PUBLICAÇÕES

UNIVERSIDADE DE SÃO PAULO

INSTITUTO DE FÍSICA
CAIXA POSTAL 20516
01498 - SÃO PAULO - SP
BRASIL

IFUSP/P-794

SISTEMA DE AQUISIÇÃO DE DADOS DO TBR-I

A.N. Fagundes
Laboratório de Física de Plasmas
Instituto de Física, Universidade de São Paulo

Julho/1989

Sistema de aquisição de dados do TBR-I

A.N.Fagundes, Laboratório de Fisica de Plasmas
Instituto de Física da Universidade de São Paulo

Julho, 1989

Apresentação.

O sistema de aquisição de dados do Tokamak TBR-I tem como finalidade prover meios para a obtenção de registros recuperáveis de sinais para posterior análise, originários de sistemas de diagnósticos. No extremo inicial da cadeia estão os elementos geradores dos sinais, de construção dedicada e uso específico, como as bobinas de Rogowski, para medida da corrente elétrica; as sondas eletrostáticas, para determinação de temperatura e densidade de elétrons e íons; medidores de raios-X, para determinação de disrupturas internas; bobinas magnéticas para medida de instabilidades, etc. Os sinais transientes são, como regra geral, condicionados por amplificadores e injetados em sistemas rápidos de conversão análogo-digital, transportados a um computador e finalmente armazenados em disquetes. Em especial, este trabalho prende-se aos estagios finais do processo, de registro, armazenamento e transferência dos sinais obtidos. Divide-se em duas partes. Na primeira, está descrito, de forma sucinta, o hardware disponível no laboratorio para a conversão análogo-digital dos sinais e seu armazenamento; na segunda, está descrito o software desenvolvido para sua operação, junto de identificação dos cabeçalhos e detalhes dos arquivos produzidos. Cópias dos programas estão incluidos no apêndice.

I. O Hardware.

1. **Introdução.** Optou-se pelo sistema CAMAC (Norma IEEE-583) já no início dos traba-

lhos de construção do TBR-I, em meados de 1978, para o sistema de aquisição de dados. Dispomos hoje de dois bastidores(ou "crates"); controladores que apresentam interfaciamento para computador no padrão GPIB (norma IEEE-488); oito módulos digitalizadores de transientes com características sucintamente descritas abaixo, que permitem a observação de, no máximo -- respeitados os limites de cada aparelho! --, 77 sinais transientes simultâneos; um módulo gerador de base de tempo; um módulo de teste, que dá a conhecer o trânsito de sinais nas linhas CAMAC e um módulo extensor de conectores, para manutenção "on line" dos módulos.

2. Os digitalizadores.

i. Modelo Le Croy WD2264. Apresenta as seguintes características principais:

- 1, 2, 4 ou 8 canais simultâneos de análise;
- freqüência máxima de amostragem: 4 MHz (para 1 canal ativo) a 500 kHz (para quatro ou oito canais ativos);
- Resolução de oito bits (1 parte em 256);
- Programação externa,
- Memoria para armazenamento de até 4 x 32 kwords, igualmente divididas pelo numero de canais ativos. (Normalmente, apenas um módulo de memória é atribuido a cada digitalizador, daí resultando 32 kwords/canais ativos para cada registro de sinal; por exemplo, utilizando o módulo com 8 canais ativos, cada canal terá extensão de 4096 pontos.)
- Impedância de entrada de 50 ohms;
- Sinais de entrada com amplitude máxima de 0.512 V sem polaridade, com polaridade positiva ou negativa.

Contamos com cinco modulos.

ii. Modelo Le Croy WA8210:

- 1, 2, ou 4 canais simultâneos de análise
- freqüência maxima de amostragem de 1 MHz;
- resolução de 10 bits (1 parte em 1024);
- programação externa;
- impedância de entrada de 1 Mohm
- sinais de entrada com amplitude 10 V;
- memória de armazenamento como no modulo WD2264.

Contamos com 1 módulo.

iii. Modelo Le Croy DL8210A:

- 1, 2, 4, 8, 16 ou 32 canais simultâneos de análise;
- freqüência máxima de amostragem de 40 kHz;
- resolução de 12 bits (1 parte em 4096);
- programação interna;
- impedância de entrada de 1 Mohm;
- sinais de entrada com amplitude 10 V
- memoria de armazenamento como nos módulos WD2264.

Contamos com 1 modulo.

iv. Modelo Le Croy TR8837F:

- Um único canal de análise;
- freqüência máxima de amostragem de 32 MHz;
- resolução de 8 bits (1 parte em 256);
- programação interna;
- impedância de entrada de 50 ohms;
- sinais de entrada com amplitude 0.512, sem polaridade;
- memoria de armazenamento de 8 kwords.

Contamos com 1 módulo.

3.O **gerador de base de tempo.** Modelo Le Croy 8501, cujas características principais são:

- freqüência de pulsos entre 20 Hz e 20 MHz;
- programação interna;
- 4 modos de operação:

- **-por mudança tabelada de freqüência**, quando o módulo emite pulsos pelos seis conectores de saida com freqüência f_1 até a chegada de um pulso de controle na entrada 1; emite um número programado n_2 de pulsos com freqüência f_2 e finalmente um número programado n_3 de pulsos com freqüência f_3, quando emite um pulso de parada (stop);
- **-por mudança de freqüência com pulsos de controle**, quando o módulo emite pulsos com freqüência f_1 até o recebimento de um pulso de controle na entrada 1; passa emitir pulsos com freqüência f_2 até o recebimento de um pulso de controle na entrada 2; e finalmente emite pulsos com freqüência f_3 até o recebimento de um pulso na entrada 3, quando emite um pulso de parada;
- **-pulsos defasados**, quando emite pulsos de mesma freqüência pelos seis conectores de saida, mas a saida de cada par casado (isto é, saidas 1 e 2, 3 e 4, 5 e 6) emite pulsos que estão meio período fora de fase;
- **-Trem de pulsos**, quando , disparado por um pulso na entrada 2, emite um número programado n_2 de pulsos idênticos de freqüência f_2 por todos conectores de saída.

4. **A unidade de teste.** Modelo BiRa 6102 . Permite testar as linhas CAMAC, dando a conhecer as informações enviadas pelas linhas CAMAC a uma estação específica de um bastidor através de uma estrutura de diodos luminosos. Também permite monitorar a atividade nestas linhas. Não tem uso direto na tarefa de aquisição de dados.

5.O **Controlador.** Modelo Le Croy 8901A. Sua principal característica é a de apresentar

uma porta GPIB, incorporando portanto as características deste padrão de interfaciamento. Tem capacidade de operaçoes de transferência em bloco, quer controlada por número de posições de memoria pré-especificado, quer pelo processo em que a linha de barramento CAMAC "Q" é testada, terminando a transferência quando Q = 0; pode ser programado para a transferência automática de 1, 2 ou 3 bytes de cada vez. Apresenta características favoráveis à operação de módulos digitalizadores CAMAC Le Croy, com a possibilidade de uso de duas velocidades de transferência, compatíveis com as restrições de tempo impostas por condições interna de uso dos digitalizadores. O módulo controlador é de uso transparente ao usuário, isto é, não se faz notar durante o uso do equipamento.

6.O **Computador.** No outro extremo da conexão CAMAC está o computador, que escolhemos compatível com a familia IBM-PC de microcomputadores. Dedicado exclusivamente à tarefa de controle e gerenciamento do sistema de aquisição de dados, este computador é um dos disponíveis no laboratório, normalmente um computador XT, fabricação Microtec, modelo 2002, equipado com placa aceleradora, com disco Winchester de 10 Mbytes; ou um modelo AT, fabricação NOVADATA, com memoria RAM de 1,152 kBytes, divididos 640 kbytes para o sistema operacional e o restante utilizado com disco virtual; disco Winchester com 20 MBytes.(Este arranjo é o que apresenta melhor desempenho). A placa de interfaciamento GPIB é fabricada por STD-Sistemas Técnicos Digitais S.A., modelo STD-8410. Esta placa, (compatível com o modelo PC2 National Instruments -- geralmente citada como padrão de comparação para softwares) tem características técnicas favoraveis ao processo de controle do sistema, tais como a operação em acesso direto à memoria (DMA), permitindo velocidades de transferência de dados da ordem centenas de kbytes por segundo, e a possibilidade gerar "Interrupt Requests" com ação prevista por "software". (Esta facilidade, que permite, por exemplo, proceder a transferências automatizadas de dados, pela interpretação do sinal CAMAC "Look-At-Me"

gerado na controladora, ao fim da aquisição de dados, ainda não está implementado.)

O sistema é completado por uma placa de vídeo modelo ATI "VGAWonder" com resolução gráfica de 1024 x 768 pontos, no modo de mais alta resolução, com compatibilidade para o padrão VGA, EGA, CGA e monitor SONY "Multisync II" condizente com as exigência técnicas da placa. (O sistema é utilizado presentemente no modo VGA, com resolução de 640 pontos na horizontal e 480 pontos na vertical.)

II. O software.

1. Introdução. O software de operação do sistema de aquisição de dados foi totalmente desenvolvido em nossos laboratorio, utilizando-se a linguagem C de programação, com o compilador Microsoft versão 5.1 e Compilador Turbo-C versão 2.0. As informações necessárias foram obtidas dos manuais dos módulos, suporte de software de operação da placa STD-8410, fornecido pelo fabricante, e o pacote de software descrito em (1) para entrada e saída de dados.

A estratégia adotada na preparação dos programas consistiu em que fazer com que o produto final -- o registro dos dados em disquete -- deve conter o máximo de elementos que permitam sua recuperação mais fácil, bem como sua identificação e caracterização. Isto deve ser feito sem perguntas repetitivas ao usuário e de forma rotineira, contribuindo para que, durante o processo junto ao Tokamac, as atenções estejam primariamente voltadas à operação da máquina, dos sistemas de diagnóstico e do ambiente; enfim, da física envolvida no processo de investigação científica.

O sistema de transferência é composto de quatro programas:

i. PlanaÇao (chamada: pc)

ii. ArmaCam (chamada: ac)

iii. ToCamac (chamada: tc)

1. Al Stevens, "QuickC - Memory Resident Utilities, Screen, I/O and programming Techniques", Mis Press, 1988.

iv. VeCurva (chamada: vc)

2. Descrição resumida. Grosso modo, o programa **pc** constroi um plano de trabalho que representa a atividade global de aquisição de dados pretendida. O programa **ac** interpreta este plano, armando os diversos módulos de programação interna com as necessárias informações. O programa **tc** obedece à seqüência descrita neste plano para proceder à tarefa de transferência. Finalmente, o programa **vc** irá mostrar na forma gráfica os arquivos escolhidos. Assim, o programa **pc** é utilizado uma unica vez, enquanto os programas **ac** e **tc** são empregados na oportunidade de cada transferência, para inicializar o sistema e posteriormente transferir os arquivos digitalizados dos módulos para a memoria de disco do computador.

3. Descrição detalhada dos programas.

i. O programa PlanoAcao (pc). Foi escrito de forma a facilitar ao máximo a preparação do plano de trabalho, pelo uso extensivo de menus, janelas de auxílio ao usuário e mensagens de erro quando pertinentes. Para que haja auxílio, o arquivo MODULO.HLP deve ser acessível, quer através do comando pelo comando do sistema operacional DOS "APPEND", quer se localizando no mesmo diretório do qual **pc** é invocado. Qualquer tecla, de F2 a F10, termina a janela; F1 é a tecla de "Help" e a tecla "ESC" reinicializa a janela. Ao ser invocado, abre-se uma janela indagando o nome a ser dado ao arquivo de saída; se nada é informado, o nome "default", isto é, o nome assumido pelo programa, é "C:\DADOS\PLANO.TRB". Abre-se a seguir uma janela mostrando um menu com os módulos de aquisição de dados de que dispomos. A primeira linha é a do gerador de base de tempo. Se escolhido, então indaga-se o modo de operação: se 0, o módulo não será utilizado na cadeia; se 1, 2, 3 ou 4, abrem-se janelas específicas indagando as características de operação do módulo. Por exemplo, no modo 1, será indagado o bastidor e a estação onde o módulo de base de tempo se localiza; depois, a 1ª freqüência de operação; a 2ª

freqüência de operação, o número de pulsos para esta freqüência; a 3ª freqüência de operação e o número de pulsos antes que um pulso de parada seja ativado pelas linhas de "stop". O programa retorna então ao menu principal.

A escolha sobre um módulo digitalizador programável internamente faz com que sejam indagados o bastidor e sua estação, o número de canais ativos, isto é, o número de canais que paticipará da tarefa de digitalização; os canais de interêsse, ou seja, aqueles efetivamente utilizados para guardar sinais que devem ser posteriormente transferidos e guardados no disco fixo. Há a possibilidade de particularizar as informações para cada um dos canais de interêsse -- o que, sugere-se fortemente, seja sempre feito! Em caso positivo abre-se uma janela para cada canal quando indaga-se a grandeza física medida, sua abreviação, suas unidades; o fator numérico de conversão "overall", ou seja, o fator númerico que converte o sinal gravado nas unidades da grandeza medida desconhecendo-se todos os elementos intervenientes no processo; e uma linha de informações sobre aquele canal. Estas informações serão transferidas ao arquivo de saída sem outras alterações contribuindo para uma interpretação posterior mais fácil, e possivelmente automatizada, dos arquivos.

A escolha de um modulo digitalizador programável externamente se reduz à identificação do bastidor e sua estação, a escolha dos canais de interêsse, e a particularização de cada canal. É tarefa do programador garantir que não haja conflito entre a informação prestada e a efetiva disposição das chaves sobre o módulo. Em caso de conflito, é a disposição das chaves que será obedecida no momento da transferência.

A escolha de "Fim" termina o programa, fazendo com que seja gerado o arquivo de plano de trabalho no diretorio presente. No apêndice 1 está um exemplo de plano de trabalho.

ii. O programa **ArmaCam** (**ac**). É empregado para armar o conjunto de módulos para receber os sinais. Sua ação consiste em primeiro estabelecer o protocolo de comunicação GPIB, para então enviar um sinal CAMAC "CLEAR and INITIALIZE" para todos os bastidores; a seguir, um sinal de "STOP TRIGGER" para todas as estações de todos os

bastidores -- haja ou não módulo ali. Desta forma, quer o sistema já esteja em uso, quer tenha sido apenas ligado, ele é levado a condições iniciais bem determinadas (processo de inicialização). A seguir, abre e interpreta o arquivo contendo o plano de trabalho (caso nenhum nome seja informado é o arquivo "C:\DADOS\PLANO.TRB" que será carregado), transferindo as informações dos módulos programaveis internamente a cada módulo respectivo, terminando por armá-lo. Importante: É tarefa do programador certificar-se de que os módulos informados no plano de trabalho estejam efetivamente em seus lugares. O programa retorna para novo processo até que o arquivo de plano de trabalho seja exaurido. As informações adicionais contantes no plano de trabalho, começadas por "#" são ignoradas pelo programa. Seu uso admite um parâmetro (-r) que, se informado, faz com que seja produzido um relatório dos módulos que compõem a cadeia de aquisição de dados.

iii. O programa **ToCamac** (**tc**). É o responsável pela efetiva transferência dos dados armazenados nos módulos digitalizadores para a memória do computador. Está provido de alguns avisos e alertas para certas condições inadequadas de operação. A seqüência de transferência é aquela indicada no arquivo de plano de trabalho. O módulo é acessado e interrogado internamente sobre sua configuração e, a seguir, faz-se a transferência para a memória do computador, canal a canal. Um nome contendo informações mnemônicas sobre o registro, é atribuido ao arquivo, especificamente

<mes><num.pulso><crate><estacao><canal>.CHR.

Por exemplo, o arquivo citado no apêndice 2 tem nome U0251J04.CHR, in-dicando que foi tomado no mes de Junho, com numero 025, de um módulo situado no crate 1, estacao 10, canal 4. (O plano de trabalho informa que este canal registrava o sinal da bobina poloidal de Mirnov 12.) O número do arquivo é obtido do arquivo "C:\DADOS\CONTADOR.NUM", ao qual é somado um e novamente gravado.

Estrutura do arquivo produzido por tc

Exatamente 240 bytes iniciais estão reservados para informações sobre o pulso. O

apêndice 2 representa o cabeçalho de um arquivo típico, com a interpretação de cada número presente. São anexadas informações sobre o pulso, informações sobre a base de tempo (de utilidade se o módulo gerador de base de tempo CG8501 comanda a aquisição nos modos de 1 a 4) e finamente as informações adicionais que permitem caracterizar o sinal ali guardado. Seguem-se os sinais, guardados no formato inteiro, (2 bytes para cada número), tais como são lidos do digitalizador. O registro de um canal com 4096 pontos, portanto, tem extensão de

$$4096 \times 2 + 240 = 8432 \text{ bytes}$$

O arquivos de sinais se compõem de números num intervalo que depende da resolução do módulo que o gerou. Assim modulos de 8 bits produzem números entre 0 e 255; de 10 bits, entre 0 e 1023; e 12 bits, entre 0 e 4095. Dois numeros consecutivos do registro correspondem a amostragens separadas no tempo de um período da base de tempo. Nenhum processo de correção é aplicado ao arquivo quanto ao modo de geração ("Offset binary", "Complementary binary") , e sua interpretação quanto à fase e ao sinal devem ser feita conforme indicado no manual do módulo correspondente. (O processo de identificação de fase dos sinais é geralmente auto-evidente na maioria das aplicações comuns, não exigindo procedimentos especiais.) Durante o funcionamento, o programa tc cria um arquivo , mostrado no apêndice 3, cujo radical do nome é o mesmo do pulso sendo transferido, mas com terminação GRF, contendo a data presente, o número do pulso e os nomes dos arquivos informados no plano de trabalho para retorno gráfico. Este arquivo será interpretado pelo programa **vc** para produzir a tela gráfica que termina o processo de transferência.

iv. O programa **VeCurva** (vc). Projetado para auxiliar no diagnóstico rápido do pulso, pela visualização de sinais escolhidos, o programa **vc** busca por um arquivo com terminação GRF localizado na saida default (em d:, se a técnica de disco virtual está sendo empregada; ou no diretório C:\DADOS\). Abre os arquivos ali relacionados e automaticamente divide o espaço de tela pelos graficos solicitados, num máximo de 15.

Cada gráfico traz informações contidas no cabeçalho, tais como a grandeza medida e unidades. Há um menu na linha inferior da tela solicitando uma ação específica quanto aos registros como um todo. Assim, se a opção SALVA é escolhido os arquivos serão transferidos (ou mantidos) no diretório C:\DADOS\; se a opção ABANDONA é escolhida, **todos** os registros relativos àquele disparo serão apagados (e não apenas os mostrados em gráficos). IMPRIME leva à impressão da tela gráfica (Atenção: os comandos são compatíveis com impressoras gráficas do tipo EPSON). A opção EXAMINA não está implementada e representa uma porta para desenvolvimentos futuros do programa. FIM termina a execução sem qualquer ação sobre os arquivos. Este programa foi escrito prevendo-se sua expansão, devendo permitir de forma relativamente fácil a inclusão de ações específicas sobre arquivos escolhidos, com uso particular a uma determinada experiência.

4. **Exemplo de recuperação.** O Programa **lc.c**, escrito em "C", mostra como podemos recuperar um arquivo gerado por tc. Especificamente, lc lê o arquivo e imprime os cem primeiros números; ao se pressionar uma tecla qualquer, mais cem números são lidos, até que, ao se pressionar "q", o programa termina.

```
/*
        Programa para decodificar o arquivo de dados produzido
        por tc. Assume-se que o arquivo está guardado no
        diretório "C:\DADOS".
*/

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <conio.h>


void main(int argc, char**argv);


void main(int argc, char *argv[])
{
FILE *flux;
int i, c, balde[100];
```

```
static char nomearq   [24], grandeza_fisica[13],
        abreviacao[ 3], unidades      [ 8],
        comentario[34];
float fator_multiplicativo;

static struct param                    /* Estrutura para os parametros */
 {
 char data [25];
 char polar[4];
 char tipo_modulo[8];
 unsigned int canais_interesse;
 unsigned int amostragens;
 unsigned int num_pontos;
 unsigned int okay;
 unsigned int canal;
 unsigned int resol;
 unsigned long int shot;
 float tempo;
 } cab;


struct complex_time /* Estrutura para a base de tempo */
 {
 unsigned int modo;
 unsigned int range;
 unsigned long int numpul_2;
 unsigned long int numpul_3;
 float per_1;
 float per_2;
 float per_3;
 } ajuste, *aj = &ajuste;

        if (argc < 2)
                {
                printf ("\n Uso do programa: lc <nome>\n");
                exit(0);
                }
        strcat( nomearq, argv[1]);
        strcat( nomearq, ".CHR"); /* Coloca terminacao CHR */

        if( (flux = fopen(nomearq, "r")) == NULL)
                {
                printf ("Arquivo %s nao pode ser aberto!", nomearq);
                exit(0);
                }

        /* As informações do cabeçalho */

        fscanf( flux, "%s %U %u %u %u %u %s %s %u %u",
                 cab.data,          /* Quando ocorreu o evento  */
                &cab.shot,          /* Num do disparo           */
                &cab.canais_interesse, /* Canais a considerar     */
                &cab.amostragens,     /* posicao do pulso no int. */
```



```
                &cab.num_pontos,      /* Extensao do registro    */
                &cab.okay,            /* Pulso esta correto      */
                 cab.polar,           /* Polaridade              */
                 cab.tipo_modulo,     /* Modulo de origem        */
        &cab.resol,          /* Resolucao do modulo     */
                &cab.canal);          /* Cnl de origem no modulo */


fscanf( flux, "%d %d %f %f %f %U %U",
        &aj->modo,     /* Modo da base de tempo */
        &aj->range,    /* Parametro  do modulo 8501 */
        &aj->per_1,    /* Periodo da 1a. freq.. trabalho */
        &aj->per_2,    /*          2a.            */
        &aj->per_3,    /*          3a.            */
        &aj->numpul_2, /* pts de atividade 2a. freq.    */
        &aj->numpul_3); /*                3a. freq.    */

fscanf( flux, " %s %s %7c %f %s",
        grandeza_fisica, /* Grandeza fisica registrada: */
        abreviacao,      /*  - Nome pela qual a indicamos; */
        unidades,        /*  - suas unidades;            */
        &fator_multiplicativo, /*  - Fator num.conversao   */
        comentario);     /* Coment. relativo ao canal   */

/* Imprime o cabeçalho */

 printf( "\n\n arquivo: %s", nomearq);
 printf( "\n  Modulo: %s     data: %s", cab.tipo_modulo, cab.data);
 printf( "\n   Pulso: %lu    Amost.pos/trig:%u", cab.shot,
       cab.amostragens);
 printf( "\n Num.Pts: %u     Pol: %s", cab.num_pontos, cab.polar);
 printf( "\n   Resol: 1/%u   Can.Origem: %u", cab.resol, cab.canal);
 printf( "\n\n");
 printf( " \nBase de tempo: ");
 printf( "\n modo: %d  range: %d", aj->modo, aj->range);
 printf( "\n per_1: %f   per_2: %f   per_3: %f",
                        aj->per_1, aj->per_2, aj->per_3);
 printf( "\n     extensao 2: %ld   extensao 3: %ld",
                        aj->numpul_2, aj->numpul_3);


 printf( "\n Informacoes sobre o canal:");
 printf( "\n    - grandeza fisica: %s (%s)", grandeza_fisica,
       abreviacao);
 printf( "\n    - unidades: %s  fator multiplicativo: %f",
       unidades, fator_multiplicativo);
 printf( "\n    comentario: %s", comentario);

        /* Posiciona ponteiro para leitura de dados do arquivo */

 fseek(flux, 239L, SEEK_SET); /* Pula 240 bytes: 0 a 239 inclusive */

 while (1)
        {
```

```
        fread((char *) balde, 2, 100, flux);
        printf("\n");
        for( i = 0; i < 100; i++)
                printf(" %u ", balde[i]);

    if( (c = getch()) == 'q')
                exit(0);
        }

}
```


Apêndice 1

A estrutura do Plano de Trabalho

Abaixo está o plano de trabalho produzido por **pc**. Em cada linha, as informações após ";" são tratadas como comentários. Os módulos aqui relacionados são identificados por **K** - Clock Generator CG8501 e **D** - Waveform digitizer WD2264 -- outros códigos existem para os demais módulos. O número que se segue refere-se á estação onde se localiza o módulo. **C** identifica o crate; **Z** é uma palavra de comando que aparece para alguns módulos especiais; **I** representa o número de canais ativos para módulos com programação externa. As linhas começadas por **#**, quando não se referem ao módulo de base de tempo, representam informações que serão adicionadas ao cabeçalho do canal correspondente. Os números que se seguem constituem: o canal no módulo digitalizador; a informação se deve retornar na forma de gráfico (1 = sim; 0 = não); a grandeza física sendo medida; abreviação da grandeza física e suas unidades; o fator de conversão numérica "overall"; e comentários sobre a grandeza física contida naquela canal. Após uma informação de base de tempo, **#** representa a indicação de programação daquele módulo, indicando respectivamente: modo de operação; range; periodo em us das bases de tempo 1, 2 e 3; número de pulsos para a base de tempo 2 e 3. Estas linhas são ignoradas pelos vários programas, quando não aplicáveis. A última linha traz a data em que o plano de trabalho foi elaborado.

Este plano envolve 5 módulos CAMAC (1 base de tempo e 4 digitalizadores WD 2264), gerando a cada disparo 32 arquivos de dados, 10 curvas na tela gráfica e cerca de 128 kbytes de informação a serem guardados para posterior análise. (Foi preparado e utilizado por Celso Ribeiro, em Junho-Julho/89, dentro de seu programa de medidas.)

```
; Tudo é comentario após ponto-e-virgula.
;
K020 C02 Z0002 ;        Modulo da base de tempo(modo2, Fsr=500kHz)
#02 01  2.000000  2.000000  2.000000 00000000 00000000
;
D003 C01 I08 ; Sinais  das bobinas poloidais de Mirnov(1 a 8)
#00 1 B~ poloidal  B01(T) 1.2e-4  Sinal bob. poloidal de Mirnov 1
#01 0 B~ poloidal  B02(T) 1.17e-4  Sinal bob. poloidal de Mirnov 2
#02 0 B~ poloidal  B03(T) 1.22e-4  Sinal bob. poloidal de Mirnov 3
#03 0 B~ poloidal  B04(T) 1.19e-4  Sinal bob. poloidal de Mirnov 4
#04 0 B~ poloidal  B05(T) 1.14e-4  Sinal bob. poloidal de Mirnov 5
#05 0 B~ poloidal  B06(T) 1.22e-4  Sinal bob. poloidal de Mirnov 6
#06 0 B~ poloidal  B07(T) 1.2e-4  Sinal bob. poloidal de Mirnov 7
#07 0 B~ poloidal  B08(T) 1.2e-4  Sinal bob. poloidal de Mirnov 8
;
D010 C01 I08 ; Sinais das bobinas poloidais de Mirnov(9 a 16)
#00 1 B~ poloidal  B09(T) 1.2e-4  Sinal bob. poloidal de Mirnov 9
#01 0 B~ poloidal  B10(T) 1.16e-4 Sinal bob. poloidal de Mirnov 10
#02 0 B~ poloidal  B11(T) 1.18e-4 Sinal bob. poloidal de Mirnov 11
#03 0 B~ poloidal  B12(T) 1.18e-4 Sinal bob. poloidal de Mirnov 12
#04 0 B~ poloidal  B13(T) 1.18e-4 Sinal bob. poloidal de Mirnov 13
#05 0 B~ poloidal  B14(T) 1.19e-4 Sinal bob. poloidal de Mirnov 14
#06 0 B~ poloidal  B15(T) 1.18e-4 Sinal bob. poloidal de Mirnov 15
#07 0 B~ poloidal  B16(T) 1.12e-4 Sinal bob. poloidal de Mirnov 16
;
D015 C01 I08 ; Sin.bob.tor.Mir(2,3,4), sond.eletr. e pot.flut.
```



```
#00 0 B~ toroidal  Bt2(T) 1.62e-4  Sinal bob. toroidal de Mirnov 2
#01 0 B~ toroidal  Bt3(T) 2.97e-4  Sinal bob. toroidal de Mirnov 3
#02 0 B~ toroidal  Bt4(T) 2.93e-4  Sinal bob. toroidal de Mirnov 4
#03 1 Tensao sonda  Vs(V)  40.0    Tensao varredura sonda eletrost.
#04 1 Corr.sonda-R  Ir(A) 30.5e-3  Corrente da sonda medida sobre R
#05 1 Corr.sonda-T Itr(A)  5.0e-2  Corrente da sonda util. transfo.
#06 1 Pot.flu.plas  Vf(V) 20.0     Potencial flutuante de plasma
#07 1 Pot.f.plas-f  Vff(V) 20.0    Pot.flu.plas.c/filtro3.8kHz-3dB
;
D020 C01 I08 ; Sinais padrao:Ip,Vl,Ph,Pv,Rx,Rxi,Ih-m/n=4/1,Ibt
#00 1 Corr. plasma  Ip(kA)  2.87   Corrente de plasma
#01 0 Tens. enlace   Vl(V) 1.25   Tensao de enlace
#02 1 Pos.horizont  Ph(cmxkA) 19.82 Posicao horizontal coluna plasma
#03 0 Pos.vertical  Pv(cmxkA) 12.0  Posicao vertical coluna plasma
#04 0 Raios-x dur.  Rx(u.a.)   1.0  Raios-x de alta energia(duros)
#05 0 Rx dur. int. Rxi(u.a.)   1.0  Raios-x duros(sinal integrado)
#06 1 Corr. helic.  Ih(A)   667.0  Corrente nas espiras helicoidais
#07 0 Corr. toroi. Ibt(kA)     1.19 Corrente do campo magn. toroidal
;
@ - Fim. Arquivo gerado em 29-6-89
```

Apêndice 2

O Arquivo de dados

Este cabeçalho pertence ao arquivo U0251J04.CHR. (É produzido na subrotina TC_TRANS.C)

Thu Jun 29 17:51:53 1989 0000025 8 0004 04096 0 zer WD2264 256 04 02 01 2.000000e+000 2.000000e+000 2.000000e+000 00000000 00000000 B~ poloidal B12(Tesla) 1.180000e-004 Sinal bob. poloidal de Mirnov 12
Ô[illegible]

Os campos possuem o seguinte significado:

Thu Jun 29 17:51:53 1989 é a data em que o pulso foi obtido (atenção: os espaços de separação na frase correspondem ao caráter ASCII 255 e não ao caráter normalmente utilizado ASCII 32);

0000025 Corresponde ao número do pulso, tal como é contabilizado no arquivo C:\DADOS\CONTADOR.NUM;

8 O sinal teve origem de um módulo ajustado para a digitalização de 8 canais;

0004 Informação sobre a disposição da chave relativa aos "pontos pós-trigger";

04096 Corresponde à extensão do arquivo, em pontos;

0 Constitui a condição de erro, expressa pelo módulo WD2264. Se igual a 1, o pulso foi obtido em condições anormais;

zer Expressa a polaridade do sinal; zer para pontos bipolares, pos para sinais com polaridade positiva e neg para sinais com polaridade negativa (presentemente só se aplica ao sinais obtidos através do módulo digitalizador WD2264; nenhum outro módulo de que dispomos possui seleção de polaridade.);

WD2264 Modulo que deu origem ao sinal ;

256 é a resolução do módulo; corresponde a 2 ^(n.bits resolução)

04 O sinal foi digitalizado pelo canal 4 do módulo;

Informações do módulo da base de tempo:

02 01 Corresponde, respectivamente, ao modo de trabalho e ao "range" do módulo da base de tempo;

2.000000e+000 2.000000e+000 2.000000e+000 Intervalo entre pontos, em microsegundos, para as três freqüências; (frequentemen-te só se utiliza a primeira das freqüências, obtendo-se todo o registro com separação uniforme no tempo)

00000000 00000000 número de pulsos para as freqüências f_2 e f_3;

B~ poloidal Corresponde à grandeza sendo medida; (espacos em branco no interior da frase correspondem ao caráter ASCII 255)

B12(T) Abreviação desta grandeza e suas unidades;

1.180000e-004 Fator de conversão "overall", que traduz os sinais registrados na grandeza física medida;

Sinal bob. poloidal de Mirnov 12 Comentário sobre o registro. (Espaços em branco na frase correspondem ao caráter ASCII 255.)

Seguem-se os sinais relativos aos números digitalizados gravados que constituem o produto final do proceso de digitalização. (Que são interpretados pelo pelo computador como sinais ASCII, já que este arquivo está sendo reproduzido através de um editor de textos e não lido como números inteiros).

Apêndice 3

Conteúdo do arquivo de lista gráfica

O arquivo que instrui o programa vc na elaboração da tela gráfica tem a seguinte apresentação (corresponde ao arquivo U025.GRF):

```
25 Thu Jun 29 17:51:53 1989
U0251C01.CHR
U0251J01.CHR
U0251O04.CHR
U0251O05.CHR
U0251O06.CHR
U0251O07.CHR
U0251O08.CHR
U0251T01.CHR
U0251T03.CHR
U0251T07.CHR
```

A primeira linha contém o número do pulso e a data em que foi obtido; as linhas consecutivas indicam os arquivos que devem ser abertos e mostrados. Um máximo de 15 (para operação em modo VGA de 640 x 480 pontos) é admitido pelo programa.

Apêndice 4

O programa ac

```
/* Inicio módulo arma.h */
union PalavraComunicacao        /* Leitura e escrita no modulo 8901A */
 {
 unsigned    char cc [6];
 unsigned long int ll;
 unsigned     int ii;
 };

struct complex_time
 {
 unsigned int modo;
 unsigned int range;
 unsigned long int numpul_2;
 unsigned long int numpul_3;
 float par_1;
 float par_2;
 float par_3;
 };

struct disp
 {
 unsigned int amostragens;
 float tempo;
 unsigned int canais_interesse;
 unsigned int canais_ativos;
 unsigned int num_pontos;
 unsigned int flag;
 unsigned int okay;
 char tipo_modulo[7];
 };

#define  NUM_CRATES  2
typedef unsigned int unsint;


void _8212(int crate, int estacao, unsigned long palcom);
void _8501(int crate, int estacao, struct complex_time *aj_tempo);
void _8837(int crate, int estacao, unsigned long palcom);
void _2264(int crate, int estacao);
void _8210(int crate, int estacao);
void arma (int crate, int estacao, int comando);
void carmac(char *);
void executa (union PalavraComunicacao pt, int num_bytes, int crate);
void lei(FILE *);

/* Fim  módulo arma.h */

/* Inicio módulo lgpib.h */
/*
```

```
        Prototipo das funcoes C para o STD-8410
  Modelo LARGE – Nao esta documentado no manual !
          * acrescentamos L na frente das funcoes;
          * instrucao (far) torna-se desnecessaria.

(Informacoes obtidas junto ao representante da placa)
                fevereiro-1989
*/

void libcac (int, int);
void libclr (int, int);
void libcmd (int, unsigned char  *, int);
void libconf(int, int, int);
void libdma (int, int);
void libeos (int, int);
void libeot (int, int);
void libesc (int, int, unsigned char  *, unsigned int);
void libgts (int, int);
void libist (int, int);
void libler (int, int, char  *, int);
void libipe (int, int);
void libslp (int, int,  char);
void libonl (int, int);
void libpad (int, int);
void librd  (int, unsigned char  *, int);
void librpp (int, unsigned char);
void librsc (int, int);
void librsv (int, int);
void librtl (int);
void libsad (int, int);
void libsic (int);
void libsre (int, int);
void libtmo (int, int);
void libtrg (int, int);
void libwait(int, unsigned int);
void libwrt (int, unsigned char *, int);

  /* variavels declaradas publicas pelo cb.obj     */
        extern  int  ibsta;  /* palavra de status    */
        extern  int  iberr;  /* codigo de erro GPIB  */
        extern  int  ibcnt;  /* numero de bytes enviados */

  /* constantes usadas em programas de exemplos      */
  #define UNL  0x3f  /* GPIB comando unlisten        */
  #define UNT  0x5f  /* GPIB comando untalk          */
  #define GTL  0x01  /* GPIB comando go to local     */
  #define SDC  0x04  /* GPIB selected device clear   */
  #define PPC  0x05  /* GPIB parallel poll conf.     */
  #define GET  0x08  /* GPIB group execute trigger   */
  #define TCT  0x09  /* GPIB tome o controle         */
  #define LLO  0x11  /* GPIB local lock out          */
  #define DCL  0x14  /* GPIB device clear            */
  #define PPU  0x15  /* GPIB ppoll unconfigure       */
  #define SPE  0x18  /* GPIB habilitar poll serial   */
```



```
  #define SPD  0x19  /* GPIB desabilitar poll serial  */
  #define PPE  0x60  /* GPIB habilitar poll paralelo  */
  #define PPD  0x70  /* GPIB desabilitar poll paralelo*/
  #define S    0x08  /* especifica o sentido do PPR   */
  #define REOS 0x400
  #define XEOS 0x800
  #define BIN  0x1000
  #define LF   0x0a
  #define TIMO 0x10000
  #define SRQI 0x1000
  #define CIC  0x20
  #define TACS 0x08
  #define LACS 0x04

  /* variaveis do programa de aplicacao passadas  */
  /* para as funcoes STD-8410 .                   */
  int    bd;                  /* numero do cartao GPIB no compuutador   */
  int    cnt;          /* variavel cont bytes  */
  int    v;            /* uso geral            */
  int    conf;         /* config. de enderecos */
  int    disp;         /* dispositivo GPIB     */
  char   ppr;          /* byte de poll paralelo*/
  char   spbyte;       /* byte para resposta de*/
                       /* poll serial          */
  unsigned int   mask;  /* eventos esperados   */

/* Fim módulo lgpib.h */

/* Início módulo ac.c */
#include <string.h>
#include <stdio.h>
#include <process.h>
#include "arma.h"

void main(int argc, char *argv[]);
int getopt(int, char**, char *);
void camarc(char*);

extern int optind;
int relato = 0;

void main(int argc, char *argv[])
{
int ch;
static char plano_trab[] = "c:\\dados\\plano.trb"; /* arquivo default */

        if(argc > 1)
                {
                ch = getopt(argc, argv, "r");
                switch(ch)
                        {
                        case 'r' :          relato++;
                                break;
```

```
                case '?' : printf(" Uso do programa: ac [-r] [arquivo de trabalho]\n");
                        exit(0);
                }

        argc -= optind;
        argv += optind;

        if(argc)
                strcpy((plano_trab + 9), argv[0]);
        }

    camarc(plano_trab);
}

/* Fim módulo ac.c */

/* Inicio módulo ac_geral.c */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include "lgpib.h"
#include "arma.h"

extern int bd;
extern int crate, estacao, canint;
extern unsigned long int palcom;
extern char *tipo_modulo;
extern struct complex_time *aj_tempo;

void executa(union PalavraComunicacao pt, int num_bytes, int crate)
{
union PalavraComunicacao pp;

 libesc (bd, crate, pt.cc, num_bytes);

 pp.ll = 0x4020 | (crate << 8);
 libcmd (bd, pp.cc, 2);
}


void iel(FILE *flux)
{
char line[80], *linha, *p;

 crate = estacao = -1;
 while( crate == -1 || estacao == -1)
  {
  linha = fgets(line, 80, flux);

  if (linha != NULL)
   {
   p = strtok(linha, " ");
   while (p != NULL)
```



```
   {
   switch ( toupper(*p) )
    {
/* Informacoes gerais */
    /* Identificacao do crate */
            case 'C' : p++;
                    sscanf( p, "%d", &crate);
            break;

    /* Palavra contendo a programacao */
    case 'Z' : sscanf( ++p, "%X", &palcom);
            if(toupper(tipo_modulo[1]) == 'K')
             if(palcom)
              {
              fgets(line, 80, flux);
              sscanf(line+1,"%d %d %f %f %f %d %d",
                      &aj_tempo->modo,
                      &aj_tempo->range,
                      &aj_tempo->per_1,
                      &aj_tempo->per_2,
                      &aj_tempo->per_3,
                      &aj_tempo->numpul_2,
                      &aj_tempo->numpul_3);
              goto aa;
              }
            break;

    case 'I' : sscanf( ++p, "%d", &canint);
            break;


/* Modulos com programacao interna */
    /* Data Logger 8212a */
    case 'L' : tipo_modulo = "DL8212a";
            sscanf( ++p, "%d", &estacao);
            break;

    /* Transient Recorder tr8837f */
            case 'R' : tipo_modulo = "TR8837f";
            sscanf( ++p, "%d", &estacao);
            break;

    /* Clock Generator 8501 */
    case 'K' : tipo_modulo = "ck8501";
            sscanf( ++p, "%d", &estacao);
            break;

/* Modulos com programacao externa */
    /* Waveform Analyzer WA8210 */
    case 'A' : tipo_modulo = "WA8210";
            sscanf( ++p, "%d", &estacao);
            break;

    /* Waveform Analyzer WD2264 */
```

```
        case 'D'  : tipo_modulo = "WD2264";
                    sscanf( ++p, "%d", &estacao);
                    break;

                                        case '#'  :
        case ';'  : goto aa;

        case '@'  : exit(0);

        default   : break;
        }
      p = strtok(NULL, " ");
      }
    }
aa: ;
  }
}
/* Fim módulo ac_geral.c */

/* Inicio módulo getopt.c */
/*
                                rotina getopt(ion)
                                        Origem: Proficient C, A. Hansen, p. 127.
*/

#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#define EOF             (-1)

int opterr = 1;
int optind = 1;
int optopt;
char *optarg;

int getopt(int argc, char **argv, char *opts)
{
static int sp = 1;
register int c;
register char *cp;

        if(sp == 1)
                if(
                        (       optind >= argc) ||
                        (argv[optind][0] != '-' ) ||
                                (argv[optind][1] == '\0')
                        ) return EOF;
          else if (strcmp(argv[optind], "--") == 0)
                        {
                                        optind++;
                                        return EOF;
                                        }
        optopt = c = argv[optind][sp];
```



```
        if( c == ':' || (cp = strchr (opts, c)) == 0)
                {
                printf(": opcao ilegal -- %c\n", c);
                if (argv[optind][++sp] == '\0')
                        {
                        optind++;
                        sp = 1;
                        }
                return ('?');
                }

        if (*++cp == ':')
                {
                if (argv[optind][sp+1] != '\0')
                        optarg = &argv[optind++][sp+1];
                else if(++optind >= argc)
                        {
                        printf (": a opcao requer um argumento -- %c\n", c);
                        sp = 1;
                        return('?');
                        }
                else
                        optarg = argv[optind++]:
                sp = 1;
                }
        else
                {
                if(argv[optind][++sp] == '\0')
                        {
                        sp = 1;
                  optind++;
                        }
                optarg = 0;
                }
        return (c);
}

/* Fim módulo getopt.c */

/* Inicio módulo ac_2264.c */
#include <stdio.h>
#include "lgpib.h"
#include "arma.h"

extern int bd, relato,canint;

void _2264(int crate, int estacao)
{
struct disp ch, *chave = &ch;
union PalavraComunicacao pt, px;
int i, polaridade[8];

static unsigned int post_trigger_2264[8] =  { 8, 7, 6, 5, 4, 3, 2, 1};
static float samp_period_2264 [8] =
```

```
    { (float) 0.0, (float) 0.0, (float) -1.0, (float)25.0, (float)5.0,
     (float) 2.5, (float) 0.5, (float) 0.25};
static unsigned int channels_2264   [4] =   { 8, 4, 2, 1};
static char *polar[] = {"pos", "neg", "zer"};

/* Posicao das chaves no painel frontal do modulo */

 /* Configuracao */
 pt.cc[0] = 1;      /* Funcao Camac: F = 1 */
 pt.cc[1] = 0;      /* Comando Camac: A = 0 */
 pt.cc[2] = (char) estacao;

 libesc(bd, crate, pt.cc, 3);
 libler(bd, crate, px.cc, 3);

 chave->amostragens   = post_trigger_2264  [px.ll & 0x00000007L];
        px.ll > > = 3;
        chave->tempo        = samp_period_2264   [px.ll & 0x00000007L];
        px.ll > > = 3;
 chave->canais_ativos = channels_2264     [px.ll & 0x00000003L];
        px.ll > > = 2;
 chave->flag        = (unsigned int)(px.ll & 0x00000001L);
        px.ll > > = 1;
 chave->okay        = (unsigned int)(px.ll & 0x00000001L);
 chave->num_pontos    = (unsigned int) 32768/chave->canais_ativos;

/* Polaridade dos canais */

pt.cc[0] = 0;  /* Funcao Camac: F = 0 */
pt.cc[1] = 0;

libesc(bd, crate, pt.cc, 3);
libler(bd, crate, px.cc, 3);

for(i = 0; i < chave->canais_ativos; i+ +)
  {
  polaridade[i] = (int) (px.ll & 0x00000003L);
  px.ll > > = 2;
  }
        printf("\n");
printf("\n Modulo 2264, localizado no crate %d estacao %d", crate, estacao);
        printf("\n instruido para digitalizar %d canais", chave->canais_ativos);
        printf("\n      Canais de interesse: %d", canint);
        if(chave->tempo > (float) 0)
          printf("\n   Intervalo entre pontos: %f us", chave->tempo);
        else
                printf("\n   Intervalo entre pontos: base de tempo externa");
printf("\n      Pontos pos-trigger: %d",  chave->amostragens);
        printf("\n              Polaridade:");

        for (i = 0; i< canint; i+ +)
                printf( "\n           canal %i %s", (i+1), polar[polaridade[i]-1]);

        printf("\n");
```



```
}
/* Fim módulo ac_2264.c */

/* Início módulo ac_8210.c */
#include <stdio.h>
#include "Igpib.h"
#include "arma.h"

extern int bd, relato, canint;

void _8210(int crate, int estacao)
{

union PalavraComunicacao pt, px;
struct disp ch, *chave=&ch;

static unsigned int post_trigger_8210[8] = { 7, 6, 5, 4, 3, 2, 1, 0};
static float samp_period_8210 [8] =
  { (float) -1.0, (float) 100.0, (float) 40.0, (float) 20.0,
   (float) 10.0, (float)  4.0, (float)  2.0, (float)  1.0};
static unsigned int channels_8210   [4] = { 1, 2, 0, 4};


/* Posicao das chaves no painel frontal do modulo */

 /* Configuracao */
 pt.cc[0] = 1;  /* Funcao Camac: F = 1 */
 pt.cc[1] = 0;  /* Comando Camac: A = 0 */
        pt.cc[2] = (char) estacao;

 libesc(bd, crate, (char *)  pt.cc, 3);
 libler(bd, crate, (char *)  px.cc, 3);

 chave->amostragens   = post_trigger_8210  [px.ll & 0x00000007L];
                        px.ll > > = 3;
 chave->tempo        = samp_period_8210   [px.ll & 0x00000007L];
                        px.ll > > = 3;
 chave->canais_ativos = channels_8210     [px.ll & 0x00000003L];
                        px.ll > > = 2;
 chave->flag        = (unsigned int)(px.ll & 0x00000001L);
                        px.ll > > = 1;
 chave->okay = 0;
 chave->num_pontos    = (unsigned int) 16384/chave->canais_ativos;

 printf("\n Modulo 8210, localizado no crate %d estacao %d", crate, estacao);
        printf("\n instruido para digitalizar %d canais", chave->canais_ativos);
        printf("\n      Canais de interesse: %d", canint);
 if( chave->tempo > (float) 0)
                printf("\n   Intervalo entre pontos: %f us", chave->tempo);
        else
                printf("\n   Intervalo entre pontos: Base de tempo externa");
 printf("\n      Pontos pos-trigger: %d",  chave->amostragens);
        printf("\n");
```

```
        }
/* Fim módulo ac_8210.c  */

/* Inicio módulo ac_8212.c */
#include <stdio.h>
#include "lgpib.h"
#include "arma.h"

extern int bd, relato;
extern unsigned long int palcom;
extern char *tipo_modulo;


void _8212(int crate, int estacao, unsigned long int palcom) _
{
static char *interv[] =
 { "   ", "5000", "1000", " 500", " 200", " 100", "  50", "  25"};
static char *frequencia[] =
     { "Ext", " 0,2", "  1", "  2", "  5", " 10", " 20", " 40"};

static char *post_samp[] =
 { "1024", " 896", " 768", " 640", " 512", " 384", " 256", " 128"};
static char *can_perm[] =
 {" 4", " 8", "16", "32"};

int noc, clk, ptsl;
union PalavraComunicacao pv;

 pv.cc[0] = 0x11; /* 8212 : F(17) */
 pv.cc[1] = 0;   /*      A( 0)  */
 pv.cc[2] = (char) estacao;
 pv.cc[3] = (char) palcom;

 executa (pv, 4, crate);

 /* Verificacao dos parametros transmitidos ao modulo  */
 pv.cc[0] = 0x03;          /* 8212 : F(03) */
 libesc(bd, crate, (char  *) pv.cc, 3);
 libler(bd, crate, (char  *) pv.cc, 3);

 noc  = (int) (pv.ll & 0x00000003L);
      pv.ll  > > = 2;
 clk  = (int) (pv.ll & 0x00000007L);
      pv.ll > > = 3;
 ptsl = (int) (pv.ll & 0x00000007L);

 if (relato)
  {
  printf( "\n Modulo 8212a instruido para digitalizar %s canais",
       can_perm[noc]);
  printf( "\n   Intervalo entre pontos: %s micro-s (%s kHz)",
       interv[clk], frequencia[clk]);
```



```
  printf( "\n      Pontos pos-trigger: %s ",   post_samp[ptsl]);
  printf( "\n");
  }

}
/* Fim módulo 8212.c */

/* Inicio módulo ac_8501.c */
#include <stdio.h>
#include "lgpib.h"
#include "arma.h"

extern int bd, relato;

void _8837(int crate, int estacao, unsigned long palcom)
{
static char *interva[] =
 { " 31", "  62", " 125", " 250", " 500", "1000", "2000", "   "};
static char *frequencia[] =
 { " 32", "  16", "   8", "   4", "   2", "   1", " 0.5", " Ext"};
static char *post_samp[] =
 { "8", "7", "6", "5", "4", "3", "2", "1"};
static char *mem_perm[] =
 { "1", "2", "3", "4", "5", "6", "7", "8"};
int mem_size, ptsl, clk;

union PalavraComunicacao pv;

 pv.cc[0] = 0x10;                    /* 8837 : F(16) */
 pv.cc[1] = 0;                     /*      A( 0) */
 pv.cc[2] = (char) estacao;
 pv.cc[3] = (char) (palcom & 0x000000FF);
        palcom > > = 8;
 pv.cc[4] = (char) (palcom & 0x000000FF);

 executa(pv, 4, crate);

 /* Verificacao dos parametros transmitidos ao modulo  */
 pv.cc[0] = 0x00;          /* 8837f : F(00) */
 libesc(bd, crate, (char  *) pv.cc, 3);
 libler(bd, crate, (char  *) pv.cc, 3);

 ptsl     = (int) (pv.ll & 0x00000007L);
            pv.ll > > = 4;
 clk      = (int) (pv.ll & 0x00000007L);
            pv.ll > > = 4;
 mem_size = (int) (pv.ll & 0x00000007L);


 if(relato)
  {
  printf( "\n Modulo 8837f instruido para operar com %s k de memoria",
       mem_perm[mem_size]);
  printf( "\n   Intervalo entre pontos: %s nano-s (%s MHz)",
```

```
        interva[clk], frequencia[clk]);
    printf( "\n        Pontos pos-trigger: %s ",   post_samp[ptsl]);
    printf( "\n");
    }


}
/* Fim módulo ac_8837.c */
```


Apêndice 5

O programa pc

```
/* Início módulo pc_menu.h */

struct complex_time
        {
 unsigned int modo;
 unsigned int range;
 unsigned long int numpul_2;
 unsigned long int numpul_3;
 float per_1;
 float per_2;
 float per_3;
 };

void menu_modulos(void);
void detalha_canal(char *inform, int ncl);
char *troca_espaco (char *frase);
int mod_dl8212a(WINDOW *, unsigned long int *,  int *, char *);
int mod_np     (WINDOW *, char *,  int *, char *);
int mod_tr8837f(WINDOW *, unsigned long int *, char *);
int mod_cg8501 (WINDOW *, struct complex_time *, char *);

/* Fim módulo pc_menu.h */

/Início módulo keys.h */
/*-----------------  Keys.h  ------------------------------*/

 #define HT          9
 #define RUBOUT                          8
 #define BELL                    7
 #define ESC         27
 #define SHIFT_HT                143
 #define CTRL_T                          20
 #define CTRL_B                          2
 #define CTRL_D                          4
 #define ALT_D                   160

 #define F1          187
 #define F2          188
 #define F3          189
 #define F4          190
 #define F5          191
 #define F6          192
 #define F7          193
 #define F8          194
 #define F9          195
 #define F10         196

 #define HOME                    199
 #define UP          200
```

```
#define PGUP            201
#define BS      203
#define FWD      205
#define END      207
#define DN      208
#define PGDN            209
#define INS     210
#define DEL      211

#define CTRL_HOME       247
#define CTRL_BS                 243
#define CTRL_FWD        244
#define CTRL_END    245

/* Fim módulo keys.h */

/* Início módulo qwindows.h */
/*----------------------------qwindow.h

        Uncomment this for stacked windows
                rather than layered windows.

#define FASTWINDOWS

*/


/*-------------Window colors----------------------------*/
#define RED             4
#define GREEN           2
#define BLUE            1
#define WHITE           (RED+GREEN+BLUE)
#define YELLOW                  (RED+GREEN)
#define AQUA            (GREEN+BLUE)
#define MAGENTA                 (RED+BLUE)
#define BLACK           0
#define BRIGHT          8
#define DIM         0

#define BORDER                  0
#define TITLE           1
#define ACCENT                  2
#define NORMAL                  3
#define ALL             4

#define TRUE            1
#define FALSE           0
#define ERROR       -1
#define OK              0

/*------------    Window controller structures----------*/
typedef struct field
        {
        char *fmask;
```



```
        int  fprot;
        char *fbuff;
        int  ftype;
        int  frow;
        int  fcol;
        void (*fhelp)(char *);
        char *fhwin;
        int  fix;
        int  fly;
        int (*fvalid) (char *);
        struct  field *fnxt;
        struct  field *fprv;
        } FIELD;


typedef struct _wnd
        {
        int _wv;
        int _hd;
        char *_ws;
        char *_tl;
        int _wx;
        int _wy;
        int _ww;
        int _wh;
        int _wsp;
        int _sp;
        int _cr;
        int btype;
        int wcolor[4];
        int _pn;
        struct _wnd *_nx;
        struct _wnd *_pv;
        FIELD *_fh;
        FIELD *_ft;
        } WINDOW;

typedef struct w_menu
        {
        char *mname;
        char **mselcs;
        void (**func)(int, int);
        } MENU;

#define SAV     (wnd -> _ws)
#define WTITLE  (wnd -> _tl)
#define COL     (wnd -> _wx)
#define ROW     (wnd -> _wy)
#define WIDTH   (wnd -> _ww)
#define HEIGHT  (wnd -> _wh)
#define SCROLL  (wnd -> _wsp)
#define SELECT  (wnd -> _sp)
#define WCURS   (wnd -> _cr)
#define WBORDER (wnd -> wcolor[BORDER])
```

```
#define WTITLEC (wnd -> wcolor[TITLE])
#define WACCENT (wnd -> wcolor[ACCENT])
#define WNORMAL (wnd -> wcolor[NORMAL])
#define PNORMAL (wnd -> _pn)
#define BTYPE   (wnd -> btype)
#define NEXT    (wnd -> _nx)
#define PREV    (wnd -> _pv)
#define WCOLOR  (wnd -> wcolor)
#define VISIBLE (wnd -> _wv)
#define HIDDEN  (wnd -> _hd)
#define FHEAD   (wnd -> _fh)
#define FTAIL   (wnd -> _ft)

#define NW      (wcs[wnd -> btype].nw)
#define NE      (wcs[wnd -> btype].ne)
#define SE      (wcs[wnd -> btype].se)
#define SW      (wcs[wnd -> btype].sw)
#define SIDE    (wcs[wnd -> btype].side)
#define LINE    (wcs[wnd -> btype].line)

/* General Purpose functions and macros   */
void    clear_screen(void);
int     vmode(void);
unsigned video_address(void);
void    cursor(int, int);
void    curr_cursor(int *, int *);
int     cursor_type(void);
void    set_cursor_type(int);
int     get_char(void);
int     scroll_lock(void);
char far *getdta(void);
void    setdta(char far *);
unsigned peek( unsigned, unsigned);
char    peekb( unsigned, unsigned);
void    poke( unsigned, unsigned, unsigned);
void    vpoke( unsigned, unsigned, unsigned);
int     vpeek( unsigned, unsigned);
int     ht(char **);
int     wd(char **);

/* Window functions and macros */
WINDOW  *establish_window(int, int, int, int);
void    set_border    ( WINDOW *, int);
void    set_colors    ( WINDOW *, int, int, int, int);
void    set_intensity ( WINDOW *, int);
void    set_title     ( WINDOW *, char *);
void    display_window( WINDOW *);
void    clear_window  ( WINDOW *);
void    delete_window ( WINDOW *);
void    hide_window   ( WINDOW *);
void    wprintf       ( WINDOW *, char *, ...);
void    wputchar      ( WINDOW *, int);
void    close_all     (void);
void    wcursor       ( WINDOW *, int, int);
```



```
void    error_message ( char *);
void    clear_message ( void);
int     get_selection ( WINDOW *, int, char *);

#define reverse_video(wnd)          wnd -> wcolor[3] = wnd -> wcolor[2]
#define normal_video(wnd)           wnd -> wcolor[3] = wnd -> _pn
#define rmove_window(wnd, x, y)     repos_wnd( wnd,    x,    y, 0)
#define move_window(wnd, x, y)      repos_wnd( wnd, COL-x, ROW-y, 0)
#define forefront(wnd)              repos_wnd( wnd,    0,    0, 1)
#define rear_window(wnd)            repos_wnd( wnd,    0,    0, -1)

/* Internal to window processes */
void    scroll      ( WINDOW *, int);
void    repos_wnd   ( WINDOW *, int, int, int);
void    acline      ( WINDOW *, int);

#define accent(wnd)     acline(wnd, WACCENT)
#define deaccent(wnd)   acline(wnd, WNORMAL)
#define clr(bg, fg, in)     ( (fg) | (bg << 4) | (in) )
#define vad(x, y)           ( (y)*160 + (x)*2)

/* Editor function */
void    text_editor( WINDOW *, char *, unsigned);

/* Menu function   */

void menu_select (char *name, MENU *mn);

/* Help functions  */
void    load_help(char *);
void    set_help (char *, int, int);

/* Data Entry functions   */
void    init_template   (WINDOW *);
FIELD  *establish_field (WINDOW *, int, int, char *, char *, int);
void    clear_template  (WINDOW *);
void    field_tally     (WINDOW *);
int     data_entry      (WINDOW *);
void    wprompt         (WINDOW *, int, int, char *);
void    error_message   (char *);
void    clear_notice    (void);
void    field_window    (FIELD *, char *, int, int);

#define field_protect(f, s)     f -> fprot  = s
#define field_help(f, h)        f -> fhelp  = h
#define field_validate(f, v)    f -> fvalid = v

#ifdef MSC50

#define MF(sg, of)    ((void far *)(((unsigned long)(sg) << 16 | (unsigned)(of)))

#define poke(sg, of)  (*(( int far *)MF((sg),(of))) = (int)(wd))
#define peek(sg, of)  (*(( int far *)MF((sg),(of))))
#define peekb(sg, of) (*((char far *)MF((sg),(of))))
```

```
#endif

#ifdef NOCGA

#define vpoke(sg, of, ch)   poke(sg, of, ch)
#define vpeek(sg, of)       peek(sg, of)

#endif

/* TSR prototypes */
void   resinit   (void);
int    resident (char *);
void   terminate (void);
void   restart   (void);
void   suspend   (void);
int    get_char  (void);
void   popup     (void);

/* Fim módulo qwindow.h */

/* Início módulo pc_mod.c */
/*
        Menu dos modulos disponivels

*/

#include "qwindow.h"

void main(void);
void menu_modulos(void);

void main()
{
        load_help("modulos.hlp");
        menu_modulos();
}
/* Fim módulo pc_mod.c */

/* Início módulo pc_menu.c */
/*
 Contrucao de menus para selecao de parametros
 de operacao dos modulos de aquisicao de dados.
   1-jan-89.
    * Alteracoes para escolha do nome do arquivo
      em 2-marco-89.

    * Inclusao do modulo Clock Programavel
      em 15-3-89.
*/

#include <malloc.h>
#include <stdio.h>
#include <string.h>
```



```
#include <stdlib.h>
#include <stdlib.h>
#include <conio.h>
#include <dos.h>
#include "qwindow.h"
#include "keys.h"
#include "pc_menu.h"


char *mod_disp [] =
 {
 "8501  - (Le Croy) programmable clocK ",
 "2264  - (Le Croy) waveform Digitizer ",
 "8210  - (Le Croy) waveform Analizer  ",
 "8212A - (Le Croy) fast data Logger   ",
 "8837F - (Le Croy) transient Recorder ",
 "8901A - (Le Croy) gpib Interface     ",
 "      - Fim                          ",
 NULL
 };

void nome_arq_trab( char *);

int  crate, estacao;
struct complex_time aj, *aj_tempo=&aj;
struct dosdate_t hoje;

void menu_modulos(void)
{
WINDOW *mn_mod, *wnda;
FILE *flux;
char **cp;
unsigned long int palcom;
int xold, yold, detail, alt, larg, nci, j, escolha=1, jj=0, s=0;
static char nomearq[25], coment[50], arq[7000], informa[2700];

 curr_cursor( &xold, &yold);
 nome_arq_trab( nomearq);

 alt  = ht(mod_disp);
 larg = wd(mod_disp);
 mn_mod = establish_window(20, 8, alt-1, larg+2);
 set_help("progmod ", 40, 10);
 set_title (mn_mod, " Modulos Camac Disponiveis ");
 set_colors(mn_mod,    ALL,  BLUE, WHITE, BRIGHT);
 set_colors(mn_mod, ACCENT, WHITE, BLACK,    DIM);
 display_window (mn_mod);

 cp = mod_disp;
 while (*cp)
   wprintf( mn_mod, "\n%s", *cp++);

 while (escolha)
   {
```

```
s = get_selection(mn_mod, s+1, "KDALRiF");
switch (s)
 {
 case 1 : wnda = establish_window(10, 5, 3, 30);
        detail = mod_cg8501(wnda, aj_tempo, coment);
        j = sprintf(arq + jj, ";\nK%03d C%02d Z%04x ; %50s\n",
                estacao, crate, aj_tempo->modo, coment);
        jj += j;
                                        j = sprintf(arq+jj,
                                                "#%02d %02d %10.6f %10.6f
%10.6f %08ld %08ld\n",
                                                aj_tempo->modo,
                                                aj_tempo->range,
                                                aj_tempo->per_1,
                                                aj_tempo->per_2,
                                                aj_tempo->per_3,
                                                aj_tempo->numpul_2,
                                                aj_tempo->numpul_3);
                                        jj += j;
        delete_window(wnda);
        break;

 case 2 : wnda = establish_window(10, 8, 8, 50);
        detail = mod_np(wnda,"D", &nci, coment);
        if (crate != 0 )
         {
         j = sprintf(arq + jj, ";\nD%03d C%02d I%02d ; %s\n",
                estacao, crate, nci, coment);
         jj += j;
         if(detail)
          {
          detalha_canal(informa, nci);
          j = sprintf(arq + jj, informa);
          jj += j;
          }
         }
        delete_window(wnda);
        break;

 case 3 : wnda = establish_window(10, 8, 8, 50);
        detail = mod_np(wnda,"A", &nci, coment);
        if (crate != 0 )
         {
         j = sprintf(arq + jj, ";\nA%03d C%02d I%02d ; %s\n",
                estacao, crate, nci, coment);
         jj += j;
         if(detail)
          {
          detalha_canal(informa, nci);
          j = sprintf(arq + jj, informa);
          jj += j;
          }
         }
        delete_window(wnda);
```



```
        break;

 case 4 : wnda = establish_window(10, 5, 15, 50);
        palcom = 0;
        detail = mod_dl8212a(wnda, &palcom, &nci, coment);
        if (crate != 0 )
         {
         j = sprintf(arq + jj,";\nL%03d C%02d I%02d Z%08lx ; %s\n",
                estacao, crate, nci, palcom, coment);
         jj += j;
         if(detail)
          {
          detalha_canal(informa, nci);
          j = sprintf(arq + jj, informa);
          jj += j;
          }
         }
        delete_window(wnda);
        break;

 case 5 : wnda = establish_window(10, 5, 15, 50);
        palcom = 0;
        detail = mod_tr8837f(wnda, &palcom, coment);
        if (crate != 0 )
         {
         j = sprintf(arq + jj, ";\nR%03d C%02d Z%08lx ; %s\n",
                estacao, crate, palcom, coment);
         jj += j;
         if(detail)
          {
          detalha_canal(informa, 1);
          j = sprintf(arq + jj, informa);
          jj += j;
          }
         }
        delete_window(wnda);
        break;

 case 6 : break;

 case 7 : escolha = 0;
        break;
 }
}
if(jj != 0)
 {
        _dos_getdate (&hoje);
                sprintf(arq + jj, ";\n@ - Fim. Arquivo gerado em %d-%d-%d",
                                hoje.day, hoje.month, (hoje.year - 1900) );
                flux = fopen(nomearq, "w");
fprintf(flux, "%s", arq);
fclose(flux);
}
```

```
  delete_window(mn_mod);
  cursor(xold, yold);

}



int ht(char **tb)
{
int h = 0;

  while(*(tb + h++));
  return (h + 2);
}

/*------- Compute width of a window display table------ */
int wd(char **tb)
{
int w = 0;
  while(*tb)
   {
   w = max(w, strlen(*tb));
   tb++;
   }
  return (w + 2);
}

void nome_arq_trab( char *nomearq)
{
WINDOW *wnd;
FIELD *fld;
int c;


  wnd = establish_window(10, 5, 3, 40);
  set_border(wnd, 1);
  set_title(wnd, " Registro de Instrucoes ");
  set_colors(wnd,   ALL,  BLUE, WHITE, BRIGHT);
  set_colors(wnd, ACCENT, WHITE, BLACK,   DIM);
  display_window (wnd);

  wprompt(wnd,  1, 0, " Arquivo: ");
  fld = establish_field(wnd, 12, 0, "____________________", nomearq, 'A');
  field_window(fld, "nomeprog", 12, 10);

  do
   {

   strcpy(nomearq, "c:\\dados\\PLANO.TRB   "); /* Nome Default          */
                                  /* para arquivo de trabalho. */
   clear_template;
   field_tally(wnd);
```



```
                c = data_entry(wnd);

                troca_espaco(nomearq);
   } while (c == ESC);
   delete_window(wnd);
                nomearq[23] = '\0';
}
char *troca_espaco (char *frase);


char *troca_espaco(char *frase)

/*
        Objetivo: Retirar todos os espacos a direita de uma
                cadela alfanumerica; trocar os espacos intermediarios
                (ascii = 32) por "hard space", (ascii= 255).
                Frase nunca volta vazia; Retorna pelo menos um espaco "duro".
*/

{
int c = 0;

/* Eliminamos brancos a direita */
        while( frase[++c] );
        for (; (c > 0 && frase[c] <= (char) 32); c--);
        if( (unsigned) frase[c] > 32)
                frase[c + 1] = (char) 0;
        else
                {
                if (c == 0)
                        {
                        frase[c] = (char) 255;
                        frase[c + 1] = (char) 0;
                        return (frase);
                        }
                else
                        frase[c] = (char) 0;
                }

/* Trocamos os espacos intermediarios (ascii = 32) por (ascii = 255)      */
        c = 0;
        while(frase[++c])
                if ( *(frase + c) == (char) 32)
                        *(frase + c) = (char) 255;

        return (frase);
}
/* Fim módulo pc_menu.c */

/* Início módulo pc_detal.c */
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
```

```
#include <stdlib.h>
#include <conio.h>
#include "qwindow.h"
#include "keys.h"
#include "pc_menu.h"

extern int crate, estacao, seq;

void detalha_canal( char *buf, int nci)
{
WINDOW *wndb;
FIELD *fld;
static char title[10];
int c, canal, j, jj = 0;
static char L0[13], L1[4], L2[8], L3[13], L4[34], L5[2];
static char L6[14];

 L0[12] = L1[2] = L2[7] = L3[12] = L4[32] = '\0';
 wndb = establish_window(5, 14, 7, 56);
 set_border(wndb, 1);
 set_colors(wndb,   ALL,  BLUE, WHITE, BRIGHT);
 set_colors(wndb, ACCENT, WHITE, BLACK,   DIM);
 wprompt(wndb,  1, 0, " Grandeza:  ");
 wprompt(wndb, 28, 0, "Abreviacao:  ");
 wprompt(wndb,  1, 1, " Unidades: ");
 wprompt(wndb, 20, 1, "Fator Multiplicativo: ");
 wprompt(wndb,  1, 2, " Linha de comentario: ");
 wprompt(wndb,  1, 4, " Mostra em grafico ? (S/N): ");

 init_template(wndb);
 fld = establish_field(wndb, 11, 0,  "____________", L0, 'a');
 field_window(fld, "grandeza", 40, 5);
 fld = establish_field(wndb, 41, 0,          "___", L1, 'a');
 field_window(fld, "abrevia ", 40, 5);
 fld = establish_field(wndb, 11, 1,      "_______", L2, 'a');
 field_window(fld, "unidades", 40, 5);
 fld = establish_field(wndb, 42, 1,  "____________", L3, 'a');
 field_window(fld, "fatmult ", 40, 5);
 fld = establish_field(wndb,  1, 3,
                "__________________________________", L4, 'a');

 fld = establish_field(wndb, 30, 4,           "_", L5, 'A');

 for(canal = 0; canal < nci; canal++)
  {
  strcpy(title, " Canal: ");
  strcat(title, itoa((canal + 1), "   ", 10) );
  set_title (wndb, title);
  display_window(wndb);
  do
   {
   clear_template(wndb);
   } while ( (c = data_entry(wndb)) == ESC);
```



```
                 troca_espaco(L0);
                 troca_espaco(L1);
                 troca_espaco(L2);
                 troca_espaco(L3);
                 troca_espaco(L4);
                 strcpy(L6, L1);
                 strcat(L6, "(");
                 strcat(L6, L2);
                 strcat(L6, ")");
                 c = (L5[0] == 'S') ? 1 : 0;

  j = sprintf(buf + jj, "#%0.2u %u %12s %12s %12s %32s\n",
                                 canal, c, L0, L6, L3, L4);
                 jj += j;

  }
 delete_window(wndb);
}
/* Fim módulo pc_detal.c */

/* Início módulo pc_np.c */
/*
 Menu para modulos nao programaveis internamente;
 registro servira para compor arquivo para a tarefa de
 transferencia de dados. A unica selecao ativa refere-se
 ao numero de canais de interesse.
  Presta-se aos modulos 8210 e 2264.

    Fevereiro de 1989.
*/

#include <stdlib.h>
#include "qwindow.h"
#include "keys.h"
#include "pc_menu.h"

extern int crate, estacao;
static char I5[2], I6[3], I7[2], I8[2];

int mod_np(WINDOW *wnda, char *mod, int *nci, char *coment)
{
FIELD *fld;
int c;

 coment[0] = '\0';

 set_border(wnda, 1);
 set_colors(wnda,   ALL,  BLUE, WHITE, BRIGHT);
 set_colors(wnda, ACCENT, WHITE, BLACK,   DIM);
 wprompt(wnda,  1, 0, " Crate:  ");
 wprompt(wnda, 15, 0, "Estacao:  ");
 wprompt(wnda,  1, 2, "   Canais Interesse: ");
 wprompt(wnda,  1, 3, " Linha de comentario: ");
 wprompt(wnda,  1, 5, " Particulariza cada canal ? (S/N)");
```

```
switch (mod[0])
 {
 case 'D' : set_title(wnda, " Waveform Digitizer 2264 ");
        break;

 case 'A' : set_title(wnda, " Waveform Analizer  8210 ");
        break;
 }

 init_template(wnda);
 fld = establish_field(wnda,  9, 0,   "_",  I5,  'N');
 field_window(fld, "crate   ",12,10);
 fld = establish_field(wnda, 25, 0,   "__",  I6,  'N');
 field_window(fld, "estacao ",12,10);
 fld = establish_field(wnda, 24, 2,   "_",  I7,  'N');
 field_window(fld, "canint  ",12,10);
 fld = establish_field(wnda,  1, 4,
        "                                                    ", coment,'a');

 fld = establish_field(wnda, 35, 5,   "_",  I8,  'A');
 field_window(fld, "detail  ",12,10);

 display_window (wnda);


do
 {
 clear_template(wnda);
 } while ( (c = data_entry(wnda)) = = ESC);

crate   = atoi(I5);
estacao = atoi(I6);
*nci    = atoi(I7);

        troca_espaco(coment);

        if(I8[0] = = 'S')
  return (1);

 return (0);
}

/* Fim módulo pc_np.c */

/* Início módulo pc_8212.c */
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <stdlib.h>
#include <conio.h>
#include "qwindow.h"
#include "keys.h"
#include "pc_menu.h"
```



```
extern int crate, estacao, seq;

int interv_perm (char *);
int escolha_perm(char *);
int canais_perm (char *);



static char *interv[] =
 { "    ", "5000", "1000", " 500", " 200", " 100", "  50", "  25"};
static char *post_samp[] =
 { "1024", " 896", " 768", " 640", " 512", " 384", " 256", " 128"};
static char *can_perm[] =
 {" 4", " 8", "16", "32"};

static char I1[3], I3[5], I4[5], I5[3], I6[3], I7[3], I8[2];
int ptsl = 0, clk = 0, nca = 0, nci = 0;


int mod_dl8212a(WINDOW *wnda,unsigned long int *palcom,int *nci,char *coment)
{
FIELD *fld;
int c;

 coment[0] = '\0';

 set_border(wnda, 1);
 set_title(wnda, " Fast Data Logger 8212A ");
 set_colors(wnda,   ALL,  BLUE, WHITE, BRIGHT);
 set_colors(wnda, ACCENT, WHITE, BLACK,   DIM);
 display_window (wnda);
 wprompt(wnda,  1,  0, " Crate:  ");
 wprompt(wnda, 15,  0, "Estacao:  ");
 wprompt(wnda,  1,  2, "      Canais ativos: ");
 wprompt(wnda,  1,  3, "   Canais Interesse: ");
 wprompt(wnda,  1,  5, "Ptos. pos-amostragem: ");
 wprompt(wnda,  1,  7, " Interv.entre pontos:      (microseg)");
 wprompt(wnda,  1,  9, " Linha de comentario: ");
    wprompt(wnda,  1, 11, " Particulariza cada canal ? (S/N)");
 init_template(wnda);


 fld = establish_field(wnda,  9, 0,   "_",  I5,  'N');
 field_window(fld, "crate   ", 12, 10);

 fld = establish_field(wnda, 25, 0,   "__",  I6,  'N');
 field_window(fld, "estacao ", 12, 10);

 fld = establish_field(wnda, 24, 2,   "__",  I1,  'N');
 field_window(fld, "canat   ", 12, 10);
 field_validate(fld, canais_perm);

 fld = establish_field(wnda, 24, 3,   "__",  I7,  'N');
```

```
field_window(fld, "canint  ", 12, 10);

fld = establish_field(wnda, 24, 5, "____", l3, 'N');
field_window(fld, "postrig ", 12, 10);
field_validate(fld, escolha_perm);

fld = establish_field(wnda, 24, 7, "____", l4, 'N');
field_window(fld, "tempo   ", 12, 10);
field_validate(fld, interv_perm);

fld = establish_field(wnda,  1,10,
        "____________________________________________", coment,'a');
fld = establish_field(wnda, 35, 11,   "_", l8, 'A');
field_window(fld, "detail  ", 12, 10);
do
  {
  clear_template(wnda);
  } while ( (c = data_entry(wnda)) = = ESC);

crate = atoi(l5);
estacao = atoi(l6);
*palcom = nca | (clk < < 2) | (ptsl < < 5);
*ncl  = atoi(l7);

        troca_espaco(coment);

if(l8[0] = = 'S')
  return (1);
return (0);
}


int interv_perm(char *bf)
{
int i = 0;
 char **seta = interv;

 while(*seta)
   {
   if(strcmp(*seta+ +, bf) = = 0)
     {
     clk = i;
     return OK;
     }
   i+ +;
   }
 error_message( "Int.Poss(em micr-seg):5000,1000,500,100,50,25, (ext)");
 return ERROR;
}

int escolha_perm(char *bf)
{
int i = 0;
char **seta = post_samp;
```



```
 while(*seta)
   {
   if(strcmp(*seta+ +, bf) = = 0)
     {
     ptsl = i;
     return OK;
     }
   i+ +;
   }
 error_message("Amost.pos-trig: 1024,896,768,640,512,384,256,128" );
 return ERROR;
}

int canais_perm(char *bf)
{
int i = 0;
char **seta = can_perm;

 while(*seta)
   {
   if(strcmp(*seta+ +, bf) = = 0)
     {
     nca = i;
     return OK;
     }
   i+ +;
   }
 error_message( "Canais ativos possiveis sao: 4, 8, 16 ou 32");
 return ERROR;
}

/* Fim módulo pc_8212.c */

/* Início módulo pc_8501.c */

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <stdlib.h>
#include <conio.h>
#include "qwindow.h"
#include "keys.h"
#include "pc_menu.h"

int  val_permitido(char *bf);
void escolhe_freq(int i, int *range, char *periodo);
void num_pulsos(int i, unsigned long int *numpul);

extern int crate, estacao;

int freq;

int mod_cg8501(WINDOW *wndc, struct complex_time *aj_tempo, char *coment)
```

```
{
WINDOW *wnda;
FIELD *fld, *fla;
int c, rang = 2;
char per[20], L0[2], L1[3], L2[2];

 coment[49] = L0[1] = L1[2] = L2[1] = '\0';

 set_border(wndc, 1);
 set_title (wndc, " Base de tempo ");
 set_colors(wndc,   ALL,  BLUE, WHITE, BRIGHT);
 set_colors(wndc, ACCENT, WHITE, BLACK,   DIM);
 wprompt(wndc,  1,  0, " Modo de operacao: ");
 display_window(wndc);

 init_template(wndc);
 fla = establish_field(wndc, 21, 0,   " ",  L2,  'N');
 field_window(fla, "modoper ", 12, 10);

 do
  {
  clear_template(wndc);
  } while ( (c = data_entry(wndc)) == ESC);

 aj_tempo->modo = atoi(L2);

 if (aj_tempo->modo == 0)
  {
  crate = 0;
  estacao = 1;
  return(0);
  }

 wnda = establish_window(10, 7, 5, 50);
 set_border(wnda, 1);
 set_title(wnda, " Clock Generator 8501 ");
 set_colors(wnda,   ALL,  BLUE, WHITE, BRIGHT);
 set_colors(wnda, ACCENT, WHITE, BLACK,   DIM);
 wprompt(wnda,  1,  0, " Crate:  ");
 wprompt(wnda, 15,  0, "Estacao:  ");
 wprompt(wnda,  1,  1, " Linha de comentario: ");

 display_window (wnda);
 init_template(wnda);

 fld = establish_field(wnda,  9, 0,   " ",  L0,  'N');
 field_window(fld, "crate   ", 12, 10);
 fld = establish_field(wnda, 25, 0,   " ",  L1,  'N');
 field_window(fld, "estacao ", 12, 10);
 fld = establish_field(wnda,  1, 2,
        "                                                  ", coment,'a');


 do
```



```
  {
  clear_template(wnda);
  } while ( (c = data_entry(wnda) ) == ESC);

 crate   = atoi(L0);
 estacao = atoi(L1);

 switch (aj_tempo->modo)
  {
  case 1  :  escolhe_freq(1, &rang, per);
             aj_tempo->per_1 = (float) atof(per);

             escolhe_freq(2, &rang, per);
             aj_tempo->per_2 = (float) atof(per);
                    num_pulsos( 2, &aj_tempo->numpul_2);

             escolhe_freq(3, &rang, per);
             aj_tempo->per_3 = (float) atof(per);
                    num_pulsos( 3, &aj_tempo->numpul_3);

             aj_tempo->range = rang;
                    break;

  case 2  :  escolhe_freq(1, &rang, per);
             aj_tempo->per_1 = (float) atof(per);
             escolhe_freq(2, &rang, per);
             aj_tempo->per_2 = (float) atof(per);
             escolhe_freq(3, &rang, per);
             aj_tempo->per_3 = (float) atof(per);
             aj_tempo->range = rang;
             break;

  case 3  :  escolhe_freq(1, &rang, per);
             aj_tempo->per_1 = (float) atof(per);
             aj_tempo->range = rang;
             break;

  case 4  :  escolhe_freq(2, &rang, per);
             aj_tempo->per_2 = (float) atof(per);
                    num_pulsos(2, &aj_tempo->numpul_2);

             aj_tempo->range = rang;
             break;
  }


  delete_window(wnda);
             troca_espaco(coment);

                    return(0);
}


int val_permitido(char *bf)
```

```
{
static char *val_perm[] = {" 1", " 2", " 5",
                    " 10", " 20", " 50",
                    "100", "200", "500"};
int c, i = 0;
 char **seta = val_perm;

 while(*seta)
  {
  if((c = strcmp(*seta++, bf)) == 0)
   {
   freq = i;
   return OK;
   }
  i++;
  }
 error_message(" So sao permitidos numeros na sequencia 1, 2, 5");
 return ERROR;
}


void escolhe_freq(int i, int *range, char *periodo)
{
WINDOW *wnd;
FIELD *fle;
char titulo[15];
int c, erro;
static char L3[4], L4[2];

static char *freq_Hz [] = {      " 20", " 50",
                    "100", "200", "500"};

static char *freqkHz [] = {" 1", " 2", " 5",
                    " 10", " 20", " 50",
                    "100", "200", "500"};

static char *freqMHz [] = {" 1", " 2", " 5",
                    " 10", " 20"};

static char *interv  [] = {          "50000  ", "20000  ",
                    "10000  ", " 5000  ", " 2000  ",
                    " 1000  ", " 500  ", " 200  ",
                    " 100  ", " 50  ", " 20  ",
                    "  10  ", "  5  ", "  2  ",
                    "   1  ", "  0.50", "  0.20",
                    "  0.10", "  0.05"};
 *range = 1;

 wnd = establish_window(5, 7, 3, 20);
 strcpy(titulo, itoa(i, " ", 10));
 strcat(titulo,"a. frequencia");
 set_title(wnd, titulo);
 set_colors(wnd,   ALL,  BLUE, WHITE, BRIGHT);
 set_colors(wnd, ACCENT, WHITE, BLACK,   DIM);
```



```
 wprompt( wnd, 9, 0,"Hz");

 Init_template(wnd);
 fle = establish_field(wnd, 1, 0, "___", L3, 'N');
 field_validate(fle, val_permitido);
 fle = establish_field(wnd, 8, 0,  "_", L4, 'a');
 display_window(wnd);

 do
  {
  erro = 0;
  clear_template(wnd);
  c = data_entry(wnd);

  switch( L4[0] )
   {
   case ' ' :  if (freq == 0)
               {
               error_message(" Minima freq.Amostragem: 20Hz ");
               erro++;
               }
               else if (freq < 3)
               *range = 1;

              strcpy(periodo, interv[freq]);
              break;

   case 'k' :  strcpy(periodo, interv[freq + 5]);
              break;

   case 'M' :  if( freq > 4)
               {
               error_message(" Maxima freq.Amostragem: 20 MHz ");
               erro++;
               }
              strcpy(periodo, interv[freq + 14]);
              break;

   default  :  error_message(" Apenas \" \"(x1), k (kilo) ou M (mega) ");
              erro++;
   }
  } while ((c == ESC) || (erro) );

  delete_window(wnd);
}

void num_pulsos(int i, unsigned long int *numpul)
{
WINDOW *wnd;
FIELD *fle;
int c, erro;
static char titulo[14], L0[9];

 L0[8] = '\0';
```

```
  wnd = establish_window(5, 7, 3, 16);
  strcpy(titulo, itoa(i, " ", 10));
  strcat(titulo,":N.Pulsos");
  set_title(wnd, titulo);
  set_colors(wnd,   ALL,  BLUE, WHITE, BRIGHT);
  set_colors(wnd, ACCENT, WHITE, BLACK,   DIM);

  init_template(wnd);
  fle = establish_field(wnd, 3, 0, "________", L0, 'N');
  display_window(wnd);

  do
   {
   erro = 0;
   clear_template(wnd);
   c = data_entry(wnd);
   if ( (*numpul = atoi(L0)) > 0xffff)
            {
            error_message( "Numero maximo e de 64k");
            erro++;
            }

   } while ( (c == ESC) || erro);

 delete_window(wnd);
}

/* Fim módulo 8212.c */

/* Início módulo pc_8501.c */


#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <stdlib.h>
#include <conio.h>
#include "qwindow.h"
#include "keys.h"
#include "pc_menu.h"

int  val_permitido(char *bf);
void escolhe_freq(int i, int *range, char *periodo);
void num_pulsos(int i, unsigned long int *numpul);

extern int crate, estacao;

int freq;

int mod_cg8501(WINDOW *wndc, struct complex_time *aj_tempo, char *coment)
{
WINDOW *wnda;
FIELD *fld, *fla;
```



```
int c, rang = 2;
char per[20], L0[2], L1[3], L2[2];

 coment[49] = L0[1] = L1[2] = L2[1] = '\0';

 set_border(wndc, 1);
 set_title (wndc, " Base de tempo ");
 set_colors(wndc,   ALL,  BLUE, WHITE, BRIGHT);
 set_colors(wndc, ACCENT, WHITE, BLACK,   DIM);
 wprompt(wndc,  1,  0, " Modo de operacao: ");
 display_window(wndc);

 init_template(wndc);
 fla = establish_field(wndc, 21, 0,   " ",  L2,  'N');
 field_window(fla, "modoper ", 12, 10);

 do
  {
  clear_template(wndc);
  } while ( (c = data_entry(wndc)) == ESC);

 aj_tempo->modo = atoi(L2);

 if (aj_tempo->modo == 0)
  {
  crate = 0;
  estacao = 1;
  return(0);
  }

 wnda = establish_window(10, 7, 5, 50);
 set_border(wnda, 1);
 set_title(wnda, " Clock Generator 8501 ");
 set_colors(wnda,   ALL,  BLUE, WHITE, BRIGHT);
 set_colors(wnda, ACCENT, WHITE, BLACK,   DIM);
 wprompt(wnda,  1,  0, " Crate:  ");
 wprompt(wnda, 15,  0, "Estacao:  ");
 wprompt(wnda,  1,  1, " Linha de comentario: ");

 display_window (wnda);
 init_template(wnda);

 fld = establish_field(wnda,  9, 0,   "_",  L0,  'N');
 field_window(fld, "crate   ", 12, 10);
 fld = establish_field(wnda, 25, 0,   "__",  L1,  'N');
 field_window(fld, "estacao ", 12, 10);
 fld = establish_field(wnda,  1, 2,
        "____________________________________________", coment,'a');


do
 {
 clear_template(wnda);
 } while ( (c = data_entry(wnda) ) == ESC);
```

```
    crate   = atoi(L0);
    estacao = atoi(L1);

    switch (aj_tempo->modo)
     {
     case 1 : escolhe_freq(1, &rang, per);
            aj_tempo->per_1 = (float) atof(per);

            escolhe_freq(2, &rang, per);
            aj_tempo->per_2 = (float) atof(per);
                  num_pulsos( 2, &aj_tempo->numpul_2);

            escolhe_freq(3, &rang, per);
            aj_tempo->per_3 = (float) atof(per);
                  num_pulsos( 3, &aj_tempo->numpul_3);

            aj_tempo->range = rang;
                  break;

     case 2 : escolhe_freq(1, &rang, per);
            aj_tempo->per_1 = (float) atof(per);
            escolhe_freq(2, &rang, per);
            aj_tempo->per_2 = (float) atof(per);
            escolhe_freq(3, &rang, per);
            aj_tempo->per_3 = (float) atof(per);
            aj_tempo->range = rang;
            break;

     case 3 : escolhe_freq(1, &rang, per);
            aj_tempo->per_1 = (float) atof(per);
            aj_tempo->range = rang;
            break;

     case 4 : escolhe_freq(2, &rang, per);
            aj_tempo->per_2 = (float) atof(per);
                  num_pulsos(2, &aj_tempo->numpul_2);

            aj_tempo->range = rang;
            break;
     }


    delete_window(wnda);
            troca_espaco(coment);

                  return(0);
}


int val_permitido(char *bf)
{
static char *val_perm[] = {" 1", " 2", " 5",
                  " 10", " 20", " 50",
```



```
                  "100", "200", "500"};
int c, i = 0;
 char **seta = val_perm;

 while(*seta)
  {
  if((c = strcmp(*seta++, bf)) == 0)
   {
   freq = i;
   return OK;
   }
  i++;
  }
 error_message(" So sao permitidos numeros na sequencia 1, 2, 5");
 return ERROR;
}


void escolhe_freq(int i, int *range, char *periodo)
{
WINDOW *wnd;
FIELD *fle;
char titulo[15];
int c, erro;
static char L3[4], L4[2];

static char *freq_Hz [] = {      " 20", " 50",
                  "100", "200", "500"};

static char *freqkHz [] = {" 1", " 2", " 5",
                  " 10", " 20", " 50",
                  "100", "200", "500"};

static char *freqMHz [] = {" 1", " 2", " 5",
                  " 10", " 20"};

static char *interv  [] = {          "50000  ", "20000  ",
                  "10000  ", " 5000  ", " 2000  ",
                  " 1000  ", " 500  ", " 200  ",
                  " 100  ", " 50  ", " 20  ",
                  " 10  ", " 5  ", " 2  ",
                  " 1  ", " 0.50", " 0.20",
                  " 0.10", " 0.05"};
 *range = 1;

 wnd = establish_window(5, 7, 3, 20);
 strcpy(titulo, itoa(i, " ", 10));
 strcat(titulo,"a. frequencia");
 set_title(wnd, titulo);
 set_colors(wnd,   ALL,  BLUE, WHITE, BRIGHT);
 set_colors(wnd, ACCENT, WHITE, BLACK,   DIM);
 wprompt( wnd, 9, 0,"Hz");

 init_template(wnd);
```

```
  fle = establish_field(wnd, 1, 0, "___", L3, 'N');
  field_validate(fle, val_permitido);
  fle = establish_field(wnd, 8, 0,  "_", L4, 'a');
  display_window(wnd);

  do
   {
   erro = 0;
   clear_template(wnd);
   c = data_entry(wnd);

   switch( L4[0] )
    {
    case ' ' :  if (freq = = 0)
                 {
                 error_message(" Minima freq.Amostragem: 20Hz ");
                 erro+ +;
                 }
                else if (freq < 3)
                 *range = 1;

               strcpy(periodo, interv[freq]);
               break;

    case 'k' :  strcpy(periodo, interv[freq + 5]);
               break;

    case 'M' :  if( freq > 4)
                 {
                 error_message(" Maxima freq.Amostragem: 20 MHz ");
                 erro+ +;
                 }
                strcpy(periodo, interv[freq + 14]);
                break;

    default :  error_message(" Apenas \" \"(x1), k (kilo) ou M (mega) ");
               erro+ +;
    }
   } while ((c = = ESC) | | (erro) );

   delete_window(wnd);
}

void num_pulsos(int i, unsigned long int *numpul)
{
WINDOW *wnd;
FIELD *fle;
int c, erro;
static char titulo[14], L0[9];

 L0[8] = '\0';

 wnd = establish_window(5, 7, 3, 16);
 strcpy(titulo, itoa(i, " ", 10));
```



```
 strcat(titulo,":N.Pulsos");
 set_title(wnd, titulo);
 set_colors(wnd,   ALL,  BLUE, WHITE, BRIGHT);
 set_colors(wnd, ACCENT, WHITE, BLACK,   DIM);

 init_template(wnd);
 fle = establish_field(wnd, 3, 0, "________", L0, 'N');
 display_window(wnd);

 do
  {
  erro = 0;
  clear_template(wnd);
  c = data_entry(wnd);
  if ( (*numpul = atoi(L0)) > 0xffff)
          {
          error_message( "Numero maximo e de 64k");
          erro+ +;
          }

  } while ( (c = = ESC) | | erro);

 delete_window(wnd);
}

/* Fim módulo pc_8501.c */
/* Início módulo makefile para Microsoft C 5.1 */
#
#       Program: pc_menu
#


LINC=pc_mod+pc_menu+pc_8212+pc_8837+pc_np+pc_detal+pc_8501


.c.obj:
  cl -c -W3 -Od -Ze -Zi -AL -V"Lab.Fis.Plasmas, A.N.Fagundes, 1989" $*.c

pc_mod.obj  :  pc_mod.c

pc_menu.obj  :  pc_menu.c        pc_menu.h

pc_8212.obj  :  pc_8212.c        pc_menu.h

pc_8837.obj  :  pc_8837.c        pc_menu.h

pc_np.obj   :   pc_np.c   pc_menu.h

pc_detal.obj :  pc_detal.c       pc_menu.h

pc_8501.obj  :  pc_8501.c        pc_menu.h

mod.exe: pc_mod.obj pc_menu.obj pc_8212.obj pc_8837.obj pc_np.obj \
      pc_detal.obj  pc_8501.obj
```

```
        link /CO $(LINC), pc_menu,, windllib;
        exepack pc_menu.exe c:\util\pc.exe

/* Fim módulo makefile */

/* Início do arquivo módulos.hlp */
```

<nomeprog>
O nome a ser dado ao programa de instrucoes para a transferencia de arquivos pode ser qualquer um, com qualquer terminacao. Se, no entanto, nenhum nome e informado, PLANO.TRB sera assumido como o nome "default".

<progmod >
Escolha o modulo a ser programado atraves das setas verticais ou pela letra destacada em maiusculo.

<crate >
Informe o CRATE onde se localiza o modulo. E necessariamente um numero 1 ou 2 (por enquanto, pelo menos.)

<estacao >
Informe a ESTACAO onde se localiza o modulo Observe para que numero aponta a seta identificada com "N" na parte inferior do modulo; necessariamente e um numero entre 1 e 23.

<canint >
O numero aqui informado sera o numero maximo de canais a serem transferidos. E necessariamente um numero entre 1 e o de canais ativos.

<detail >
Se a resposta for "S", abrir-se-a uma janela para informacoes adicionais que serao adicionadas ao arquivo no momento da transferencia. E recomendavel seu preenchimento.

<canat >
Este item especifica o numero de canais ativos com que se fara a conversao de sinais. E necessariamente maior ou igual ao numero de canais de interesse.

<postrig >
Especifica o numero de pontos amostrais a serem guardados apos o recebimento do pulso de "Stop Trigger". Veja manual para detalhes e modificacoes.

<tempo >

Indica a escolha da frequencia de amostragem, atraves de seu inverso, o periodo, ou intervalo entre dois pontos consecutivos. Cuidado para com a taxa de amostragem, que depende do numero de canais ativos selecionados.

<memoria >
Especifica a fracao de memoria a ser utilizada, em kbytes. Recomendavel utilizar sempre toda a memoria, isto e 8/8.

<post8837>
Numeros de pontos amostrais a serem guardados apos o recebimento do pulso de "Stop Trigger". (Cuidado ao ler o manual, que especifica o numero complementar a este.)

<temp8837>
Especifica a taxa de amostragem, atraves do intervalo de tempo entre pontos consecutivos.

<grandeza>
Grandeza fisica registrada no pulso. Use chaves mnemonicas, se necessario. Por exemplo, a corrente de Plasma poderia ser indicada "Cor.Plasma".

<abrevia >
Indicacao curta da grandeza medida, tal como poderia aparecer junto ao eixo de um grafico. Por exemplo, para a corrente de Plasma poderiamos indicar "Ip".

<unidades>
Unidades da grandeza medida, tal como poderia aparecer junto ao eixo de um grafico, discriminando valores. Por exemplo, para a corrente de Plasma, "kA"

<fatmult >
Fator multiplicativo global que transforma o sinal armazenado no modulo nas unidades especificadas da grandeza medida. Por exemplo, para a corrente de plasma, este fator de conversao seria de 33 (kA/V) x o ganho do amplificador/redutor intermediario.

<modoper >
Ha cinco modos diferentes de trabalho:

0. Este modulo nao e acionado; a base de tempo e gerada, quer internamente em cada modulo, quer por outro modulo que nao o 8501.
1. Mudanca programada de frequencia.
2. Mudanca de frequencia por pulsos.
3. Pulsos intercalados de mesma frequencia.
3. Trem de pulsos com extensao programada

Refira-se ao manual (CG8501) para maiores explicacoes.

```
<end>

/* fim módulo módulos.hlp */
```


Apendice 6

O programa TC

```
/* g8901a.h - Definições para o módulo controlador LeCroy G8901A */
/*      Comandos de modo de transferencia      */

#define NT_08          0x61    /*      Transferencia normal 8 bits        */
#define NT_16          0x62    /*      16 bits  */
#define NT_24          0x64    /*      24 bits  */

#define  SBT_08  0x79  /* Leitura em Bloco (250 kHz) 8 bits */
#define  SBT_16  0x7A  /* 16 bits  */
#define  SBT_24  0x64  /* 24 bits  */

#define FBT_08         0x69    /*      Leitura em bloco (1 MHz) 8 bits */
#define FBT_16         0x6A    /*      16 bits  */
#define FBT_24         0x6C    /*      24 bits  */


#define DISRQ      0x40  /* Desabilita SRQ */

        /*      Comandos CAMAC      gerais  */
#define INITIALIZE      0x21    /*      Z       */
#define CLEAR      0x22  /* C  */
#define CLZ        0x23  /* C+Z  */
#define INH        0x48  /* I  */
#define DINH       0x40  /* not I */

#define NUM_CRATES  2          /* Numero de crates do sistema */

/* Fim g8901a.h */

/*  Igpib.h

        Prototipo das funcoes C para o STD-8410
  Modelo LARGE -- Nao esta documentado no manual !
          * acrescentamos L na frente das funcoes;
          * instrucao (far) torna-se desnecessaria.

(informacoes obtidas junto ao representante da placa)
                fevereiro-1989
*/

void libcac (int, int);
void libclr (int, int);
void libcmd (int, char  *, int);
void libconf(int, int, int);
void libdma (int, int);
void libeos (int, int);
void libeot (int, int);
void libesc (int, int, char  *, unsigned int);
void libgts (int, int);
void libist (int, int);
```

```
void libler (int, int, char *, int);
void libipe (int, int);
void libslp (int, int, char);
void liboni (int, int);
void libpad (int, int);
void librd (int, char *, int);
void librpp (int, char);
void librsc (int, int);
void librsv (int, int);
void librtl (int);
void libsad (int, int);
void libsic (int);
void libsre (int, int);
void libtmo (int, int);
void libtrg (int, int);
void libwait(int, unsigned int);
void libwrt (int, char *, int);

/* variaveis declaradas publicas pelo cb.obj */
        extern int ibsta; /* palavra de status */
        extern int iberr; /* codigo de erro GPIB */
        extern int ibcnt; /* numero de bytes enviados */

/* constantes usadas em programas de exemplos */
#define UNL 0x3f /* GPIB comando unlisten */
#define UNT 0x5f /* GPIB comando untalk */
#define GTL 0x01 /* GPIB comando go to local */
#define SDC 0x04 /* GPIB selected device clear */
#define PPC 0x05 /* GPIB parallel poll conf. */
#define GET 0x08 /* GPIB group execute trigger */
#define TCT 0x09 /* GPIB tome o controle */
#define LLO 0x11 /* GPIB local lock out */
#define DCL 0x14 /* GPIB device clear */
#define PPU 0x15 /* GPIB ppoll unconfigure */
#define SPE 0x18 /* GPIB habilitar poll serial */
#define SPD 0x19 /* GPIB desabilitar poll serial */
#define PPE 0x60 /* GPIB habilitar poll paralelo */
#define PPD 0x70 /* GPIB desabilitar poll paralelo*/
#define S 0x08 /* especifica o sentido do PPR */
#define REOS 0x400
#define XEOS 0x800
#define BIN 0x1000
#define LF 0x0a
#define TIMO 0x10000
#define SRQI 0x1000
#define CIC 0x20
#define TACS 0x08
#define LACS 0x04

/* variaveis do programa de aplicacao passadas */
/* para as funcoes STD-8410 . */
int bd;          /* numero do cartao GPIB no compuutador */
int cnt;         /* variavel cont bytes */
int v;           /* uso geral */
```



```
int conf;        /* config. de enderecos */
int disp;        /* dispositivo GPIB */
char ppr;        /* byte de poll paralelo*/
char spbyte;     /* byte para resposta de*/
                 /* poll serial */
unsigned int mask; /* eventos esperados */

/* Fim Lgpib.h */

/*      qwindow.h

        Uncomment this for stacked windows
                rather than layered windows.

#define FASTWINDOWS

*/

/*-------------Window colors-----------------------*/
#define RED          4
#define GREEN        2
#define BLUE         1
#define WHITE        (RED+GREEN+BLUE)
#define YELLOW       (RED+GREEN)
#define AQUA         (GREEN+BLUE)
#define MAGENTA      (RED+BLUE)
#define BLACK        0
#define BRIGHT       8
#define DIM          0

#define BORDER       0
#define TITLE        1
#define ACCENT       2
#define NORMAL       3
#define ALL          4

#define TRUE         1
#define FALSE        0
#define ERROR       -1
#define OK           0

/*------------  Window controller structures----------*/
typedef struct field
        {
        char *fmask;
        int fprot;
        char *fbuff;
        int ftype;
        int frow;
        int fcol;
        void (*fhelp)(char *);
        char *fhwin;
```

```
        int  fix;
        int  fly;
        int (*fvalid) (char *);
        struct field *fnxt;
        struct field *fprv;
        } FIELD;


typedef struct _wnd
        {
        int _wv;
        int _hd;
        char *_ws;
        char *_tl;
        int _wx;
        int _wy;
        int _ww;
        int _wh;
        int _wsp;
        int _sp;
        int _cr;
        int btype;
        int wcolor[4];
        int _pn;
        struct _wnd *_nx;
        struct _wnd *_pv;
        FIELD *_fh;
        FIELD *_ft;
        } WINDOW;

typedef struct w_menu
        {
        char *mname;
        char **mselcs;
        void (**func)(int, int);
        } MENU;

#define SAV    (wnd -> _ws)
#define WTITLE  (wnd -> _tl)
#define COL    (wnd -> _wx)
#define ROW    (wnd -> _wy)
#define WIDTH   (wnd -> _ww)
#define HEIGHT  (wnd -> _wh)
#define SCROLL  (wnd -> _wsp)
#define SELECT  (wnd -> _sp)
#define WCURS   (wnd -> _cr)
#define WBORDER (wnd -> wcolor[BORDER])
#define WTITLEC (wnd -> wcolor[TITLE])
#define WACCENT (wnd -> wcolor[ACCENT])
#define WNORMAL (wnd -> wcolor[NORMAL])
#define PNORMAL (wnd -> _pn)
#define BTYPE   (wnd -> btype)
#define NEXT    (wnd -> _nx)
#define PREV    (wnd -> _pv)
```



```
#define WCOLOR  (wnd -> wcolor)
#define VISIBLE (wnd -> _wv)
#define HIDDEN  (wnd -> _hd)
#define FHEAD   (wnd -> _fh)
#define FTAIL   (wnd -> _ft)

#define NW     (wcs[wnd -> btype].nw)
#define NE     (wcs[wnd -> btype].ne)
#define SE     (wcs[wnd -> btype].se)
#define SW     (wcs[wnd -> btype].sw)
#define SIDE   (wcs[wnd -> btype].side)
#define LINE   (wcs[wnd -> btype].line)

/* General Purpose functions and macros   */
void   clear_screen(void);
int    vmode(void);
unsigned video_address(void);
void   cursor(int, int);
void   curr_cursor(int *, int *);
int    cursor_type(void);
void   set_cursor_type(int);
int    get_char(void);
int    scroll_lock(void);
char far *getdta(void);
void   setdta(char far *);
unsigned peek( unsigned, unsigned);
char   peekb( unsigned, unsigned);
void   poke( unsigned, unsigned, unsigned);
void   vpoke( unsigned, unsigned, unsigned);
int    vpeek( unsigned, unsigned);
int    ht(char **);
int    wd(char **);

/* Window functions and macros */
WINDOW  *establish_window(int, int, int, int);
void   set_border   ( WINDOW *, int);
void   set_colors   ( WINDOW *, int, int, int, int);
void   set_intensity ( WINDOW *, int);
void   set_title    ( WINDOW *, char *);
void   display_window( WINDOW *);
void   clear_window  ( WINDOW *);
void   delete_window ( WINDOW *);
void   hide_window   ( WINDOW *);
void   wprintf      ( WINDOW *, char *, ...);
void   wputchar     ( WINDOW *, int);
void   close_all    (void);
void   wcursor      ( WINDOW *, int, int);
void   error_message ( char *);
void   clear_message ( void);
int    get_selection ( WINDOW *, int, char *);

#define reverse_video(wnd)        wnd -> wcolor[3] = wnd -> wcolor[2]
#define normal_video(wnd)         wnd -> wcolor[3] = wnd -> _pn
#define rmove_window(wnd, x, y)    repos_wnd( wnd,    x,    y,  0)
```

```
#define move_window(wnd, x, y)      repos_wnd( wnd, COL-x, ROW-y, 0)
#define forefront(wnd)              repos_wnd( wnd,   0,   0, 1)
#define rear_window(wnd)            repos_wnd( wnd,   0,   0, -1)

/* Internal to window processes */
void   scroll     ( WINDOW *, int);
void   repos_wnd  ( WINDOW *, int, int, int);
void   acline     ( WINDOW *, int);

#define  accent(wnd)     acline(wnd, WACCENT)
#define deaccent(wnd)     acline(wnd, WNORMAL)
#define clr(bg, fg, in)    ( (fg) | (bg < < 4) | (in) )
#define vad(x, y)          ( (y)*160 + (x)*2)

/* Editor function */
void   text_editor( WINDOW *, char *, unsigned);

/* Menu function  */

void menu_select (char *name, MENU *mn);

/* Help functions */
void   load_help(char *);
void   set_help (char *, int, int);

/* Data Entry functions  */
void   init_template  (WINDOW *);
FIELD  *establish_field (WINDOW *, int, int, char *, char *, int);
void   clear_template  (WINDOW *);
void   field_tally    (WINDOW *);
int   data_entry     (WINDOW *);
void   wprompt       (WINDOW *, int, int, char *);
void   error_message  (char *);
void   clear_notice   (void);
void   field_window   (FIELD *, char *, int, int);

#define field_protect(f, s)      f -> fprot  = s
#define field_help(f, h)         f -> fhelp  = h
#define field_validate(f, v)     f -> fvalid = v

#ifdef MSC50

#define MF(sg, of)   ((void far *)(((unsigned long)(sg) < <16 | (unsigned)(of)))

#define poke(sg, of)  (*(( int far *)MF((sg),(of))) = (int)(wd))
#define peek(sg, of)  (*(( int far *)MF((sg),(of))))
#define peekb(sg, of) (*((char far *)MF((sg),(of))))

#endif

#ifdef  NOCGA

#define vpoke(sg, of, ch)   poke(sg, of, ch)
#define vpeek(sg, of)       peek(sg, of)
```



```
#endif

/* TSR prototypes */
void   resinit   (void);
int    resident (char *);
void   terminate  (void);
void   restart   (void);
void   suspend   (void);
int    get_char  (void);
void   popup     (void);

/* Fim qwindows.h */

/*————————  Keys.h ———————————————— */

#define HT          9
#define RUBOUT                          8
#define BELL                    7
#define ESC         27
#define SHIFT_HT                143
#define CTRL_T                          20
#define CTRL_B                          2
#define CTRL_D                          4
#define ALT_D                   160

#define F1          187
#define F2          188
#define F3          189
#define F4          190
#define F5          191
#define F6          192
#define F7          193
#define F8          194
#define F9          195
#define F10         196

#define HOME                    199
#define UP          200
#define PGUP                    201
#define BS          203
#define FWD          205
#define END          207
#define DN          208
#define PGDN                    209
#define INS         210
#define DEL         211

#define CTRL_HOME               247
#define CTRL_BS                         243
#define CTRL_FWD                244
#define CTRL_END       245

/* Fim keys.h */
```

```
/*      tc.h
   Include file para o programa TC
*/

struct disp
 {
 unsigned int amostragens;
 float tempo;
 unsigned int canais_interesse;
 unsigned int canais_ativos;
 unsigned int num_pontos;
 unsigned int flag;
 unsigned int okay;
 char tipo_modulo[7];
 };

static union matran          /* Matriz para transferencia      */
 {                           /*   e ordenacao dos dados        */
 unsigned char ch [65536];
 unsigned int  it [32768];
 };

union PalavraComunicacao      /* Leitura e escrita no modulo 8901A */
 {
 unsigned long int rep;
 unsigned char lin [4];
 };

struct complex_time          /* Registro de variaveis para o modulo 8501 */
 {
 unsigned int modo;
 unsigned int range;
 unsigned long int numpul_2;
 unsigned long int numpul_3;
 float per_1;
 float per_2;
 float per_3;
 };

struct infs_detal
        {
        char  grandeza  [13];
        char  unidades  [15];
        char  comentario[35];
        int   mostra;
        float fator_mult;
        };

#define NUM_CRATES  2
```



```
void executa(union PalavraComunicacao, int, int);
void espera(double);
void exit(int);
void grava_int (int, int  *, unsigned int, char *, char *);
void nome_final(char nomearq[][24], char *radical, int canais);
void prep_inicial (int crate);
void primeira_linha(int canais, int resol, char memoria[][240]);
void problemas(int num, char *nome_modulo, int crate, int estacao);

/* fim tc.h */


/* tc.c - Modulo principal do programa tc   */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <process.h>
#include "tc.h"
#include "qwindow.h"

extern int ibsta, iberr, ibcnt; /*  Variaveis do STD-8410 */

union PalavraComunicacao pt;
union matran   z, bf;

struct disp ch, *chave = &ch;
struct complex_time aj1, aj2,
      ajref = { 0, 0, 0L, 0L, (float) 0.0, (float) 0.0, (float) 0.0};
struct complex_time *aj_tempo = &aj1, *aj_ref = &ajref, *aj = &aj2;

struct infs_detal infs[32];


float ref_tempo;


char radical[18], deriv[18], data[25], arq_resumo[800];
int larq_res;
unsigned long shot;

WINDOW wtr, *wndtr = &wtr;

int curx, cury;

static char buffer_leitura[100][80]; /* No maximo 100 instrucoes */

char    nomearq [32][ 24];  /* Dimensionamento pelo modulo maior */
char    memoria [32][160];
char complemento [32][ 80];
char mostra_graf [32][  2];
char caminho[80];
```

```
int bd = 0, crate, estacao;

void main (int argc, char *argv[]);
void mod_2264(int, char *);
void mod_8210(int, char *);
void mod_8212(int, char *);
void mod_8837(int, char *);
void linha_comando_pronta(void);
void linha_comando_remota(char buffer_leitura[][80]);
void trato_inicial(void);
void nome_sugerido(char *radical, char *data, unsigned long *shot);
void termino (int ok);

void main(int argc, char *argv[])
{
FILE *flux;
int manual = 0, nolin = 0;
char line[80], *fim_arq;
static char plano_trabalho[25] = {"c:\\dados\\plano.trb"};
static char *estac[] = {"A", "B", "C", "D", "E", "F", "G", "H", "I",
                        "J", "K", "L", "M", "N", "O", "P", "Q", "R",
                        "S", "T", "U", "V", "X", "W", "Y", "Z"};


 curr_cursor( &curx, &cury);

 wndtr = establish_window( 10, 5, 10, 40);
 set_title (wndtr, " Transferencia em Progresso ");
 set_colors(wndtr,   ALL,  BLUE,  AQUA, BRIGHT);
 set_border(wndtr,  2);



 trato_inicial();
 nome_sugerido(radical, data, &shot);
        larq_res = sprintf(arq_resumo , "%lu %s\n\r",shot, data);

        /* Leitura do arquivo de plano de trabalho. Se um nome e fornecido na
                linha de comando, procuramos por aquele arquivo no diretorio "c:\dados\".
                Se nenhum nome e fornecido, procuramos por "c:\dados\plano.trb"; se
                nao existir, invocamos a entrada manual.
        */

        switch(argc)
                {
                case 1  : if( (flux = fopen(plano_trabalho, "r") ) = = NULL)
                                                        manual+ + ;
                        break;

        case 2  : strcpy(plano_trabalho+9, argv[1]);
        if( (flux = fopen(plano_trabalho, "r") ) = = NULL)
                                                        {
                                                        error_message("Plano de trabalho nao
```



```
encontrado em C:\DADOS");
                                                        espera(2.0);
                                                        termino(0);
                                                        }
                        break;

                default : error_message(" Uso do programa: tc [plano de trabalho]\n");
                        espera(2.0);
                                                        termino(0);
                }

        if(!manual)
         {
                while ( (fim_arq = fgets(line, 80, flux)) != NULL)
   strcpy(buffer_leitura[nolin+ +], line);
  *buffer_leitura[nolin] = '\0';
                }

 while(1)
  {
  chave->canais_interesse = 32;

  if(manual = = 0)
   linha_comando_remota(buffer_leitura);
  else
   {
   linha_comando_pronta();
   if(crate = = 0 && estacao = = 0)
    termino(0);
   }

  strcpy(deriv, radical);
  strcat(deriv, itoa(crate, "  ", 10) );
  strcat(deriv, estac[estacao - 1]);

  switch(chave->tipo_modulo[1])
   {
   case 'D' : mod_2264( chave->canais_interesse, deriv);
           break;

   case 'L' : mod_8212( chave->canais_interesse, deriv);
           break;

   case 'A' : mod_8210( chave->canais_interesse, deriv);
           break;

   case 'R' : mod_8837( chave->canais_interesse, deriv);


   }
  memset(complemento, ' ', chave->canais_interesse*80);
  }
}
```

```
void termino(int ok)
{
FILE *fl;
static char listagraf[23];
static char grafador [23];

        cursor(curx, cury); /* Cursor volta para a posicao original  */

        if (ok)       /* So abre arquivo numa terminacao normal do programa      */
                {
                strcpy(listagraf, caminho);
                strcat(listagraf, radical);
                strcat(listagraf, ".GRF");

                if ( (fl = fopen(listagraf, "w")) != NULL)
                        {
                        fprintf (fl, "%s", arq_resumo);
                 fclose(fl);
                        }

                        execl("vc.exe", "vc.exe", listagraf, NULL);
                }

        exit(ok);          /* Tarefa terminada.              */
}
/* Fim módulo tc.c */

/* modulo tc_nomes.c */
/*
Aqui damos nomes aos arquivos. Nome devolvido consta dos campos
 (11 letras)  (1 letra)       ( 3 numeros )
  diretoria     mes        num_pulso do painel
  c:\dados\   corrente

e a este arquivo sera acrescentado, no programa principal,
 - um numero, identificador do crate;
 - uma letra, identificadora da estacao no crate;
 - dois numeros, identicadores do canal no modulo.

 O numero de pulsos esta num arquivo  com nome "c:\dados\contador.num"
 e e atualizado a cada chamada.
*/

#include <time.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "tc.h"
#include "qwindow.h"

extern char caminho[];

void nome_sugerido(char *radical, char *data, unsigned long *shot);
```



```
unsigned long num_pul(void);
void abre_reg_contador(void);
void termino(int ok);
char *troca_espaco (char *frase);


void nome_sugerido (char *nome, char *data, unsigned long *shot)
{
long ltime;
unsigned long num_pulso;
static char *mes [] = { "J",  "F",  "M",  "A",  "I", "U",
                "L",  "G",  "S",  "O",  "N", "D" };
char tmp1[30], *tmp;
struct tm *agora;

 time(&ltime);
 agora = gmtime(&ltime);

 strcpy( tmp1, ctime( &ltime));
 tmp1[24] = '\0';
        strcpy(data, troca_espaco(tmp1));

 *shot = num_pulso = num_pul();

 while ( num_pulso > 1000)
  num_pulso %= 1000;

        if(  (tmp = getenv("CAMINHO")) == NULL)
  strcpy( caminho, "C:\\DADOS\\");
        else
                strcpy( caminho, tmp);

 strcat(nome, mes[agora->tm_mon]);
 if(num_pulso < 100)
   strcat(nome, "0");
 if(num_pulso < 10)
   strcat(nome, "0");
 strcat(nome, ltoa(num_pulso, tmp1, 10));
}

unsigned long  num_pul()
{
FILE *fcontador;
unsigned long num_pulso;

 fcontador = fopen( "c:\\dados\\contador.num", "r+");
 if (fcontador == NULL)
  {
  abre_reg_contador();
  fcontador = fopen( "c:\\dados\\contador.num", "r+");
  }
 fscanf( fcontador," %ld", &num_pulso );
 rewind(fcontador);
 fprintf(fcontador, "%ld", num_pulso + 1);
```

```
  fclose(fcontador);

  return(num_pulso);
 }

void abre_reg_contador()
{
WINDOW *wnd;
FIELD *fld;
FILE *fabre;
static char L0 [11];

 fabre = fopen("c:\\dados\\contador.num", "w+");
 if(fabre == NULL)
  {
  printf ("\nNao e possivel abrir CONTADOR.NUM\n");
  termino(0);
  }
 else
  {
  wnd = establish_window(20, 15, 3, 42);
  set_title (wnd, " Marca Inicial de Pulsos ");
  set_colors(wnd,   ALL,  BLUE,  AQUA, BRIGHT);
  set_colors(wnd, ACCENT, WHITE, BLACK,   DIM);
  display_window(wnd);

  wprompt(wnd, 2, 0, "Numero marcado no painel: ");
  init_template(wnd);
  fld = establish_field(wnd, 28, 0, "__________", L0, 'N');
  clear_template(wnd);
  data_entry(wnd);
  delete_window(wnd);

  fprintf(fabre, "%lu",(unsigned long) atol(L0) );
  fclose(fabre);
  }
}


char *troca_espaco(char *frase)

/*
        Objetivo: Retirar todos os espacos a direita de uma
                cadeia alfanumerica; trocar os espacos intermediarios
                (ascii = 32) por "hard space", (ascii= 255).
                Frase nunca volta vazia; Retorna pelo menos um espaco "duro".
*/

{
int c = 0;

/* Eliminamos brancos a direita */
        while( frase[++c] );
        for (; (c > 0 && frase[c] <= (char) 32); c--);
```



```
        if( (unsigned) frase[c] > 32)
                frase[c + 1] = (char) 0;
        else
                {
                if (c == 0)
                        {
                        frase[c] = (char) 255;
                        frase[c + 1] = (char) 0;
                        return (frase);
                        }
                else
                        frase[c] = (char) 0;
                }

/* Trocamos os espacos intermediarios (ascii = 32) por (ascii = 255)      */
        c = 0;
        while(frase[++c])
                if ( *(frase + c) == (char) 32)
                        *(frase + c) = (char) 255;

        return (frase);
}
/* Fim módulo tc_nomes.c */

/* Inicio módulo tc_trans.c */
/*  Leitura CAMAC via interface GPIB  */

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <sys\types.h>
#include <sys\stat.h>
#include <io.h>
#include "g8901a.h"
#include "tgpib.h"
#include "tc.h"


extern int ibsta, iberr, ibcnt;
extern struct disp ch, *chave;
extern int bd;
extern char data[];
extern char caminho[];
extern unsigned long  shot;
extern struct complex_time *aj;
extern struct infs_detal infs[];
extern char arq_resumo[];
extern int iarq_res;

extern char nomearq    [] [ 24];
extern char memoria    [] [240];
extern char complemento [] [ 80];
```

```
void erro_gpib( int errno, int crate);
void termino (int ok);


void executa(union PalavraComunicacao pt, int num_bytes, int crate)
{
union PalavraComunicacao pp;

 libesc (bd, crate, pt.lin, num_bytes);
 if(iberr)
   erro_gpib(iberr, crate);

 pp.rep = 0x4020 | (crate < < 8);
 libcmd (bd, pp.lin, 2);
 if(iberr)
   erro_gpib(iberr, crate);
}


/* Completamos o nome dos arquivos a serem gravados em disco */

void nome_final(char nomearq[][24], char *radical,int canais)
{
int i;

static char *ident[] =
        {
        "01.CHR","02.CHR","03.CHR","04.CHR","05.CHR","06.CHR","07.CHR","08.CHR",
        "09.CHR","10.CHR","11.CHR","12.CHR","13.CHR","14.CHR","15.CHR","16.CHR",
        "17.CHR","18.CHR","19.CHR","20.CHR","21.CHR","22.CHR","23.CHR","24.CHR",
        "25.CHR","26.CHR","27.CHR","28.CHR","29.CHR","30.CHR","31.CHR","32.CHR"
        };

 for(i = 0; i < canais; i+ +)
  {
  strcpy(nomearq[i], radical);
  strcat(nomearq[i], ident[i]);
  }
}

void grava_int(int canal, int *buffer, unsigned int pontos,
        char *nome, char *linha)
{
static FILE *fo;
int i;
char arquivo[80];

                strcpy(arquivo, caminho);
                strcat(arquivo, nome);

 if( (fo = fopen(arquivo,"w+b")) = = NULL)
  {
  printf("\n Nao foi possivel abrir arquivo %s", arquivo);
```



```
 termino(0);
 }

                fwrite( linha, 1,   240, fo);
 fwrite(buffer, 2, pontos, fo);

                if (infs[canal].mostra)
                        {
                i = sprintf((arq_resumo+larq_res), "%s\n\r", nome);
                        larq_res + = i;
                        }

                fclose(fo);
}


void primeira_linha(int canais, int resol, char memoria[][240])
{
static char linha[100];
static char *polar[] = {" ", "pos ", "neg ", "zer "};
extern int polaridade[];

int i, j;

 sprintf( linha, "%23s %07lu %2u %04u %05u %1u",
  data,                   /* Data do disparo        */
  shot,                   /* Numero do disparo      */
  chave->canais_interesse,        /* Canais efetivamente usados */
  chave->amostragens,             /* Pontos apos trigger      */
  chave->num_pontos,              /* Extensao do registro     */
  chave->okay );                  /* Pulso tomado corretamente? */

for ( i = 0; i < canais; i+ +)
 {
 j = sprintf(memoria[i], "%48s %3s %7s %02u %02u ",
      linha,              /* Valores gerais     */
      polar[polaridade[i]],       /* Polaridade do canal */
      chave->tipo_modulo,         /* Modulo de origem    */
      resol,              /* Resolucao em bits  */
      (i+1));             /* Canal de Origem    */

 j + = sprintf((memoria[i] +j),"%02d %02d %10.6e %10.6e %10.6e %08ld %08ld ",
      aj->modo,                   /* Base de tempo: Modo; */
                                  aj->range,          /* faixa de valores    */
      aj->per_1,                  /*  1a. frequencia    */
      aj->per_2,                  /*  2a. frequencia    */
      aj->per_3,                  /*  3a. frequencia    */
      aj->numpul_2,               /* Numero de pulsos 2a f.*/
      aj->numpul_3);              /*            3a f.*/

 j + = sprintf((memoria[i] +j), " %12s %9s %10.6e %34s",
      infs[i].grandeza,                   /* Grandeza Medida      */
                                  infs[i].unidades,       /* Abreviacao e unidades */
                                  infs[i].fator_mult,     /* Fator multiplicativo */
```

```
                                        infs[i]. comentario);        /* Linha de informacao  */
  }
}

/* Fim módulo tc_trans.c */

/* Inicio módulo tc_geral.c */
/*
 Rotina de atraso. Precisao e de 0.055 s.
 Origem: "Proficient C", A. Hansen, p. 145.
*/
#include <stdlib.h>
#include <ctype.h>
#include <stdio.h>
#include <string.h>
#include <dos.h>
#include <conio.h>
#include "g8901a.h"
#include "igplb.h"
#include "tc.h"
#include "qwindow.h"
#include "keys.h"

#define TIMER_CLK 1193180L
#define TIMER_MAX 65536L
#define TICKRATE  TIMER_CLK/TIMER_MAX
#define TIMER_CTRL  0x43
#define TIMER_COUNT 0x42
#define TIMER_PREP  0xB6


extern int ibsta, iberr, ibcnt;
extern int bd, crate, estacao, interesse;
extern int curx, cury;
extern union PalavraComunicacao pt;
extern struct disp *chave;
extern struct complex_time *aj_tempo;
extern struct infs_detal infs[];

extern char    nomearq [][ 24];
extern char    memoria [][240];
extern char complemento [][ 80];
extern char arq_resumo[];

void linha_comando_pronta(void);
void linha_comando_remota(char buffer_leitura[][80]);
void erro_gplb( int errno, int crate);
void espera (double);
void trato_inicial(void);
void termino (int ok);
long getticks(void);
int validate_module(char *);
```



```
void espera (double d)
{
long ticks, then;

 ticks = (long) d * TICKRATE;
 then = getticks() + ticks;
 while(1)
  if(getticks() >= then)
    break;
}

long getticks( void )
{
long count;
union REGS inregs, outregs;

 inregs.h.ah = 0;
 int86( 0x1a, &inregs, &outregs);

 count = (outregs.h.al != 0) ? 0x01800B0L : 0;

 count += (outregs.x.dx + (outregs.x.cx << 16));
 return (count);
}

void trato_inicial()
{
int i;
/* Estabelecemos condicoes iniciais de uso da porta GPIB  */
                    /* Interface STD-8410  */

 libsic (bd);        /* Inicializa a cadeia de instrumentacao  */
        libsre (bd, 1);      /* Permite comunicacao remota dos modulos */
        libeos (bd, 0);      /* Desativa terminacao por carater     */
 libdma (bd, 0);      /* Desliga DMA     */

          /* Controladora 8901A de cada crate    */

 pt.lin[0] = DISRQ;
 for(i = 0; i < NUM_CRATES; i++)
  executa (pt, 3, i+1);   /* Desabilitamos Service ReQuest  */
}

static int nolin = 0;

void linha_comando_remota(char buffer_leitura[][80])
{
char linha[80], linhab[80], *p, *lb;
int canal, next;

 crate = estacao = -1;
 while( crate == -1 || estacao == -1 || next == -1)
  {
```

```
                next = -1;
                strcpy(linha, buffer_leitura[nolin++]);
    lb = linhab;
    strcpy(lb, linha);
    if (*buffer_leitura[nolin] != '#')
            next = 1;

    p = strtok(linha, " ");
    while (p != NULL)
     {
     switch ( toupper(*p) )
      {
      case 'C' : sscanf((p + 1) , "%d", &crate);
              break;

      case 'I' : sscanf((p + 1), "%d", &chave->canais_interesse);
              break;


              case 'D' : strcpy(chave->tipo_modulo, "WD2264");
              sscanf((p + 1), "%d", &estacao);
              break;

      case 'A' : strcpy(chave->tipo_modulo, "WA8210");
              sscanf((p + 1), "%d", &estacao);
              break;

      case 'L' : strcpy(chave->tipo_modulo, "DL8212a");
              sscanf((p + 1), "%d", &estacao);
              break;

      case 'R' : strcpy(chave->tipo_modulo, "TR8837f");
              sscanf((p + 1), "%d", &estacao);
              break;

      case 'K' : strcpy(chave->tipo_modulo, "CK8501");
                      sscanf((p + 1), "%d", &estacao);
                              break;

      case '#' : if(chave->tipo_modulo[1] == 'K')
                 sscanf(lb+1,"%d %d %f %f %f %d %d",
                              &aj_tempo->modo,
                              &aj_tempo->range,
                              &aj_tempo->per_1,
                              &aj_tempo->per_2,
                              &aj_tempo->per_3,
                              &aj_tempo->numpul_2,
                              &aj_tempo->numpul_3);

                              else
                                      {
                                                                      sscanf((lb + 1),
"%d", &canal);
                              if ( (canal >= 0) && (canal < 32) )
```



```
                                      sscanf((lb + 3), "%d %s %s %f %s",
&infs[canal].mostra,
                              infs[canal].grandeza,
                                      infs[canal].unidades,
                                      &infs[canal].fator_mult,
                                      infs[canal].comentario);

                                                                      }

      case ';' : goto aa;    /* Comentarios */

      case '@' : termino(1);
                                      break;
              case 'Z' :          /* Palavra contendo a programacao */
      default  : break;
      }
    p = strtok(NULL, " ");
    }
aa: ;
   }
}


void problemas(int num, char *nome_modulo, int crate, int estacao)
{
WINDOW *wnd;

static char *erro [] =
        {
        " * Transferencia prejudicada. ",
        0
        };

        wnd = establish_window(10, 0, 7, 40);
        set_colors(wnd, ALL, RED, YELLOW, BRIGHT);
        set_title(wnd," Problemas ");
        display_window(wnd);

        wprintf(wnd,"%s", erro[num]);
        wprintf(wnd, "\n");
        wprintf(wnd,"\n              Modulo: %s", nome_modulo);
        wprintf(wnd,"\n        Crate: %d       Estacao: %d ",
                crate, estacao);
        wprintf(wnd,"\n          (Aperte uma tecla)");
        putchar(BELL);

        getch();

        delete_window(wnd);
```

```
}


void linha_comando_pronta(void)
{
static char L0 [3], L1[3], L2[7], L3[3];
WINDOW *wnd;
FIELD *fld;

        wnd = establish_window(2, 3, 3, 72);
        set_title (wnd, " Processo Imediato de Transferencia ");
        set_colors(wnd,   ALL,  BLUE,  AQUA, BRIGHT);
        set_colors(wnd, ACCENT, WHITE, BLACK,   DIM);
        set_border(wnd, 3);
        display_window(wnd);

        wprompt(wnd,  2,  0, "Crate: ");
        wprompt(wnd, 15,  0, "Estacao: ");
        wprompt(wnd, 28,  0, "Modulo: ");
        wprompt(wnd, 45,  0, "Canais de interesse: ");
        init_template(wnd);

        fld = establish_field(wnd,  9, 0,     "_", L0, 'N');
        fld = establish_field(wnd, 24, 0,     "__", L1, 'N');
        fld = establish_field(wnd, 36, 0, "_______", L2, 'A');
        field_validate (fld, validate_module);
        fld = establish_field(wnd, 66, 0,     "__", L3, 'N');

        clear_template(wnd);
        data_entry(wnd);
        delete_window(wnd);

        crate              = atoi(L0);
        estacao            = atoi(L1);
        strcpy(chave->tipo_modulo, L2);
        chave->canais_interesse = atoi(L3);

}

int validate_module(char *bf)
{
static char *modules [] = {
        "WD2264",
        "WA8210",
        "DL8212",
        "TR8837",
        0};
char **md = modules;

        while(*md)
                if(strcmp(*md++, bf) == 0)
                        return (OK);
        error_message(" Modulos possiveis: WD2264, WA8210, DL8212, TR8837 ");
        return (ERROR);
```



```
}

void erro_gpib(int errno, int crate)
{
static char *tab_Std_8410 [] = {
        " Placa 8410 deve ser Controller-in-Charge. ",
        " Inoperante (Nao ha \"listeners\"). ",
        " Placa STD8410 incorretamente enderecada. ",
        " Argumento invalido para a funcao. ",
        " Placa deve ser o System Active Controller. ",
        " Operacao de Entrada/Saida foi abortada. ",
        " Cartao de interface nao existe. ",
        " Placa Std-8410 incapaz de operar. ",
        " Sinal  CIC ativado, mas nao o de ATN. ",
        " ",
        " ",
        " ",
        " ",
        " Mudanca de enderecamento durante a operacao. ",
        " Tempo de operacao excedido. ",
        " Ocorreu comando DCL ou SDC durante a operacao. ",
        " Ocorreu comando GET durante uma transmissao de dados. ",
        0};

WINDOW *wnd;

        wnd = establish_window(5, 0, 3, 70);
        set_colors(wnd, ALL, RED, YELLOW, BRIGHT);
        set_title(wnd," Erro na Porta GPIB ");
        display_window(wnd);

        wprintf(wnd,"Crate: %d", crate);
        wprintf(wnd," - %s", tab_Std_8410[errno - 1]);

        putchar(BELL);

        espera(1.0);
        delete_window(wnd);
}

/* Fim módulo tc_geral.c */

/* Inicio módulo tc_2264.c */
/*     Leitura CAMAC via interface GPIB      */
   /*  modulo 2264     */

#include <stdio.h>
#include "lgpib.h"
#include "tc.h"
#include "g8901a.h"
#include "qwindow.h"

  /* Variaveis globais, definidas no MAIN */

extern int bd, crate, estacao;
```

```
extern union PalavraComunicacao pt;
extern union matran  z, bf;
extern WINDOW *wndtr;

extern struct disp *chave;
extern float ref_tempo;
extern struct complex_time *aj_tempo, *aj_ref, *aj;

extern char nomearq [] [ 24];
extern char memoria [] [240];

int polaridade[8];

int       le_2264(int modo_transf);
void      mod_2264(int, char *);
void pos_chaves_2264(void);


void mod_2264(int num_canais, char *radical)
{
register unsigned int j, *buff;
int i;

 pt.lin[2] = (char) estacao;   /*  Comando Camac:  N */


/*  Preparacao inicial do modulo para leitura */
 pt.lin[0] = 26;   /* Comando Camac F = 26 */
 executa(pt, 3, crate);
 espera( 1.0);

 pos_chaves_2264();

 chave->canais_interesse = (chave->canais_ativos > num_canais) ?
        num_canais : chave->canais_ativos;

 if(chave->tempo = = (float) -1.0)  /* padrao externo de tempo */
  {
                if (aj_tempo->modo)          /* Comando pelo ck8501 */
          aj = aj_tempo;
                else
                        aj->per_1 = ref_tempo;       /* Comando por outro modulo */
                }
        else
                aj->per_1 = ref_tempo = chave->tempo;

 primeira_linha(chave->canais_interesse+1, 256, memoria);
 nome_final(nomearq, radical, chave->canais_interesse+1);

 clear_window(wndtr);
 wcursor   (wndtr,  2, 0);
 wprintf   (wndtr, " Modulo: ");
 wcursor   (wndtr, 23, 0);
 wprintf   (wndtr, " Estacao: ");
```



```
 wcursor   (wndtr, 2, 2);
 wprintf   (wndtr, " Bytes transferidos: ");
 display_window(wndtr);
 wcursor(wndtr, 11, 0);
 wprintf(wndtr, chave->tipo_modulo);
 wcursor(wndtr, 34, 0);
 wprintf(wndtr, "%d", estacao);
 wcursor(wndtr, 2, 3);

 switch( chave->canais_ativos )
  {
  case 1 : pt.lin[1] =  0;
        if( le_2264(FBT_16) )
         {
         for (j = 0, buff = bf.it; j < 32768; j++, buff++)
          {
          *buff     = (int) *(z.ch + j) ;
          }
         grava_int(0, bf.it, chave->num_pontos, nomearq[0], memoria[0]);
         }
        else
         problemas(0, chave->tipo_modulo, crate, estacao);
        break;

  case 2  :
  case 4  :
  case 8  : buff = bf.it;
         for (i = 0; i < chave->canais_interesse; i += 2)
          {
          pt.lin[1] = (char) i/2;
          if (le_2264 (SBT_16) )
           {
           for(j = 0; j < 2*chave->num_pontos; j += 2, buff++)
            {
            *buff                    = (int) *(z.ch + j);
            *(buff + chave->num_pontos)  = (int) *(z.ch + j+1);
            }
           buff += chave->num_pontos + 1;
           }
          else
           {
           problemas(0, chave->tipo_modulo, crate, estacao);
           break;
           }
          }
         for (i = 0; i < chave->canais_interesse; i++)
          grava_int(i, (bf.it + i*chave->num_pontos),
           chave->num_pontos, nomearq[i], memoria[i]);
         break;
  }
  hide_window(wndtr);
}
```

```
void pos_chaves_2264()
{
int i;
union PalavraComunicacao px;

static unsigned int post_trigger_2264[8] =  { 8, 7, 6, 5, 4, 3, 2, 1};
static float samp_period_2264 [8] =
    { (float) 0.0, (float) 0.0, (float) -1.0, (float)25.0, (float)5.0,
     (float) 2.5, (float) 0.5, (float) 0.25};
static unsigned int channels_2264   [4] =  { 8, 4, 2, 1};

/* Posicao das chaves no painel frontal do modulo */

 /* Configuracao */
 pt.lin[0] = 1;      /* Funcao Camac: F = 1 */
 pt.lin[1] = 0;      /* Comando Camac: A = 0 */

 libesc(bd, crate, pt.lin, 3);
 libler(bd, crate, px.lin, 3);

 chave->amostragens  = post_trigger_2264  [px.rep & 0x00000007L];
        px.rep >>= 3;
 chave->tempo        = samp_period_2264   [px.rep & 0x00000007L];
        px.rep >>= 3;
 chave->canais_ativos = channels_2264     [px.rep & 0x00000003L];
        px.rep >>= 2;
 chave->flag         = (unsigned int)(px.rep & 0x00000001L);
        px.rep >>= 1;
 chave->okay         = (unsigned int)(px.rep & 0x00000001L);
 chave->num_pontos   = (unsigned int) 32768/chave->canais_ativos;

 /* Polaridade dos canais */

 pt.lin[0] = 0; /* Funcao Camac: F = 0 */
 pt.lin[1] = 0;

 libesc(bd, crate, pt.lin, 3);
 libler(bd, crate, px.lin, 3);

 for(i = 0; i < chave->canais_ativos; i++)
  {
  polaridade[i] = (int) (px.rep & 0x00000003L);
  px.rep >>= 2;
  }
}

int le_2264( int modo_transf)
{
 int ok = 1;
 /*
  Procedimento de leitura:  Modulo 2264
 */

 pt.lin[0] = 10;      /* Reseta LAM              */
```



```
 executa(pt, 3, crate);

 pt.lin[0] = 16;      /* Funcao Camac: F=16        */
 executa(pt, 3, crate);

 pt.lin[0] = 2;       /* Funcao Camac: F=2         */
 executa(pt, 3, crate);

 pt.lin[0] = (char) modo_transf;
 executa(pt, 3, crate);

 libdma (bd, 1);      /*   Liga DMA                */
 librd  (bd, z.ch, 65535);
 wprintf (wndtr,"\n %u (iberr = %2d, ibsta = %4xh)", ibcnt, iberr, ibsta);

 if (ibcnt == 3)
  ok = 0;

 pt.lin[0] = 10;      /* Reseta LAM              */
 executa(pt, 3, crate);
 libdma(bd, 0);       /* E/S programada          */

 return (ok);

}

/* Fim módulo tc_2264.c */

/* Inicio módulo tc_8210.c */
/* Leitura CAMAC via interface GPIB */
/* Estudo do modulo 8210 */

#include <stdio.h>
#include "lgpib.h"
#include "tc.h"
#include "g8901a.h"
#include "qwindow.h"

extern union matran  z, bf;
extern union PalavraComunicacao pt;
extern WINDOW *wndtr;

extern int bd, crate, estacao;
extern struct disp *chave;
extern struct complex_time *aj_tempo, *aj_ref, *aj;
extern float ref_tempo;

extern char nomearq [] [ 24];
extern char memoria [] [240];

int polaridade[8];

int     le_8210(int modo_transf);
void    mod_8210(int, char *);
```

```
void pos_chaves_8210(void);


void mod_8210(int num_canais, char *radical)
{
int i, modo_transf;

 pt.lin[2] = (char) estacao; /*  Comando Camac:  N */

/*  Preparacao inicial do modulo para leitura */
 pt.lin[0] = 26;          /* Comando Camac F = 26 */
 executa(pt, 3, crate);
 espera( 1.0);

 pos_chaves_8210();

 chave->canais_interesse = (chave->canais_ativos > num_canais) ?
     num_canais : chave->canais_ativos;

 if(chave->tempo = = (float) -1.0)  /* padrao externo de tempo */
  {
                if (aj_tempo->modo)          /* Comando pelo ck8501 */
          aj = aj_tempo;
                else
                        aj->per_1 = ref_tempo;    /* Comando por outro modulo */
                }
        else
                aj->per_1 = ref_tempo = chave->tempo;

 primeira_linha(chave->canais_ativos, 1024, memoria);
 nome_final(nomearq, radical, chave->canais_ativos);

 modo_transf = (chave->canais_ativos = = 1) ? FBT_16 : SBT_16;

 clear_window(wndtr);
 wcursor   (wndtr,  2, 0);
 wprintf   (wndtr, " Modulo: ");
 wcursor   (wndtr, 23, 0);
 wprintf   (wndtr, " Estacao: ");
 wcursor   (wndtr, 2, 2);
 wprintf   (wndtr, " Bytes transferidos: ");
 display_window(wndtr);
 wcursor(wndtr, 11, 0);
 wprintf(wndtr, chave->tipo_modulo);
 wcursor(wndtr, 34, 0);
 wprintf(wndtr, "%d", estacao);
 wcursor(wndtr, 2, 3);

 for (i = 0; i < chave->canais_interesse; i+ +)
  {
  pt.lin[1] = (char) i;
  if( le_8210(modo_transf) )
   grava_int(i, z.it, chave->num_pontos, nomearq[i], memoria[i]);
  else
```



```
    problemas(0, chave->tipo_modulo, crate, estacao);
   }
  hide_window(wndtr);
 }


void pos_chaves_8210()
{
union PalavraComunicacao px;
int i;

static unsigned int post_trigger_8210[8] = { 7, 6, 5, 4, 3, 2, 1, 0};
static float samp_period_8210 [8] =
 { (float) -1.0, (float) 100.0, (float) 40.0, (float) 20.0,
   (float) 10.0, (float)  4.0, (float)  2.0, (float)  1.0};
static unsigned int channels_8210    [4] = { 1, 2, 0, 4};


/*  Posicao das chaves no painel frontal do modulo  */

 /*  Configuracao  */
 pt.lin[0] = 1;  /*  Funcao Camac: F = 1 */
 pt.lin[1] = 0;  /* Comando Camac: A = 0 */

 libesc(bd, crate, (char  *)  pt.lin, 3);
 libler(bd, crate, (char  *)  px.lin, 3);

 chave->amostragens   = post_trigger_8210  [px.rep & 0x00000007L];
                        px.rep > > = 3;
 chave->tempo         = samp_period_8210   [px.rep & 0x00000007L];
                        px.rep > > = 3;
 chave->canais_ativos = channels_8210      [px.rep & 0x00000003L];
                        px.rep > > = 2;
 chave->flag        = (unsigned int)(px.rep & 0x00000001L);
                        px.rep > > = 1;
 chave->okay = 0;
 chave->num_pontos    = (unsigned int) 32768/chave->canais_ativos;

/*
 Polaridade dos canais:

  Modulo 8210 nao admite controle de polaridade. Offset e
    sempre zero
*/

 for(i = 0; i < chave->canais_ativos; i+ +)
  polaridade[i] = 3;
}



int le_8210( int modo_transf)
```

```
{
 int ok = 1;
  /*
   Procedimento de leitura:   Modulo 8210
  */

  pt.lin[0] = 10;        /*  Reseta LAM                  */
  executa(pt, 3, crate);

  pt.lin[0] = 16;        /*  Funcao Camac: F=16          */
  executa(pt, 3, crate);

  pt.lin[0] = 2;         /*  Funcao Camac: F=2           */
  executa(pt, 3, crate);

  pt.lin[0] = (char) modo_transf;
  executa(pt, 3, crate);

  libdma (bd, 1);        /*  Liga DMA                    */
  librd  (bd, z.ch, 65535);
  wprintf (wndtr,"\n %u (iberr = %2d, ibsta = %4xh)", ibcnt, iberr, ibsta);

  if (ibcnt = = 3)
    ok = 0;

  pt.lin[0] = 10;        /*  Reseta LAM                  */
  executa(pt, 3, crate);
  libdma(bd, 0);         /*  E/S programada              */

  return (ok);

}

/* Fim módulo tc_8210.c */

/* Inicio módulo tc_8212.c */
/*    Leitura CAMAC via interface GPIB      */
/*    Estudo do modulo Data Logger 8212     */

#include <stdio.h>
#include "igpib.h"
#include "tc.h"
#include "g8901a.h"
#include "qwindow.h"


extern union matran z, bf;
extern WINDOW *wndtr;

extern int ibsta, iberr, ibcnt;
extern int bd, crate, estacao;
extern struct disp *chave;
extern struct complex_time *aj_tempo, *aj_ref, *aj;
extern float ref_tempo;
```



```
extern char nomearq [] [ 24];
extern char memoria [] [240];

void  mod_8212(int, char *);
void prog_8212(void);
int   le_8212(char *buf);

int polaridade[32];
union PalavraComunicacao pt;



void mod_8212(int num_canais, char *radical)
{
int register *buff, j;
int i, ii;

 pt.lin[2] = (char) estacao;  /*  Comando Camac:  N */

 prog_8212();

 chave->canais_interesse = (chave->canais_ativos > num_canais) ?
        num_canais : chave->canais_ativos;

 if(chave->tempo = = (float) -1.0)  /* padrao externo de tempo */
  {
                if (aj_tempo->modo)          /* Comando pelo ck8501 */
          aj = aj_tempo;
                else
                        aj->per_1 = ref_tempo;    /* Comando por outro modulo */
                }
        else
 aj->per_1 = ref_tempo = chave->tempo;

 primeira_linha(chave->canais_interesse, 4096, memoria);

 nome_final(nomearq, radical, chave->canais_interesse);

 clear_window(wndtr);
 wcursor   (wndtr,  2, 0);
 wprintf   (wndtr, " Modulo: ");
 wcursor   (wndtr, 23, 0);
 wprintf   (wndtr, " Estacao: ");
 wcursor   (wndtr, 2, 2);
 wprintf   (wndtr, " Bytes transferidos: ");
 display_window(wndtr);
 wcursor(wndtr, 11, 0);
 wprintf(wndtr, chave->tipo_modulo);
 wcursor(wndtr, 34, 0);
 wprintf(wndtr, "%d", estacao);
 wcursor(wndtr, 2, 3);

 buff = bf.it;
```

```
 if(le_8212(z.ch))   /* Matriz z.ch contem os dados digitalizados */

 for (i = 0; i < chave->canais_interesse; i++)
  {
  ii = i-1;
  j = 0;
  while (j < chave->num_pontos)
   *buff++ = *(z.it + ii + chave->canais_ativos*(j++) );

  grava_int(i, (bf.it+i*chave->num_pontos), chave->num_pontos,
          nomearq[i], memoria[i]);
  }
 else
  problemas(0, chave->tipo_modulo, crate, estacao);

 hide_window(wndtr);
}



void prog_8212()
{
int i;
union PalavraComunicacao px;

static float samp_period_8212 [] =
{ (float)- 1.0, (float) 5000.0, (float)1000.0, (float) 500.0,
 (float)200.0, (float)  100.0, (float)  50.0, (float)  25.0};

static int post_trigger_8212 []  = {1024, 896, 768, 640, 512, 384, 256, 128};

static int channels_8212 []     = {4, 8, 16, 32};

/*  Decodificacao da Instrucao de trabalho  */

 pt.lin[0] = 0x03;    /* 8212 : F(03) */
 pt.lin[1] = 0;       /* Comando Camac: A = 0 */

 libesc(bd, crate, pt.lin, 3);
 libler(bd, crate, px.lin, 3);


 chave->canais_ativos = channels_8212     [px.rep & 0x00000003L];
                        px.rep >> = 2;
 chave->tempo         = samp_period_8212  [px.rep & 0x00000007L];
                        px.rep >> = 3;
 chave->amostragens   = post_trigger_8212  [px.rep & 0x00000003L];


 chave->flag        = 1;  /* Nao se aplica  */
 chave->okay        = 0;  /* Nao se aplica  */
 chave->num_pontos  = (unsigned int) 32768/chave->canais_ativos;
```



```
/*
 Polaridade dos canais:
  Modulo 8212 nao admite controle de polaridade. Offset e sempre zero.
*/

 for(i = 0; i < chave->canais_interesse; i++)
  polaridade[i] = 3;
}



int le_8212(char *buf)
{
int ok = 1;

 pt.lin[0] = 16;    /* Select channel f(16) */
 pt.lin[3] = 32;    /* leitura interlacada de toda a memoria */
 executa(pt, 4, crate);

 pt.lin[0] =  2;    /* Leitura sucessiva ate Q = 0 */
 executa(pt, 3, crate);

 pt.lin[0] = FBT_16; /* Leitura  do Modulo a maxima velocidade */
 executa(pt, 3, crate);

 libdma(bd, 1);       /* Liga transferencia DMA */
 librd (bd, buf, 65535);
 wprintf (wndtr,"\n %u (iberr = %2d, ibsta = %4xh)", ibcnt, iberr, ibsta);

 if (ibcnt = = 3)
  ok = 0;
 pt.lin[0] = 10;      /* Reseta LAM */
 executa(pt, 3, crate);

 libdma(bd, 0);       /* Desliga DMA */

 return (ok);
}
/* Fim módulo tc_8212.c */

/* Inicio módulo tc_8837.c */
/*    Leitura CAMAC via interface GPIB       */
/*    Estudo do modulo 8210                  */

#include <stdio.h>
#include "lgpib.h"
#include "tc.h"
#include "g8901a.h"
#include "qwindow.h"


extern union matran  z, bf;
```

```
extern union PalavraComunicacao pt;
extern WINDOW *wndtr;

extern int bd, crate, estacao;
extern struct disp *chave;
extern float ref_tempo;
extern int polaridade[];

extern struct complex_time *aj_tempo, *aj_ref, *aj;

extern char nomearq [] [ 24];
extern char memoria [] [240];

int   le_8837(char *buffer);
void  mod_8837(int, char *);
void prog_8837(void);


union PalavraComunicacao pt;

void mod_8837(int num_canais, char *radical)
{
register int *buf, j;

 pt.lin[2] = (char) estacao;   /*  Comando Camac:  N */

/*  Preparacao inicial do modulo para leitura */
 pt.lin[0] = 26;   /* Comando Camac F = 26 */
 executa(pt, 3, crate);
 espera( 1.0);

 prog_8837();

 num_canais = 1;
 chave->canais_interesse = 1;

 if(chave->tempo = = (float) -1.0)  /* padrao externo de tempo */
  {
                if (aj_tempo->modo)          /* Comando pelo ck8501 */
          aj = aj_tempo;
                else
                        aj->per_1 = ref_tempo;    /* Comando por outro modulo */
                }
        else
  aj->per_1 = ref_tempo = chave->tempo;

 primeira_linha(1, 256, memoria);
 nome_final(nomearq, radical, 1);

 clear_window(wndtr);
 wcursor   (wndtr,  2, 0);
 wprintf   (wndtr, " Modulo: ");
 wcursor   (wndtr, 23, 0);
 wprintf   (wndtr, " Estacao: ");
```



```
 wcursor   (wndtr, 2, 2);
 wprintf   (wndtr, " Bytes transferidos: ");
 display_window(wndtr);
 wcursor(wndtr, 11, 0);
 wprintf(wndtr, chave->tipo_modulo);
 wcursor(wndtr, 34, 0);
 wprintf(wndtr, "%d", estacao);
 wcursor(wndtr, 2, 3);

 if( le_8837(z.ch) )
  {
  j = 0;
  buf = bf.it;
  while (j < 8192)
   *buf++ = (int) *(z.ch + j++);

  grava_int (0, bf.it, 8192, nomearq[0], memoria[0]);
  }
 else
  problemas( 1, chave->tipo_modulo, crate, estacao);

 hide_window(wndtr);
}


void prog_8837()
{

static float samp_period_8837[] =
       {(float)0.03125, (float)0.0625, (float)0.125, (float)0.250,
        (float)0.500,   (float)1.0,    (float)2.0,   (float)-1.0};

static int post_trigger_8837 [] = { 8, 7, 6, 5, 4, 3, 2, 1};

static int mem_perm_8837    [] = {1024,2048,3072,4096,5120,6144,7168,8192};

/*  Decodificacao da instrucao dada ao modulo  */

 pt.lin[0] = 0x00;          /* 8837f : F(00) */
 libesc(bd, crate, pt.lin, 3);
 libler(bd, crate, pt.lin, 3);

 chave->amostragens   = post_trigger_8837 [pt.rep & 0x00000007L];
                        pt.rep > > = 4;
 chave->tempo         = samp_period_8837  [pt.rep & 0x00000007L];
                        pt.rep > > = 4;
     chave->num_pontos    = mem_perm_8837    [pt.rep & 0x00000007L];

 chave->canais_ativos = 1; /* Um so canal */

 chave->flag        = 1;    /* Nao se aplica  */

 chave->okay        = 0; /* Nao se aplica  */
```

```
 polaridade[0] = 3;          /* Bipolar */

}


int le_8837(char *buffer)
{
int ok = 1;
 /* Procedimento de leitura */

 pt.lin[1] = 0;
 pt.lin[2] = (char) estacao;

 pt.lin[0] = 10;  /* Reseta LAM */
 executa(pt, 3, crate);

 pt.lin[0] = 17;    /* Funcao Camac: F=17 */
 executa(pt, 3, crate);

 pt.lin[0] = 2;     /* Funcao Camac: F=2 */
 executa(pt, 3, crate);

 pt.lin[0] = FBT_08;  /* Transf. rapida 8 bits */
 executa(pt, 3, crate);

 libdma (bd, 1);       /* Liga DMA     */
 librd  (bd, buffer, 8193);
 wprintf (wndtr,"\n %u (iberr = %d, ibsta = %xh)", ibcnt, iberr, ibsta);

 if( ibcnt = = 3)
   ok = 0;

 pt.lin[0] = 10;          /* Reseta LAM   */
 executa(pt, 3, crate);
 libdma(bd, 0);              /* Volta a E/S programada    */

 return (ok);
}
/* Fim módulo tc_8837.c */
/* Inicio módulo tc (make file para o programa TC com Microsoft C 5.1) */
#
# Tc - Programa para transferencia de arquivos
#

LNC= tc+tc_trans+tc_geral+tc_nome+tc_2264+tc_8210+tc_8212+tc_8837

.c.obj:
  cl -c -W3 -Gs -Ze -Zl -AL -V"Lab.Fis.Plasmas, A.N.Fagundes, 1989" $*.c

tc.obj   : tc.c    tc.h

tc_2264.obj : tc_2264.c  tc.h

tc_8210.obj : tc_8210.c  tc.h
```



```
tc_8212.obj : tc_8212.c  tc.h

tc_8837.obj : tc_8837.c  tc.h

tc_trans.obj: tc_trans.c tc.h

tc_nome.obj : tc_nome.c  tc.h

tc_geral.obj: tc_geral.c tc.h


tc.exe  : tc.obj  tc_trans.obj tc_2264.obj  tc_8210.obj \
            tc_8212.obj tc_8837.obj  tc_geral.obj tc_nome.obj
    link /CO /M $(LNC), trg,,gpiblib+windllib;
        exepack trg.exe c:\util\tc.exe
```

Apêndice 7

O programa vc

O Programa vc foi escrito com o compilador Turbo-C, versão 2.0

```
/* Início módulo vc.h */
#include <stdio.h>

typedef unsigned int unsint;
typedef unsigned char byte;

struct Cabecalho
 {
 char data [25];
 char polar[4];
 char tipo_modulo[8];
 unsigned int canais_interesse;
 unsigned int amostragens;
 unsigned int num_pontos;
 unsigned int okay;
 unsigned int canal;
 unsigned int resol;
 unsigned long int shot;
        };


struct ComplexTime
 {
 unsigned int modo;
 unsigned int range;
 unsigned long int numpul_2;
 unsigned long int numpul_3;
 float per_1;
 float per_2;
 float per_3;
        };

typedef struct QuadroGrafico
        {
        char nomearq[25];                /* Arquivo contendo registro   */
        char grandeza_fisica[13];     /* Grandeza fisica registrada */
        char abreviacao[13];          /* Indicacao das ordenadas   */
        char comentario[35];
        unsint xini, yini, xfin, yfin; /* Espaco total alocado ao grafico  */
        float fat_esc;              /* Fator de escala - ordenadas */
        struct ComplexTime ct;       /* Informacoes de tempo - abscissas */
        struct Cabecalho cb;         /* Informacoes gerais  do pulso */
        } QUADRO;

#define NUM_MAX_CURVAS    15                    /* E 15 (por enquanto) num.max.curvas */
#define PORTRAIT                        0     /* Vai assim ou asssado para o papel ? */
#define LANDSCAPE                       1
```



```
#define CRT_I                                   4     /* Densidade de impressao - Vide manual
*/
#define PLOTTER                                 5     /*        da impressora        */
#define CRT_II      6     /*        Olivia.          */
#define UP                          72
#define DOWN                                    80
#define LEFT                                    75
#define RIGHT                                   77
#define ENTER                                   13

extern unsint maxx, maxy;
extern unsint npts, eixoX[], eixoY[], *dados;
extern char radical[], nome[];

int  abandona_arqs(void);
void extremos    (QUADRO *qd, int num_canais, int i);
int  gprintf     (int *xloc, int *yloc, char *fmt, ...);
void imprime_tela (int mode,int direction);
void intervalo   (QUADRO *qd, unsint *inicio, unsint *incx);
int  le_lista    (unsigned long *pulso, char *data);
void moldura     (QUADRO *qd, int i);
void MolduraMaior (unsigned long pulso, char *data);
int  monta_quadro (QUADRO *qd);
void plots       (unsint color);
int  principio   (void);
int  ProximaAcao  (char **opcoes, char *lista, int numop);
int  pro_quadro   (QUADRO *qd);
int  salva_arqs   (void);
void termino     (char *mensagem);

/* Fim módulo vc.h */

/* Início módulo vc.c */
#include <stdlib.h>
#include <conio.h>
#include <dir.h>
#include <graphics.h>
#include <string.h>
#include "c:\comum\vc\vc.h"

char *opcoes[] = {"Salva", "Abandona", "Imprime", "Examina", "Fim"};

QUADRO  qd[NUM_MAX_CURVAS];

unsint npts, eixoX[638], eixoY[638], *dados;
char nome [MAXPATH], radical [MAXPATH];

unsigned long int pulso;
char data[25];
int num_canais;
```

```
void main(int argc, char *argv[])
{
int i = 0, completo = 0;

        if ( !principio() )                                     /* Inicializa
tarefa grafica    */
                termino("Sem driver grafico\n");      /* Desliga, se houver erro.  */

        if (argc < 2)
                        termino(" Nao foi especificada uma lista grafica!\n");

        strcpy(nome, argv[1]);
        num_canais = le_lista(&pulso, data);

        MolduraMaior(pulso, data);

        for (i = 0; i < num_canais; i++)
                {
                if(!pro_quadro(&qd[i]))
                        break;
                extremos (&qd[i], num_canais, i);
                moldura  (&qd[i], i);
                if(!monta_quadro(&qd[i]))
                        break;
                plots( EGA_WHITE);
                }

        do
                {
                switch( ProximaAcao(opcoes, "SAIEF", 5) )
                        {
                        case 0  :       salva_arqs();                   /* Bom pulso! Guardamos
resultados      */
                                                        completo++;
                                                        break;
                        case 1  :       abandona_arqs();  /* Mau pulso! Desprezamos        */
                                                        completo++;
                                                        break;

                        case 2  :  imprime_tela(CRT_I, PORTRAIT);  /* Copia impressa da tela*/
                                                        break;

                        default :  completo++;
                        }
                } while ( !completo);

                termino(" Programa terminado!\n");

}

/* Fim módulo vc.c */

/* Início módulo vc_geral.c */
```



```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <graphics.h>
#include <dir.h>
#include "c:\comum\vc\vc.h"

unsint maxx, maxy;

extern QUADRO qd[];

int principio()
                                        /*      Inicializa modo grafico VGAHI: 640x480 */
{
int graphdriver   = VGA;
int graphmode     = VGAHI;
char *pathtodriver = "c:\tc\bgi";


        if (registerbgidriver(EGAVGA_driver) < 0)
                return (0);

        initgraph (&graphdriver, &graphmode, pathtodriver);     /* Tela  */
        maxx = getmaxx();               /* Estabelece maximos graficos */
        maxy = getmaxy();               /* conforme modo utilizado     */
        setbkcolor(LIGHTBLUE);
        return (1);
}


void termino(char *mensagem)
/*
        Fecha o programa, voltando ao modo texto e
        restabelecendo as cores iniciais.
*/
{
        setbkcolor(BLACK);
        closegraph();
        printf( mensagem );
        exit (0);
}

int le_lista(unsigned long int *pulso, char *data)
/*      Le pulso, data e nome dos arquivos      */
{
FILE *fi;
int i, j;
char linha[80];

                                                        /* Identificacao da lista grafica  */
char disco  [MAXDRIVE],         /* Disco onde se localiza                          */
            diretoria[MAXDIR],          /* Diretoria               */
            arquivo [MAXFILE],          /* Nome                    */
```

```
        familia  [MAXEXT];          /* Extensao          */

    fnsplit(nome, disco, diretoria, arquivo, familia);

    strcpy (radical, disco);
    strcat (radical, diretoria);

    strcpy (nome, arquivo);

    if(     (strlen(radical) = = 0) )
            {
            getcwd(radical, MAXPATH);
            strcat(radical, "\\");
            }

    strcpy(linha, radical);
    strcat(linha, arquivo);
    strcat(linha, ".GRF");

    if ( (fl = fopen(linha, "r")) = = NULL)
            termino ("Lista para grafico nao foi encontrada\n");

    fgets (linha, 80, fl);                  /* 1a linha: contem pulso      e data  */
    sscanf(linha, " %U %s", pulso, data);


    i = 0;
    fgets (linha, 80, fl);      /* Linhas conscs. contem arquivos */
    do
            {
            j = strlen(linha);
            linha[j-1] = '\0';
            strcpy(qd[i].nomearq, radical);
            strcat(qd[i].nomearq,  linha);
            i++;
            fgets (linha, 80, fl);
            } while ( (!feof (fl)) && (i < NUM_MAX_CURVAS) );

    fclose(fl);
    return (i);
}


void imprime_tela(int mode,int direction)
{
char m;
int i, j, k, msb, lsb;

    fprintf (stdprn, "\x1B\x41%c", 7);

    switch  (direction)
            {
            case PORTRAIT :         lsb = maxx & 0x00ff;  /* maxx e maxy calculados */
                                                          msb = maxx > > 8;
```



```
/* em MolduraMaior()  */

13)/8; j+ +)

"\x1B*%c%c%c", mode, lsb, msb);

maxx; i+ +)


= 0; k < 8; k+ +)


< <= 1;

f (getpixel (i, j*8 + k) ) m+ +;

dprn, "%c", m);

dprn, "\x0D\x0A");

                    case LANDSCAPE: lsb = (maxy - 13) & 0x00ff;

8;

+ = 8)

"\x1B*%c%c%c", mode, lsb, msb);

13); i > = 0; i--)


= 0; k < 8; k+ +)


< <= 1;

f (getpixel (j + k, i) ) m+ +;

dprn, "%c", m);

"\x0D\x0A");

for (j = 0; j < = (maxy -

        {
        fprintf (stdprn,

        for (i = 0; i <

                {
                m = 0;
                for ( k

                        {
                        m

                        i

                        }
                fprintf(st

                }
                fprintf(st

        }
break;

msb = (maxy - 13) > >

for (j = 0; j < = maxx; j

        {
        fprintf (stdprn,

        for (i = (maxy -

                {
                m = 0;
                for ( k

                        {
                        m

                        i

                        }
                fprintf(st

                }
        fprintf(stdprn,

        }
break;
```

```
                }
                fprintf(stdprn, "\f");
}

/* Fim módulo vc_geral.c */

/* Início módulo vc_cpdl.c */
/*
*
*               Copia dos arquivos .CHR do disco virtual d: para o diretorio
*                       C:\DADOS\  do disco fixo. Representa a acao de salvar os re
*                       gistros do disparo.
*       Origem: Adaptacao do programa existente VCOPY, em
*                                                       "Turbo C - Dos Utilities, R. Alonso, p.
127.
*/

#include <stdio.h>
#include <dir.h>
#include <alloc.h>
#include <string.h>
#include <io.h>
#include "c:\comum\vc\vc.h"

int bufcopy (char *fromname, char *toname);

/*      Variaveis globais       */
char newname  [MAXPATH], oldname  [MAXPATH];
int  done;
unsigned char *buffer;
struct ffblk dta;


int salva_arqs()
{
char lista[MAXPATH];


        if( (buffer = (unsigned char *) calloc (32767, 1)) = = NULL)
                {
                printf( " Erro na alocacao de memoria ");
                return(0);
                }

        if( strlen(nome) ! = 0)
                {
                strcpy(lista, radical);
                strcat(lista, nome);
                strcat(lista, "*.*");
                }

        done = findfirst(lista, &dta, 47);
```



```
        if( strcmp(strupr(radical), "C:\\DADOS\\") = = NULL )
                done = 1;

        while (!done)
                {
                strcpy(oldname, radical);
                strcat(oldname, dta.ff_name);
                strcpy(newname, "C:\\DADOS\\");
                strcat(newname, dta.ff_name);

                if( bufcopy(oldname, newname) = = 0)
                        {
                        printf ("Erro: a copia nao foi feita\n");
                        }
                        done = findnext (&dta);
                }
        return (1);
}

int bufcopy (char *fromname, char *toname)
{
long numread, filepos;
unsigned numtoget, size;
FILE *fromfile, *tofile;
int inhandle, outhandle;

        fromfile = fopen(fromname, "r");
        tofile  = fopen( toname, "w");

        if( (tofile = = NULL) | | (fromfile = = NULL) )
                return (0);

        inhandle  = fileno(fromfile);
        outhandle = fileno( tofile);

        size = (unsigned) filelength (inhandle);

        numtoget = (size < 32767) ? size : 32767;

        numread = 0;
        do
                {
                filepos = ftell (fromfile);
                fseek( fromfile, numread, 0);
                _read(inhandle, buffer, numtoget);

                numread + = numtoget;
                fseek(tofile, filepos, 0);
                _write(outhandle, buffer, numtoget);

                if ( (numread + numtoget) > size)
                        numtoget = size - (unsigned) numread;
                } while (numtoget ! = 0);
```

```
        fclose(fromfile);
        fclose( tofile);

        unlink (fromname);

        return (1);
}


/*
*
*               Delecao dos arquivos .CHR do disco virtual d: Constitui a
*               de limpeza do registro do disparo.
*       Origem: Modificacao do programa existente VCOPY, em
*                                                       "Turbo C - Dos Utilities, R. Alonso, p.
127.
*/


int abandona_arqs()
{
char lista[MAXPATH];

        if( strlen(nome) != 0)
                {
                strcpy(lista, radical);
                strcat(lista, nome);
                strcat(lista, "*.*");
                }

        if ( (done = findfirst( lista, &dta, 47)) != 0)
                        return (0);     /* Ha problemas; retorna com indicativo    */

        while (!done)       /* Segue em frente apagando arquivo a arquivo          */
                {
                strcpy(oldname, radical);
                strcat(oldname, dta.ff_name);

                unlink(oldname);

                done = findnext (&dta);
                }
        return (1);
}

/* Fim módulo vc_cpdl.c */

/* Início módulo vc_lrqs.c */
#include <string.h>
#include <stdlib.h>
#include "c:\comum\vc\vc.h"

extern unsint npts, eixoX[], eixoY[], *dados;
```



```
int pro_quadro (QUADRO *qd)
{
char a1[80];
FILE *fl;

        if ( (fl = fopen (qd->nomearq, "rb")) == NULL)
                {
                strcpy(a1, "Arquivo ");
                strcat(a1, qd->nomearq);
                strcat(a1, " nao pode ser aberto !");
                termino ( a1 );
                }
        fscanf(fl,
        "%s %U %u %*d %u %*d %s %s %u %*d %u %*d %s %f %f %U %U %s %s %f %s",
                                                qd->cb.data,
                                        &qd->cb.shot,
                                        &qd->cb.canais_interesse,
                                        &qd->cb.num_pontos,
                                                qd->cb.polar,
                                                qd->cb.tipo_modulo,
                                        &qd->cb.resol,
                                        &qd->ct.modo,
                                                a1,             /* Nao funciona de outro jeito!  */
                                        &qd->ct.per_2,
                                        &qd->ct.per_3,
                                        &qd->ct.numpul_2,
                                        &qd->ct.numpul_3,
                                                qd->grandeza_fisica,
                                                qd->abreviacao,
                                        &qd->fat_esc,
                                                qd->comentario);
        qd->ct.per_1 = (float) atof(a1);

        if( dados = (unsint *) malloc (qd->cb.num_pontos*sizeof(int)) ) == NULL)
                {
                printf("Nao ha %d bytes alocaveis!\n", qd->cb.num_pontos*2);
                return (0);
                }

        fseek(fl, 240L, SEEK_SET);
        fread((void *) dados, sizeof(int), qd->cb.num_pontos, fl);
        fclose(fl);

        return ( 1 );
}

/* Fim módulo vc_lqrs.c */

/* Início módulo vc_graf.c */
#include <string.h>
#include <stdlib.h>
```

```
#include <graphics.h>
#include <conio.h>
#include <bios.h>
#include <ctype.h>
#include "c:\comum\vc\vc.h"

void highlite(int x, int xop, int y, char *str, int flag);

void moldura(QUADRO *qd, int i)
/*
        Traca os eixos de cada grafico, com espessura maior que o normal.
*/
{
unsint j;
int xloc, yloc;

        setlinestyle(SOLID_LINE, 0, THICK_WIDTH);
        settextstyle(DEFAULT_FONT, HORIZ_DIR, 1);

        rectangle(qd->xini + 25, qd->yini + 15, qd->xfin - 3, qd->yfin - 18);
        j = strlen(qd->grandeza_fisica);

        xloc = qd->xfin-j*8-24;
        yloc = qd->yini + 2;
        gprintf(&xloc, &yloc, "%s %d", qd->grandeza_fisica, i);

        xloc = qd->xfin-80;
        yloc = qd->yfin-10;
        gprintf(&xloc, &yloc,"Tempo (ms)");

        settextstyle(DEFAULT_FONT, VERT_DIR, 1);

        j = strlen(qd->abreviacao);

        xloc = qd->xini + 10;
        yloc = qd->yini + j + 12;
        gprintf(&xloc, &yloc,"%s",qd->abreviacao);

}

void MolduraMaior(unsigned long pulso, char *data)
/*
        Delimita a area grafica da tela, imprimindo, no alto,
         numero do pulso e data; embaixo, as opcoes de acao.
*/
{
int xloc, yloc;

        setlinestyle(SOLID_LINE, 0, NORM_WIDTH);
        settextstyle(DEFAULT_FONT, HORIZ_DIR, 1);

        maxx = getmaxx();
        maxy = getmaxy();
```



```
        rectangle(0, 0, maxx, maxy);
        line (0,      12, maxx,      12);
        line (0, maxy - 13, maxx, maxy - 13);

/*      Identificacao do pulso    */
        xloc = 0;  yloc = 2;
        gprintf (&xloc, &yloc, " IFUSP - Lab.Fis.Plasmas   Pulso: %lu", pulso);

        xloc = maxx - 8*(strlen(data));   yloc = 2;
        gprintf (&xloc, &yloc, "%s", data);


}


/*
                GPRINTF: Used like PRINTF except the output is sent to the
                screen in graphics mode at the specified co-ordinate.

                Origem: Bgidemo.c - Program de demostracao grafica que acompanha

Turbo-C.
*/

int gprintf( int *xloc, int *yloc, char *fmt, ... )
{
 va_list  argptr;                /* Argument list pointer */
 char str[140];                  /* Buffer to build sting into        */
 int cnt;                        /* Result of SPRINTF for return */

 va_start( argptr, format );            /* Initialize va_ functions          */

 cnt = vsprintf( str, fmt, argptr ); /* prints string to buffer  */
 outtextxy( *xloc, *yloc, str );     /* Send string in graphics mode */

 va_end( argptr );                      /* Close va_ functions               */

 return( cnt );                  /* Return the conversion count  */

}


void plots(unsint color)
/*
        Coloca os pontos em cada grafico
*/
{
int i = 0;

        while (i++ < npts)
                putpixel(*(eixoX + i), *(eixoY + i), color);

}
```

```
int dx;

int ProximaAcao(char **opcoes, char *lista, int numop)
{
int  i, key, prev, curr;
int xopc[10], yopc;

        setlinestyle(SOLID_LINE, 0, NORM_WIDTH);
        settextstyle(DEFAULT_FONT, HORIZ_DIR, 1);

        dx = maxx/numop;

        yopc = maxy - 11;
        for( i = 0; i < numop; i++)
                {
                xopc[i] = 1 + i*dx + (dx - 8*strlen(*(opcoes + i)))/2;
                highlite(i*dx+1, *(xopc + i), yopc, *(opcoes + i), 0);
                }

                highlite(1, xopc[0], yopc, opcoes[0], 1);

                curr = 0;
        for (;;)
                {

                key = bioskey(0);                                /* Le e decodifica tecla
pressionada     */
                if ((key & 0x00ff) == 0)
                        key = (key & 0xff00) >> 8;
                else
                        key &= 0x00ff;

                switch (key)
                        {
                        case RIGHT : prev = curr;
                                                                 curr = (curr >= 0) ? (++curr
% numop) : numop-1;
                                                                 break;

                        case LEFT  : prev = curr;
                                                                 curr = (curr >  0) ? (--curr %
numop) : numop-1;
                                                                 break;

                        case ENTER : return(curr);

                        default :   for (i = 0; *(lista + i); i++)
                                                                         if( toupper(key)
== *(lista + i) )
                                                                                 {
                                                                                 prev =
curr;
                                                                                 curr = i;
                                                                                 break;
```



```
                                                                                 }
                                                                         break;
                        }
                highlite(prev*dx, xopc[prev], yopc, opcoes[prev], 0);
                highlite(curr*dx, xopc[curr], yopc, opcoes[curr], 1);
                }
}


void highlite(int x, int xop, int y, char *str, int flag)
{
        if (flag)
                setfillstyle (SOLID_FILL, LIGHTGRAY);
        else
                setfillstyle (SOLID_FILL, LIGHTBLUE);
        bar(x, y, x + dx, y + 10);
        outtextxy(xop, y+1, str);
}

/* Fim módulo vc_graf.c */

/* Início módulo vc_extr.c */
#include <alloc.h>
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>
#include "c:\comum\vc\vc.h"
#include "c:\comum\vc\vc_jan.d"

#define BITS_RESOL    qd->cb.resol

void extremos (QUADRO *qd, int num_canais, int i)
/*
Implementa escolha dos pontos extremos dos quadros, guardando
        valores em qd.
*/
{

static int *tab[15] =
        {g1, g2, g3, g4, g6, g6, g8, g8, g10, g10, g12, g12, g15, g15, g15};


        qd->xini = tab[num_canais - 1][4*i];
        qd->yini = tab[num_canais - 1][4*i + 1];
        qd->xfin = tab[num_canais - 1][4*i + 2];
        qd->yfin = tab[num_canais - 1][4*i + 3];

}

unsint deltax;


int monta_quadro (QUADRO *qd)
```

```
{
unsint j, k, deltay, upr, dwn, ift, rgt;
unsint inicio, incx;
float fesc;


        lft = qd->xini + 28;
        rgt = qd->xfin - 3;
        upr = qd->yini + 15;
        dwn = qd->yfin - 18;

        npts = deltax = rgt - lft;      /* Intervalo em x    */
        deltay = dwn - upr;             /* Intervalo em y */

        fesc = (float) deltay/BITS_RESOL;

        Intervalo(qd, &inicio, &incx);


        for (j = inicio, k = 0; k < deltax; j += incx, k++)
                {
                *(eixoY + k) = (unsint) (*(dados +j))*fesc + upr;
                *(eixoX + k) = k + lft;
                }

                free(dados);

        return (1);
}

void Intervalo(QUADRO *qd, unsint *inicio, unsint *incx)
{
char *interval;
float tin, tfn;
unsint j = 0, fim = 0, tmp = 0;

        interval = getenv("INTERVALO");         /* Le intervalo do environment     */

        if (interval == NULL)           /* Se nao existe, toma pts extremos */
                {
                tin = (float) 0.0;
                tfn = (float)50.0;
                }
        else                            /* Achou; decodifica valores       */
                {
                while (!isalpha(*(interval  + j)))
                        j++;
                sscanf((interval + j), "%f", &tin);

                j += 3;
                while (!isalpha(*(interval + j)))
                        j++;
                sscanf((interval + j), "%f", &tfn);
                }
```



```
        *inicio = (int) (1000.0 * tin/qd->ct.per_1);
        fim    = (int) (1000.0 * tfn/qd->ct.per_1);

        if(*inicio > fim)
                {
                tmp    = *inicio;
                *inicio = fim;
                fim    = tmp;
                }

        if ( (fim - *inicio) > qd->cb.num_pontos)
                {
                *inicio = 0;
                fim    = qd->cb.num_pontos;
                }

        *incx = (fim - *inicio)/deltax;
}

/* Fim módulo vc_extr.c */

/* Início módulo vc.prj */
c:\comum\vc\vc.c        (c:\comum\vc\vc.h)
c:\comum\vc\vc_geral.c  (c:\comum\vc\vc.h)
c:\comum\vc\vc_lrqs.c   (c:\comum\vc\vc.h)
c:\comum\vc\vc_graf.c   (c:\comum\vc\vc.h)
c:\comum\vc\vc_extr.c   (c:\comum\vc\vc.h)
c:\comum\vc\vc_cpdl.c   (c:\comum\vc\vc.h)


/* Fim módulo vc.prj */

/* Início módulo vc_jan.dat */

static xini [4] [8] =
    { { 10,  0,  0,  0,  0,  0,  0,  0},
      { 10, 10,  0,  0,  0,  0,  0,  0},
      { 10, 10, 335, 335,  0,  0,  0,  0},
      { 10, 10, 10, 10, 335, 335, 335, 335} };

static yini [4] [8] =
    { { 13,  0,  0,  0,  0,  0,  0,  0},
      { 13, 181,  0,  0,  0,  0,  0,  0},
      { 13, 181, 13, 181,  0,  0,  0,  0},
      { 13, 97, 181, 265, 13, 97, 181, 265} };

static xfin [4] [8] =
    { { 639,  0,  0,  0,  0,  0,  0,  0},
      { 639, 639,  0,  0,  0,  0,  0,  0},
      { 315, 315, 639, 639,  0,  0,  0,  0},
      { 315, 315, 315, 315, 639, 639, 639, 639} };

static yfin [4] [8] =
```

```
{ { 339,  0,  0,  0,  0,  0,  0,  0},
  { 171, 339,  0,  0,  0,  0,  0,  0},
  { 171, 339, 171, 339,  0,  0,  0,  0},
  {  87, 171, 255, 339,  87, 171, 255, 339} };


static int larg [4] = {629, 629, 305, 305};
static int  alt [4] = {326, 158, 158,  77};


/* Fim módulo vc_jan.dat */
```


Apêndice 8

Resumo operacional dos programas

Laboratorio de Física de Plasmas - IFUSP

Programa **ArmaCam**
ac

Abril de 1989

Orígem: Desenvolvido localmente.

Autor: A.N. Fagundes

Finalidade: Inicializar os crates CAMAC; interpretar o arquivo de plano de trabalho, programando os modulos de programação interna e finalmente armando todos os módulos que constam do plano de trabalho.

Formato: ac [-r] [<Arquivo contendo plano de trabalho>]

Comentários: Quando invocado, este programa estabelece o proptocolo GPIB e estão envia um sinal CAMAC "Send Clear and Initialize" F(35) a cada bastidor (2, presentemente). A seguir envia o sinal CAMAC "Stop Trigger" F(25) a cada estação de cada bastidor, estabelecendo as condições iniciais. Abre e interpreta o arquivo de plano de trabalho ("C:\DADOS\PLANO.TRB" é o arquivo default) programando os módulos de programação interna e armando a todos com o comando F(9). Se a opção "r" é ligada, o programa produz um relatório das características de programação de cada módulo componente da cadeia.

Nota: Deve ser aplicado antes de cada pulso cujos sinais devem ser guardados.

Laboratorio de Física de Plasmas - IFUSP

Programa **PlanoAcao**
pc

Abril de 1989

Orígem: Desenvolvido localmente.

Autor: A.N. Fagundes

Finalidade: Produzir o arquivo que relaciona os módulos e discriminação da tarefa de cada módulo na cadeia de aquisição de dados.

Formato: pc

Comentários: Através do uso abundante de janelas, com perguntas específicas a cada caso, e "help", obtido pressionando-se F1, o usuário é levado à construção de um arquivo que contém as informações necessárias à programação dos módulos com programação interna, informações adicionais que devem ser incorporadas ao registro de cada canal e de cada modulo e informação de quais gráficos devem retornar sob a forma gráfica. (O arquivo "MODULOS.HLP" deve ser acessível.) Qualquer tecla, de F2 a F10, termina o preenchimento de qualquer tela; "ESC" reinicializa a janela. O programa tem capacidade para reconhecer alguns (poucos) erros de programação dos módulos.

Nota: O programa pc é empregado uma única vez no processo de aquisição de dados.

Laboratorio de Física de Plasmas - IFUSP

Programa **ToCamac**
tc

Abril de 1989

Orígem: Desenvolvido localmente.

Autor: A.N. Fagundes

Finalidade: Proceder à transferência dos sinais guardados nas unidades de memoria associadas aos módulos digitalizadores que compõem a cadeia de aquisição de dados para o computador.

Formato: tc [<Arquivo contendo plano de trabalho>]

Comentários: O programa procura pelo arquivo de plano de trabalho e procede à transferência de cada modulo linha a linha. Caso seja invocado apenas por tc, o programa irá procurar pelo nome default "C:\DADOS\PLANO.TRB". Caso também não seja encontrado, abre-se uma janela para operação manual. (Apenas os módulos de programação externa podem ser lidos desta forma e não haverá inserção de detalhes sobre canal do módulo; operando desta forma o programa desliga-se com "ESC"). O nome produzido para o arquivo com sinais digitalizados é formado de letras e numeros que informam aproximadamente a data em que foi obtido, o numero do pulso e sua orígem. O numero do pulso é obtido do arquivo "C:\DADOS\CONTADOR.NUM". Um conjunto maior de informações está contido em 240 bytes iniciais do arquivo. O restante do arquivo corresponde ao sinal vindo do módulo, no formato inteiro (2 bytes por numero componente do registro). Aceita um parâmetro de "enviroment" CAMINHO, que define onde serão guardados os arquivos. (É muito útil operando-se com disco virtual, para maior velocidade; neste caso deve-se fazer SET CAMINHO=d:\.)

Nota: Este program produz um arquivo com as iniciais do pulso e terminação GRF que será utilizado pelo programa **vc** para construção da tela gráfica com os arquivos escolhidos em PLANO.TRB.

Laboratorio de Física de Plasmas - IFUSP

Programa **VeCurva**
vc

Julho de 1989

Orígem: Desenvolvido localmente.

Autor: A.N. Fagundes

Finalidade: Mostrar arquivos de dados escolhidos na forma gráfica.

Formato: vc [arquivo grafico<.GRF>]

Comentários: Na tela gráfica há opções na linha inferior destinadas a guardar o sinal (SALVA) ou apagar completamente os registros de um disparo (ABANDONA). Estas opções envolvem todos os registros, e não apenas aqueles mostrados pem gráficos. A opção IMPRIME produz uma impressão numa impressora compatível com impressoras EPSON e FIM termina o programa sem qualquer ação sobre os arquivos. A opção EXAMINA não esta implementada.

Nota: Os sinais a serem mostrados graficamente devem estar no mesmo diretorio que o arquivo que os relaciona, com terminação GRF.