITE;
  if(n>0) setfillstyle(SOLID_FILL,BLUE); else setfillstyle(SOLID_FILL,c);
  bar(16*j+4,18,16*j+11,19); PUTSETTINGS(); return; }

```



```

/* ***** */

int TIME_UPDATE(void)
{
    int j,n,flag;

    if(BUF==NULL) flag=0; else flag=1;
    if(flag==0) IER=LOAD_BUF(); if(IER>0) return(IER);
    for(j=0;j<M;j++)
    { if(SPEC[15*j]=='\0') strcpy(&SPEC[15*j],"*-----*");
      if(GET_EV(j)==0)
      { TIME[j]=0; strcpy(&SPEC[15*j],"*-----*"); TAG[j]='*'; } }
    for(j=0;j<M;j++)
    { if(GET_EV(j)==2) TBL_EVAL(&SPEC[15*j],&TIME[j],TB,&n); }
    for(j=0;j<M;j++) { if(GET_EV(j)==3) FCT_EVAL(&SPEC[15*j],&TIME[j]); }
    for(j=0;j<M;j++)
    { if(GET_EV(j)==8)
      { TIME[j]=10000; strcpy(&SPEC[15*j],"ADC+"); TAG[j]='*'; }
      if(GET_EV(j)==9)
      { TIME[j]=10000; strcpy(&SPEC[15*j],"ADC-"); TAG[j]='*'; } }
    if(flag==0) IER=FREE_BUF(); return(IER);
}

/* ***** */

/* ***** LINE.C ***** */

#include "psp_dcl.h"

/* ***** */

int LINE_EDT(void)
{
    FILE *chn;
    unsigned long int *pulse;
    char *label=NULL, *bnc=NULL, *code=NULL, str[80], ch[2];
    int i,j,ii,k,l,p,n=0,m=0,chg=0,ncode=72,nmax=100;
    char pskbnc[75][5]=
    { "PH00","PH01","CCHA","SSIG","HHLD","AADC","SP00","SP05","ETRG","CP00",
      "CP01","CP02","CP03","CP04","CP05","CP06","CP07","CP08","CP10","CP11",
      "CP12","CP13","CP14","CP15","GX00","GX01","GX02","GX03","GX04","GX05",
      "GX06","GX07","GX08","GX09","GX10","GX11","GY00","GY01","GY02","GY03",
      "GY04","GY05","GY06","GY07","GY08","GY09","GY10","GY11","GZ00","GZ01",
      "GZ02","GZ03","GZ04","GZ05","GZ06","GZ07","GZ08","GZ09","GZ10","GZ11",
      "PL00","PL01","PL02","PL03","PL04","PL05","PL06","PL07","PL08","PL09",
      "PL10","PL11" };
    char pskcode[73][4]=
    { "082","083","090","091","092","093","080","081","117","084",
      "085","086","087","088","089","094","095","107","118","119",
      "120","121","122","123","000","001","002","003","004","005",
      "006","007","008","009","010","011","012","013","014","015",
      "016","017","018","019","020","021","022","023","024","025",
      "026","027","028","029","030","031","032","033","034","035",
      "036","037","038","039","040","041","042","043","044","045",
      "046","047" };

    IER=GET_DEF(1); if(IER>0) return(IER);
    label=calloc(11*nmax,sizeof(char));
    if(!label) { strcpy(ERS,"Memory allocation error - label - (LINE_EDT)");
                IER=1; fprintf(OUT,"\n%d %s",IER,ERS); return(IER); }
}

```

```

bnc=calloc(5*nmax,sizeof(char));
if(!bnc) { strcpy(ERS,"Memory allocation error - bnc - (LINE_EDT)"); IER=2;
    fprintf(OUT,"\n%d %s",IER,ERS); free(label); return(IER); }
code=calloc(4*nmax,sizeof(char));
if(!code) { strcpy(ERS,"Memory allocation error - code - (LINE_EDT)");
    free(label); free(bnc); IER=3;
    fprintf(OUT,"\n%d %s",IER,ERS); return(IER); }
if((chn=fopen(DEF,"r"))==NULL)
{ sprintf(ERS,"WARNING: Channels file '%s' cannot be open (LINE_EDT)",DEF);
    IER=4; fprintf(OUT,"\n%d%s",IER,ERS); free(label); free(bnc);
    free(code); return(IER); }
for(;;)
{ if(n==nmax)
    { fprintf(OUT,"\nWARNING: Channels file will be truncated (LINE_EDT)");
        break; }
    if((fscanf(chn,"%s%s%s",&label[11*n],&bnc[5*n],&code[4*n]))==EOF) break;
    n=n+1; }
fclose(chn);
GETSETTINGS(); WINDOW(2); GETIMAGE(X1,Y1,X2,Y2); MOLDURA(); WINDOW(3);
GETIMAGE(X1,Y1,X2,Y2); MOLDURA(); MESSAGE(35,1); MESSAGE(36,1);
MESSAGE(37,1); MESSAGE(38,1); MESSAGE(39,1); MESSAGE(40,1);
MESSAGE(41,1); MESSAGE(42,1); MESSAGE(43,1); MESSAGE(44,1);
IER=LOAD_BUF(); if(IER>0)
{ free(label); free(bnc); free(code); WINDOW(3); IER=PUTIMAGE(X1,Y1,X2,Y2);
    WINDOW(2); IER=PUTIMAGE(X1,Y1,X2,Y2); PUTSETTINGS(); return(IER); }
settextstyle(0,0,1); setlinestyle(SOLID_LINE,0,NORM_WIDTH);
WINDOW(3); setcolor(RED); line(130,4,130,Y2-Y1-4); line(256,4,256,Y2-Y1-4);
line(382,4,382,Y2-Y1-4); line(508,4,508,Y2-Y1-4); setcolor(BLUE);
i=0; j=0; ii=0; l=1; p=0;
WRITE_LABELS(n,ii,i,j,label,bnc);
for(;;)
{ setcolor(BLUE);
    if(p!=0) { WINDOW(3); rectangle(5+126*i,9+16*j,129+126*i,21+16*j); }
    if(p==0) { WINDOW(6); rectangle(2,16*(1-II)+2,84,16*(1-II)+12); }
    setcolor(RED);
    if(p==0) { WINDOW(3); rectangle(5+126*i,9+16*j,129+126*i,21+16*j); }
    if(p!=0) { WINDOW(6); rectangle(2,16*(1-II)+2,84,16*(1-II)+12); }
    BIOSKEY(ch);
    if(M2==1)
    { if(MW==3)
        { if(p==1) { p=0; WINDOW(6); setcolor(WHITE);
            rectangle(2,16*(1-II)+2,84,16*(1-II)+12); }
            WINDOW(3); if((M3>X1+5)&&(M3<X2-5)&&(M4>Y1+9)&&(M4<Y2-9))
            { k=(M3-X1-5)/126; m=(M4-Y1-9)/16;
                if(7*(k+ii)+m<n)
                { setcolor(WHITE); rectangle(5+126*i,9+16*j,129+126*i,21+16*j);
                    i=k; j=m; } } }
            if(MW==6)
            { if(p==0) { p=1; WINDOW(3); setcolor(WHITE);
                rectangle(5+126*i,9+16*j,129+126*i,21+16*j); }
                WINDOW(6); setcolor(WHITE);
                rectangle(2,16*(1-II)+2,84,16*(1-II)+12); l=II+(M4-Y1-2)/16; }
            if(MW==2)
            { MESSAGE(35,-1); if((M3>X1)&&(M3<X2)&&(M4>Y1)&&(M4<Y2)) ch[1]=LEFT;
                MESSAGE(36,-1); if((M3>X1)&&(M3<X2)&&(M4>Y1)&&(M4<Y2)) ch[0]='r';
                MESSAGE(37,-1); if((M3>X1)&&(M3<X2)&&(M4>Y1)&&(M4<Y2)) ch[0]='c';
                MESSAGE(38,-1); if((M3>X1)&&(M3<X2)&&(M4>Y1)&&(M4<Y2)) ch[0]='d';
                MESSAGE(39,-1); if((M3>X1)&&(M3<X2)&&(M4>Y1)&&(M4<Y2)) ch[1]=RIGHT;
                MESSAGE(40,-1); if((M3>X1)&&(M3<X2)&&(M4>Y1)&&(M4<Y2)) ch[0]='i';
                MESSAGE(41,-1); if((M3>X1)&&(M3<X2)&&(M4>Y1)&&(M4<Y2)) ch[0]='s';

```

```

MESSAGE(42,-1); if((M3>X1)&&(M3<X2)&&(M4>Y1)&&(M4<Y2)) ch[0]='a';
MESSAGE(43,-1); if((M3>X1)&&(M3<X2)&&(M4>Y1)&&(M4<Y2)) ch[0]='w';
MESSAGE(44,-1); if((M3>X1)&&(M3<X2)&&(M4>Y1)&&(M4<Y2)) ch[0]='m'; } }
switch(ch[0])
{ case 'r':
    if(chg!=0) /* return */
    { strcpy(str,"Labels are not saved !");
      if(SAVE_ASK(str)==1) SAVE_LABELS(n,label,bnc,code); }
    WINDOW(6); setcolor(WHITE); rectangle(2,16*(1-II)+2,84,16*(1-II)+12);
    free(label); free(bnc); free(code); IER=FREE_BUF();
    WINDOW(3); IER=PUTIMAGE(X1,Y1,X2,Y2);
    if(IER>0) { PUTSETTINGS(); return(IER); }
    WINDOW(2); IER=PUTIMAGE(X1,Y1,X2,Y2); PUTSETTINGS(); return(IER);
  case 's': /* save */
    SAVE_LABELS(n,label,bnc,code); chg=0; break;
  case 'c': /* copy */
    if(p==0)
    { if(n==0) break;
      CHG=1; WINDOW(6); sprintf(str,"%10.10s",&NAME[11*1]);
      setcolor(WHITE); outtextxy(4,16*(1-II)+4,str);
      strcpy(&NAME[11*1],&label[11*(7*(i+ii)+j)]);
      strcpy(&LINE[ 5*1],&bnc [ 5*(7*(i+ii)+j)]);
      strcpy(&CODE[ 4*1],&code [ 4*(7*(i+ii)+j)]);
      sprintf(str,"%10.10s",&NAME[11*1]);
      setcolor(BLUE); outtextxy(4,16*(1-II)+4,str); break; }
    if(p==1)
    { if(n==0) break; if(n==nmax) break;
      chg=1; setcolor(WHITE); WINDOW(3);
      rectangle(5+126*i,9+16*j,129+126*i,21+16*j);
      strcpy(&label[11*n],&NAME[11*1]); strcpy(&bnc[5*n],&LINE[5*1]);
      strcpy(&code[4*n],&CODE[4*1]); n=n+1; chg=1;
      if(ii!=n/7-4)
      { ii=(n-1)/7-4; if(ii<0) ii=0; WRITE_LABELS(n,ii,0,0,label,bnc); }
      i=(n-1)/7-ii; j=(n-1)%7; break; }
    break;
  case 'd': /* delete */
    if(p==0)
    { if(n==0) break;
      setfillstyle(SOLID_FILL,WHITE); WINDOW(3);
      bar(5+126*i,9+16*j,129+126*i,21+16*j);
      for(k=7*(i+ii)+j;k<n-1;k++)
      { strcpy(&label[11*k],&label[11*(k+1)]);
        strcpy(&bnc [ 5*k],&bnc [ 5*(k+1)]);
        strcpy(&code [ 4*k],&code [ 4*(k+1)]); }
      n=n-1; if(7*(i+ii)+j==n) { if(j>0) j=j-1; else { j=6; i=i-1; } }
      if(n==0) { i=0; j=0; }
      WRITE_LABELS(n,ii,i,j,label,bnc); chg=1; break; }
    if(p==1)
    { CHG=1; WINDOW(6); setcolor(WHITE);
      sprintf(str,"%10.10s",&NAME[11*1]); outtextxy(4,16*(1-II)+4,str);
      strcpy(&NAME[11*1],"*-----*"); strcpy(&LINE[5*1],"*---*");
      strcpy(&CODE[4*1],"*-"); sprintf(str,"%10.10s",&NAME[11*1]);
      setcolor(BLUE); outtextxy(4,16*(1-II)+4,str); break; }
  case 'i': /* insert */
    if(p==1) break; if(n==0) break; if(n==nmax) break;
    m=7*(i+ii)+j; for(k=n;k>m;k--)
    { strcpy(&label[11*k],&label[11*(k-1)]);
      strcpy(&bnc [ 5*k],&bnc [ 5*(k-1)]);
      strcpy(&code [ 4*k],&code [ 4*(k-1)]); }
    n=n+1; chg=1; WRITE_LABELS(n,ii,i,j,label,bnc); k=0;

```

```

ins:
HIDE_MOUSE(); setfillstyle(SOLID_FILL,WHITE); setlinestyle(0,0,1);
WINDOW(2); GETIMAGE(X1,Y1,X2,Y2); MOLDURA(); setcolor(RED);
outtextxy(16,16,"Type your line label"); setcolor(BLUE);
outtextxy(16,32,
"You can use keys: right, left, backspace, delete, insert");
outtextxy(16,48,"Press 'enter' when you are finished");
WINDOW(3); bar(5+126*i,9+16*j,129+126*i,21+16*j);
setcolor(RED); rectangle(5+126*i,9+16*j,89+126*i,21+16*j);
INPUT(7+126*i,11+16*j,&label[11*m],10,10,k,1);
REM_BLANK(&label[11*m]);
if(label[11*m]=='\0') strcpy(&label[11*m],"*-----*");
sprintf(str,"%10.10s",&label[11*m]);
setcolor(BLUE); outtextxy(7+126*i,11+16*j,str);
setcolor(WHITE); rectangle(5+126*i,9+16*j,89+126*i,21+16*j);
setcolor(RED); rectangle(93+126*i,9+16*j,129+126*i,21+16*j);
WINDOW(2); bar(6,6,X2-X1-6,Y2-Y1-6); setcolor(RED);
outtextxy(16,16,"Enter your line code"); setcolor(BLUE);
outtextxy(16,32,
"You can use keys: up (or right), down (or left)");
outtextxy(16,48,"Press 'enter' when you are finished");
for(k=0;k<ncode;k++) { if(strcmp(&bnc[5*m],pskbnc[k])==0) break; }
if(k==ncode) k=0;
WINDOW(3); setcolor(BLUE); outtextxy(95+126*i,11+16*j,pskbnc[k]);
for(;;)
{ setcolor(BLUE); outtextxy(95+126*i,11+16*j,pskbnc[k]);
  BIOSKEY(ch); if(ch[0]=='\r') break;
  switch(ch[1])
  { case RIGHT: case UP:
    setcolor(WHITE); outtextxy(95+126*i,11+16*j,pskbnc[k]);
    if(k<ncode-1) k=k+1; else k=0; break;
    case LEFT: case DOWN:
    setcolor(WHITE); outtextxy(95+126*i,11+16*j,pskbnc[k]);
    if(k>0) k=k-1; else k=ncode-1; break; } }
  setcolor(WHITE); rectangle(93+126*i,9+16*j,129+126*i,21+16*j);
  strcpy(&bnc[5*m],pskbnc[k]); strcpy(&code[4*m],pskcode[k]);
  sprintf(str,"%10.10s:%4.4s",&label[11*m],&bnc[5*m]);
  setcolor(BLUE); outtextxy(7+126*i,11+16*j,str); SHOW_MOUSE();
  ch[0]='0'; ch[1]='0'; WINDOW(2); PUTIMAGE(X1,Y1,X2,Y2); break;
case 'a':
  if(p==1) break; if(n==0) break; if(n==nmax) break; /* append */
  setcolor(WHITE); WINDOW(3);
  rectangle(5+126*i,9+16*j,129+126*i,21+16*j);
  label[11*n]='\0'; bnc[5*n]='\0'; code[4*n]='\0'; n=n+1; chg=1;
  if(ii!=n/7-4)
  { ii=(n-1)/7-4; if(ii<0) ii=0; WRITE_LABELS(n,ii,0,0,label,bnc); }
  i=(n-1)/7-ii; j=(n-1)%7; m=7*(i+ii)+j; k=0; goto ins;
case 'w':
  if(p==1) break; m=7*(i+ii)+j; chg=1; /* write */
  if(n==0) { n=n+1; k=0; } else k=1; goto ins;
case 'm':
  WINDOW(3); setcolor(WHITE); /* move */
  rectangle(5+126*i,9+16*j,129+126*i,21+16*j);
  WINDOW(6); rectangle(2,16*(1-II)+2,84,16*(1-II)+12);
  if(p==0) { p=1; break; } if(p==1) { p=0; break; }
  break;
}
switch(ch[1])
{ case RIGHT:
  if(p==1) break; if(n==0) break; if(i+ii==(n-1)/7) break; WINDOW(3);

```

```

    if(i<4)
    { setcolor(WHITE); rectangle(5+126*i,9+16*j,129+126*i,21+16*j);
      i=i+1; if(7*(ii+i)+j+1>n) j=(n-1)%7; break; }
    if(i==4)
    { ii=ii+1; if(7*(i+ii)+j+1>n)
      { setcolor(WHITE); rectangle(5+126*i,9+16*j,129+126*i,21+16*j);
        j=(n-1)%7; }
      WRITE_LABELS(n,ii,0,0,label,bnc); break; }
    break;
case LEFT:
    if(p==1) break; if(n==0) break; if(i+ii==0) break; WINDOW(3);
    if(i>0)
    { setcolor(WHITE); rectangle(5+126*i,9+16*j,129+126*i,21+16*j);
      i=i-1; break; }
    if(i==0) { ii=ii-1; WRITE_LABELS(n,ii,0,0,label,bnc); break; }
    break;
case UP:
    if(p==0)
    { if(n==0) break; if(j==0) break; WINDOW(3); setcolor(WHITE);
      rectangle(5+126*i,9+16*j,129+126*i,21+16*j); j=j-1; break; }
    if(p==1)
    { if(l==II) break; WINDOW(6); setcolor(WHITE);
      rectangle(2,16*(l-II)+2,84,16*(l-II)+12); l=l-1; break; }
    break;
case DOWN:
    if(p==0)
    { if(n==0) break; if(j==6) break; if(7*(ii+i)+j>n-1) break;
      WINDOW(3); setcolor(WHITE);
      rectangle(5+126*i,9+16*j,129+126*i,21+16*j); j=j+1; break; }
    if(p==1)
    { if(l==II+NL-1) break; WINDOW(6); setcolor(WHITE);
      rectangle(2,16*(l-II)+2,84,16*(l-II)+12); l=l+1; break; }
    break;
  }
}
}

/* ***** */

void WRITE_LABELS(n,ii,i,j,label,bnc) /* reescreve a tabela de n ele- */
int n,ii,i,j; /* mentos, a partir do (i,j), */
char *label, *bnc; /* apagando os labels da tabela */
{ /* antiga. */
    char str[80];
    int k,m,l;

    GETSETTINGS(); WINDOW(3);
    setfillstyle(SOLID_FILL,WHITE); settextstyle(0,0,1);
    for(l=7*(i+ii)+j;l<7*(i+ii+5);l++)
    { k=l/7-ii; m=l%7; bar(7+126*k,11+16*m,127+126*k,19+16*m);
      if(l<n) { sprintf(str,"%10.10s:%4.4s",&label[l*1],&bnc[5*1]);
        setcolor(BLUE); outtextxy(7+126*k,11+16*m,str); } }
    PUTSETTINGS(); return;
}

/* ***** */

void SAVE_LABELS(n,label,bnc,code)
int n;
char *label, *bnc, *code;

```

```

{
    FILE *chn;
    char str[80];
    int i;

    IER=GET_DEF(1); if(IER>0) return;
    if((chn=fopen(DEF,"w"))!=NULL)
    { for(i=0;i<n;i++)
        { sprintf(str,"%10.10s %4.4s %3.3s",&label[11*i],&bnc[5*i],&code[4*i]);
          fprintf(chn,"%s\n",str); }
        fclose(chn); }
    else
    { sprintf(str,"%s' cannot be open, labels not saved (SAVE_LABELS)",DEF);
      fprintf(OUT,"\n%d %s",str); }
    return;
}

/* ***** */

int CHK_CHN(void)
{
    FILE *chn;
    int i;
    char name[11],line[5],code[4];

    IER=GET_DEF(1); if(IER>0) return(IER);
    if((chn=fopen(DEF,"r"))==NULL)
    { fprintf(OUT,"\nWARNING: Channels file cannot be find - CHK_CHN");
      fprintf(OUT,"\n          A new channels file will be created");
      if((chn=fopen(DEF,"w"))==NULL)
      { strcpy(ERS,"Failure trying to create a new channels file - CHK_CHN");
        IER=1; fprintf(OUT,"\n%d%s",IER,ERS); return(IER); }
      for(i=20;i<100;i++)
      { if(GET_DEF(i)==0)
          { sscanf(DEF," %s %s %s",name,line,code);
            sprintf(DEF,"%10.10s %4.4s %3.3s",name,line,code);
            fprintf(chn,"%s\n",DEF); } } }
      fclose(chn); return(IER);
}

/* ***** */

/* ***** PSP_AUX.C ***** */

#include "psp_dcl.h"

/* ***** */

int DIR_LIST(x,y,filename) /* x,y : canto superior esquerdo (entrada) */
int x,y; /* filename : nome da file escolhida (saida) */
char *filename; /* cabecalho da janela (entrada) */
{ /* retorno : 0 = ok */

    struct ffbblk ffbblk;
    int i,j,k,ii,jj,n,x1,y1,x2,y2;
    char str[80],ch[2], *ff,path[80],drive[3],dir[66],file[9],ext[5];

    IER=GET_DEF(4); if(IER>0) return(IER);
    GETSETTINGS(); x1=x; y1=y; x2=x+570; y2=y+240;
    IER=GETIMAGE(x1,y1,x2,y2); if(IER>0) { PUTSETTINGS(); return(IER); }
    setviewport(x1,y1,x2,y2,1); MOLDURA();

```

```

SET_MOUSE_WINDOW(x1+5,y1+48,x2-5,y2-5); SET_MOUSE_XY((x1+x2)/2,(y1+y2)/2);
settextstyle(0,0,1); setcolor(BLUE);
strcat(filename," directory listing of:"); outtextxy(12,8,filename);
setcolor(RED); setlinestyle(SOLID_LINE,1,NORM_WIDTH);
rectangle(7,25,563,39); strcpy(path,DEF); strcat(path,"\\*.seq");
begin:
i=fnsplit(path,drive,dir,file,ext);
if(!(i&FILENAME)) strcpy(file,""); if(!(i&EXTENSION)) strcpy(ext,".");
fnmerge(path,drive,dir,file,ext); setcolor(BLUE);
sprintf(str,"%58s",path); outtextxy(12,29,path);
n=0; k=findfirst(path,&ffblk,0);
if(k!=0)
{ setcolor(BLUE); settextstyle(0,0,2); outtextxy(173,144,"No files found");
  outtextxy(101,180,"Press any key to return"); getch();
  PUTIMAGE(x1,y1,x2,y2); PUTSETTINGS(); return(1ER); }
while(k==0) { n=n+1; k=findnext(&ffblk); }
ff=calloc(13*n,sizeof(char));
if(!ff) { strcpy(ERS,"Memory allocation error - ff - (DIR_LIST)");
  fprintf(OUT,"\n\n%s",ERS); 1ER=1;
  1ER=PUTIMAGE(x1,y1,x2,y2); PUTSETTINGS(); return(1ER); }
setcolor(RED); line(5,48,565,48); line(5,212,565,212); line(477,26,477,38);
line(117,48,117,212); line(229,48,229,212); line(341,48,341,212);
line(453,48,453,212); setcolor(BLUE); sprintf(str,"%3d files",n);
outtextxy(485,29,str); outtextxy(20,221," <<"); rectangle(12,217,60,231);
setcolor(BLUE); outtextxy(108,221,"Open"); rectangle(100,217,148,231);
setcolor(RED); outtextxy(108,221,"O"); setcolor(BLUE);
outtextxy(208,221,"New directory"); rectangle(200,217,320,231);
setcolor(RED); outtextxy(208,221,"N"); setcolor(BLUE);
outtextxy(388,221,"Cancel"); rectangle(380,217,444,231);
setcolor(RED); outtextxy(388,221,"C"); setcolor(BLUE);
outtextxy(518,221," >>"); rectangle(510,217,558,231); setcolor(BLUE);
i=0; j=0; k=findfirst(path,&ffblk,0);
while(k==0)
{ strcpy(&ff[13*(10*i+j)],ffblk.ff_name);
  if(10*i+j<50) outtextxy(i*112+13,j*16+56,&ff[13*(10*i+j)]);
  k=findnext(&ffblk); j=j+1; if(j>9) { j=0; i=i+1; } }
i=0; j=0; ii=0;
for(;;)
{ setcolor(RED); rectangle(i*112+11,j*16+54,i*112+111,j*16+66);
  BIOSKEY(ch);
  if(M2==1)
  { if((M3>x1+5)&&(M3<x2-5)&&(M4>y1+52)&&(M4<y1+212))
    { setcolor(WHITE); rectangle(i*112+11,j*16+54,i*112+111,j*16+66);
      i=(M3-x1-5)/112; j=(M4-y1-52)/16; }
    else
    { if((M3>x1+12)&&(M3<x1+60)&&(M4>y1+217)&&(M4<y1+231)) ch[1]=LEFT;
      if((M3>x1+510)&&(M3<x1+558)&&(M4>y1+217)&&(M4<y1+231)) ch[1]=RIGHT;
      if((M3>x1+100)&&(M3<x1+148)&&(M4>y1+217)&&(M4<y1+231)) ch[0]='o';
      if((M3>x1+200)&&(M3<x1+320)&&(M4>y1+217)&&(M4<y1+231)) ch[0]='n';
      if((M3>x1+380)&&(M3<x1+444)&&(M4>y1+217)&&(M4<y1+231)) ch[0]='c'; } }
  switch(ch[1])
  { case UP:
    if(j==0) break;
    setcolor(WHITE); rectangle(i*112+11,j*16+54,i*112+111,j*16+66);
    j=j-1; break;
    case DOWN:
    if(j==9) break;
    if(10*(ii+i)+j>=n-1) break;
    setcolor(WHITE); rectangle(i*112+11,j*16+54,i*112+111,j*16+66);
    j=j+1; break;
  }
}

```

```

case RIGHT:
    if(i+ii==(n-1)/10) break;
    if(i<4)
    { setcolor(WHITE); rectangle(i*112+11,j*16+54,i*112+111,j*16+66);
      i=i+1; if(10*(ii+i)+j>=n) j=(n-1)%10; break; }
    if(i==4)
    { setfillstyle(SOLID_FILL,WHITE); ii=ii+1; setcolor(BLUE);
      for(k=0;k<5;k++)
      { bar(k*112+11,54,k*112+111,206);
        for(jj=0;(jj<10)&&(10*(ii+k)+jj<n);jj++)
        { outtextxy(k*112+13,jj*16+56,&ff[13*(10*(ii+k)+jj)]); } }
      if(10*(ii+4)+j>=n)
      { setcolor(WHITE); rectangle(i*112+11,j*16+54,i*112+111,j*16+66);
        j=(n-1)%10; } break; }
case LEFT:
    if(i+ii==0) break;
    if(i>0)
    { setcolor(WHITE); rectangle(i*112+11,j*16+54,i*112+111,j*16+66);
      i=i-1; break; }
    if(i==0)
    { setfillstyle(SOLID_FILL,WHITE); ii=ii-1; setcolor(BLUE);
      for(k=0;k<5;k++)
      { bar(k*112+11,54,k*112+111,206);
        for(jj=0;(jj<10)&&(10*(ii+k)+jj<n);jj++)
        outtextxy(k*112+13,jj*16+56,&ff[13*(10*(ii+k)+jj)]); } break; }
}
switch(ch[0])
{
    case 'n':
        /* new */
        setfillstyle(SOLID_FILL,WHITE); bar(5,48,565,235); bar(485,29,557,37);
        free(ff); INPUT(12,29,path,79,58,0,1); goto begin;
    case 'o': case '\r':
        fnsplit(path,drive,dir,file,ext);
        sprintf(filename,"%s%s%s",drive,dir,&ff[13*(10*(ii+i)+j)]);
        free(ff); IER=PUTIMAGE(x1,y1,x2,y2); PUTSETTINGS();
        SET_MOUSE_WINDOW(0,0,639,479); SET_MOUSE_XY(320,240); return(IER);
    case 'c':
        filename[0]='\0'; free(ff); IER=PUTIMAGE(x1,y1,x2,y2); PUTSETTINGS();
        SET_MOUSE_WINDOW(0,0,639,479); SET_MOUSE_XY(320,240); return(IER);
}
}
}

/* ***** */

void BIOSKEY(str)
char *str;
{
    union { char ch[2]; int q; } u;

    begin:
    if(IER<0) { M2=0; str[0]=0; str[1]=0; return; }
    while(bioskey(1)) bioskey(0); while(M2!=0) GET_MOUSE_STATUS();
    SHOW_MOUSE(); u.q=0;
    for(;;)
    { if(ITR>=0)
      { if((ITR=((inportb(DADOS_C)&224)/32))>0)
        { M2=0; str[0]=0; str[1]=0; return; } }
      if(bioskey(1)) { u.q=bioskey(0); break; }
      else { GET_MOUSE_STATUS(); if(M2!=0) break; } }
}

```



```

HIDE_MOUSE();
switch(u.ch[1])
{ case F1:
    HFLP(); goto begin;
  case F2:
    IER = SAVE_DATA(0); DATA_WINDOW(2); goto begin;
  case F3:
    COLOR(0); goto begin; /* branco e preto */
  case F4:
    COLOR(1); goto begin; /* colorido normal */
}
str[0]=tolower(u.ch[0]); str[1]=u.ch[1]; return;
}

/* ***** */

void COLOR(n)
int n;
{
  int i;

  switch(n)
  { case 0: /* branco e preto */
      for(i=1;i<15;i++) setpalette(i,63); setpalette(7,0);
      setrgbpalette(0,0,0,0); setrgbpalette(63,63,63,63); break;
    case 1: /* colorido normal */
      setrgbpalette(63,0,0,0); setrgbpalette(0,63,63,63);
      setpalette( 1, 1); setpalette( 2, 2); setpalette( 3, 3);
      setpalette( 4, 4); setpalette( 5, 5); setpalette( 6,20);
      setpalette( 7, 7); setpalette( 8,56); setpalette( 9,57);
      setpalette(10,58); setpalette(11,59); setpalette(12,60);
      setpalette(13,61); setpalette(14,62); break;
    }
  return;
}

/* ***** */

void MOLDURA(void)
{
  GETSETTINGS();
  clearviewport(); setlinestyle(SOLID_LINE,1,THICK_WIDTH); setcolor(BLUE);
  rectangle(1,1,(OLDVIEW[SETLEVEL].right-OLDVIEW[SETLEVEL].left-1),
            (OLDVIEW[SETLEVEL].bottom-OLDVIEW[SETLEVEL].top-1));
  PUTSETTINGS(); return;
}

/* ***** */

int GET_DEF(n) /* Atualiza a string DEF (externa) com a infor- */
int n; /* macao contida na linha numero n do arquivo */
{ /* psp.def localizado no diretorio corrente. */
  FILE *def; /* retorna : 0 : ok */
  char *ptr; /* 1 : nao pode abrir psp.def */
  unsigned long int i; /* 2 : nao encontrou linha n */
  /*

  if(n<=0) { DEF[0]='\0'; return(2); }
  getcwd(DEF,68); strcat(DEF,"\\psp.def");
  if((def=fopen(DEF,"r"))==NULL)

```

```

{ sprintf(ERS,"Default file '%s' cannot be open - GET_DEF",DEF);
  IER=1; DEF[0]='\0'; fprintf(OUT,"\n%d %s",IER,ERS); return(IER); }
for(;;)
{ if(fgets(DEF,80,def)==NULL) { DEF[0]='\0'; fclose(def); return(2); }
  if((ptr=strchr(DEF,'\n'))!=NULL) DEF[ptr-DEF]='\0';
  if((ptr=strchr(DEF,'*'))!=NULL) DEF[ptr-DEF]='\0';
  i=strtoul(DEF,&ptr,10);
  if((i!=0)&&(ptr!=NULL))
  { if((i-(unsigned long int)n)==0)
    { for(i=0;i<ptr-DEF;i++) DEF[i]=' '; REM_BLANK(DEF); break; } } }
  fclose(def); return(IER);
}

/* ***** */

void INPUT(x,y,str,m,n,new,s) /* m : no. total de caracteres da string */
int x,y,m,n,new,s;          /* n : no. de caracteres do visor */
char *str;                  /* new : 1 - atualiza uma string */
{                             /* 0 - inicia uma nova string */
  char c[2];                /* s : tamanho do caracter (1,2,3...) */
  int i,j,k;
  union { char ch[2]; int t; } u;

  GETSETTINGS();
  s=s*8; /* s: tamanho do caracter (multiplo de 8*8 pixels) */
  if(n>m) n=m;
  setviewport(OLDVIEW[SETLEVEL].left+x,OLDVIEW[SETLEVEL].top+y,
              OLDVIEW[SETLEVEL].left+x+s*n-1,OLDVIEW[SETLEVEL].top+y+s-1,1);
  settextstyle(0,0,s/8); setcolor(BLUE);
  setfillstyle(SOLID_FILL,WHITE); bar(s,0,s*n-1,s-1); /* fundo */
  setfillstyle(SOLID_FILL,DARKGRAY); bar(0,0,s-1,s-1); /* cursor */
  if(new==1) for(j=0;j<m;j++) if(str[j]!='\0') { str[j]=' '; str[j+1]='\0'; }
  if(new==0) for(j=0;j<m;j++) str[j]=' '; str[m]='\0';
  outtextxy(0,0,str); i=0; k=0;
  for(;;)
  { while(bioskey(1)) bioskey(0);
    for(;;)
    { setfillstyle(SOLID_FILL,DARKGRAY); bar(s*(i-k),0,s*(i-k+1)-1,s-1);
      c[0]=str[i]; c[1]='\0'; outtextxy(s*(i-k),0,c);
      delay(100); if(bioskey(1)) break;
      setfillstyle(SOLID_FILL,WHITE); bar(s*(i-k),0,s*(i-k+1)-1,s-1);
      c[0]=str[i]; c[1]='\0'; outtextxy(s*(i-k),0,c);
      delay(100); if(bioskey(1)) break; }
    u.t=bioskey(0); if(u.ch[0]=='\r') break;
    switch(u.ch[1])
    {
      case LEFT:
        if(i>0)
        { i=i-1;
          if(i<k)
          { k=k-1;
            setfillstyle(SOLID_FILL,WHITE); bar(s,0,s*n-1,s-1);
            setfillstyle(SOLID_FILL,DARKGRAY); bar(0,0,s-1,s-1);
            outtextxy(0,0,&str[k]); }
          else
          { setfillstyle(SOLID_FILL,WHITE); bar(s*(i-k+1),0,s*(i-k+2)-1,s-1);
            c[0]=str[i+1]; c[1]='\0'; outtextxy(s*(i-k+1),0,c);
            setfillstyle(SOLID_FILL,DARKGRAY);
            bar(s*(i-k),0,s*(i-k+1)-1,s-1);
            c[0]=str[i]; c[1]='\0'; outtextxy(s*(i-k),0,c); } } }
    }
  }
}

```

```

break;
case RIGHT:
    if(i<m-1)
    { i=i+1;
      if(i>k+n-1)
      { k=k+1;
        setfillstyle(SOLID_FILL,WHITE); bar(0,0,s*(n-1)-1,s-1);
        setfillstyle(SOLID_FILL,DARKGRAY); bar((n-1)*s,0,n*s-1,s-1);
        outtextxy(0,0,&str[k]); }
      else
      { setfillstyle(SOLID_FILL,WHITE); bar(s*(i-k-1),0,s*(i-k)-1,s-1);
        c[0]=str[i-1]; c[1]='\0'; outtextxy(s*(i-k-1),0,c);
        setfillstyle(SOLID_FILL,DARKGRAY);
        bar(s*(i-k),0,s*(i-k+1)-1,s-1);
        c[0]=str[i]; c[1]='\0'; outtextxy(s*(i-k),0,c); } }
    break;
case INSERT:
    for(j=m-1;j>i;j--) str[j]=str[j-1]; str[i]=' ';
    setfillstyle(SOLID_FILL,WHITE); bar((i-k+1)*s,0,s*n-1,s-1);
    setfillstyle(SOLID_FILL,DARKGRAY); bar((i-k)*s,0,(i-k+1)*s-1,s-1);
    outtextxy((i-k)*s,0,&str[i]); break;
case DELETE:
    for(j=i;j<m;j++) str[j]=str[j+1]; str[m-1]=' ';
    setfillstyle(SOLID_FILL,WHITE); bar((i-k+1)*s,0,s*n-1,s-1);
    setfillstyle(SOLID_FILL,DARKGRAY); bar((i-k)*s,0,(i-k+1)*s-1,s-1);
    outtextxy((i-k)*s,0,&str[i]); break;
default:
    if(u.ch[0]==BACK)
    { if(i<=0) break;
      for(j=i;j<m;j++) str[j-1]=str[j]; str[m-1]=' '; i=i-1;
      if(i<k)
      { k=k-1;
        setfillstyle(SOLID_FILL,WHITE); bar(s,0,s*n-1,s-1);
        setfillstyle(SOLID_FILL,DARKGRAY); bar(0,0,s-1,s-1);
        outtextxy(0,0,&str[k]); }
      else
      { setfillstyle(SOLID_FILL,WHITE); bar((i-k+1)*s,0,s*n-1,s-1);
        setfillstyle(SOLID_FILL,DARKGRAY);
        bar((i-k)*s,0,(i-k+1)*s-1,s-1);
        outtextxy((i-k)*s,0,&str[i]); } break; }
    if(u.ch[0]) str[i]=u.ch[0]; else break;
    if(i>=m-1)
    { setfillstyle(SOLID_FILL,DARKGRAY); bar((i-k)*s,0,(i-k+1)*s-1,s-1);
      c[0]=str[i]; c[1]='\0'; outtextxy(s*(i-k),0,c); }
    else
    { i=i+1;
      if(i-k<n)
      { setfillstyle(SOLID_FILL,WHITE); bar((i-k-1)*s,0,(i-k)*s-1,s-1);
        c[0]=str[i-1]; c[1]='\0'; outtextxy(s*(i-k-1),0,c);
        setfillstyle(SOLID_FILL,DARKGRAY);
        bar((i-k)*s,0,(i-k+1)*s-1,s-1);
        c[0]=str[i]; c[1]='\0'; outtextxy(s*(i-k),0,c); }
      else
      { k=k+1;
        setfillstyle(SOLID_FILL,WHITE); bar(0,0,s*(n-1)-1,s-1);
        setfillstyle(SOLID_FILL,DARKGRAY); bar((n-1)*s,0,n*s-1,s-1);
        outtextxy(0,0,&str[k]); } } break; } }
i=0; while(str[i]==' ') i=i+1;
j=m-1; while(str[j]==' ') { str[j]='\0'; j=j-1; } m=j+1;
for(j=i;j<m;j++) if(str[j]==' ')

```

```

        { for(k=j;k<m;k++) str[k]=str[k+1]; j=j-1; m=m-1; }
    setfillstyle(SOLID_FILL,WHITE); bar(0,0,s*n-1,s-1);
    PUTSETTINGS(); return;
}

/* ***** */

int SIZEARQ(arq,nl,nc) /* determina no. de linhas e colunas do arquivo arq */
FILE *arq;             /* nl : ponteiro para numero de linhas */
int *nl,*nc;           /* nc : ponteiro para numero de colunas */
{
    /* retorna 0 se nao houver erros */
    char str[80];       /* retorna -1 se todas as linhas nao tiverem o */
    int i,j,ns,first;   /* mesmo numero de strings. */

    rewind(arq); *nl=0;
    while(fgets(str,80,arq)!=NULL)
    { *nl=*nl+1; ns=strlen(str)-1; str[ns]='\0'; j=1; *nc=0;
      for(i=0;i<ns;i++)
      { if(isspace(str[i])==0) { if(j>0) { *nc=*nc+1; j=-1; } }
        else j=1; }
      if(*nl==1) first=*nc;
      else
        if(*nc!=first)
        { strcpy(ERS,"Format error (SIZE_ARQ)"); return (-1); } }
    rewind(arq); return(0);
}

/* ***** */

void REM_BLANK(str) /* remove os caracteres brancos do inicio e do */
char *str;         /* final da string str. */
{
    int n,i;

    n=strlen(str);
    while(str[0]==' ') { for(i=0;i<n;i++) str[i]=str[i+1]; n=n-1; }
    while((n>0)&&(str[n-1]==' ')) { str[n-1]='\0'; n=n-1; }
    return;
}

/* ***** */

/* ***** TIME.C ***** */

#include "psp_dcl.h"

/* ***** */

void STRTIME(str,t) /* input t: tempo em multiplos de 100 ns */
unsigned long int t; /* output str */
char *str;
{
    unsigned int tt;

    if(t>=100000000)
    { tt=t/100000000; t=t%100000000; sprintf(str,"%3u.%07lu sg",tt,t); return; }
    if(t>=10000)
    { tt=t/10000; t=t%10000; sprintf(str,"%6u.%04lu ms",tt,t); return; }
    tt=t/10; t=t%10; sprintf(str,"%9u.%01lu us",tt,t); return;
}

```

```

/* ***** */

void INTTIME(str,t)      /* output t: tempo em multiplos de 100 ns */
char *str;              /* input str (modificada na execucao) */
unsigned long int *t;
{
    char st[80],*last;
    int m,i;

    if(strstr(str,"sg")!=NULL) m=7;
    if(strstr(str,"ms")!=NULL) m=4;
    if(strstr(str,"us")!=NULL) m=1;
    *t=strtoul(str,&last,10); sprintf(st,"%lu",*t); str=last+1;
    *t=strtoul(str,&last,10); *last='\0';
    if(strlen(str)<m+1) for(i=strlen(str);i<m;i++) strcat(str,"0");
    else str[m]='\0';
    strcat(st,str); *t=strtoul(st,&last,10); if(*t<5) *t=5;
    return;
}

/* ***** */

void INPUTIME(str,m,x,y) /* input str : string a ser modificada (m=1) */
char *str;              /*          lixo (m=0) */
int m,x,y;              /* output str: tempo no formato n.nnn xx */
{                        /* m=0 : new      m=1 : modifica */
    char ch[3];          /* x,y : posicao na tela (relativa ao ultimo */
    int i,s;             /* viewport) */
    unsigned long int t;
    char unit[3][3] = { "sg","ms","us" };

    GETSETTINGS(); HIDE_MOUSE();
    s=8; /* s: tamanho do caracter (multiplo de 8*8 pixels) */
    setviewport(OLDVIEW[SETLEVEL].left+x,OLDVIEW[SETLEVEL].top+y,
                OLDVIEW[SETLEVEL].left+x+14*s-1,OLDVIEW[SETLEVEL].top+y+s-1,1);
    INPUT(0,0,str,11,11,m,1); setcolor(BLUE); outtextxy(0,0,str);
    i=0; setcolor(BLUE); outtextxy(12*s,0,unit[i]);
    for(;;)
    { setcolor(BLUE); outtextxy(12*s,0,unit[i]);
      BIOSKEY(ch); if(ch[0]=='\r') break;
      switch(ch[1])
      { case UP:
          setcolor(WHITE); outtextxy(12*s,0,unit[i]); if(i<2) i=i+1; else i=0;
          break;
        case DOWN :
          setcolor(WHITE); outtextxy(12*s,0,unit[i]); if(i>0) i=i-1; else i=2;
          break; } }
    strcat(str," "); strcat(str,unit[i]); INTIME(str,&t); STRTIME(str,t);
    clearviewport(); PUTSETTINGS(); SHOW_MOUSE(); return;
}

/* ***** */

/* ***** TBL.C ***** */

#include "psp_dcl.h"

/* ***** */

int TBL_EDT(choice)      /* choice (saida): titulo da tabela apontada pelo */

```

```

char *choice:          /* cursor no momento de retorno.          */
{
    FILE *tbl;
    int i,ii,j,jj,isave,jsave,k,l,q,r,n=0,nmax=100,chg=0,ns=0,*size,buflag;
    char *titul,*sel,ch[2],str[80];
    unsigned long int t,*tempo;

    if(BUF==NULL) buflag=0;
    else { IER=FREE_BUF(); if(IER>0) return(IER); buflag=1; }
    IER=GET_DEF(2); if(IER>0) { if(buflag==1) IER=LOAD_BUF(); return(IER); }
    titul=calloc(15*nmax,sizeof(char));
    if(!titul)
    { strcpy(ERS,"Memory allocation error - titul - (TBL_EDT)");
      IER=1; fprintf(OUT,"\n%d %s",IER,ERS);
      if(buflag==1) IER=LOAD_BUF(); return(IER); }
    if((tbl=fopen(DEF,"r"))==NULL)
    { sprintf(ERS,"WARNING: Table file '%s' could not be open (TBL_EDT)",DEF);
      IER=2; fprintf(OUT,"\n%d%s",IER,ERS); free(titul);
      if(buflag==1) IER=LOAD_BUF(); return(IER); }
    for(;;)
    { if(n==nmax)
      { sprintf(ERS,"Too many tables in table file '%s' (TBL_EDT)",DEF);
        IER=4; fprintf(OUT,"\n%d %s",IER,ERS); fclose(tbl); free(titul);
        if(buflag==1) IER=LOAD_BUF(); return(IER); }
      if(fscanf(tbl,"%s",&titul[15*n])==EOF) break;
      if(fscanf(tbl,"%d",&i)!=1)
      { sprintf(ERS,"Format error in table file '%s' (TBL_EDT)",DEF);
        IER=5; fprintf(OUT,"\n%d %s",IER,ERS); fclose(tbl); free(titul);
        if(buflag==1) IER=LOAD_BUF(); return(IER); }
      for(j=0;j<i;j++) if(fscanf(tbl,"%lu",t)!=1)
      { sprintf(ERS,"Format error in table file '%s' (TBL_EDT)",DEF);
        IER=6; fprintf(OUT,"\n%d %s",IER,ERS); fclose(tbl); free(titul);
        if(buflag==1) IER=LOAD_BUF(); return(IER); }
      n=n+1; }
    fclose(tbl);
    sel=calloc(nmax,sizeof(char));
    if(!sel)
    { strcpy(ERS,"Memory allocation error - sel - (TBL_EDT)");
      IER=7; fprintf(OUT,"\n%d %s",IER,ERS); free(titul);
      if(buflag==1) IER=LOAD_BUF(); return(IER); }
    for(k=0;k<nmax;k++) sel[k]=0;
    tempo=calloc(5*nmax,sizeof(unsigned long int));
    if(!tempo)
    { strcpy(ERS,"Memory allocation error - tempo - (TBL_EDT)");
      IER=10; fprintf(OUT,"\n%d %s",IER,ERS); free(titul);
      if(buflag==1) IER=LOAD_BUF(); return(IER); }
    size=calloc(5,sizeof(int));
    if(!size)
    { strcpy(ERS,"Memory allocation error - size - (TABLE)"); IER=11;
      fprintf(OUT,"\n%d %s",IER,ERS); free(titul); free(tempo);
      if(buflag==1) IER=LOAD_BUF(); return(IER); }
    GETSETTINGS(); setfillstyle(SOLID_FILL,WHITE); settextstyle(0,0,1);
    setlinestyle(SOLID_LINE,0,NORM_WIDTH); WINDOW(2); IER=GETIMAGE(X1,Y1,X2,Y2);
    if(IER>0) { free(titul); free(sel); PUTSETTINGS(); return(IER); }
    WINDOW(3); IER=GETIMAGE(X1,Y1,X2,Y2);
    if(IER>0) { free(titul); free(sel); WINDOW(2); IER=PUTIMAGE(X1,Y1,X2,Y2);
      PUTSETTINGS(); if(buflag==1) IER=LOAD_BUF(); return(IER); }
    WINDOW(2); MOLDURA(); WINDOW(3); MOLDURA();
    MESSAGE(80,1); MESSAGE(81,1); MESSAGE(82,1); MESSAGE(83,1);
    MESSAGE(84,1); MESSAGE(85,1); MESSAGE(86,1); MESSAGE(87,1);
}

```

```

MESSAGE(88,1); WINDOW(3); setcolor(RED);
line(127,4,127,Y2-Y1-4); line(255,4,255,Y2-Y1-4);
line(383,4,383,Y2-Y1-4); line(511,4,511,Y2-Y1-4);
i=0; j=0; ii=0; WRITE_TITUL(n,ii,i,j,titul,sel);
for(;;)
{ cont:
  WINDOW(3); setcolor(RED); rectangle(5+128*i,9+16*j,121+128*i,21+16*j);
  BIOSKEY(ch);
  if(M2>0)
  { if(MW==3)
    { WINDOW(3);
      if((M3>X1+5)&&(M3<X2-5)&&(M4>Y1+9)&&(M4<Y2-9))
      { k=(M3-X1-5)/128; m=(M4-Y1-9)/16;
        if(7*(k+ii)+m<n)
        { setcolor(WHITE); rectangle(5+128*i,9+16*j,121+128*i,21+16*j);
          i=k; j=m; } } if(M2==2) ch[0]='l'; }
    if(MW==2)
    { MESSAGE(80,-1); if((M3>X1)&&(M3<X2)&&(M4>Y1)&&(M4<Y2)) ch[1]=LEFT;
      MESSAGE(81,-1); if((M3>X1)&&(M3<X2)&&(M4>Y1)&&(M4<Y2)) ch[0]='r';
      MESSAGE(82,-1); if((M3>X1)&&(M3<X2)&&(M4>Y1)&&(M4<Y2)) ch[0]='d';
      MESSAGE(83,-1); if((M3>X1)&&(M3<X2)&&(M4>Y1)&&(M4<Y2)) ch[0]='i';
      MESSAGE(84,-1); if((M3>X1)&&(M3<X2)&&(M4>Y1)&&(M4<Y2)) ch[1]=RIGHT;
      MESSAGE(85,-1); if((M3>X1)&&(M3<X2)&&(M4>Y1)&&(M4<Y2)) ch[0]='s';
      MESSAGE(86,-1); if((M3>X1)&&(M3<X2)&&(M4>Y1)&&(M4<Y2)) ch[0]='a';
      MESSAGE(87,-1); if((M3>X1)&&(M3<X2)&&(M4>Y1)&&(M4<Y2)) ch[0]='w';
      MESSAGE(88,-1); if((M3>X1)&&(M3<X2)&&(M4>Y1)&&(M4<Y2)) ch[0]='l';
      MESSAGE(89,-1); if((M3>X1)&&(M3<X2)&&(M4>Y1)&&(M4<Y2)) ch[0]='e'; } }
    switch(ch[0])
    { case 'r':
      if(chg!=0) /* return */
      { strcpy(str,"Modifications are not saved !");
        if(SAVE_ASK(str)==1) SAVE_TITUL(n,titul); }
      strcpy(choice,&titul[15*(7*(i+ii)+j)]);
      free(titul); free(sel); free(tempo); free(size);
      WINDOW(3); IER=PUTIMAGE(X1,Y1,X2,Y2);
      WINDOW(2); IER=PUTIMAGE(X1,Y1,X2,Y2); PUTSETTINGS();
      if(buflag==1) IER=LOAD_BUF(); return(IER);
    case 's':
      SAVE_TITUL(n,titul); chg=0; break; /* save */
    case 'd':
      if(n==0) break; if(sel[7*(i+ii)+j]!=0) ns=ns-1; /* delete */
      setfillstyle(SOLID_FILL,WHITE); WINDOW(3);
      bar(5+128*i,9+16*j,121+128*i,21+16*j);
      for(k=7*(i+ii)+j;k<n-1;k++)
      { strcpy(&titul[15*k],&titul[15*(k+1)]); strcpy(&sel[k],&sel[k+1]); }
      n=n-1; if(7*(i+ii)+j==n) { if(j>0) j=j-1; else { j=6; i=i-1; } }
      if(n==0) { i=0; j=0; }
      WRITE_TITUL(n,ii,i,j,titul,sel); chg=1; break;
    case 'i':
      if(n==0) break; if(n==nmax) break; /* insert */
      m=7*(i+ii)+j;
      for(k=n;k>m;k--)
      { strcpy(&titul[15*k],&titul[15*(k-1)]);
        strcpy(&sel[k],&sel[k-1]); }
      n=n+1; chg=1; WRITE_TITUL(n,ii,i,j,titul,sel); sel[m]=0; k=0;
    ins1:
      HIDE_MOUSE(); WINDOW(3); bar(5+128*i,9+16*j,121+128*i,21+16*j);
      setcolor(RED); rectangle(5+128*i,9+16*j,121+128*i,21+16*j);
      INPUT(7+128*i,11+16*j,&titul[15*m],14,14,k,1);
      REM_BLANK(&titul[15*m]);

```

```

    if(titul[15*m]=='\0') strcpy(&titul[15*m],"*-----*");
    sprintf(str,"%14.14s",&titul[15*m]);
    if(sel[m]==0) setcolor(BLUE); else setcolor(RED);
    outtextxy(7+128*i,11+16*j,str); SHOW_MOUSE(); break;
case 'a':
    if(n==0) break; if(n==nmax) break; /* append */
    setcolor(WHITE); WINDOW(3);
    rectangle(5+128*i,9+16*j,121+128*i,21+16*j);
    titul[15*n]='\0'; n=n+1; chg=1;
    if(ii!=n/7-4)
    { ii=(n-1)/7-4; if(ii<0) ii=0; WRITE_TITUL(n,ii,0,0,titul,sel); }
    i=(n-1)/7-ii; j=(n-1)%7; m=7*(i+ii)+j; sel[m]=0; k=0; goto ins1;
case 'w':
    m=7*(i+ii)+j; chg=1; /* write */
    if(n==0) { n=n+1; k=0; } else k=1; goto ins1;
case 'l':
    if(n==0) break; /* select */
    m=7*(i+ii)+j; if((ns==5)&&(sel[m]==0)) break;
    sprintf(str,"%14.14s",&titul[15*m]); WINDOW(3);
    if(sel[m]==0) { sel[m]=1; setcolor(RED); ns=ns+1; }
    else { sel[m]=0; setcolor(BLUE); ns=ns-1; }
    outtextxy(7+128*i,11+16*j,str);
    if(ns==0) MESSAGE(89,0); else MESSAGE(89,1); break;
case 'e':
    if((n==0);!(ns==0)) break; /* edit */
    if(chg!=0) { SAVE_TITUL(n,titul); chg=0; }
    goto edit;
}
switch(ch[1])
{ case RIGHT:
    if(n==0) break; if(i+ii==(n-1)/7) break; WINDOW(3);
    if(i<4)
    { setcolor(WHITE); rectangle(5+128*i,9+16*j,121+128*i,21+16*j);
      i=i+1; if(7*(ii+i)+j+1>n) j=(n-1)%7; break; }
    if(i==4)
    { ii=ii+1; if(7*(i+ii)+j+1>n)
      { setcolor(WHITE); rectangle(5+128*i,9+16*j,121+128*i,21+16*j);
        j=(n-1)%7; }
      WRITE_TITUL(n,ii,0,0,titul,sel); break; }
    break;
case LEFT:
    if(n==0) break; if(i+ii==0) break; WINDOW(3);
    if(i>0)
    { setcolor(WHITE); rectangle(5+128*i,9+16*j,121+128*i,21+16*j);
      i=i-1; break; }
    if(i==0) { ii=ii-1; WRITE_TITUL(n,ii,0,0,titul,sel); break; }
    break;
case UP:
    if(n==0) break; if(j==0) break; WINDOW(3); setcolor(WHITE);
    rectangle(5+128*i,9+16*j,121+128*i,21+16*j); j=j-1; break;
case DOWN:
    if(n==0) break; if(j==6) break; if(7*(ii+i)+j>n-1) break;
    WINDOW(3); setcolor(WHITE);
    rectangle(5+128*i,9+16*j,121+128*i,21+16*j); j=j+1; break;
}
}
edit:
if((tbl=fopen(DEF,"r"))==NULL)
{ sprintf(ERS,"Table file '%s' cannot be open (TBL_EDT)",DEF);
  IER=10; fprintf(OUT,"\n%d %s",IER,ERS);
}

```



```

free(titul); free(sel); free(tempo); free(size); WINDOW(3);
IER=PUTIMAGE(X1,Y1,X2,Y2); WINDOW(2); IER=PUTIMAGE(X1,Y1,X2,Y2);
PUTSETTINGS(); if(buflag==1) IER=LOAD_BUF(); return(IER); }
WINDOW(51); IER=GETIMAGE(X1,Y1,X2,Y2); if(IER>0)
{ fclose(tbl); free(titul); free(sel); free(tempo); free(size); WINDOW(3);
  IER=PUTIMAGE(X1,Y1,X2,Y2); WINDOW(2); IER=PUTIMAGE(X1,Y1,X2,Y2);
  PUTSETTINGS(); if(buflag==1) IER=LOAD_BUF(); return(IER); }
WINDOW(3); setcolor(WHITE); rectangle(5+128*i,9+16*j,121+128*i,21+16*j);
WINDOW(51); MOLDURA(); setcolor(RED); line(5,20,633,20);
line(127,4,127,Y2-Y1-4); line(255,4,255,Y2-Y1-4); line(383,6,383,Y2-Y1-4);
line(511,6,511,Y2-Y1-4); line(5,240,633,240);
MESSAGE(88,0); MESSAGE(89,0); WINDOW(51); setcolor(BLUE); m=-1; l=-1;
for(;;)
{ m=m+1; if(fscanf(tbl,"%s",str)==EOF) break;
  if(sel[m]!=0)
  { l=l+1; sprintf(str,"%14.14s",&titul[15*m]); outtextxy(7+128*l,8,str);
    fscanf(tbl,"%d",&size[l]);
    sprintf(str,"size: %2d",size[l]); outtextxy(31+128*l,244,str);
    for(k=0;k<size[l];k++) fscanf(tbl,"%lu",&tempo[100*l+k]); }
  else { fscanf(tbl,"%d",&q); for(k=0;k<q;k++) fscanf(tbl,"%lu",&t); } }
for(l=0;l<ns;l++) WRITE_TABLE(size[l],0,l,0,tempo);
fclose(tbl); isave=i; jsave=j; i=0; j=0; jj=0;
for(;;)
{ WINDOW(51); setcolor(RED); rectangle(5+128*i,22+12*j,121+128*i,34+12*j);
  bar(79+128*i,244,95+128*i,252); sprintf(str,"%2d",size[i]);
  setcolor(BLUE); outtextxy(79+128*i,244,str); BIOSKEY(ch);
  if(M2==1)
  { if(MW==2)
    { MESSAGE(80,-1); if((M3>X1)&&(M3<X2)&&(M4>Y1)&&(M4<Y2)) ch[1]=DOWN;
      MESSAGE(81,-1); if((M3>X1)&&(M3<X2)&&(M4>Y1)&&(M4<Y2)) ch[0]='r';
      MESSAGE(82,-1); if((M3>X1)&&(M3<X2)&&(M4>Y1)&&(M4<Y2)) ch[0]='d';
      MESSAGE(83,-1); if((M3>X1)&&(M3<X2)&&(M4>Y1)&&(M4<Y2)) ch[0]='i';
      MESSAGE(84,-1); if((M3>X1)&&(M3<X2)&&(M4>Y1)&&(M4<Y2)) ch[1]=UP;
      MESSAGE(85,-1); if((M3>X1)&&(M3<X2)&&(M4>Y1)&&(M4<Y2)) ch[0]='s';
      MESSAGE(86,-1); if((M3>X1)&&(M3<X2)&&(M4>Y1)&&(M4<Y2)) ch[0]='a';
      MESSAGE(87,-1); if((M3>X1)&&(M3<X2)&&(M4>Y1)&&(M4<Y2)) ch[0]='w'; }
    if((M3>5)&&(M3<633)&&(M4>22)&&(M4<238))
    { if(M3/128<ns)
      { m=size[M3/128]; if(m==0) m=1; if((M4-22)/12+jj<m)
        { WINDOW(51); setcolor(WHITE);
          rectangle(5+128*i,22+12*j,121+128*i,34+12*j);
          j=(M4-22)/12; i=M3/128; setcolor(RED);
          rectangle(5+128*i,22+12*j,121+128*i,34+12*j); } } } }
    switch(ch[0])
    { case 'r':
      if(chg!=0)
      { strcpy(str,"Modifications are not saved !");
        if(SAVE_ASK(str)==1) SAVE_TABLE(sel,size,tempo); }
      chg=0; i=isave; j=jsave; MESSAGE(88,1); MESSAGE(89,1);
      WINDOW(51); PUTIMAGE(X1,Y1,X2,Y2); WINDOW(3); goto cont;
      case 's':
        SAVE_TABLE(sel,size,tempo); chg=0; break;
        case 'i':
        if(size[i]==0) break; if(size[i]==nmax) break;
        for(k=size[i];k>j+jj;k--) tempo[100*i+k]=tempo[100*i+k-1];
        size[i]=size[i]+1; chg=1; WRITE_TABLE(size[i],jj,i,j,tempo); k=0;
        ins2:
        HIDE_MOUSE(); WINDOW(51); bar(7+128*i,24+12*j,119+128*i,32+12*j);
        setcolor(RED); rectangle(5+128*i,22+12*j,121+128*i,34+12*j);
        STRTIME(str,tempo[100*i+j+jj]); INPUT(7+128*i,24+12*j,str,14,14,k,1);

```

```

    INTIME(str,&tempo[100*i+j+jj]); STRTIME(str,tempo[100*i+j+jj]);
    setcolor(BLUE); outtextxy(7+128*i,24+12*j,str);
    SHOW_MOUSE(); break;
case 'w':
    /* write */
    if(size[i]==0) { size[i]=size[i]+1; k=0; } else k=1;
    chg=1; goto ins2;
case 'a':
    /* append */
    if(size[i]==0) break; if(size[i]==nmax) break;
    setcolor(WHITE); rectangle(5+128*i,22+12*j,121+128*i,34+12*j);
    j=size[i]-jj;
    if(j>17)
    { j=17; jj=size[i]-17;
      for(m=0;m<ns;m++) WRITE_TABLE(size[m],jj,m,0,tempo); }
    chg=1; size[i]=size[i]+1; k=0; goto ins2;
case 'd':
    /* delete */
    if(size[i]==0) break;
    if((j==0)&&(jj>0)&&(j+jj+1==size[i]))
    { jj=jj-1; j=j+1;
      for(m=0;m<ns;m++) WRITE_TABLE(size[m],jj,m,0,tempo); }
    for(k=j+jj;k<size[i]-1;k++) tempo[100*i+k]=tempo[100*i+k+1];
    size[i]=size[i]-1; chg=1;
    if(j+jj<size[i]) { WRITE_TABLE(size[i],jj,i,j,tempo); break; }
    if(j+jj==size[i])
    { setcolor(WHITE); rectangle(5+128*i,22+12*j,121+128*i,34+12*j);
      WRITE_TABLE(size[i],jj,i,j,tempo); j=j-1; if(j<0) j=0; break; }
    break;
}
switch(ch[1])
{ case RIGHT:
    if(i==ns-1) break; WINDOW(51);
    setcolor(WHITE); rectangle(5+128*i,22+12*j,121+128*i,34+12*j);
    if(j+jj==0) { i=i+1; break; }
    m=i+1; while(j+jj+1>size[m]) { m=m+1; if(m>ns) { m=i; break; } }
    i=m; break;
case LEFT:
    if(i==0) break; WINDOW(51);
    setcolor(WHITE); rectangle(5+128*i,22+12*j,121+128*i,34+12*j);
    if(j+jj==0) { i=i-1; break; }
    m=i-1; while(j+jj+1>size[m]) { m=m-1; if(m<0) { m=i; break; } }
    i=m; break;
case UP:
    if(size[i]==0) break; if(j+jj==0) break; WINDOW(51);
    setcolor(WHITE); rectangle(5+128*i,22+12*j,121+128*i,34+12*j);
    if(j>0) { j=j-1; break; }
    jj=jj-1; for(m=0;m<ns;m++) WRITE_TABLE(size[m],jj,m,0,tempo); break;
case DOWN:
    if(size[i]==0) break; if(j+jj+1==size[i]) break; WINDOW(51);
    setcolor(WHITE); rectangle(5+128*i,22+12*j,121+128*i,34+12*j);
    if(j+1<18) { j=j+1; break; }
    jj=jj+1; for(m=0;m<ns;m++) WRITE_TABLE(size[m],jj,m,0,tempo); break;
}
}
}

/* ***** */

void WRITE_TITUL(n,ii,i,j,titul,sel) /* reescreve a tabela de titulos de n */
int n,ii,i,j; /* elementos a partir do (i,j), apa- */
char *titul,*sel; /* gando os titulos da tabela antiga. */

```

```

{
    char str[80];
    int k,m,l;

    GETSETTINGS(); WINDOW(3);
    setfillstyle(SOLID_FILL,WHITE); settextstyle(0,0,1);
    for(l=7*(i+ii)+j;l<7*(i+ii+5);l++)
    { k=l/7-ii; m=l%7; bar(7+128*k,11+16*m,119+128*k,19+16*m);
      if(l<n) { sprintf(str,"%14.14s",&titul[15*l]);
               if(sel[l]==0) setcolor(BLUE); else setcolor(RED);
               outtextxy(7+128*k,11+16*m,str); } }
    PUTSETTINGS(); return;
}

/* ***** */

void WRITE_TABLE(n,jj,i,j,tempo) /* reescreve a coluna i de n elementos */
int n,jj,i,j; /* partir do elemento j, apagando os */
unsigned long int *tempo; /* elementos da coluna antiga. */
{
    char str[80];
    int k;

    WINDOW(51); setcolor(BLUE);
    for(k=j;k<18;k++)
    { bar(7+128*i,24+12*k,119+128*i,32+12*k);
      if(k+jj<n)
      { STRTIME(str,tempo[100*i+k+jj]); outtextxy(7+128*i,24+12*k,str); } }
    return;
}

/* ***** */

int SAVE_TITUL(n,titul)
int n;
char *titul;
{
    FILE *tbl, *lixo;
    char str[80],file[80],drive[3],dir[66],name[9],ext[5];
    unsigned long int t;
    int i,j,m;

    IER=GET_DEF(2); if(IER>0) return(IER);
    fnsplit(DEF,drive,dir,name,ext);
    sprintf(file,"%s%s%s",drive,dir,"lixo.tbl");
    if((lixo=fopen(file,"w"))==NULL)
    { sprintf(ERS,"Temporary table file '%s' cannot be open (SAVE_TITUL)",file);
      IER=1; fprintf(OUT,"\n%d %s",IER,ERS); return(IER); }
    if((tbl=fopen(DEF,"r"))==NULL)
    { sprintf(ERS,"Table file '%s' cannot be open (SAVE_TITUL)",DEF);
      IER=2; fprintf(OUT,"\n%d %s",IER,ERS); return(IER); }
    for(i=0;i<n;i++)
    { REM_BLANK(&titul[15*i]); fprintf(lixo,"%14.14s\n",&titul[15*i]);
      rewind(tbl);
      for(;;)
      { if(fscanf(tbl,"%s",str)==EOF) { m=0; fprintf(lixo,"%d\n",m); break; }
        if(strcmp(str,&titul[15*i])==0)
        { fscanf(tbl,"%d",&m); fprintf(lixo,"%d\n",m);
          for(j=0;j<m;j++) { fscanf(tbl,"%lu",&t); fprintf(lixo,"%lu\n",t); }
          break; }
      }
    }
}

```

```

        else { fscanf(tbl,"%d",&m); for(j=0;j<m;j++) fscanf(tbl,"%lu",t); } } }
fclose(lixo); fclose(tbl); remove(DEF); rename(file,DEF); return(IER);
}

/* ***** */

int SAVE_TABLE(sel,size,tempo)
char *sel;
int *size;
unsigned long *tempo;
{
    FILE *tbl, *lixo;
    char str[80],file[80],drive[3],dir[66],name[9],ext[5];
    unsigned long int t;
    int i,j,n,m;

    IER=GET_DEF(2); if(IER>0) return(IER);
    fnsplit(DEF,drive,dir,name,ext);
    sprintf(file,"%s%s%s",drive,dir,"lixo.tbl");
    if((lixo=fopen(file,"w"))==NULL)
    { sprintf(ERS,"Temporary table file '%s' cannot be open (SAVE_TABLE)",file);
      IER=1; fprintf(OUT,"\n%d %s",IER,ERS); return(IER); }
    if((tbl=fopen(DEF,"r"))==NULL)
    { sprintf(ERS,"Table file '%s' cannot be open (SAVE_TABLE)",DEF);
      IER=2; fprintf(OUT,"\n%d %s",IER,ERS); return(IER); }
    n=-1; m=-1;
    for(;;)
    { n=n+1; if(fscanf(tbl,"%s",str)==EOF) break; fprintf(lixo,"%14.14s\n",str);
      if(sel[n]!=0)
      { m=m+1; fscanf(tbl,"%d",&i); fprintf(lixo,"%d\n",size[m]);
        for(j=0;j<i;j++) fscanf(tbl,"%lu",&t);
        for(j=0;j<size[m];j++) fprintf(lixo,"%lu\n",tempo[100*m+j]); }
      else
      { fscanf(tbl,"%d",&i); fprintf(lixo,"%d\n",i);
        for(j=0;j<i;j++) { fscanf(tbl,"%lu",&t); fprintf(lixo,"%lu\n",t); } } }
    fclose(lixo); fclose(tbl); remove(DEF); rename(file,DEF); return(IER);
}

/* ***** */

int TBL_EVAL(choice,value,line,n) /* devolve o tempo (value) da linha */
char *choice; /* (line) e o numero de elementos (n) */
unsigned long int *value; /* da tabela de titulo (choice). */
int line, *n; /* Se o titulo nao existe, ou se a */
{ /* linha procurada nao existe, esta */
    FILE *tbl; /* funcao devolve value=0 e n=0. */
    int i;
    char str[80],titulo[15];

    IER=GET_DEF(2); if(IER>0) return(IER);
    if((tbl=fopen(DEF,"r"))==NULL)
    { sprintf(ERS,"WARNING: Table file '%s' could not be open (TBL_EVAL)",DEF);
      IER=1; fprintf(OUT,"\n%d %s",IER,ERS); return(IER); }
    REM_BLANK(choice);
    for(;;)
    { if(fscanf(tbl,"%s",titulo)==EOF) { *value=0; *n=0; break; }
      fscanf(tbl,"%d",&n);
      if(strcmp(choice,titulo)!=0)
      { for(i=0;i<*n;i++) fscanf(tbl,"%lu",value); }
      else

```

```

    { if(line>*n-1) { *value=0; *n=0; }
      else for(i=0;i<=line;i++) fscanf(tbl,"%lu",value);
      break; } }
fclose(tbl); return(IER);
}

/* ***** */

int CHK_TBL(void)
{
    FILE *tbl;

    IER=GET_DEF(2); if(IER>0) exit(IER);
    if((tbl=fopen(DEF,"r"))==NULL)
    { fprintf(OUT,"\nWARNING: Table file cannot be find - CHK_TBL");
      fprintf(OUT,"\n          A new table file will be created");
      if((tbl=fopen(DEF,"w"))==NULL)
      { strcpy(ERS,"Failure trying to create a new table file - CHK_TBL");
        IER=1; fprintf(OUT,"\n%d%s",IER,ERS); return(IER); } }
    fclose(tbl); return(IER);
}

/* ***** */

/* ***** SAVE.C ***** */

#include "psp_dcl.h"

/* ***** */

int GET_DATA(n)
int n;
{
    FILE *dat;
    unsigned int size;
    int i,j;
    char str[80],drive[3],dir[66],name[9],ext[5];

    /* n=1 : tenta abrir FILE_NAME para leitura e copiar o seu conteudo nos */
    /* buffers. Se nao conseguir retorna com mensagem de erro. Apos a */
    /* leitura da FILE_NAME, o driver e diretorio passam a ser aquele */
    /* da linha 4 do arquivo psp.def. O conteudo da FILE_NAME e' co- */
    /* piado nos buffers e depois transferido para c:\psp\temp\psp.buf, */
    /* atraves da FREE_BUF, onde os buffers sao liberados. */
    /* n=0 : inicializa uma nova sequencia nos buffers, copia o conteudo */
    /* deles na c:\psp\temp\psp.buf, libera-os e retorna. Neste caso */
    /* faz FILE_NAME = "default(4)\psp.seq" para futuro save. */
    /* Em ambos os casos, atualiza a tela antes de retornar. */

    IER=GET_DEF(4); if(IER>0) return(IER);
    GETSETTINGS(); if(BUF!=NULL) { free(BUF); BUF=NULL; }
    size = 11*N*sizeof(char) + 5*N*sizeof(char) + 4*N*sizeof(char)
          + N*(M/NC)*sizeof(unsigned long int)
          + 4*(M/NC)*sizeof(unsigned long int)
          + 46*sizeof(char) + M*sizeof(unsigned long int)
          + 15*M*sizeof(char) + M*sizeof(char)
          + 5*(M/NC)*sizeof(unsigned long int)
          + 5*sizeof(unsigned int);
    BUF = malloc(size);
    if(!BUF) { strcpy(ERS,"Memory allocation error - BUF - (GET_DATA)");

```

```

        IER=1; fprintf(OUT, "\n%d %s", IER, ERS);
        fclose(dat); return(IER); }

NAME = (char *) BUF;
LINE = (char *) BUF + 11*N;
CODE = (char *) BUF + 11*N + 5*N;
PULSE = (unsigned long int *) ( (char *) BUF + 11*N + 5*N + 4*N );
TYPE = PULSE + N*(M/NC);
SEQ = (char *) ( TYPE + 4*(M/NC) );
TIME = (unsigned long int *) ( SEQ + 46 );
SPEC = (char *) (TIME + M);
TAG = SPEC + 15*M;
LOOP = (unsigned long int *) ( TAG + M );
COUNT = (unsigned int *) ( LOOP + 5*(M/NC) );
if(n==1)
{ MESSAGE(101,1);
  if((dat=fopen(FILE_NAME, "r"))==NULL)
  { sprintf(ERS, "WARNING: file '%s' cannot be open (GET_DATA)", FILE_NAME);
    IER=2; fprintf(OUT, "\n%d %s", IER, ERS); return(IER); }
  if((fscanf(dat, "%s", &SEQ[0]))!=1)
  { sprintf(ERS, "Reading error at file '%s' - (GET_DATA)", FILE_NAME);
    IER=3; fprintf(OUT, "\n%d %s", IER, ERS);
    fclose(dat); free(BUF); return(IER); }
  for(i=0; i<N; i++)
  { if((fscanf(dat, "%s%s%s", &NAME[11*i], &LINE[5*i], &CODE[4*i]))!=3)
    { sprintf(ERS, "Reading error at file '%s' - (GET_DATA)", FILE_NAME);
      IER=4; fprintf(OUT, "\n%d %s", IER, ERS);
      fclose(dat); free(BUF); return(IER); } }
  for(i=0; i<4; i++) for(j=0; j<M/NC; j++)
  { if((fscanf(dat, "%lX", &TYPE[i*(M/NC)+j]))!=1)
    { sprintf(ERS, "Reading error at file '%s' - (GET_DATA)", FILE_NAME);
      IER=5; fprintf(OUT, "\n%d %s", IER, ERS);
      fclose(dat); free(BUF); return(IER); } }
  for(i=0; i<M; i++)
  { if((fscanf(dat, "%lu%s%ls", &TIME[i], &SPEC[15*i], &TAG[i]))!=3)
    { sprintf(ERS, "Reading error at file '%s' - (GET_DATA)", FILE_NAME);
      IER=6; fprintf(OUT, "\n%d %s", IER, ERS);
      fclose(dat); free(BUF); return(IER); } }
  for(i=0; i<N; i++) for(j=0; j<M/NC; j++)
  { if((fscanf(dat, "%lX", &PULSE[i*(M/NC)+j]))!=1)
    { sprintf(ERS, "Reading error at file '%s' - (GET_DATA)", FILE_NAME);
      IER=7; fprintf(OUT, "\n%d %s", IER, ERS);
      fclose(dat); free(BUF); return(IER); } }
  for(i=0; i<5; i++)
  { if((fscanf(dat, "%u", &COUNT[i]))!=1)
    { sprintf(ERS, "Reading error at file '%s' - (GET_DATA)", FILE_NAME);
      IER=8; fprintf(OUT, "\n%d %s", IER, ERS);
      fclose(dat); free(BUF); return(IER); } }
  for(i=0; i<5; i++) for(j=0; j<M/NC; j++)
  { if((fscanf(dat, "%lX", &LOOP[i*(M/NC)+j]))!=1)
    { sprintf(ERS, "Reading error at file '%s' - (GET_DATA)", FILE_NAME);
      IER=9; fprintf(OUT, "\n%d %s", IER, ERS);
      fclose(dat); free(BUF); return(IER); } }
  fnsplit(FILE_NAME, drive, dir, name, ext);
  sprintf(FILE_NAME, "%s%s%s%s", DEF, "\\ ", name, ext);
  fclose(dat); }
if(n==0)
{ sprintf(FILE_NAME, "%s%s", DEF, "\\psp.seq"); MESSAGE(102,1);
  strcpy(&SEQ[0], "*-----*");
  for(j=0; j<M; j++)
  { TIME[j]=0; strcpy(&SPEC[15*j], "*-----*"); TAG[j]='*'; }
}

```

```

for(i=0;i<N;i++)
{ strcpy(&NAME[11*i],"*-----*");
  strcpy(&LINE[ 5*i],"*--*");
  strcpy(&CODE[ 4*i],"*-");
  if(i<4) for(j=0;j<M/NC;j++) TYPE[i*(M/NC)+j]=0;
  if(i<5) { COUNT[i]=0; for(j=0;j<M/NC;j++) LOOP[i*(M/NC)+j]=0; }
  for(j=0;j<M/NC;j++) PULSE[i*(M/NC)+j]=0; } }
CHG=0; WINDOW(6); setcolor(BLUE);
clearviewport(); setcolor(BLUE); settextstyle(0,0,1);
for(i=0;i<NL;i++)
{ sprintf(str,"%10.10s",&NAME[11*i]); outtextxy(4,16*i+4,str); }
WINDOW(9); clearviewport(); I=0; for(J=0;J<NC;J++) PUTCHAR(8+GET_EV(J));
WINDOW(7); for(I=0;I<NL;I++) for(J=0;J<NC;J++) PUTCHAR(GETCHAR(I,J));
WINDOW(4); setfillstyle(SOLID_FILL,LIGHTGRAY); bar(0,0,X2-X1,Y2-Y1);
for(i=0;i<5;i++)
{ sprintf(str,"    LOOP %ld",i+1); outtextxy(4,16*i+4,str); }
WINDOW(5); for(I=0;I<5;I++) LOOP_DRAW();
IER=FREE_BUF(); I=0; J=0; II=0; JJ=0; MESSAGE(101,0); MESSAGE(102,0);
PUTSETTINGS(); return(IER);
}

/* ***** */

int SAVE_DATA(n) /* Carrega o buffer BUF (se ele estiver vazio) com o */
int n; /* conteudo da file "c:\psp\temp\psp.buf" e salva-o */
{ /* na file FILE_NAME. */
  FILE *dat;
  int i,j,flag;
  char str[80],drive[3],dir[66],name[9],ext[5];

  /* n = 0 : salva sem perguntar na FILE_NAME */
  /* n = 1 : pergunta se quer salvar. Sim: salva na FILE_NAME */
  /* n = 2 : save as.. Pergunta o path, nome e extensao da file. */
  /* FILE_NAME passa a conter o path, nome e extensao */
  /* fornecido pelo usuario. Neste caso, apos a sequencia */
  /* ser salva FILE_NAME passa a ter o path dado pelo de- */
  /* fault 4, com o nome e extensao que foi dado pelo */
  /* usuario. */

  IER=GET_DEF(4); if(IER>0) return(IER);
  if(BUF==NULL) flag=0; else flag=1;
  if(flag==0) IER=LOAD_BUF(); if(IER>0) return(IER);
  if(n==1)
  { strcpy(str,"Sequence not saved !");
    if(SAVE_ASK(str)==0) { if(flag==0) IER=FREE_BUF(); return(IER); } }
  strcpy(str,FILE_NAME);
  if(n==2)
  { strcpy(str,"Save sequence in file:"); SAVE_AS(str); }
  if((dat=fopen(str,"w"))==NULL)
  { sprintf(ERS,"Data file '%s' cannot be open (SAVE_DATA)",str); IER=1;
    fprintf(OUT,"\n%d %s",IER,ERS); if(flag==0) IER=FREE_BUF(); return(IER); }
  MESSAGE(100,1);
  fprintf(dat,"%45.45s\n",&SEQ[0]);
  for(i=0;i<N;i++)
    fprintf(dat,"%10.10s %4.4s %3.3s\n",&NAME[11*i],&LINE[5*i],&CODE[4*i]);
  for(i=0;i<4;i++)
  { for(j=0;j<M/NC;j++) fprintf(dat,"%08lx ",TYPE[i*(M/NC)+j]);
    fprintf(dat,"\n"); }
  for(i=0;i<M;i++)
  { fprintf(dat,"%10lu %14.14s %c\n",TIME[i],&SPEC[15*i],TAG[i]); }
}

```

```

for(i=0;i<N;i++)
{ for(j=0;j<M/NC;j++) fprintf(dat,"%08lx ",PULSE[i*(M/NC)+j]);
  fprintf(dat,"\n"); }
for(i=0;i<5;i++) fprintf(dat,"%u ",COUNT[i]); fprintf(dat,"\n");
for(i=0;i<5;i++)
{ for(j=0;j<M/NC;j++) fprintf(dat,"%08lx ",LOOP[i*(M/NC)+j]);
  fprintf(dat,"\n"); }
if(n==2) { fnsplit(str,drive,dir,name,ext);
          sprintf(FILE_NAME,"%s%s%s%s",DEF,"\\",name,ext); }
CHG=0; fclose(dat); if(flag==0) IER=FREE_BUF(); MESSAGE(100,0); return(IER);
}

/* ***** */

int SAVE_ASK(str)      /* str (entrada) : cabecalho */
char *str;             /* retorno      : 1=Yes  0=No */
{
    char ch[2];
    int n,x1,y1,x2,y2;

    GETSETTINGS(); WINDOW(2); GETIMAGE(X1,Y1,X2,Y2); clearviewport();
    x1=100; y1=390; x2=540; y2=478;
    setviewport(x1,y1,x2,y2,1); GETIMAGE(x1,y1,x2,y2);
    SET_MOUSE_WINDOW(x1+6,y1+6,x2-6,y2-6);
    SET_MOUSE_XY((x1+x2)/2,(y1+y2)/2);
    setfillstyle(SOLID_FILL,LIGHTGRAY); bar(0,0,(x2-x1),(y2-y1));
    setcolor(YELLOW); setlinestyle(SOLID_LINE,0,THICK_WIDTH);
    rectangle(5,5,(x2-x1-5),(y2-y1-5)); settextstyle(0,0,2);
    setcolor(BLUE); outtextxy(16,10,"Warning:");
    outtextxy(16,34,str); outtextxy(16,59,"Save ?");
    outtextxy(150,59,"Yes"); outtextxy(250,59,"No");
    rectangle(142,56,206,78); rectangle(234,56,298,78);
    setcolor(RED); outtextxy(150,59,"Y"); outtextxy(250,59,"N");
    for(;;)
    { BIOSKEY(ch);
      if(M2!=0)
      { if((M3>x1+142)&&(M3<x1+206)&&(M4>y1+56)&&(M4<y1+78)) ch[0]='y';
        if((M3>x1+234)&&(M3<x1+298)&&(M4>y1+56)&&(M4<y1+78)) ch[0]='n'; }
      switch(ch[0])
      { case 'y': n=1; goto end;
        case 'n': n=0; goto end;
        default : break; } }
    end:
    PUTIMAGE(x1,y1,x2,y2); WINDOW(2); PUTIMAGE(X1,Y1,X2,Y2);
    SET_MOUSE_WINDOW(0,0,639,479); SET_MOUSE_XY(320,240);
    PUTSETTINGS(); return(n);
}

/* ***** */

void SAVE_AS(str)      /* str (entrada) : cabecalho */
char *str;             /* (saida)   : file name */
{
    int x1,y1,x2,y2;

    GETSETTINGS(); WINDOW(2); GETIMAGE(X1,Y1,X2,Y2); clearviewport();
    x1=100; y1=390; x2=540; y2=478;
    setviewport(x1,y1,x2,y2,1); GETIMAGE(x1,y1,x2,y2);
    SET_MOUSE_WINDOW(x1+6,y1+6,x2-6,y2-6);
    SET_MOUSE_XY((x1+x2)/2,(y1+y2)/2);

```



```

setfillstyle(SOLID_FILL,LIGHTGRAY); bar(0,0,(x2-x1),(y2-y1));
setcolor(YELLOW); setlinestyle(SOLID_LINE,0,THICK_WIDTH);
rectangle(5,5,(x2-x1-5),(y2-y1-5)); settextstyle(0,0,2);
setcolor(BLUE); outtextxy(16,16,str);
setfillstyle(SOLID_FILL,WHITE); bar(10,40,x2-x1-10,60);
INPUT(12,42,str,79,26,0,2);
PUTIMAGE(x1,y1,x2,y2); WINDOW(2); PUTIMAGE(x1,y1,x2,y2);
SET_MOUSE_WINDOW(0,0,639,479); SET_MOUSE_AY(320,240);
PUTSETTINGS(); return;
}

/* ***** */

int LOAD_BUF(void)
{
    int buf;
    unsigned int size;

    if(BUF!=NULL) return(IER);
    if((buf=open("c:\\psp\\temp\\psp.buf",
        O_RDONLY | O_BINARY, S_IWRITE | S_IREAD))===-1)
    { strcpy(ERS,"File 'c:\\psp\\temp\\psp.buf' cannot be open (LOAD_BUF)");
      IER=1; fprintf(OUT,"\n%d %s",IER,ERS); return(IER); }
    size = 11*N*sizeof(char) + 5*N*sizeof(char) + 4*N*sizeof(char)
        + N*(M/NC)*sizeof(unsigned long int)
        + 4*(M/NC)*sizeof(unsigned long int)
        + 46*sizeof(char) + M*sizeof(unsigned long int)
        + 15*M*sizeof(char) + M*sizeof(char)
        + 5*(M/NC)*sizeof(unsigned long int)
        + 5*sizeof(unsigned int);
    BUF = malloc(size);
    if(!BUF) { strcpy(ERS,"Memory allocation error - BUF - (LOAD_BUF)"); IER=2;
      fprintf(OUT,"\n%d %s",IER,ERS); close(buf); return(IER); }
    if((_read(buf,BUF,size))!=size)
    { strcpy(ERS,"Reading error - 'c:\\psp\\temp\\psp.buf' - (LOAD_BUF)");
      IER=3; fprintf(OUT,"\n%d %s",IER,ERS);
      free(BUF); BUF=NULL; close(buf); return(IER); }
    NAME = (char *) BUF;
    LINE = (char *) BUF + 11*N;
    CODE = (char *) BUF + 11*N + 5*N;
    PULSE = (unsigned long int *) ( (char *) BUF + 11*N + 5*N + 4*N );
    TYPE = PULSE + N*(M/NC);
    SEQ = (char *) ( TYPE + 4*(M/NC) );
    TIME = (unsigned long int *) ( SEQ + 46 );
    SPEC = (char *) (TIME + M);
    TAG = SPEC + 15*M;
    LOOP = (unsigned long int *) ( TAG + M );
    COUNT = (unsigned int *) ( LOOP + 5*(M/NC) );
    close(buf); return(IER);
}

/* ***** */

int FREE_BUF(void)
{
    int buf;
    unsigned int size;

    if(BUF==NULL) return(IER);
    if((buf=open("c:\\psp\\temp\\psp.buf",

```

```

O_CREAT | O_WRONLY | O_BINARY, S_IWRITE | S_IREAD) == -1)
{ strcpy(ERS, "Data file c:\\psp\\temp\\psp.buf cannot be open - FREE_BUF");
  IER=1; fprintf(OUT, "\n%d %s", IER, ERS); return(IER); }
size = 11*N*sizeof(char) + 5*N*sizeof(char) + 4*N*sizeof(char)
      + N*(M/NC)*sizeof(unsigned long int)
      + 4*(M/NC)*sizeof(unsigned long int)
      + 46*sizeof(char) + M*sizeof(unsigned long int)
      + 15*M*sizeof(char) + M*sizeof(char)
      + 5*(M/NC)*sizeof(unsigned long int)
      + 5*sizeof(unsigned int);
if((_write(buf, BUF, size))!=size)
{ strcpy(ERS, "Writting error - 'c:\\psp\\temp\\psp.buf' - (FREE_BUF)");
  IER=2; fprintf(OUT, "\n%d %s", IER, ERS); close(buf); return(IER); }
free(BUF); BUF=NULL; close(buf); return(IER);
}

/* ***** */

void GETSETTINGS(void) /* save graphics settings */
{
  SETLEVEL=SETLEVEL+1;
  getfillsettings(&OLDFILL[SETLEVEL]); getviewsettings(&OLDVIEW[SETLEVEL]);
  OLDCOLOR[SETLEVEL]=getcolor(); OLDBKCOLOR[SETLEVEL]=getbkcolor();
  gettextsettings(&OLDTEXT[SETLEVEL]); getlinesettings(&OLDLINE[SETLEVEL]);
  return;
}

/* ***** */

void PUTSETTINGS(void) /* restore graphics settings */
{
  setcolor(OLDCOLOR[SETLEVEL]); setbkcolor(OLDBKCOLOR[SETLEVEL]);
  setfillstyle(OLDFILL[SETLEVEL].pattern, OLDFILL[SETLEVEL].color);
  setviewport(OLDVIEW[SETLEVEL].left, OLDVIEW[SETLEVEL].top,
              OLDVIEW[SETLEVEL].right, OLDVIEW[SETLEVEL].bottom, 1);
  settextrstyle(OLDTEXT[SETLEVEL].font, OLDTEXT[SETLEVEL].direction,
                OLDTEXT[SETLEVEL].charsize);
  setlinestyle(OLDLINE[SETLEVEL].linestyle, OLDLINE[SETLEVEL].upattern,
               OLDLINE[SETLEVEL].thickness);
  SETLEVEL=SETLEVEL-1; return;
}

/* ***** */

int GETIMAGE(x1,y1,x2,y2) /* salva imagem definida pelo retangulo */
int x1,y1,x2,y2; /* (x1,y1,x2,y2), em coordenadas absolutas */
{ /* da tela, na file "c:\\psp\\temp\\img.nnn". */
  int handle;
  void *buf;
  unsigned int size;
  int i,n,j,ystart,yend;
  char file[80];

  IMGLEVEL=IMGLEVEL+1; sprintf(file, "c:\\psp\\temp\\img.%03d", IMGLEVEL);
  if((handle=open(file, O_CREAT|O_WRONLY|O_BINARY, S_IWRITE|S_IREAD))==-1)
  { sprintf(ERS, "Image save file '%s' cannot be open (GETIMAGE)", file);
    IER=1; fprintf(OUT, "\n%d %s", IER, ERS); return(IER); }
  GETSETTINGS(); setviewport(0,0,getmaxx(),getmaxy(),1);
  n= y2-y1+1; i= n/4;
  if((n%4)!=0) n=i+1; else n=i;

```

```

ystart=y1; yend=ystart+n-1; size=imagesize(x1,ystart,x2,yend);
if( (buf=malloc(size))==NULL)
{ strcpy(ERS,"Memory allocation error - buf - (GETIMAGE)");
  IER=2; fprintf(OUT,"\n%d %s",IER,ERS); return(IER); }
for(i=0;i<4;i++)
{ getimage(x1,ystart,x2,yend,buf);
  if(!_write(handle,buf,size)!=size)
  { sprintf(ERS,"Write in '%s' failed (GETIMAGE)",file);
    IER=3; fprintf(OUT,"\n%d %s",IER,ERS); return(IER); }
  ystart = yend+1; yend = ystart+n-1;
  if(yend>y2) { yend=y2; size=imagesize(x1,ystart,x2,yend); } }
PUTSETTINGS(); close(handle); free(buf); return(IER);
}

/* ***** */

int PUTIMAGE(x1,y1,x2,y2)
int x1,y1,x2,y2;
{
  void *buf;
  int handle,bytes;
  unsigned int size;
  int i,n,j,ystart,yend;
  char file[80];

  sprintf(file,"c:\\psp\\temp\\img.%03d",IMGLEVEL);
  if((handle=open(file,O_RDONLY | O_BINARY, S_IWRITE | S_IREAD))!=-1)
  { sprintf(ERS,"Image save file '%s' cannot be open (PUTIMAGE)",file);
    IER=1; fprintf(OUT,"\n%d %s",IER,ERS); return(IER); }
  GETSETTINGS(); setviewport(0,0,getmaxx(),getmaxy(),1);
  n= y2-y1+1; i= n/4;
  if((n%4)!=0) n=i+1; else n=i;
  ystart = y1; yend = ystart+n-1; size=imagesize(x1,ystart,x2,yend);
  if( (buf=malloc(size))==NULL)
  { strcpy(ERS,"Memory allocation error - buf - (PUTIMAGE)");
    IER=2; fprintf(OUT,"\n%d %s",IER,ERS); return(IER); }
  for(i=0;i<4;i++)
  { if(!_read(handle,buf,size)!=size)
    { sprintf(ERS,"Read from '%s' failed (PUTIMAGE)",file);
      IER=3; fprintf(OUT,"\n%d %s",IER,ERS); return(IER); }
    putimage(x1,ystart,buf,0);
    ystart = yend+1; yend = ystart+n-1;
    if(yend>y2) { yend=y2; size=imagesize(x1,ystart,x2,yend); } }
  IMGLEVEL=IMGLEVEL-1; PUTSETTINGS(); close(handle); free(buf); remove(file);
  return(IER);
}

/* ***** */

/* ***** PULSE.C ***** */

#include "psp_dcl.h"

/* ***** */

int PULSE_EDT(void)
{
  char ch[2],str[80];
  int i,flag;

```

```

IER=LOAD_BUF(); if(IER>0) return(IER);
GETSETTINGS(); WINDOW(2); IER=GETIMAGE(X1,Y1,X2,Y2);
if(IER>0) { PUTSETTINGS(); return(IER); }
MOLERA(); MESSAGE(20,1); MESSAGE(21,1); WINDOW(7); CURSOR(1); flag=1;
for(;;)
{ i=GET_EV(J); if(((i==1)|| (i==2)|| (i==3))) { DATA_WINDOW(4); flag=1; }
  else if(flag==1) { DATA_WINDOW(4); flag=0; }
  FIOSKEY(ch);
  if(M2!=0)
  { if(MW==2)
    { MESSAGE(20,-1); if((M3>X1)&&(M3<X2)&&(M4>Y1)&&(M4<Y2)) ch[0]='r';
      MESSAGE(21,-1); if((M3>X1)&&(M3<X2)&&(M4>Y1)&&(M4<Y2)) ch[0]='t'; }
    else
    { WINDOW(7);
      if((M3>X1)&&(M3<X2)&&(M4>Y1)&&(M4<Y2))
      { CURSOR(0); J=JJ+(M3-X1)/16; I=II+(M4-Y1)/16; CURSOR(1);
        if(M2==2) ch[0]='t'; } } }
  switch(ch[0])
  { case 'r':
    WINDOW(7); CURSOR(0); IER=FREE_BUF(); WINDOW(2);
    IER=PUTIMAGE(X1,Y1,X2,Y2); PUTSETTINGS(); return(IER);
    case 't':
    CHG=1; TOGGLE(I,J); PUTCHAR(GETCHAR(I,J));
    if(J>0) { J=J-1; PUTCHAR(GETCHAR(I,J)); J=J+1; }
    if(J<M-1) { J=J+1; PUTCHAR(GETCHAR(I,J)); J=J-1; }
    CURSOR(1); DATA_WINDOW(2); break;
  }
  switch(ch[1])
  { case RIGHT:
    if(J==M-1) break;
    if(J<JJ+NC-1) { CURSOR(0); J=J+1; CURSOR(1); break; }
    PULSE_UPDATE(1,0,0,0); JJ=JJ+1; J=J+1; break;
    case LEFT:
    if(J==0) break; if(J>JJ) { CURSOR(0); J=J-1; CURSOR(1); break; }
    PULSE_UPDATE(0,1,0,0); JJ=JJ-1; J=J-1; break;
    case DOWN:
    if(I==N-1) break;
    if(I<II+NL-1) { CURSOR(0); I=I+1; CURSOR(1); break; }
    PULSE_UPDATE(0,0,0,1); II=II+1; I=I+1; break;
    case UP:
    if(I==0) break; if(I>II) { CURSOR(0); I=I-1; CURSOR(1); break; }
    PULSE_UPDATE(0,0,1,0); II=II-1; I=I-1; break;
  }
}
}

/* ***** */

void CURSOR(n)
char n;
{
  register int i,j,c;

  GETSETTINGS(); ; i=I-II; j=J-JJ; if(j%2==0) c=LIGHTGRAY; else c=WHITE;
  if(n>0) setfillstyle(SOLID_FILL,BLUE); else setfillstyle(SOLID_FILL,c);
  bar(16*j+4,16*i+8,16*j+11,16*i+9); PUTSETTINGS(); return;
}

/* ***** */

```

```

void TOGGLE(i,j)
unsigned int i,j;
{
    unsigned int coluna,bit;
    unsigned long int one;

    coluna = j/NC; bit = j%NC; one = 0x80000000;
    PULSE[i*(M/NC)+coluna] = (PULSE[i*(M/NC)+coluna]) ^ (one>>bit);
    return;
}

/* ***** */

void PULSE_UPDATE(r,l,u,d) /* r:right; l:left; u:up; d:down */
int r,l,u,d; /* r,l,u,d = ..-3,-2,-1,0,1,2.. */
{
    int i,j,ii,j1,j2,oldchar,newchar,oldlevel,newlevel,cor,this,last,next;
    static int tc[4][4]
        = { -1,-1,RED,RED,-1,-1,RED,RED,WHITE,WHITE,-1,-1,WHITE,WHITE,-1,-1 };
    static int td[4][4]
        = { -1,RED,-1,RED,WHITE,-1,WHITE,-1,-1,RED,-1,RED,WHITE,-1,WHITE,-1 };
    char str[80];

    GETSETTINGS();

    /* atualiza nome das linhas */

    if((u-d)!=0)
    { WINDOW(6); setcolor(WHITE); setttextstyle(0,0,1);
      for(i=II;i<(II+NL);i++)
      { sprintf(str,"%10.10s",&NAME[11*i]); outtextxy(4,16*(i-II)+4,str); }
      setcolor(BLUE);
      for(i=II+d-u;i<(II+NL+d-u);i++)
      { sprintf(str,"%10.10s",&NAME[11*i]);
        outtextxy(4,16*(i-II-d+u)+4,str); } }

    /* atualiza tipos de eventos */

    if((r-l)!=0)
    { WINDOW(9); j=J; i=I; ii=II; II=0; I=0;
      for(J=JJ;J<JJ+NC;J++)
      { oldchar=GET_EV(J); newchar=GET_EV(J+r-l);
        if(newchar!=oldchar) PUTCHAR(8+newchar); } }

    /* atualiza loops */

    if((r-l)!=0)
    { WINDOW(5);
      for(I=0;I<5;I++)
      { LOOP_LIMITS(&j1,&j2);
        if(j1>=0)
        { if(r-l<0)
          { for(J=j1;J<j1-r+1;J++) if((J>=JJ)&&(J<JJ+NC)) PUTCHAR(8);
            if(j2==j1) { J=j1-r+1; if((J>=JJ)&&(J<JJ+NC)) PUTCHAR(3); }
            else
            { J=j1-r+1; if((J>=JJ)&&(J<JJ+NC)) PUTCHAR(2);
              for(J=j2;J<j2-r+1;J++) if((J>=JJ)&&(J<JJ+NC)) PUTCHAR(0);
              J=j2-r+1; if((J>=JJ)&&(J<JJ+NC)) PUTCHAR(1); } }
          if(r-l>0)
          { for(J=j2;J>j2-r+1;J--) if((J>=JJ)&&(J<JJ+NC)) PUTCHAR(8);

```

```

        if(j2==j1) { J=j2-r+1; if((J>=JJ)&&(J<JJ+NC)) PUTCHAR(3); }
        else
        { J=j2-r+1; if((J>=JJ)&&(J<JJ+NC)) PUTCHAR(1);
          for(J=j1;J>j1-r+1;J--) if((J>=JJ)&&(J<JJ+NC)) PUTCHAR(0);
          J=j1-r+1; if((J>=JJ)&&(J<JJ+NC)) PUTCHAR(2); } }
    J=j; I=i; II=ii; }

/* atualiza tabela de pulsos */

WINDOW(7); setlinestyle(SOLID_LINE,1,NORM_WIDTH);
for(i=II;i<II+NL;i++)
{ for(j=JJ;j<JJ+NC;j++)
  { oldchar = GETCHAR(i,j); newchar = GETCHAR(i+d-u,j+r-1);
    if(newchar!=oldchar)
    { if(oldchar>3) { oldlevel=1; oldchar=oldchar-4; } else oldlevel=0;
      if(newchar>3) { newlevel=1; newchar=newchar-4; } else newlevel=0;
      if((cor=tc[oldchar][newchar])>=0) /* linha c */
      { if((cor==WHITE)&&((j-JJ)%2==0)) cor=LIGHTGRAY; setcolor(cor);
        line(16*(j-JJ),16*(i-II)+3+2*newlevel,
              16*(j-JJ),16*(i-II)+10+2*newlevel); }
      if((cor=td[oldchar][newchar])>=0) /* linha d */
      { if((cor==WHITE)&&((j-JJ)%2==0)) cor=LIGHTGRAY; setcolor(cor);
        line(16*(j-JJ)+15,16*(i-II)+3+2*newlevel,
              16*(j-JJ)+15,16*(i-II)+10+2*newlevel); }
      if(oldlevel<newlevel) /* acende a, apaga b */
      { setcolor(RED);
        line(16*(j-JJ),16*(i-II)+3,16*(j-JJ)+15,16*(i-II)+3);
        line(16*(j-JJ),16*(i-II)+4,16*(j-JJ)+15,16*(i-II)+4);
        if((j-JJ)%2==0) setcolor(LIGHTGRAY); else setcolor(WHITE);
        line(16*(j-JJ)+((newchar&2)>>1),16*(i-II)+11,
              16*(j-JJ)+15-(newchar&1),16*(i-II)+11);
        line(16*(j-JJ)+((newchar&2)>>1),16*(i-II)+12,
              16*(j-JJ)+15-(newchar&1),16*(i-II)+12); }
      if(oldlevel>newlevel) /* acende b, apaga a */
      { setcolor(RED);
        line(16*(j-JJ),16*(i-II)+11,16*(j-JJ)+15,16*(i-II)+11);
        line(16*(j-JJ),16*(i-II)+12,16*(j-JJ)+15,16*(i-II)+12);
        if((j-JJ)%2==0) setcolor(LIGHTGRAY); else setcolor(WHITE);
        line(16*(j-JJ)+((newchar&2)>>1),16*(i-II)+3,
              16*(j-JJ)+15-(newchar&1),16*(i-II)+3);
        line(16*(j-JJ)+((newchar&2)>>1),16*(i-II)+4,
              16*(j-JJ)+15-(newchar&1),16*(i-II)+4); } } }
    PUTSETTINGS(); return;
  }
}

/* ***** */

int GET_PULSE(i,j) /* get pulse status at line i, coluna j */
int i,j; /* retorna : 0 (off) , 1 (on) */
{
    unsigned int coluna;
    unsigned long int bit,one=0x80000000;

    if((i>=NL)||((i<0)||((j>=NC)||((j<0)) return(0);
    coluna=j/NC; bit=one>>(j%NC);
    if((PULSE[i*(M/NC)+coluna]&bit)!=0) return(1); else return(0);
}

/* ***** */

```

```

void SET_PULSE(i,j,n) /* set pulse status at line i, coluna j */
int i,j,n;           /* to off (if n=0) on (if n!=0) */
{
    unsigned int coluna;
    unsigned long int bit,one=0x80000000;

    if((i>=NL)||!(i<0)||!(j>=NC)||!(j<0)) return;
    coluna=j/NC; bit=one>>(j%NC);
    if(n==0) PULSE[i*(M/NC)+coluna]=(PULSE[i*(M/NC)+coluna]^bit)^bit;
    else     PULSE[i*(M/NC)+coluna]= PULSE[i*(M/NC)+coluna]^bit;
    return;
}

/* ***** */
/* ***** LOOP.C ***** */

#include "psp_dcl.h"

/* ***** */

int LOOP_EDT(void)
{
    char ch[2],str[80],*lixo;
    int coluna1,bit1,coluna2,bit2;
    unsigned long int t,one=0x80000000,f=0xFFFFFFFF;
    int i,j,isave,jsave,iisave,jjsave,nl=5,this,last,next;

    IER=LOAD_BUF(); if(IER>0) return(IER);
    GETSETTINGS(); WINDOW(2); IER=GETIMAGE(X1,Y1,X2,Y2);
    if(IER>0) { PUTSETTINGS(); return(IER); }
    MOLDURA(); MESSAGE(120,1); MESSAGE(121,1); MESSAGE(122,1); MESSAGE(123,1);
    MESSAGE(124,1); WINDOW(5);
    isave=I; jsave=J; iisave=II; jjsave=JJ; I=0; II=0; CURSOR(1);
    for(;;)
    { BIOSKEY(ch);
      if(M2!=0)
      { if(MW==2)
        { MESSAGE(120,-1); if((M3>X1)&&(M3<X2)&&(M4>Y1)&&(M4<Y2)) ch[0]='r';
          MESSAGE(121,-1); if((M3>X1)&&(M3<X2)&&(M4>Y1)&&(M4<Y2)) ch[0]='b';
          MESSAGE(122,-1); if((M3>X1)&&(M3<X2)&&(M4>Y1)&&(M4<Y2)) ch[0]='e';
          MESSAGE(123,-1); if((M3>X1)&&(M3<X2)&&(M4>Y1)&&(M4<Y2)) ch[0]='o';
          MESSAGE(124,-1); if((M3>X1)&&(M3<X2)&&(M4>Y1)&&(M4<Y2)) ch[0]='c'; }
        else
        { WINDOW(5);
          if((M3>X1)&&(M3<X2)&&(M4>Y1)&&(M4<Y2))
          { CURSOR(0); J=JJ+(M3-X1)/16; I=II+(M4-Y1)/16; CURSOR(1); } } }
      switch(ch[0])
      { case 'r': /* return */
        WINDOW(5); CURSOR(0); IER=FREE_BUF();
        I=isave; J=jsave; II=iisave; JJ=jjsave;
        WINDOW(2); IER=PUTIMAGE(X1,Y1,X2,Y2); PUTSETTINGS(); return(IER);
      case 'b': /* begin */
        CHG=1; coluna1=J/NC; bit1=J%NC; coluna2=coluna1; bit2=bit1;
        for(j=0;j<coluna1;j++) LOOP[I*(M/NC)+j]=0;
        LOOP[I*(M/NC)+coluna1]=(LOOP[I*(M/NC)+coluna1])^(one>>bit1);
        LOOP[I*(M/NC)+coluna1]=(LOOP[I*(M/NC)+coluna1])&(f>>bit1);
        for(j=M/NC-1;j>=coluna1;j--)
        { if(LOOP[I*(M/NC)+j]!=0) { coluna2=j; break; } }
        t=LOOP[I*(M/NC)+coluna2]; bit2=-1; while(t!=0) { t<<=1; bit2++; }
      }
    }
}

```

```

fill;
if(coluna1==coluna2)
{ for(j=bit1+1;j<bit2;j++)
  { LOOP[I*(M/NC)+coluna1]=(LOOP[I*(M/NC)+coluna1])|(one>>j); } }
else
{ for(j=bit1+1;j<32;j++)
  { LOOP[I*(M/NC)+coluna1]=(LOOP[I*(M/NC)+coluna1])|(one>>j); }
  for(j=coluna1+1;j<coluna2;j++) { LOOP[I*(M/NC)+j]=f; }
  for(j=0;j<bit2;j++)
  { LOOP[I*(M/NC)+coluna2]=(LOOP[I*(M/NC)+coluna2])|(one>>j); } }
LOOP_DRAW(); CURSOR(1); DATA_WINDOW(5); break;
case 'e':
CHG=1; coluna2=J/NC; bit2=J%NC; coluna1=coluna2; bit1=bit2;
for(j=M/NC-1;j>coluna2;j--) LOOP[I*(M/NC)+j]=0;
LOOP[I*(M/NC)+coluna2]=(LOOP[I*(M/NC)+coluna2])|(one>>bit2);
LOOP[I*(M/NC)+coluna2]=(LOOP[I*(M/NC)+coluna2])&(f<<(31-bit2));
for(j=0;j<=coluna2;j++)
{ if(LOOP[I*(M/NC)+j]!=0) { coluna1=j; break; } }
t=LOOP[I*(M/NC)+coluna1]; bit1=32; while(t!=0) { t>>=1; bit1--; }
goto fill;
case 'o':
COUNT[I]=0; for(j=0;j<M/NC;j++) LOOP[I*(M/NC)+j]=0;
LOOP_DRAW(); CURSOR(1); DATA_WINDOW(5); CHG=1; break;
case 'c':
WINDOW(3); setlinestyle(SOLID_LINE,1,NORM_WIDTH);
setcolor(RED); rectangle(250,26+16*I,294,38+16*I);
INPUT(252,28+16*I,str,5,5,0,1); t=strtoul(str,&lixo,10);
if(t>65535) t=65535; COUNT[I]=(unsigned int) t;
setcolor(WHITE); rectangle(250,26+16*I,294,38+16*I);
DATA_WINDOW(5); CHG=1; WINDOW(5); break;
}
switch(ch[1])
{ case RIGHT:
  if(J==M-1) break;
  if(J<JJ+NC-1) { CURSOR(0); J=J+1; CURSOR(1); break; }
  i=I; j=J; I=isave; II=iisave; PULSE_UPDATE(1,0,0,0);
  I=i; J=j; JJ=JJ+1; J=J+1; CURSOR(1); break;
case LEFT:
  if(J==0) break;
  if(J>JJ) { CURSOR(0); J=J-1; CURSOR(1); break; }
  i=I; j=J; I=isave; II=iisave; PULSE_UPDATE(0,1,0,0);
  I=i; J=j; JJ=JJ-1; J=J-1; CURSOR(1); break;
case DOWN:
  if(I==nl-1) break; CURSOR(0); I=I+1; CURSOR(1); break;
case UP:
  if(I==0) break; CURSOR(0); I=I-1; CURSOR(1); break;
}
}
}

/* ***** */

void LOOP_DRAW(void)
{
  int j,coluna,bit,this,last,next;
  unsigned long int one=0x80000000;

  j=J;
  for(J=JJ;J<JJ+32;J++)
  { coluna=J/NC; bit=J%NC;

```



```

        if((LOOP[I*(M/NC)+coluna]&(one>>bit))!=0) this=1; else this=0;
        if(J>0)
        { coluna=(J-1)/NC; bit=(J-1)%NC;
          if((LOOP[I*(M/NC)+coluna]&(one>>bit))!=0) last=1; else last=0; }
        else last=0;
        if(J<M-1)
        { coluna=(J+1)/NC; bit=(J+1)%NC;
          if((LOOP[I*(M/NC)+coluna]&(one>>bit))!=0) next=1; else next=0; }
        else next=0;
        if(this==0) PUTCHAR(8);
        if(this==1)
        { if( (last==0) && (next==0) ) PUTCHAR(3);
          if( (last==0) && (next==1) ) PUTCHAR(2);
          if( (last==1) && (next==0) ) PUTCHAR(1);
          if( (last==1) && (next==1) ) PUTCHAR(0); } }
J=j; return;
}

/* ***** */

void LOOP_LIMITS(j1,j2)      /* Retorna os limites do loop I:      */
int *j1,*j2;                /* j1 - inicio      j2 - fim      */
{                            /* Se o loop nao esta' definido: j1=j2=-1 */
    int j,coluna1,bit1,coluna2,bit2;
    unsigned long t;

    for(j=0;j<M/NC;j++) if(LOOP[I*(M/NC)+j]!=0) { coluna1=j; break; }
    if(j==M/NC) { *j1=-1; *j2=-1; return; }
    for(j=M/NC-1;j>=coluna1;j--) if(LOOP[I*(M/NC)+j]!=0) { coluna2=j; break; }
    t=LOOP[I*(M/NC)+coluna1]; bit1=32; while(t!=0) { t>>=1; bit1--; }
    t=LOOP[I*(M/NC)+coluna2]; bit2=-1; while(t!=0) { t<<=1; bit2++; }
    *j1=coluna1*NC+bit1; *j2=coluna2*NC+bit2; return;
}

/* ***** */

/* ***** FCT.C ***** */

#include "psp_dcl.h"

/* ***** */

int FCT_EDT(choice)      /* choice (saida) : nome da funcao esolhida. */
char *choice;
{
    FILE *fct;
    char *name, *expr,str[80],ch[2];
    int i,j,k,l,n=0,m,chg=0,nmax=35;

    IER=GET_DEF(3); if(IER>0) return(IER);
    if((fct=fopen(DEF,"r"))==NULL)
    { sprintf(ERS,"Function file '%s' cannot be open (FCT_EDT)",DEF);
      IER=1; fprintf(OUT,"\n%d%s",IER,ERS); return(IER); }
    name = calloc(15*nmax,sizeof(char)); if(!name)
    { strcpy(ERS,"Memory allocation error - name - (FCT_EDT)");
      IER=2; fprintf(OUT,"\n%d %s",IER,ERS); return(IER); }
    expr = calloc(100*nmax,sizeof(char)); if(!expr)
    { strcpy(ERS,"Memory allocation error - expr - (FCT_EDT)");
      IER=3; fprintf(OUT,"\n%d %s",IER,ERS); return(IER); }
}

```

```

for(;;)
{ if(n==nmax)
  { fprintf(OUT,"\\nWARNING: Function file has been truncated - FCT_EDT");
    break; }
  if((fscanf(fct,"%s%s",&name[15*n],&expr[100*n]))==EOF) break;
  n=n+1; }
fclose(fct);
GETSETTINGS(); WINDOW(3); GETIMAGE(X1,Y1,X2,Y2); MOLDURA(); WINDOW(2);
GETIMAGE(X1,Y1,X2,Y2); MOLDURA(); MESSAGE(50,1); MESSAGE(51,1);
MESSAGE(52,1); MESSAGE(53,1);
setfillstyle(SOLID_FILL,WHITE);
settextstyle(0,0,1); setlinestyle(SOLID_LINE,0,THICK_WIDTH); setcolor(BLUE);
WINDOW(2); line(3,18,635,18); setlinestyle(SOLID_LINE,0,NORM_WIDTH);
WINDOW(3); setcolor(RED); line(130,4,130,Y2-Y1-4); line(256,4,256,Y2-Y1-4);
line(382,4,382,Y2-Y1-4); line(508,4,508,Y2-Y1-4); setcolor(BLUE);
for(l=0;l<n;l++) { k=l/7; m=l%7; sprintf(str,"%14.14s",&name[15*l]);
                  outtextxy(7+126*k,11+16*m,str); }
i=0; j=0;
for(;;)
{ WINDOW(2); setcolor(BLUE); sprintf(str,"%-78.78s",&expr[100*(7*i+j)]);
  outtextxy(6,8,str); WINDOW(3); setcolor(RED);
  rectangle(5+126*i,9+16*j,129+126*i,21+16*j); BIOSKEY(ch);
  if(M2==1)
  { if(MW==3)
    { WINDOW(3);
      if((M3>X1+5)&&(M3<X2-5)&&(M4>Y1+9)&&(M4<Y2-9))
      { k=(M3-X1-5)/126; m=(M4-Y1-9)/16;
        if(7*k+m<n)
        { setcolor(WHITE); rectangle(5+126*i,9+16*j,129+126*i,21+16*j);
          WINDOW(2); bar(6,8,630,16); i=k; j=m; } } }
    if(MW==2)
    { MESSAGE(50,-1); if((M3>X1)&&(M3<X2)&&(M4>Y1)&&(M4<Y2)) ch[0]='r';
      MESSAGE(51,-1); if((M3>X1)&&(M3<X2)&&(M4>Y1)&&(M4<Y2)) ch[0]='d';
      MESSAGE(52,-1); if((M3>X1)&&(M3<X2)&&(M4>Y1)&&(M4<Y2)) ch[0]='a';
      MESSAGE(53,-1); if((M3>X1)&&(M3<X2)&&(M4>Y1)&&(M4<Y2)) ch[0]='w'; } }
    switch(ch[0])
    { case 'r':
      /* return */
      strcpy(choice,&name[15*(7*i+j)]);
      if(chg!=0)
      { strcpy(str,"Functions are not saved !");
        if(SAVE_ASK(str)==1)
        { if((fct=fopen(DEF,"w"))==NULL)
          { strcpy(ERS,"WARNING: Functions were not saved (FCT_EDT)");
            fprintf(OUT,"\\n%s",ERS); }
          for(l=0;l<n;l++)
            fprintf(fct,"%-14.14s %s\\n",&name[15*l],&expr[100*l]); } }
        fclose(fct); WINDOW(2); IER=PUTIMAGE(X1,Y1,X2,Y2);
        if(IER>0) { PUTSETTINGS(); return(IER); }
        WINDOW(3); IER=PUTIMAGE(X1,Y1,X2,Y2);
        PUTSETTINGS(); return(IER);
      }
    case 'd':
      /* delete */
      if(n==1) break; WINDOW(2); bar(6,8,630,16);
      WINDOW(3); bar(5+126*i,9+16*j,129+126*i,21+16*j);
      for(l=7*i+j;l<n;l++)
      { k=l/7; m=l%7; sprintf(str,"%14.14s",&name[15*l]);
        setcolor(WHITE); outtextxy(7+126*k,11+16*m,str); }
      n=n-1;
      for(l=7*i+j;l<n;l++)
      { strcpy(&name[15*l],&name[15*(l+1)]);
        strcpy(&expr[100*l],&expr[100*(l+1)]); }
    }
  }
}

```

```

    for(l=7*i+j;l<n;l++)
    { k=l/7; m=l%7; sprintf(str,"%14.14s",&name[15*l]);
      setcolor(BLUE); outtextxy(7+126*k,11+16*m,str); }
    chg=1; if(7*i+j==n) { if(j>0) j--; else { j=6; i--; } }
    break;
case 'w':
    WINDOW(3); INPUT(7+126*i,11+16*j,&name[15*(7*i+j)],14,14,1,1);
    sprintf(str,"%14.14s",&name[15*(7*i+j)]);
    setcolor(BLUE); outtextxy(7+126*i,11+16*j,str);
    WINDOW(2); INPUT(6,8,&expr[100*(7*i+j)],99,78,1,1);
    chg=1; break;
case 'a':
    if(n==nmax) break; WINDOW(2); bar(6,8,630,16); WINDOW(3);
    setcolor(WHITE); rectangle(5+126*i,9+16*j,129+126*i,21+16*j);
    n=n+1; i=(n-1)/7; j=(n-1)%7;
    setcolor(BLUE); rectangle(5+126*i,9+16*j,129+126*i,21+16*j);
    INPUT(7+126*i,11+16*j,&name[15*(7*i+j)],14,14,0,1);
    sprintf(str,"%14.14s",&name[15*(7*i+j)]);
    setcolor(BLUE); outtextxy(7+126*i,11+16*j,str);
    WINDOW(2); INPUT(6,8,&expr[100*(7*i+j)],99,78,0,1);
    chg=1; break;
}
switch(ch[1])
{ case RIGHT:
    if((i==(n-1)/7)||i==4) break; WINDOW(2); bar(6,8,630,16); WINDOW(3);
    setcolor(WHITE); rectangle(5+126*i,9+16*j,129+126*i,21+16*j);
    i=i+1; if(7*i+j+1>=n) j=n%7-1; break;
case LEFT:
    if(i==0) break; WINDOW(2); bar(6,8,630,16); WINDOW(3);
    setcolor(WHITE); rectangle(5+126*i,9+16*j,129+126*i,21+16*j);
    i=i-1; break;
case UP:
    if(j==0) break; WINDOW(2); bar(6,8,630,16); WINDOW(3);
    setcolor(WHITE); rectangle(5+126*i,9+16*j,129+126*i,21+16*j);
    j=j-1; break;
case DOWN:
    if((j==6)||7*i+j>=n-1) break; WINDOW(2); bar(6,8,630,16); WINDOW(3);
    setcolor(WHITE); rectangle(5+126*i,9+16*j,129+126*i,21+16*j);
    j=j+1; break;
}
}
}

/* ***** */

int CHK_FCT(void)
{
    FILE *fct;

    IER=GET_DEF(3); if(IER>0) exit(IER);
    if((fct=fopen(DEF,"r"))==NULL)
    { fprintf(OUT,"\nWARNING: Function file cannot be find - CHK_FCT");
      fprintf(OUT,"\n          A new function file will be created");
      if((fct=fopen(DEF,"w"))==NULL)
      { strcpy(ERS,"Failure trying to create a new table file - CHK_FCT");
        IER=1; fprintf(OUT,"\n%d%s",IER,ERS); return(IER); } }
    fclose(fct); return(IER);
}

/* ***** */

```

```

int FCT_EXPR(choice,expr) /* Devolve a expressao (expr) definida para a */
char *choice;             /* funcao (choice). */
char *expr;               /* Se nao encontrar devolve expr = *-----* */
{
    FILE *fct;
    int i;
    char str[80],name[15];

    IER=GET_DEF(3); if(IER>0) return(IER);
    if((fct=fopen(DEF,"r"))==NULL)
    { sprintf(ERS,"Function file '%s' cannot be open (FCT_EXPR)",DEF);
      IER=1; fprintf(OUT,"\n%d%s",IER,ERS); return(IER); }
    REM_BLANK(choice);
    for(;;)
    { if(fscanf(fct,"%s",name)==EOF) { strcpy(expr,"*-----*"); break; }
      fscanf(fct,"%s",expr); if(strcmp(choice,name)==0) break; }
    fclose(fct); return(IER);
}

/* ***** */

int FCT_EVAL(name,answer) /* calcula o valor (answer) da funcao definida */
char *name;              /* por name, em multiplos de 100 ns. */
unsigned long int *answer; /* name : entrada; answer : saida */
{
    FILE *fct;
    char str[80];
    float a;
    int i,buflag;
    double ans;

    IER=GET_DEF(3); if(IER>0) return(IER);
    if(BUF==NULL) buflag=0; else buflag=1;
    if(buflag==0) { IER=LOAD_BUF(); if(IER>0) return(IER); }
    for(i=0;i<26;i++) VARS[i]=0.0;
    for(i=0;i<M;i++) /* valor do tempo em microsegundos */
    { if(TAG[i]!='') VARS[TAG[i]-'a']=((double) TIME[i]) / 10.0; }
    if(buflag==0) { IER=FREE_BUF(); if(IER>0) return(IER); }
    PROG = calloc(100,sizeof(char));
    if(!PROG) { strcpy(ERS,"Memory allocation error - FCT_EVAL");
      IER=1; fprintf(OUT,"\n%d %s",IER,ERS); return(IER); }
    if((fct=fopen(DEF,"r"))==NULL)
    { sprintf(ERS,"Function file '%s' cannot be open (FCT_EVAL)",DEF);
      IER=2; fprintf(OUT,"\n%d%s",IER,ERS); free(PROG); return(IER); }
    REM_BLANK(name);
    for(;;)
    { if(fscanf(fct,"%s%s",str,PROG)!=2)
      { sprintf(ERS,
        "WARNING: Function '%s' not defined in '%s' - FCT_EVAL",name,DEF);
        fprintf(OUT,"\n%s",ERS); break; }
      if(strcmp(str,name)==0) /* retorna tempo em multiplos de 100 ns */
      { EVAL_EXP(&ans); sprintf(str,"%9.1lf us",ans); INTTIME(str,answer);
        break; } }
    free(PROG); fclose(fct); return(0);
}

/* ***** */

void EVAL_EXP(answer) /* ponto de entrada do analisador */
double *answer;

```

```

{
    GET_TOKEN(); if(!*TOKEN) { ERROR(2); return; }
    EVAL_EXP1(answer); return;
}

/* ***** */

void EVAL_EXP1(answer)                /* processa uma atribuicao */
double *answer;
{
    int slot;
    char ttok_type,temp_token[80];

    if(TOK_TYPE == VARIABEL)          /* salva o token antigo */
    { strcpy(temp_token,TOKEN); ttok_type = TOK_TYPE;
      slot = toupper(*TOKEN)-'A';      /* calcula o indice da variavel */
      GET_TOKEN();
      if(*TOKEN != '=')
      { PUTBACK();                    /* devolve o token atual */
        strcpy(TOKEN,temp_token);    /* restaura token antigo - nenhuma atrib */
        TOK_TYPE = ttok_type; }
      else
      { GET_TOKEN();                  /* pega proxima parte da expressao */
        EVAL_EXP2(answer);
        VARS[slot] = *answer; return; } }
    EVAL_EXP2(answer); return;
}

/* ***** */

void EVAL_EXP2(answer)                /* soma ou subtrai dois termos */
double *answer;
{
    register char op;
    double temp;

    EVAL_EXP3(answer);
    while((op = *TOKEN) == '+' || op == '-')
    { GET_TOKEN(); EVAL_EXP3(&temp);
      switch(op)
      { case '-': *answer = *answer - temp; break;
        case '+': *answer = *answer + temp; break; } }
    return;
}

/* ***** */

void EVAL_EXP3(answer)                /* multiplica ou divide dois fatores */
double *answer;
{
    register char op;
    double temp;

    EVAL_EXP4(answer);
    while((op = *TOKEN) == '*' || op == '/' || op == '%')
    { GET_TOKEN(); EVAL_EXP4(&temp);
      switch(op)
      { case '*': *answer = *answer * temp; break;
        case '/':
          if(temp==0) { ERROR(3); break; }

```

```

        *answer = *answer / temp; break;
    case '%': *answer = (int) *answer % (int) temp; break; } }
    return;
}

/* ***** */

void EVAL_EXP4(answer) /* processa um expoente */
double *answer;
{
    double temp, ex;
    register int t;

    EVAL_EXP5(answer);
    if(*TOKEN == '^')
    { GET_TOKEN(); EVAL_EXP4(&temp); ex = *answer;
      if(temp==0.0) { *answer = 1.0; return; }
      for(t=temp-1;t>0;--t) *answer = (*answer) * (double)ex; }
    return;
}

/* ***** */

void EVAL_EXP5(answer) /* avalia um + ou - unario */
double *answer;
{
    register char op;

    op=0;
    if((TOK_TYPE == DELIMITADOR) && *TOKEN == '+' || *TOKEN == '-')
    { op = *TOKEN; GET_TOKEN(); }
    EVAL_EXP6(answer); if(op == '-') *answer = -(*answer);
    return;
}

/* ***** */

void EVAL_EXP6(answer) /* processa uma expressao entre parenteses */
double *answer;
{
    if((*TOKEN == '('))
    { GET_TOKEN(); EVAL_EXP2(answer);
      if(*TOKEN != ')') SERROR(1); GET_TOKEN(); }
    else ATOM(answer); return;
}

/* ***** */

void ATOM(answer)
double *answer;
{
    switch(TOK_TYPE)
    { case VARIABEL: *answer = FIND_VAR(TOKEN); GET_TOKEN(); return;
      case NUMERO   : *answer = atof(TOKEN);      GET_TOKEN(); return;
      default       : SERROR(0); } /* erro de sintaxe na expressao */
    return;
}

/* ***** */

```

```

void PUTBACK(void)                                /* devolve um token a stream de entrada */
{
    char *t;

    t = TOKEN; for(; *t; t++) PROG--; return;
}

/* ***** */

void SERROR(error)                                /* apresenta um erro */
int error;
{
    switch(error)
    { case 0:
        strcpy(ERS,"Erro de sintaxe - FCT"); IER=10; break;
      case 1:
        strcpy(ERS,"Falta parenteses - FCT"); IER=11; break;
      case 2:
        strcpy(ERS,"Nenhuma expressao presente - FCT"); IER=12; break;
      case 3:
        strcpy(ERS,"Divisao por zero - FCT"); IER=13; break;
      default:
        strcpy(ERS,"Invalid call to function SERROR - FCT"); IER=14; break;
    }
    fprintf(OUT,"\n%d %s",IER,ERS); return;
}

/* ***** */

void GET_TOKEN(void)                               /* devolve o proximo token */
{
    register char *temp;

    TOK_TYPE = 0; temp = TOKEN; *temp = '\0';
    if(!*PROG) return;
    while(isspace(*PROG)) ++PROG;                  /* final de expressao */
    if(strchr("+-*/%^=()", *PROG))                 /* ignora espacos em branco */
    { TOK_TYPE = DELIMITADOR;
      *temp++ = *PROG++; }
    else if(isalpha(*PROG))                         /* avanca para o proximo char */
    { while(!ISDELIM(*PROG)) *temp++ = *PROG++;
      TOK_TYPE = VARIABEL; }
    else if(isdigit(*PROG))
    { while(!ISDELIM(*PROG)) *temp++ = *PROG++; TOK_TYPE = NUMERO; }
    *temp = '\0'; return;
}

/* ***** */

int ISDELIM(c)                                     /* devolve verdadeiro se c e' um delimitador */
char c;
{
    if(strchr(" +-*/%^=()",c) || c==9 || c=='\r' || c==0) return 1;
    return 0;
}

/* ***** */

double FIND_VAR(s)
char *s;

```

```

!
if(!isalpha(*s)) { SERROR(1); return 0.0; }
return VARS[toupper(*TOKEN)-'A'];
}

/* ***** */

/* ***** PLOT.C ***** */

#include "psp_def.h"

/* ***** */

void PLOT_DATA(void)
{
    int i,x,y,chn;
    float xmin,ymin,xmax,ymax,xp,yp,x1,y1,x2,y2;

    GETSETTINGS(); setfillstyle(SOLID_FILL,WHITE); x1=8.0; y1=8.0;
    xmin=1.0; xmax=(float)(CHS); ymin=-32768.0; ymax=32767.0;
    WINDOW(57); y2=(float)(Y2-Y1-8); x2=(float)(X2-X1-8);
    setcolor(RED); bar((int)x1,(int)y1,(int)x2,(int)y2);
    setlinestyle(SOLID_LINE,0,NORM_WIDTH);
    for(i=0;i<CHS;i++)
    { x=i+1; y=(int)(DAT_REAL[i]);
      xp=x1+(int)((float)x-xmin)*(x2-x1)/(xmax-xmin);
      yp=y2-(int)((float)y-ymin)*(y2-y1)/(ymax-ymin);
      if(i==0) moveto((int)xp,(int)yp); else lineto((int)xp,(int)yp); }
    setcolor(WHITE); line((int)x1,(int)((y1+y2)/2),(int)x2,(int)((y1+y2)/2));
    setlinestyle(DASHED_LINE,0,NORM_WIDTH);
    setcolor(BLUE); line((int)x1,(int)((y1+y2)/2),(int)x2,(int)((y1+y2)/2));
    WINDOW(58); y2=(float)(Y2-Y1-8); x2=(float)(X2-X1-8);
    setcolor(RED); bar((int)x1,(int)y1,(int)x2,(int)y2);
    setlinestyle(SOLID_LINE,0,NORM_WIDTH);
    for(i=0;i<CHS;i++)
    { x=i+1; y=(int)(DAT_IMAG[i]);
      xp=x1+(int)((float)x-xmin)*(x2-x1)/(xmax-xmin);
      yp=y2-(int)((float)y-ymin)*(y2-y1)/(ymax-ymin);
      if(i==0) moveto((int)xp,(int)yp); else lineto((int)xp,(int)yp); }
    setcolor(WHITE); line((int)x1,(int)((y1+y2)/2),(int)x2,(int)((y1+y2)/2));
    setlinestyle(DASHED_LINE,0,NORM_WIDTH);
    setcolor(BLUE); line((int)x1,(int)((y1+y2)/2),(int)x2,(int)((y1+y2)/2));
    PUTSETTINGS(); return;
}

/* ***** */

void PLOT_FFT(void)
{
    int i,x,y;
    float xmin,ymin,xmax,ymax,xp,yp,x1,y1,x2,y2;

    GETSETTINGS(); setfillstyle(SOLID_FILL,WHITE); x1=8.0; y1=8.0;
    xmin=1.0; xmax=(float)(CHS/2); ymax=-32768.0; ymin=32767.0;
    for(i=0;i<CHS;i++)
    { if((float)(FFT_REAL[i])>ymax) ymax=(float)(FFT_REAL[i]);
      if((float)(FFT_REAL[i])<ymin) ymin=(float)(FFT_REAL[i]);
      if((float)(FFT_IMAG[i])>ymax) ymax=(float)(FFT_IMAG[i]);
      if((float)(FFT_IMAG[i])<ymin) ymin=(float)(FFT_IMAG[i]); }
    WINDOW(59); y2=(float)(Y2-Y1-8); x2=(float)(X2-X1-8);

```



```

bar((int)x1,(int)y1,(int)x2,(int)y2); setcolor(RED);
setlinestyle(SOLID_LINE,0,NORM_WIDTH);
for(i=0;i<CHS/2;i++)
{ x=i+1; y=FFT_REAL[i];
  xp=x1+(int)((float)x-xmin)*(x2-x1)/(xmax-xmin);
  yp=y2-(int)((float)y-ymin)*(y2-y1)/(ymax-ymin);
  if(i==0) moveto((int)xp,(int)yp); else lineto((int)xp,(int)yp); }
yp=y2-(int)((-ymin)*(y2-y1)/(ymax-ymin));
setcolor(WHITE); line((int)x1,(int)yp,(int)x2,(int)yp);
setlinestyle(DASHED_LINE,0,NORM_WIDTH);
setcolor(BLUE); line((int)x1,(int)yp,(int)x2,(int)yp);
WINDOW(60); y2=(float)(Y2-Y1-8); x2=(float)(X2-X1-8);
bar((int)x1,(int)y1,(int)x2,(int)y2); setcolor(RED);
setlinestyle(SOLID_LINE,0,NORM_WIDTH);
for(i=0;i<CHS/2;i++)
{ x=i+1; y=FFT_IMAG[i];
  xp=x1+(int)((float)x-xmin)*(x2-x1)/(xmax-xmin);
  yp=y2-(int)((float)y-ymin)*(y2-y1)/(ymax-ymin);
  if(i==0) moveto((int)xp,(int)yp); else lineto((int)xp,(int)yp); }
yp=y2-(int)((-ymin)*(y2-y1)/(ymax-ymin));
setcolor(WHITE); line((int)x1,(int)yp,(int)x2,(int)yp);
setlinestyle(DASHED_LINE,0,NORM_WIDTH);
setcolor(BLUE); line((int)x1,(int)yp,(int)x2,(int)yp);
PUTSETTINGS(); return;
}

/* ***** */
/* ***** FFT.C ***** */
#include "psp_dcl.h"
/* ***** */

void FFT (void)
{
  double co,si;
  int i,ir,j,k,kdim2,l2dim,l2dim1,dim2,tmp,r,tmpi;

  for(i=0;i<CHS;i++) { FFT_REAL[i]=DAT_REAL[i]; FFT_IMAG[i]=DAT_IMAG[i]; }
  dim2=CHS; l2dim=l2dim1=LOG2(CHS);
  for(j=0;j<l2dim;j++)
  { l2dim1--; dim2 >>= 1; ir=k=0;
    do
    { co=CO[ir]; si=SI[ir];
      for(i=0;i<dim2;i++,k++)
      { kdim2=k+dim2;
        tmp=FFT_REAL[kdim2]*co+FFT_IMAG[kdim2]*si;
        tmpi=FFT_IMAG[kdim2]*co-FFT_REAL[kdim2]*si;
        FFT_REAL[kdim2]=(FFT_REAL[k]-tmp) >> 1;
        FFT_IMAG[kdim2]=(FFT_IMAG[k]-tmpi) >> 1;
        FFT_REAL[k]      =(FFT_REAL[k]+tmp) >> 1;
        FFT_IMAG[k]      =(FFT_IMAG[k]+tmpi) >> 1; }
      k += dim2; ir=BITREV(k>>l2dim1,l2dim); }
    while(k<CHS ); }
  for(i=0;i<CHS;i++)
  if((ir=BITREV(i,l2dim))>i)
  { CHGDATA(&FFT_REAL[i],&FFT_REAL[ir]);
    CHGDATA(&FFT_IMAG[i],&FFT_IMAG[ir]); }
}

```

```

/* ***** */

void CHGDATA (pr,pi)
int *pr,*pi;
{
    int tmp;

    tmp=*pr; *pr=*pi; *pi=tmp;
}

/* ***** */

int BITREV (j,nu)
int j,nu;
{
    int bitrv,i,jl;

    for(bitrv=i=0;i<nu;i++)
    { jl=j >> 1; bitrv <=< 1; bitrv += j - ( jl << 1 ); j = jl; }
    return(bitrv);
}

/* ***** */

int LOG2 (val)
int val;
{
    int ct=0;

    while(!(val&1)) { val >>= 1; ct++; }
    return(ct);
}

/* ***** */

int FFT_ON (void)
{
    double arg;
    int i;

    DAT_REAL = calloc(CHS,sizeof(int));
    if(DAT_REAL==NULL)
    { strcpy(ERS,"Memory allocation error - DAT_REAL - (FFT_ON)");
      IER=1; fprintf(OUT,"\n%d %s",IER,ERS); return(IER); }
    DAT_IMAG = calloc(CHS,sizeof(int));
    if(DAT_IMAG==NULL)
    { strcpy(ERS,"Memory allocation error - DAT_REAL - (FFT_ON)");
      IER=2; free(DAT_REAL); fprintf(OUT,"\n%d %s",IER,ERS); return(IER); }
    FFT_REAL = calloc(CHS,sizeof(int));
    if(FFT_REAL==NULL)
    { strcpy(ERS,"Memory allocation error - FFT_REAL - (FFT_ON)");
      IER=3; free(DAT_REAL); free(DAT_IMAG);
      fprintf(OUT,"\n%d %s",IER,ERS); return(IER); }
    FFT_IMAG = calloc(CHS,sizeof(int));
    if(FFT_IMAG==NULL)
    { strcpy(ERS,"Memory allocation error - FFT_IMAG - (FFT_ON)");
      IER=4; free(DAT_REAL); free(DAT_IMAG); free(FFT_REAL);
      fprintf(OUT,"\n%d %s",IER,ERS); return(IER); }
    CO = calloc(CHS/2,sizeof(double));
    if(CO==NULL)

```

```

{ strcpy(ERS,"Memory allocation error - CO - (FFT_ON)");
  IER=5; free(DAT_REAL); free(DAT_IMAG); free(FFT_REAL); free(FFT_IMAG);
  fprintf(OUT,"\n%d %s",IER,ERS); return(IER); }
SI = calloc(CHS/2,sizeof(double));
if(SI==NULL)
{ strcpy(ERS,"Memory allocation error - SI - (FFT_OFF)");
  IER=6; free(DAT_REAL); free(DAT_IMAG); free(FFT_REAL); free(FFT_IMAG);
  free(CO); fprintf(OUT,"\n%d %s",IER,ERS); return(IER); }
for(i=0;i<CHS/2;i++) { arg=TWOPI*i/CHS; CO[i]=cos(arg); SI[i]=sin(arg); }
return(IER);
}

/* ***** */

void FFT_OFF(void)
{
  free(DAT_REAL); free(DAT_IMAG); free(FFT_REAL); free(FFT_IMAG);
  free(CO); free(SI); DAT_REAL=NULL; DAT_IMAG=NULL; FFT_REAL=NULL;
  FFT_IMAG=NULL; CO=NULL; SI=NULL; return;
}

/* ***** */

int GET_PHASE(float *phase)
{
  int i,j,k,ni,n=8;
  int *sig_real,*sig_imag;
  double x[8],y[8],p[2],g[4],gi[4],d[16],cc,c[2],det,ang,new,old;

  /* saving input data */

  sig_real = calloc(CHS,sizeof(int));
  if(!sig_real)
  { strcpy(ERS,"Memory allocation error - sig_real - PHASE_SEARCH");
    IER=1; fprintf(OUT,"\n%d %s",IER,ERS); return(IER); }
  sig_imag = calloc(CHS,sizeof(int));
  if(!sig_imag)
  { strcpy(ERS,"Memory allocation error - sig_imag - PHASE_SEARCH");
    IER=2; free(sig_real); fprintf(OUT,"\n%d %s",IER,ERS); return(IER); }
  for(i=0;i<CHS;i++) { sig_real[i]=DAT_REAL[i]; sig_imag[i]=DAT_IMAG[i]; }

  /* calculo dos pares x,y a serem fitados */

  x[0]=0.0; y[0]=0.0; x[n/2]=-90.0; y[n/2]=0.0;
  for(i=0;i<CHS;i++) y[0] =y[0] +(double)(FFT_REAL[i]);
  for(i=0;i<CHS;i++) y[n/2]=y[n/2]+(double)(FFT_IMAG[i]);
  for(j=1;j<n/2;j++)
  { ang=22.5*(float)j; PHASE_SHIFT(sig_real,sig_imag,ang); FFT();
    x[j]=ang; y[j]=0.0; x[j+n/2]=ang-90.0; y[j+(n/2)]=0.0;
    for(i=0;i<CHS;i++) y[j] =y[j] +(double)(FFT_REAL[i]);
    for(i=0;i<CHS;i++) y[j+(n/2)]=y[j+(n/2)]+(double)(FFT_IMAG[i]); }
  fprintf(OUT,"\n\nFitting:");
  for(j=0;j<n;j++) fprintf(OUT,"\nx[%d] = %f  y[%d] = %f",j,x[j],j,y[j]);

  /* parametros iniciais */

  p[0]=fabs(y[0]); for(i=1;i<n;i++) if(fabs(y[i])>p[0]) p[0]=fabs(y[i]);
  p[1]=-180; ang=-180; old=0.0;
  for(i=0;i<n;i++) old=old+fabs(y[i]-p[0]*cos((p[1]+x[i])*PI/180.0));
  for(p[1]=-160.0;p[1]<=180.0;p[1]=p[1]+20.0)

```

```

{ new=0.0;
  for(i=0;i<n;i++) new=new+fabs(y[i]-p[0]*cos((p[1]+x[i])*PI/180.0));
  if(new<old) { ang=p[1]; old=new; } }
p[1]=ang;
fprintf(OUT, "\nPar. iniciais: p[0]= %f  p[1]= %f", p[0], p[1]);

/* fit routine for 2 parameters and n points */

for(;;)
{ cc=0.0;
  for(ni=1;ni<=10;ni++)
  { for(i=0;i<n;i++)
    { d[i*2]=cos((p[1]+x[i])*PI/180.0);
      d[i*2+1]=-p[0]*sin((p[1]+x[i])*PI/180.0); }
    for(i=0;i<2;i++)
    { for(j=i;j<2;j++)
      { g[i*2+j] = 0.0;
        for(k=0;k<n;k++) { g[i*2+j]=g[i*2+j]+d[k*2+i]*d[k*2+j]; } } }
    det=g[0]*g[3]-g[1]*g[1];
    gi[0]=g[3]/det; gi[1]=-g[1]/det; gi[2]=gi[1]; gi[3]=g[0]/det;
    for(i=0;i<2;i++)
    { c[i] = 0.0;
      for(k=0;k<2;k++)
      { for(j=0;j<n;j++)
        { c[i]=c[i]+gi[i*2+k]*d[j*2+k]
          *(y[j]-p[0]*cos((p[1]+x[j])*PI/180.0)); } } }
      p[0]=p[0]+c[0]; p[1]=p[1]+100.0*c[1]; cc=cc+fabs(c[1]); }
    if(cc<0.01) break; }
  fprintf(OUT, "\nPar. finais : p[0]= %f  p[1]= %f", p[0], p[1]);
  for(i=0;i<CHS;i++) { DAT_REAL[i]=sig_real[i]; DAT_IMAG[i]=sig_imag[i]; }
  *phase=-p[1]; return(IER);
}

/* ***** */

void PHASE_SHIFT(int *parte_real, int *parte_imag, float ang)
{
  double co, si, preal, pimag;
  int i;

  co=cos((double)(ang*PI/180.0)); si=sin((double)(ang*PI/180.0));
  for(i=0;i<CHS;i++)
  { preal=co*((double)parte_real[i]) - si*((double)parte_imag[i]);
    pimag=si*((double)parte_real[i]) + co*((double)parte_imag[i]);
    DAT_REAL[i]=(int)preal; DAT_IMAG[i]=(int)pimag; }
  return;
}

/* ***** */

/* ***** EXEC.C ***** */

#include "psp_dcl.h"

/* ***** */

int COMPILER(void)
{
  int i, j, n, flag, j1[5], j2[5], jj1[5], jj2[5];
  unsigned int jj;

```

```

unsigned long int t;
char *cc,str[80];

GETSETTINGS(); setfillstyle(SOLID_FILL,WHITE); settextstyle(0,0,1);
IER=TIME_UPDATE(); if(IER>0) { PUTSETTINGS(); return(IER); }
IER=LOAD_BUF(); if(IER>0) { PUTSETTINGS(); return(IER); }

/* determina tamanho dos buffers, atualiza tempos e confere a sequencia */

j=0; while(GET_EV(j)>0) { j=j+1; if(j==M) break; }
NE=j+2; fprintf(OUT,"\n\nNumber of events    : %u",NE);
if(NE==0)
{ strcpy(str,"Error: Sequence is empty (press any key to return)");
  WINDOW(1); setcolor(RED); outtextxy(16,4,str); getch();
  bar(16,4,637,12); IER=FREE_BUF(); PUTSETTINGS(); return(IER); }
for(j=NE-2;j<M;j++)
{ if(GET_EV(j)>0)
  { sprintf(str,"Error: Event %d is not defined (press any key)",NE-2);
    WINDOW(1); setcolor(RED); outtextxy(16,4,str); getch();
    bar(16,4,637,12); IER=FREE_BUF(); PUTSETTINGS(); return(IER); }
  for(i=0;i<N;i++)
  { if(GET_PULSE(i,j)==1)
    { strcpy(str,"Error: Pulse at non-defined event (press any key)");
      WINDOW(1); setcolor(RED); outtextxy(16,4,str); getch();
      bar(16,4,637,12); IER=FREE_BUF(); PUTSETTINGS(); return(IER); } } }
flag=0;
for(i=0;i<N;i++)
{ if(strcmp(&NAME[11*i],"*-----")==0)
  { flag=i+1;
    for(j=0;j<NE-2;j++)
    { if(GET_PULSE(i,j)==1)
      { strcpy(str,"Error: Pulse at non-defined line (press any key)");
        WINDOW(1); setcolor(RED); outtextxy(16,4,str); getch();
        bar(16,4,637,12); IER=FREE_BUF(); PUTSETTINGS(); return(IER); } } }
    else if(flag!=0)
    { sprintf(str,"Error: Line %d is not defined (press any key)",flag);
      WINDOW(1); setcolor(RED); outtextxy(16,4,str); getch();
      bar(16,4,637,12); IER=FREE_BUF(); PUTSETTINGS(); return(IER); } }
flag=0;
for(j=0;j<NE-2;j++)
{ if(GET_EV(j)==2)
  { TBL_EVAL(&SPEC[15*j],&TIME[j],TB,&n); if(flag==0) { NT=n; flag=1; }
    if(n!=NT)
    { strcpy(str,"Error: Tables do not match (press any key)");
      WINDOW(1); setcolor(RED); outtextxy(16,4,str); getch();
      bar(16,4,637,12); IER=FREE_BUF(); PUTSETTINGS(); return(IER); } }
    if(GET_EV(j)==3) { FCT_EVAL(&SPEC[15*j],&TIME[j]); } }
fprintf(OUT,"\nSize of table file : %u",NT);
fprintf(OUT,"\nTable index      : %d",TB+1);
setcolor(BLUE); WINDOW(3); bar(364,76,460,84);
sprintf(str,"%-d",NT); outtextxy(364,76,str);
if((flag!=0)&&(NT==0))
{ strcpy(str,"Error: Tables are empty (press any key)");
  WINDOW(1); setcolor(RED); outtextxy(16,4,str); getch();
  bar(16,4,637,12); IER=FREE_BUF(); PUTSETTINGS(); return(IER); }

/* ----- aloca o buffer KIT ----- */

KIT=malloc(16*NE+10);
if(!KIT) { IER=1; strcpy(ERS,"Memory allocation error - KIT - EXEC");

```

```

        fprintf(OUT, "\n%d %s", IER, ERS); IER=FREE_BUF();
        PUTSETTINGS(); return(IER); }
/* this buffer contain the whole pulse sequence to be send to PULSKIT */
/* - first 16*NE bytes : the sequence itself */
/* - next 10 bytes      : loop counters */

/* ----- inicializa os buffers ----- */

for(i=0; i<16*NE+10; i++) KIT[i]=0;
/* O primeiro evento e' sempre um delay de 1 segundo */
KIT[6]=0x80; KIT[7]=0x69; KIT[8]=0x67; KIT[9]=0xFF; KIT[13]=0xE0;
/* O ultimo evento mantem as linhas CP17, CP18 e CP19 ativas */
/* ( Repete um pulso de 1 ms atraves de um loop infinito tipo 2 ) */
n=16*(NE-1); KIT[n+6]=0x60; KIT[n+7]=0x79; KIT[n+8]=0xFE; KIT[n+9]=0xFF;
KIT[n+12]=(NE-1)&(0x0F); KIT[n+13]=0x20+((NE-1)>>8); KIT[n+15]=0xE0;

/* ----- compiler ----- */

fprintf(OUT, "\n 0 : software delay : 1.0000000 sg : ");
n=I; for(I=0; I<5; I++)
{ LOOP_LIMITS(&j1[I], &j2[I]); jj1[I]=-1; jj2[I]=-1; } I=n;
for(j=0; j<NE-2; j++)
{ jj=j+1; fprintf(OUT, "\n%4d : %14.14s : ", jj, &SPEC[15*j]);
  if(TIME[j]<5)
  { WINDOW(1); setcolor(RED);
    sprintf(str, "Error: Wrong time value at event %d (press any key)", j+1);
    outtextxy(16, 4, str); getch(); bar(16, 4, 637, 12); IER=FREE_BUF();
    free(KIT); KIT=NULL; PUTSETTINGS(); return(IER); }
  STRTIME(str, TIME[j]); fprintf(OUT, "%14.14s : ", str);
  t=TIME[j]; for(n=0; n<32; n++) t=t^(0x00000001<<n); t=t+1;
  KIT[16*jj+9]=(unsigned char)((t&(0xFF000000))>>24);
  KIT[16*jj+8]=(unsigned char)((t&(0x00FF0000))>>16);
  KIT[16*jj+7]=(unsigned char)((t&(0x0000FF00))>>8);
  KIT[16*jj+6]=(unsigned char)(t&(0x000000FF));
  for(n=0; n<5; n++) { if(j==j1[n]) jj1[n]=jj; if(j==j2[n]) jj2[n]=jj; }
  KIT[16*jj+13]=0xE0;
  if(GET_EV(j)==8) /* ADC + */
  { fprintf(OUT, " CP17"); /* set line CP17 (125) on */
    KIT[16*jj+(125/8)]=KIT[16*jj+(125/8)]+(0x01<<(125%8));
    fprintf(OUT, " ETRG"); /* set line ETRG (117) on */
    KIT[16*jj+(117/8)]=KIT[16*jj+(117/8)]+(0x01<<(117%8)); }
  if(GET_EV(j)==9) /* ADC - */
  { fprintf(OUT, " CP18"); /* set line CP18 (126) on */
    KIT[16*jj+(126/8)]=KIT[16*jj+(126/8)]+(0x01<<(126%8));
    fprintf(OUT, " ETRG"); /* set line ETRG (117) on */
    KIT[16*jj+(117/8)]=KIT[16*jj+(117/8)]+(0x01<<(117%8)); }
  for(i=0; i<N; i++)
  { if(strcmp(&NAME[11*i], "*-----*")==0) break;
    if(GET_PULSE(i, j)==1)
    { fprintf(OUT, " %4.4s", &LINE[5*i]); t=strtoul(&CODE[4*i], &cc, 10);
      KIT[16*jj+(t/8)]=KIT[16*jj+(t/8)]+(0x01<<(t%8)); } } }
fprintf(OUT,
"\n%4u : interrupt loop : 1.0000 ms : CP17 CP18 CP19", NE-1);
for(n=0; n<5; n++)
{ if((jj1[n]>0)&&(jj1[n]<NE-1)&&(jj2[n]>0)&&(jj2[n]<NE-1))
  { KIT[16*jj2[n]+14]=KIT[16*jj2[n]+14] + (0x01<<n);
    KIT[16*jj2[n]+13]=(KIT[16*jj2[n]+13]) & (0x0F);
    KIT[16*jj2[n]+13]=KIT[16*jj2[n]+13] + 0x30;
    KIT[16*jj2[n]+13]=KIT[16*jj2[n]+13]
      + (unsigned char) (((unsigned int)(jj1[n]))>>8) & (0x0007));
  }
}

```

```

    KIT[16*jj2[n]+12]=(unsigned char)((((unsigned int)(jj1[n]))&(0x00FF)));
    KIT[16*NE+2*n+1]=(unsigned char)(COUNT[n]>>8);
    KIT[16*NE+2*n]= (unsigned char)(COUNT[n]&(0x00FF)); }
else if(((jj1[n]>0)||((jj2[n]>0)))
{ WINDOW(1); setcolor(RED); sprintf(str,"Error: Loop %d is non valid",n);
  strcat(str," (press any key to return)"); outtextxy(16,4,str);
  getch(); bar(16,4,637,12); IER=FREE_BUF(); free(KIT); KIT=NULL;
  PUTSETTINGS(); return(IER); } }
PUTSETTINGS(); SAVE_KIT(); IER=FREE_KIT(); IER=FREE_BUF(); return(IER);
}

/* ***** */

void SAVE_KIT(void) /* save buffer KIT in file "psp.out" */
{
    /* (KIT must be loaded) */
    int n,k;

    if(KIT==NULL) return; fprintf(OUT,"\n      ");
    for(n=4;n>=0;n--) fprintf(OUT,"%02X%02X ",KIT[16*NE+2*n+1],KIT[16*NE+2*n]);
    for(n=0;n<16*NE;n=n+16)
    { fprintf(OUT,"\n%4d : ",n/16);
      for(k=15;k>=0;k=k-2)
      { fprintf(OUT,"%02X%02X ",(int) KIT[n+k],(int) KIT[n+k-1]); } }
    return;
}

/* ***** */

int EXEC(void)
{
    unsigned long int t;
    int k,j,n;
    char ch[2],file[13];

    if(ADC<0) return(IER); IER=SET_ADC(); if(IER>0) return(IER);
    IER=LDPPG(); if(IER>0) return(IER); IER=LOAD_KIT(); if(IER>0) return(IER);
    IER=LOAD_BUF(); if(IER>0) return(IER);
    GETSETTINGS(); WINDOW(2); IER=GETIMAGE(X1,Y1,X2,Y2);
    if(IER>0) { PUTSETTINGS(); IER=FREE_KIT(); IER=FREE_BUF(); return(IER); }
    MOLDURA(); MESSAGE(150,1); MESSAGE(151,1); MESSAGE(152,1); MESSAGE(153,1);
    MESSAGE(154,1); MESSAGE(155,1); MESSAGE(156,1); WINDOW(9); KURS(1);
    setfillstyle(SOLID_FILL,WHITE); settextstyle(0,0,1); setcolor(BLUE);
    ITR=-1; FTR=-1; WINDOW(3); bar(364,60,460,68);
    strcpy(file,"psp.dat"); outtextxy(364,60,file);
    for(;;)
    { BIOSKEY(ch); ITR_BREAK();
      if((M2==1)&&(MW==2))
      { MESSAGE(150,-1); if((M3>X1)&&(M3<X2)&&(M4>Y1)&&(M4<Y2)) ch[0]='r';
        MESSAGE(151,-1); if((M3>X1)&&(M3<X2)&&(M4>Y1)&&(M4<Y2)) ch[0]='a';
        MESSAGE(152,-1); if((M3>X1)&&(M3<X2)&&(M4>Y1)&&(M4<Y2)) ch[0]='s';
        MESSAGE(153,-1); if((M3>X1)&&(M3<X2)&&(M4>Y1)&&(M4<Y2)) ch[0]='i';
        MESSAGE(154,-1); if((M3>X1)&&(M3<X2)&&(M4>Y1)&&(M4<Y2)) ch[0]='d';
        MESSAGE(155,-1); if((M3>X1)&&(M3<X2)&&(M4>Y1)&&(M4<Y2)) ch[0]='t';
        MESSAGE(156,-1); if((M3>X1)&&(M3<X2)&&(M4>Y1)&&(M4<Y2)) ch[0]='f'; }
      if((M2==1)&&(MW==9))
      { WINDOW(9); KURS(0); J=(M3-X1)/16+JJ; KURS(1); }
      switch(ch[0])
      { case 'r':
          STOP_PULSKIT(); WINDOW(9); KURS(0); WINDOW(3);
          bar(364,60,460,84); IER=FREE_KIT(); IER=FREE_BUF(); WINDOW(2);
          /* return */

```

```

IER=PUTIMAGE(X1,Y1,X2,Y2); PUTSETTINGS(); return(IER);
case 'a': /* start */
    ITR=0; START_PULSKIT(); break;
case 's': /* stop */
    ITR=-1; STOP_PULSKIT(); break;
case 'f': /* data file */
    WINDOW(3); INPUT(364,60,file,13,13,0,1);
    outtextxy(364,60,file); break;
case 't': /* transfer */
    IER=TRANSFER(file);
    if(IER>0)
    { IER=FREE_KIT(); IER=FREE_BUF(); WINDOW(2);
      IER=PUTIMAGE(X1,Y1,X2,Y2); PUTSETTINGS(); return(IER); }
    WINDOW(3); bar(364,60,460,68); strcpy(file,"psp.dat");
    outtextxy(364,60,file); break;
case 'i': /* increment */
    n=GET_EV(J); if((n!=1)&&(n!=2)) break;
    if(n==1) { CHG=1; if(TIME[J]/10==0) TIME[J]=TIME[J]+1;
              else TIME[J]=TIME[J]+(TIME[J]/10); }
    if(n==2) { if(NT<=1) break; if(TB==NT-1) break; else TB=TB+1;
              TBL_EVAL(&SPEC[15*J],&TIME[J],TB,&n); }
    t=TIME[J]; for(k=0;k<32;k++) t=t^(0x00000001<<k); t=t+1;
    KIT[16*(J+1)+9]=(unsigned char)((t&(0xFF000000))>>24);
    KIT[16*(J+1)+8]=(unsigned char)((t&(0x00FF0000))>>16);
    KIT[16*(J+1)+7]=(unsigned char)((t&(0x0000FF00))>>8);
    KIT[16*(J+1)+6]=(unsigned char)(t&(0x000000FF));
    if(TAG[J]!='*')
    { for(j=0;j<NE-2;j++)
      { if(GET_EV(j)==3)
        { FCT_EVAL(&SPEC[15*j],&TIME[j]);
          t=TIME[j]; for(n=0;n<32;n++) t=t^(0x00000001<<n); t=t+1;
          KIT[16*(j+1)+9]=(unsigned char)((t&(0xFF000000))>>24);
          KIT[16*(j+1)+8]=(unsigned char)((t&(0x00FF0000))>>16);
          KIT[16*(j+1)+7]=(unsigned char)((t&(0x0000FF00))>>8);
          KIT[16*(j+1)+6]=(unsigned char)(t&(0x000000FF)); } } }
    STOP_PULSKIT(); delay(100); LDPPG(); delay(100);
    START_PULSKIT(); DATA_WINDOW(2); DATA_WINDOW(4); break;
case 'd': /* decrement */
    n=GET_EV(J); if((n!=1)&&(n!=2)) break;
    if(n==1) { CHG=1; if(TIME[J]/10==0) TIME[J]=TIME[J]-1;
              else TIME[J]=TIME[J]-(TIME[J]/10);
              if(TIME[J]<5) TIME[J]=5; }
    if(n==2) { if(NT<=1) break; if(TB==0) break; else TB=TB-1;
              TBL_EVAL(&SPEC[15*J],&TIME[J],TB,&n); }
    t=TIME[J]; for(k=0;k<32;k++) t=t^(0x00000001<<k); t=t+1;
    KIT[16*(J+1)+9]=(unsigned char)((t&(0xFF000000))>>24);
    KIT[16*(J+1)+8]=(unsigned char)((t&(0x00FF0000))>>16);
    KIT[16*(J+1)+7]=(unsigned char)((t&(0x0000FF00))>>8);
    KIT[16*(J+1)+6]=(unsigned char)(t&(0x000000FF));
    if(TAG[J]!='*')
    { for(j=0;j<NE-2;j++)
      { if(GET_EV(j)==3)
        { FCT_EVAL(&SPEC[15*j],&TIME[j]);
          t=TIME[j]; for(n=0;n<32;n++) t=t^(0x00000001<<n); t=t+1;
          KIT[16*(j+1)+9]=(unsigned char)((t&(0xFF000000))>>24);
          KIT[16*(j+1)+8]=(unsigned char)((t&(0x00FF0000))>>16);
          KIT[16*(j+1)+7]=(unsigned char)((t&(0x0000FF00))>>8);
          KIT[16*(j+1)+6]=(unsigned char)(t&(0x000000FF)); } } }
    STOP_PULSKIT(); delay(100); LDPPG(); delay(100);
    START_PULSKIT(); DATA_WINDOW(2); DATA_WINDOW(4); break;

```



```

    }
    switch(ch[1])
    { case RIGHT:
        if(J==NE-3) break;
        if(J<JJ+NC-1) { WINDOW(9); KURS0R(0); J=J+1; KURS0R(1); }
        else { PULSE_UPDATE(1,0,0,0); JJ=JJ+1; J=J+1; }
        DATA_WINDOW(2); DATA_WINDOW(4); break;
      case LEFT:
        if(J==0) break;
        if(J>JJ) { WINDOW(9); KURS0R(0); J=J-1; KURS0R(1); }
        else { PULSE_UPDATE(0,1,0,0); JJ=JJ-1; J=J-1; }
        DATA_WINDOW(2); DATA_WINDOW(4); break;
    }
  }
}

/* ***** */

int LDPPG(void) /* LOAD PULSKIT memory and set it to EXECUTE mode. */
{
  int n,i,flag;

  flag=0; if(KIT==NULL) { flag=1; IER=LOAD_KIT(); if(IER>0) return(IER); }
  /* Programa as portas (8255): porta A,B => saida; porta C (LSB) => saida */
  outportb(CTRL_PORT0,0x88);
  /* INPP */ PUTOUT(0xFF,0xFF,0xFF); /* initialize pulse programmer */
  /* LDMD */ PUTOUT(0x90,0x00,0xF9); /* set LOAD mode */
  /* INLC : disable automatic sequencing */
  PUTOUT(0x83,0xE8,0xF9);
  /* counter 1 */
  PUTOUT(0x83,0x01,0xF9); PUTOUT(0x82,0x21,0xF9);
  PUTOUT(0x82,0xA1,0xF9); PUTOUT(0x83,0x09,0xF9);
  PUTOUT(0x82,KIT[16*NE],0xF9); PUTOUT(0x82,KIT[16*NE+1],0xF9);
  /* counter 2 */
  PUTOUT(0x83,0x02,0xF9); PUTOUT(0x82,0x21,0xF9);
  PUTOUT(0x82,0xA2,0xF9); PUTOUT(0x83,0x0A,0xF9);
  PUTOUT(0x82,KIT[16*NE+2],0xF9); PUTOUT(0x82,KIT[16*NE+3],0xF9);
  /* counter 3 */
  PUTOUT(0x83,0x03,0xF9); PUTOUT(0x82,0x21,0xF9);
  PUTOUT(0x82,0xA3,0xF9); PUTOUT(0x83,0x0B,0xF9);
  PUTOUT(0x82,KIT[16*NE+4],0xF9); PUTOUT(0x82,KIT[16*NE+5],0xF9);
  /* counter 4 */
  PUTOUT(0x83,0x04,0xF9); PUTOUT(0x82,0x21,0xF9);
  PUTOUT(0x82,0xA4,0xF9); PUTOUT(0x83,0x0C,0xF9);
  PUTOUT(0x82,KIT[16*NE+6],0xF9); PUTOUT(0x82,KIT[16*NE+7],0xF9);
  /* counter 5 */
  PUTOUT(0x83,0x05,0xF9); PUTOUT(0x82,0x21,0xF9);
  PUTOUT(0x82,0xA5,0xF9); PUTOUT(0x83,0x0D,0xF9);
  PUTOUT(0x82,KIT[16*NE+8],0xF9); PUTOUT(0x82,KIT[16*NE+9],0xF9);
  /* load all counters */
  PUTOUT(0x83,0x5F,0xF9);
  /* LMEM */ for(n=0;n<16*NE;n=n+16)
    { for(i=0;i<16;i=i+2) { PUTOUT(KIT[n+i+1],KIT[n+i],0xFD); } }
  /* EXMD */ PUTOUT(0xA0,0x00,0xF9); /* set EXECUTE mode */
  if(flag==1) IER=FREE_KIT(); return(IER);
}

/* ***** */

void PUTOUT(msb,lsb,func)

```

```

unsigned char msb,lsb,func;
{
    outportb(DADOS_A,lsb); outportb(DADOS_B,msb); outportb(DADOS_C,func);
    outport(CTRL_PORT0,0x70); /* reset bit 0 da porta C */
    outport(CTRL_PORT0,0x71); /* set    bit 0 da porta C */
}

/* ***** */

void TRIG_RESET(void)
{
    outport(CTRL_PORT0,0x76); /* reset bit 3 da porta C */
    outport(CTRL_PORT0,0x77); /* set    bit 3 da porta C */
}

/* ***** */

void START_PULSKIT(void)
{
    PUTOUT(0xBF,0x7F,0xF9); return;
}

/* ***** */

void STOP_PULSKIT(void)
{
    PUTOUT(0xCF,0xDF,0xF9); return;
}

/* ***** */

int LOAD_KIT(void)
{
    int kit;
    unsigned int size;

    if(KIT!=NULL) return(1ER);
    if((kit=open("c:\\psp\\temp\\psp.kit",
O_RDONLY | O_BINARY, S_IWRITE | S_IREAD))== -1)
    { strcpy(ERS,"File 'c:\\psp\\temp\\psp.kit' cannot be open (LOAD_KIT)");
      1ER=1; fprintf(OUT,"\\n%d %s",1ER,ERS); return(1ER); }
    size = 16*NE+10; KIT=malloc(size);
    if(!KIT) { strcpy(ERS,"Memory allocation error - KIT - (LOAD_KIT)"); 1ER=2;
              fprintf(OUT,"\\n%d %s",1ER,ERS); close(kit); return(1ER); }
    if((_read(kit,KIT,size))!=size)
    { strcpy(ERS,"Reading error - 'c:\\psp\\temp\\psp.kit' - (LOAD_KIT)");
      1ER=3; fprintf(OUT,"\\n%d %s",1ER,ERS);
      free(KIT); KIT=NULL; close(kit); return(1ER); }
    close(kit); return(1ER);
}

/* ***** */

int FREE_KIT(void)
{
    int kit;
    unsigned int size;

    if(KIT==NULL) return(1ER);
    if((kit=open("c:\\psp\\temp\\psp.kit",

```

```

O_CREAT | O_WRONLY | O_BINARY, S_IWRITE | S_IREAD))==-1)
{ strcpy(ERS,"Data file c:\\psp\\temp\\psp.kit cannot be open - FREE_KIT");
  IER=1; fprintf(OUT,"\\n%d %s",IER,ERS); return(IER); }
size = 16*NE+10;
if((_write(kit,KIT,size))!=size)
{ strcpy(ERS,"Writting error - 'c:\\psp\\temp\\psp.kit' - (FREE_KIT)");
  IER=2; fprintf(OUT,"\\n%d %s",IER,ERS); close(kit); return(IER); }
free(KIT); KIT=NULL; close(kit); return(IER);
}

/* ***** */

int CHK_ADC(void)
{
  if((ADC=ibfind("dev14"))<0)
    fprintf(OUT,"\\nWARNING: ADC not found, execution will be aborted !");
  fprintf(OUT,"ADC= %d",ADC);
  return(IER);
}

/* ***** */

int SET_ADC(void)
{
  int n;
  char size[10];

  ibwrt(ADC,":STORAGE HOLDLAST",17);
  for(n=100;n<200;n++) if(GET_DEF(n)==0) ibwrt(ADC,DEF,strlen(DEF));
  ibwrt(ADC,":WAVEFORM:FORMAT 16,BLH",23); /* Format DATA */
  ibwrt(ADC,":WAVEFORM:SOURCE CHANNEL1,TIME",30); /* select channel 1 */
  ibwrt(ADC,":WAVEFORM:LENGTH?",17); /* Ask for number of points */
  ibrd(ADC,size,10); CHS=atoi(size); /* Read number of points */
  ibwrt(ADC,":WAVEFORM:SOURCE CHANNEL2,TIME",30); /* select channel 2 */
  ibwrt(ADC,":WAVEFORM:LENGTH?",17); /* Ask for number of points */
  ibrd(ADC,size,10); /* Read number of points */
  if(atoi(size)!=CHS)
  { strcpy(ERS,"Number of pts in Nic channels 1,2 do not match - SET_ADC");
    IER=1; fprintf(OUT,"\\n%d %s",IER,ERS); return(IER); }
  ibwrt(ADC,":storage live",13); ibsre(ibfind("gpib0"),0);
  DATA_WINDOW(6); return(IER);
}

/* ***** */

void ITR_BREAK(void)
{
  int i;
  char str[80];

  switch (ITR)
  { case 1 : /* ADC+ */
    if(FTR<0) { TRIG_RESET(); break; } FTR=FTR+1;
    bar(364,92,460,100); sprintf(str,"%-d",FTR); outtextxy(364,92,str);
    ibwrt(ADC,":WAVEFORM:SOURCE CHANNEL1,TIME",30); /* Select channel 1 */
    sprintf(str,":WAVEFORM:DATA? 0,%d,1",CHS);
    ibwrt(ADC,str,strlen(str)); /* ask for DATA */
    ibrd(ADC,str,10); /* read header */
    ibrd(ADC,TRF,CHS*sizeof(int)); /* read DATA */
    ibrd(ADC,str,2); /* read Delimiter */

```

```

for(i=0;i<CHS;i++) DAT[i]=DAT[i]+(long int)(TRF[i]);
ibwrt(ADC,":WAVEFORM:SOURCE CHANNEL2,TIME",30); /* Select channel 2 */
sprintf(str,":WAVEFORM:DATA? 0,%d,1",CHS);
ibwrt(ADC,str,strlen(str)); /* ask for DATA */
ibrd(ADC,str,10); /* read header*/
ibrd(ADC,TRF,CHS*sizeof(int)); /* read DATA */
ibrd(ADC,str,2); /* read Delimiter*/
for(i=0;i<CHS;i++) DAT[CHS+i]=DAT[CHS+i]+(long int)(TRF[i]);
TRIG_RESET(); break;
case 2 : /* ADC- */
if(FTR<0) { TRIG_RESET(); break; } FTR=FTR+1;
bar(364,92,460,100); sprintf(str,"%-d",FTR); outtextxy(364,92,str);
ibwrt(ADC,":WAVEFORM:SOURCE CHANNEL1,TIME",30); /* Select channel 1 */
sprintf(str,":WAVEFORM:DATA? 0,%d,1",CHS);
ibwrt(ADC,str,strlen(str)); /* ask for DATA */
ibrd(ADC,str,10); /* read header*/
ibrd(ADC,TRF,CHS*sizeof(int)); /* read DATA */
ibrd(ADC,str,2); /* read Delimiter*/
for(i=0;i<CHS;i++) DAT[i]=DAT[i]-(long int)(TRF[i]);
ibwrt(ADC,":WAVEFORM:SOURCE CHANNEL2,TIME",30); /* Select channel 2 */
sprintf(str,":WAVEFORM:DATA? 0,%d,1",CHS);
ibwrt(ADC,str,strlen(str)); /* ask for DATA */
ibrd(ADC,str,10); /* read header*/
ibrd(ADC,TRF,CHS*sizeof(int)); /* read DATA */
ibrd(ADC,str,2); /* read Delimiter*/
for(i=0;i<CHS;i++) DAT[CHS+i]=DAT[CHS+i]-(long int)(TRF[i]);
TRIG_RESET(); break;
case 7 : /* End of sequence */
STOP_PULSKIT(); ITR=-1; delay(500); break;
}
return;
}

/* ***** */

int TRANSFER(file)
char *file;
{
int i,j,jj,n,tb,nt,kitflag,buflag;
char ch[2],str[80];
FILE *arq;
unsigned long int t;

TRF=calloc(CHS,sizeof(int)); /* allocate tranfer buffer */
if(!TRF) { strcpy(ERS,"Memory allocation error - TRF - (TRANSFER)");
IER=1; fprintf(OUT,"\n%d %s",IER,ERS); return(IER); }
DAT=calloc(2*CHS,sizeof(long int)); /* allocate cummulation buffer */
if(!DAT) { strcpy(ERS,"Memory allocation error - DAT - (TRANSFER)");
free(TRF); IER=2; fprintf(OUT,"\n%d %s",IER,ERS); return(IER); }
IER=GET_DEF(5); if(IER>0) { free(TRF); free(DAT); return(IER); }
sprintf(str,"%s%s%s",DEF,"\\",file);
if((arq=fopen(str,"w"))==NULL)
{ sprintf(ERS,"Data file %s cannot be open (TRANSFER)",str); free(TRF);
free(DAT); IER=3; fprintf(OUT,"\n%d %s",IER,ERS); return(IER); }
if(KIT==NULL) { kitflag=0; IER=LOAD_KIT();
if(IER>0) { free(TRF); free(DAT); return(IER); } }
else kitflag=1;
if(BUF==NULL)
{ buflag=0; IER=LOAD_BUF(); if(IER>0)
{ fclose(arq); free(TRF); free(DAT);

```

```

    if(kitflag==0) FREE_KIT(); return(1ER); } }
else buflag=1;
GETSETTINGS(); WINDOW(2); IER=GETIMAGE(X1,Y1,X2,Y2); if(1ER>0)
{ free(TRF); free(DAT); fclose(arq); if(kitflag==0) FREE_KIT();
  if(buflag==0) FREE_BUF(); PUTSETTINGS(); return(1ER); }
WINDOW(56); IER=GETIMAGE(X1,Y1,X2,Y2); if(1ER>0)
{ free(TRF); free(DAT); fclose(arq); if(kitflag==0) FREE_KIT();
  if(buflag==0) FREE_BUF(); WINDOW(2); PUTIMAGE(X1,Y1,X2,Y2);
  PUTSETTINGS(); return(1ER); }
clearviewport(); IER=FFT_ON(); if(1ER>0)
{ free(TRF); free(DAT); fclose(arq); if(kitflag==0) FREE_KIT();
  if(buflag==0) FREE_BUF(); WINDOW(2); PUTIMAGE(X1,Y1,X2,Y2);
  WINDOW(56); PUTIMAGE(X1,Y1,X2,Y2); PUTSETTINGS(); return(1ER); }
WINDOW(57); MOLDURA(); WINDOW(58); MOLDURA();
WINDOW(59); MOLDURA();
WINDOW(60); MOLDURA();
WINDOW(2); MOLDURA(); MESSAGE(160,1);
DATA_WINDOW(6); tb=TB; if(NT>0) nt=NT; else nt=1;
settextstyle(0,0,1); setcolor(BLUE); setfillstyle(SOLID_FILL,WHITE);
sprintf(str,"%-d",nt); bar(364,76,460,84); outtextxy(364,76,str);
for(i=0;i<nt;i++)
{ for(j=0;j<2*CHS;j++) DAT[j]=0;
  WINDOW(3); setcolor(BLUE); bar(364,44,460,52); sprintf(str,"%-d",i+1);
  outtextxy(364,44,str); TB=i; IER=TIME_UPDATE(); if(1ER>0) goto end;
  for(j=0;j<NE-2;j++)
  { jj=j+1;
    if(TIME[jj]<5)
    { WINDOW(1); setcolor(RED); sprintf(str,
      "Error: Wrong time value at event %d (press any key)",j+1);
      outtextxy(16,4,str); getch(); bar(16,4,637,12); goto end; }
    STRTIME(str,TIME[jj]);
    t=TIME[jj]; for(n=0;n<32;n++) t=t^(0x00000001<<n); t=t+1;
    KIT[16*jj+9]=(unsigned char)((t&(0xFF000000))>>24);
    KIT[16*jj+8]=(unsigned char)((t&(0x00FF0000))>>16);
    KIT[16*jj+7]=(unsigned char)((t&(0x0000FF00))>>8);
    KIT[16*jj+6]=(unsigned char)(t&(0x000000FF)); }
  IER=LDPPG(); if(1ER>0) goto end; FTR=0; ITR=0; START_PULSKIT();
  for(;;)
  { BIOSKEY(ch); ITR_BREAK(); if(ITR<0) break;
    if((M2==1)&&(Mw==2))
    { MESSAGE(160,-1); if((M3>X1)&&(M3<X2)&&(M4>Y1)&&(M4<Y2)) ch[0]='s'; }
    if(ch[0]=='s') { STOP_PULSKIT(); ITR=-1; delay(500); goto end; } }
  WINDOW(3); bar(364,92,460,100);
  for(j=0;j<CHS-1;j++)
  { DAT_REAL[j]=(int)(DAT[j+1]/FTR); DAT_IMAG[j]=(int)(DAT[CHS+j+1]/FTR); }
  DAT_REAL[CHS-1]=DAT_REAL[CHS-2]; DAT_IMAG[CHS-1]=DAT_IMAG[CHS-2];
  PLOT_DATA();
  FFT(); if(i==0) { GET_PHASE(&PHASE); DATA_WINDOW(6); }
  PHASE_SHIFT(DAT_REAL,DAT_IMAG,PHASE); PLOT_DATA(); FFT(); PLOT_FFT();
  if(i>0) fprintf(arq,"\n"); fprintf(arq,"%d %d",i+1,CHS);
  for(j=0;j<CHS;j++)
  fprintf(arq,"\n%4d %8d %8d %8d %8d",
    j,DAT_REAL[j],DAT_IMAG[j],FFT_REAL[j],FFT_IMAG[j]); }
end:
FFT_OFF(); fclose(arq); ibsre(ibfind("gpib0"),0); FTR=-1; TB=tb;
WINDOW(3); bar(364,44,460,52); bar(380,92,476,100); free(TRF); free(DAT);
if(kitflag==0) FREE_KIT(); if(buflag==0) FREE_BUF();
WINDOW(56); IER=PUTIMAGE(X1,Y1,X2,Y2);
WINDOW(2); IER=PUTIMAGE(X1,Y1,X2,Y2); PUTSETTINGS(); return(1ER);}
/* ***** */

```

Bibliografia

- [1] Panepucci, H.C., Donoso, J.P., Tannus, A., Becmann, N., Bonagamba, T.J., "Tomografia por Ressonância Magnética Nuclear : Novas Imagens do Corpo", Ciencia Hoje, v.4, n.20, p.46, 1985.

- [2] Bonagamba, T.J., Tannus, A., Panepucci, H.C., "A técnica de Ressonância Magnética Nuclear e sua utilidade na formação de imagens", Anais do 2º Congresso Brasileiro de Físicos em Medicina, p.1, 1987.

- [3] Fukushima, E., Roeder, S.B.W., "Experimental Pulse NMR : a nuts and bolts approach", USA, Addison-Wesley, 1981.

- [4] Morris, P.G., "Nuclear Magnetic Resonance Imaging in Medicine and Biology", New York, Oxford University Press, 1986.

- [5] W. Koops, "MR Compendium", NETHERLANDS, Philips Medical Systems Division, 1985.

- [6] "NMRkit User's Manual", USA, Tecmag Inc.

- [7] "LN-2 Low Noise Preamplifier Installation and Operations Manual", USA, Doty Scientific Inc.

- [8] Sartori, J.C., Joaquim, M.B., "Princípios de Comunicação (Modulação angular)", São Carlos, USP-EESC-Dep. Eletricidade, 1981.

- [9] Gomes, A.T., "Telecomunicações : transmissão e recepção AM-FM", São Paulo, Érica, 1985.

[10] "8640B Signal Generator Operating and Service Manual", USA, Hewlett-Packard Company, 1978.

[11] Idoeta, I.V., Capuano, F.G., "Elementos de Eletrônica Digital", São Paulo, Érica, 1984.

[12] Taub, H., Schilling, D., "Eletrônica Digital", São Paulo, Mc Graw-Hill, 1982.

[13] "Matec R.F. Gated Amplifier Plug-In Model 515A Operation and Service Manual", USA, Matec Instruments Inc.

[14] Hayt, W.H., Kemmerly, J.E., "Análise de Circuitos em Engenharia", São Paulo, Mc Graw-Hill, 1975.

[15] Edminister, J.A., "Circuitos Elétricos", São Paulo, Mc Graw-Hill, 1985.

[16] Millman, J., "Eletrônica Dispositivos e Circuitos", São Paulo, Mc Graw-Hill, 1981.

[17] Malvino, A.P., "Eletrônica", São Paulo, Mc Graw-Hill, 1986.

[18] "PULSkit User's Manual", USA, Tecmag Inc.

[19] Osborn, A., "Introdução aos microprocessadores", São Paulo, Mc Graw-Hill, 1983.

[20] Norton, P., "Desvendando o PC", Rio de Janeiro, Campus, 1989.

[21] Rosch, W.L., "Desvendando o Hardware do PC", Rio de Janeiro, Campus, 1990.

[22] Hancock, L., Krieger, M., "Manual de linguagem C", Rio de Janeiro, Campus, 1986.

[23] Schildt, H., "Turbo C-Guia do Usuário", São Paulo, Mc Graw-Hill, 1988.

[24] "TTL Data Book", USA, Texas Instruments.

[25] "INTEL Component Data Catalog", USA, INTEL Instruments, 1981.

[26] Visconti, A.C.J.F., "Microprocessadores 8080 e 8085", São Paulo, Érica, 1981.

[27] "System 400 Benchtop Waveform Acquisition System Operation Manual", USA, Nicolet Instrument Corporation, 1990.

[28] "Remote Command Sets Using the ANSI/IEEE-488.2 Format GPIB or RS232 Manual", USA, Nicolet Instruments Corporation, 1990.

[29] "NI-488 MS-DOS Software Reference Manual", USA, National Instruments, 1990.