UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

ESCOLA DE ENGENHARIA

DEPARTAMENTO DE ENGENHARIA ELÉTRICA

RELATÓRIO DE ESTÁGIO SUPERVISIONADO

NBN Automação Industrial

RENATO HÜBNER BARCELOS

2866/97-0

Porto Alegre, 12 de março de 2003.

SUMÁRIO

LISTA DE FIGURAS

RESUMO

Este relatório trata das atividades realizadas na disciplina ENG04497 – Estágio Supervisionado III pelo aluno Renato Hübner Barcelos durante o período de dezembro de 2002 a fevereiro de 2003. Estas atividades foram realizadas na NBN Automação Industrial, empresa que desenvolve equipamentos para automação de curtumes e indústrias de outros segmentos, sob a supervisão do Eng. André Luiz Tessele Nodari e sob a orientação do Prof. Tiarajú Vasconcellos Wagner.

As atividades realizadas consistem na atualização do software da CPU de um dos produtos da empresa – a base do Water Mixer – e na implementação de um programa monitor juntamente com este software para uso do *debugger* remoto *Lucifer* a fim de facilitar a depuração do mesmo.

1. INTRODUÇÃO

A disciplina de Estágio Supervisionado é necessária para um desenvolvimento acadêmico completo, além de possibilitar ao estudante um maior contato com o mercado de trabalho e uma maior experiência num ambiente profissional. Tendo sido admitido como estagiário na NBN Automação Industrial em dezembro de 2002, o estagiário pôde desenvolver algumas atividades inseridas no contexto profissional e acadêmico de um Engenheiro Eletricista, as quais são faladas e tratadas neste relatório.

A NBN Automação Industrial é uma indústria especializada na produção de soluções visando a automação de curtumes, além de desenvolver tecnologias voltadas para a dosagem automatizada de produtos químicos, o que atende também aos segmentos têxtil, químico, polpa e papel.

As atividades do estagiário na NBN foram realizadas sobre a CPU da base de um dos produtos da empresa, o Water Mixer, que é um dosador de água com temperatura controlada para fulões e outros equipamentos. Estas atividades incluíram a atualização do software e a implementação de um programa monitor para uso do *debugger* remoto *Lucifer* a fim de facilitar a depuração do mesmo.

O relatório contém inicialmente a apresentação da empresa e de seus produtos. Na seqüência, é fornecido um conjunto de informações preliminares indispensáveis ao entendimento das atividades realizadas, sendo estas explicadas a seguir. Por último, são expostas as conclusões em cima do trabalho feito.

2. APRESENTAÇÃO DA EMPRESA

2.1. HISTÓRICO

Fundada em 1986, a NBN Automação Industrial de Porto Alegre - Brasil, iniciou sendo uma indústria especializada na produção de soluções visando a automação de curtumes. Hoje a NBN conta com uma equipe de engenheiros que cria e implementa soluções em automação para indústrias de diversos segmentos. A NBN se destaca pela preocupação com a preservação do meio ambiente – em 1999 apresentou ao mercado um sistema que permite o reciclo parcial da água utilizada no processo de curtimento, diminuindo o problema da geração de resíduos líquidos nos curtumes.

A busca de espaços no mercado internacional tem sido uma constante para a NBN, que tem participado de feiras internacionais desde 1996, com o objetivo de aumentar a participação das vendas externas e manter-se atualizada com novas tecnologias e tendências de mercado. Hoje possui clientes no Brasil, Argentina, Uruguai, Paraguai, México e Canadá totalizando mais de 130 instalações.

Ao longo destes anos foi desenvolvida uma tecnologia voltada a dosagem automatizada de produtos químicos para curtumes. Em 1998, foi iniciada a adequação desta tecnologia para atender os segmentos têxtil, químico, polpa e papel.

Atualmente são fabricados sistemas de dosagem de produtos químicos por tecnologias gravimétricas (peso) e volumétricas. Na linha de curtume, são produzidos sistemas de automação de fulões e sistemas de medição e classificação de couros.

Razão social: NBN Automação Industrial Ltda
CNPJ: 91.230.326/0001-60
Endereço: Avenida Sertório, 3133
Bairro: Santa Maria Gorete
Cidade: Porto Alegre
Estado: RS
Fone: (51) 33611026
Ramo de atividade: Automação Industrial
Nome e Cargo do Representante: André Luiz Tessele Nodari – Diretor

2.2. SISTEMAS E PRODUTOS

2.2.1. Automação de fulões

A produção do couro necessita de controles avançados para competir no mercado mundial altamente competitivo. Para tanto, a NBN dispõe de equipamentos e sistemas que controlam a produção na fase molhada.

O Smarttan automatiza os processos de caleiro, curtimento e recurtimento de um curtume. O sistema controla tempos de rotação dos fulões (Restan), adição automática de água (Water Mixer), adição automática de produtos químicos (Chemitan/Chemibatch, Reciclo, DAP/Volubatch) e protocola automaticamente todas intervenções manuais do operador. O software supervisor Smarttan ST para WIN98 gerencia a interligação dos diversos equipamentos citados. O software ST Lab controla o desenvolvimento de receitas do laboratório e está perfeitamente interligado ao Smarttan ST.

- *Restan*: A Restan é um controlador de fulão que, em conjunto com o programa de computador Smarttan SW (ST SW), forma o núcleo do sistema de automação do curtimento de couros Smarttan. Performa três funções básicas: acionamento e monitoração do fulão, interface com o operador e comunicação com o programa Smarttan SW.
- *Chemitan / Chemibatch*: O Chemitan é uma balança misturadora dosadora destinada a pesagem, mistura, diluição e dosagem de produtos químicos de forma automática até um número máximo de 24 produtos. Até 480 produtos adicionais podem ser dosados de maneira semi-automática, sob supervisão do controlador de pesagem.
- *Water Mixer*: O Water Mixer (WM) é um dosador de água com temperatura controlada para fulões ou outros equipamentos. Desenvolvido originalmente para a indústria coureira, o WM é capaz de regular a temperatura da água em um tubo com precisão de ±1 grau centígrado.
- *Reciclo*: O Reciclo é um sistema completo para o manejo do banho de caleiro/cromo reciclado. O ciclo de filtragem, realizado durante a depilação (caso do caleiro), retira o cabelo do banho resultando em economia de produtos químicos. O reuso do banho armazenado economiza produtos químicos, água e reduz a carga no sistema de efluentes.
- *Dap / Volubatch*: O DAP é um equipamento capaz de dosar volumetricamente grandes quantidades de produtos em tanques, tambores, tinas, fulões ou qualquer recipiente desejado. Destina-se basicamente a eliminar tempos de espera causados por dosagens demoradas como, por exemplo, ácido sulfúrico diluído em reatores de curtimento de couro (fulões).

2.2.2. Medição de couros

A NBN fabrica com tecnologia própria uma completa linha de equipamentos, programas e sistemas para medição de área, pesagem, classificação, paletização, marcação e etiquetação (código de barras) de couros desde Wet Blue até acabados. A medição simultânea de couros em uma mesma máquina é outra exclusividade da NBN. Esta função é utilizada para medir meios existentes simultaneamente ou medir e rachar um couro inteiro em uma única operação (com disco de corte integrado à máquina).

As máquinas e sistemas são robustas, inteligentes, de fácil uso e projetadas especificamente para o meio agressivo do curtume. Embora fáceis de operar, apresentam inteligência e conectividade capazes de as integrarem nos mais variados processos de classificação, fornecendo dados reais e atuais ao sistema de gestão do curtume.

- *Medidora de couro acabado classe Columbia ou Bismark*: Medidora com esteira convencional (cordas de nylon), tecnologia de leitura transmissiva, construída em aço carbono, específica para couros acabados.
- *Medidora de couro Wetblue - Infrablue, Nimitz e Enterprise*: Régua óptica reflexiva para medição de área de couros Wet Blue. É projetada para ser utilizada em conjunto com o computador de medição MDPlus / MDSort.
- *Sistema Logbar / Autobar*: O posto de etiquetamento de unidade logística de Logbar128 e o dispositivo de etiquetamento automático de couros Autobar128 constituem uma solução para a identificação de couros e suas unidades logísticas (pacotes, fardos, caixas ou paletes). Possibilitam etiquetar automaticamente cada couro medido e emitir etiquetas (para aplicação manual) para cada unidade logística completada, contendo toda informação do conteúdo da última.
- *Sistema Mdmon*: O Mdmon é um programa para configuração e monitoração da operação de computadores de medição de área Mdplus.

2.2.3. Mistura e dosagem de produtos químicos

- *Misturadores de água - Water Mixer*: ver item 2.2.1
- *Dosadores Volumétricos – Dap Volubatch*: ver item 2.2.1
- *Dosadores Gravimétricos – Chemitan e Accubatch*: O Accubatch é um equipamento destinado a pesagem, mistura, diluição e dosagem de produtos químicos que requeiram precisão de 1g. É, portanto ideal para aplicações com corantes. O Accubatch pode ser montado como fornecedor para o Chemitan.

3. INFORMAÇÕES PRELIMINARES

3.1. DESCRIÇÃO DO WATER MIXER

O **WATER MIXER** (WM) é um dosador de água com temperatura controlada para fulões ou outros equipamentos. Desenvolvido originalmente para a indústria coureira, o WM é capaz de regular a temperatura da água em um tubo (vazão de 1300 litros por minuto) com precisão de ±1 grau centígrado. Deste tubo a água é direcionada a um fulão (de um máximo de 24 fulões) e o volume é medido com precisão de 10 litros. Dosa-se então um dado volume de água (com precisão de ±10 litros) com temperatura desejada (precisão ±1 grau centígrado) a um fulão selecionado dentre 24 fulões possíveis.

A regulagem da temperatura da água é realizada misturando água proveniente de dois tanques (um quente e um frio) de maneira a obter qualquer temperatura intermediária desejada. O WM utiliza a técnica de processamento digital de sinais (DSP) para realizar a regulagem em um tempo muito curto - tipicamente 15 segundos. 0 tempo curto de regulagem traduz-se em substancial economia de energia devido ao fato da água misturada retornar ao tanque de água quente (recirculação) enquanto não atinge a temperatura desejada.

A alta velocidade de regulagem e a alta vazão permitem que o WM dose 2000 litros de água em menos de três minutos. O sistema tradicional de aquecimento de água na tina de preparação leva cerca de 30 minutos para os mesmos 2000 litros. Uma economia de aproximadamente meia hora por dosagem. Isto, dependendo do processo, poder resultar em economia de horas por dia, aumentando a produção instantaneamente.

O WM foi projetado para operar em um ambiente hostil como o de um curtume. É acondicionado em caixa plástica a prova d'água, possui teclas seladas (membrana) no painel do operador e tem entradas e saídas isoladas para rejeição de interferência elétrica.

A utilização de processamento digital de sinais (DSP) para o controle - ao invés de pneumática ou eletrônica analógica - elimina a necessidade de ajustes e a deriva de calibração com o tempo e temperatura. Também permite a utilização de uma bateria de

mistura de água simples, robusta e de baixo custo ao invés das complexas, frágeis e caras válvulas de controle dos sistemas pneumáticos ou eletro-analógicos.

Figura 1: Base do Water Mixer instalada

<u>As principais características do WM são:</u>

- Atende até 24 fulões.
- Precisa regulagem de temperatura (± 1 graus C) melhora a qualidade do couro.
- Precisa regulagem de temperatura (± 1 graus C) melhora a qualidade do couro.
- Precisa adição de volume de água (± 10 litros) diminui os gastos com água e tratamento de efluentes.
- Tempo de regulagem ultra rápido (10 s típico) gera economia de energia entre 20% e 30% (típica).
- Alta vazão (1300 litros por minuto) aumenta a produtividade (enche os fulões rapidamente).
- Não necessita regulagens, não perde precisão com o tempo.
- Bateria de mistura simples e robusta requer manutenção mínima.
- Operação simples e intuitiva requer treinamento mínimo.
- Conexão em computador supervisor (para integrar com um sistema de dosagem de produtos químicos).
- Cinco temperaturas e volumes pré programados **por fulão.**

- Temperaturas e volumes quaisquer programáveis.
- Caixa plástica a prova d'água (classe IP54).
- Autodiagnóstico extensivo.
- Alarmes para situações de exceção.
- Instalação e manutenção extremamente simples.

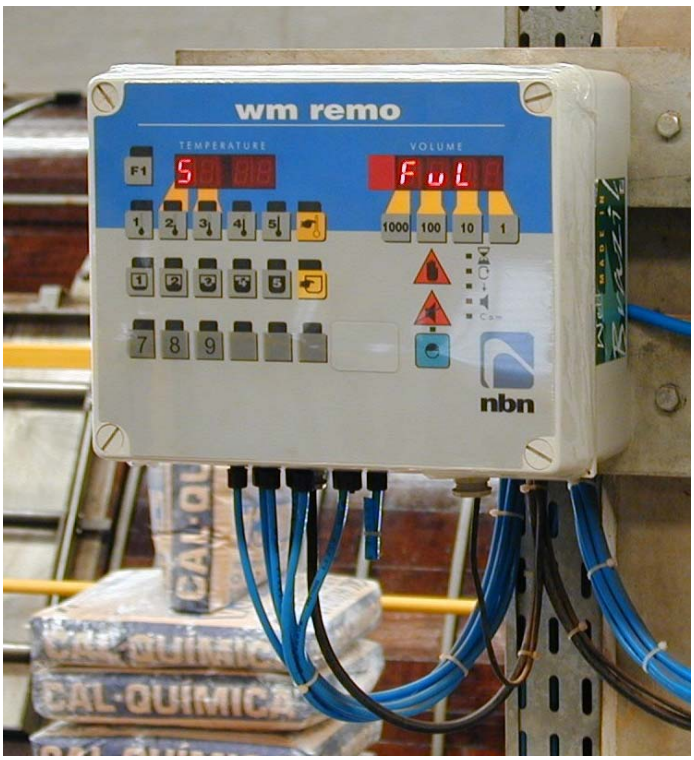

Figura 2: Remota do Water Mixer

Internamente a base do Water Mixer é controlado por um sistema com um microprocessador da família Z80 da Zilog ao qual estão ligados periféricos como o Z80-PIO (*Parallel Input/Output*), o Z80-CTC (*Counter/Timer Channels*), o Z80-SIO (*Serial Input/Output*), além de *devices* de memória RAM e ROM.

3.2. ZILOG Z80 CPU

A CPU Z80 da Zilog é um microprocessador de quarta geração com grande poder computacional e amplamente utilizado no mercado. As velocidades oferecidas de 6 a 20 MHz servem a um vasto número de aplicações. Os registradores internos incluem dois grupos de seis registradores de uso geral que podem ser usados individualmente como registradores de 8 bits ou aos pares como registradores de 16 bits. Além disso, a CPU Z80 possui dois grupos de acumuladores e *flags*.

A CPU Z80 também contem um ponteiro de pilha (*stack pointer*), um contador de programa (*program counter*), dois registradores de índice (*index registers*), um registrador de REFRESH e um de INTERRUPT. A CPU é suportada por uma família extensa de controladores periféricos.

A Figura 3 ilustra a arquitetura interna e os principais elementos da CPU Z80.

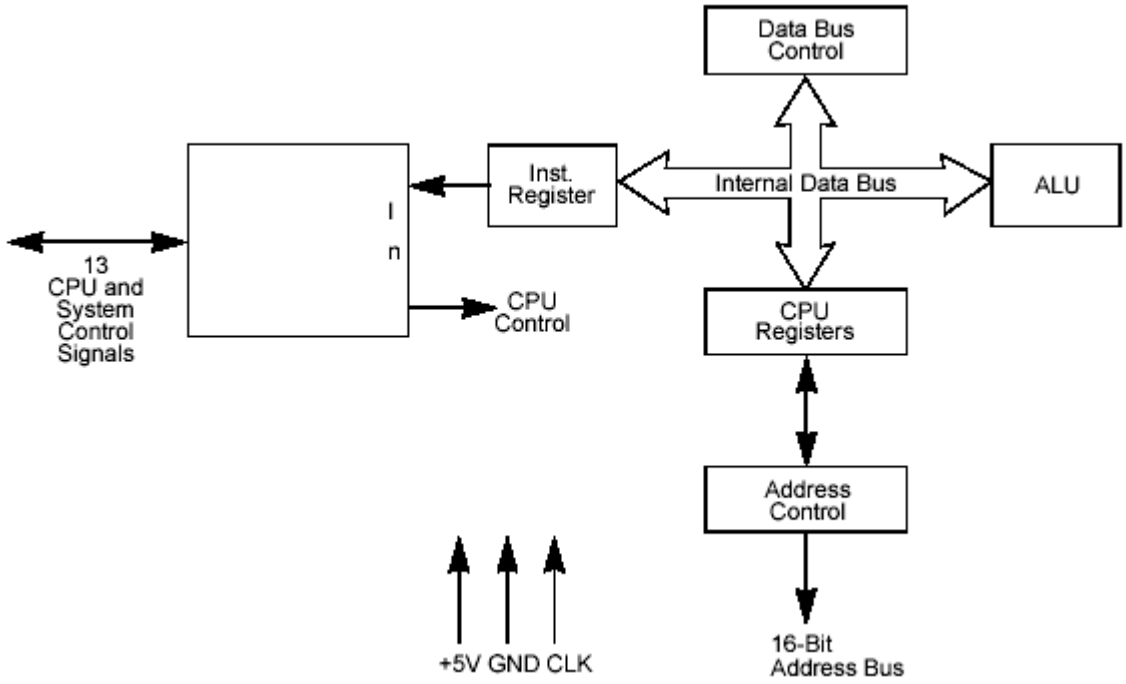

Figura 3: Diagrama de blocos da CPU Z80

<u>As principais características da CPU Z80 são:</u>

- Possui 158 instruções, incluindo um subgrupo de 78 do 8080A, que compreende operações de bit, byte, *word* e *string*, além de busca e transferência de blocos e diversos modos de endereçamento (direto, relativo, indexado, etc.).
- Clock de 6 a 20 MHz, conforme o modelo da CPU Z80.
- Os microprocessadores Z80 e a família de controladores periféricos associada são ligados por um sistema de vetores de interrupção. Este sistema pode ser conectado em cadeia para permitir a implementação de um esquema de prioridade de interrupção.
- São providos grupos duplicados de registradores de *flag* e de uso geral, facilitando o design e a operação do software. Também são providos dois registradores de índice de 16 bits.
- Existem três modos de interrupção: o modo 0, similar ao 8080; o modo 1, para dispositivos periféricos não-Z80; e o modo 2, para uso com a família Z80, com vetores de interrupção.
- Contador de *refresh* da memória dinâmica *on-chip*.

Uma das características mais interessantes da CPU Z80 e que utiliza mais eficientemente as suas capacidades e as da sua família de periféricos é o modo 2 de interrupção. Neste modo, o dispositivo que solicita a interrupção seleciona o endereço inicial da rotina de atendimento à interrupção. Isto é feito colocando um vetor de 8 bits no barramento de dados durante o ciclo de reconhecimento da interrupção. A CPU forma um ponteiro usando este byte como 8 bits menos significativos e o conteúdo do registrador I como os 8 bits mais significativos. Este ponteiro aponta para uma entrada numa tabela de endereços de rotinas de atendimento a interrupções. A CPU então pula para a rotina naquele endereço. Esta flexibilidade em selecionar o endereço da rotina de atendimento permite que um dispositivo periférico use diversas rotinas diferentes. Estas rotinas podem estar localizadas em qualquer endereço disponível de memória. Os recursos de modo 2 de interrupção são amplamente usados na programação da CPU do Water Mixer.

O microprocessador Z80 é o centro de uma família de produtos, sendo os principais (também usados internamente pelo Water Mixer):

- *PIO (Parallel Input/Output)*: opera tanto em modo de transferência I/O de byte (com handshaking) quanto de bit (sem handshaking). A PIO pode ser configurada para interfacear dispositivos paralelos padrão como impressoras e teclados.
- *CTC (Counter/Timer Circuit)*: possui quatro contadores/timers programáveis de 8 bits, cada um com uma pré-escala de 8 bits.
- *SIO (Serial Input/Output)*: controla dois canais e é capaz de operar em vários modos para comunicação tanto síncrona quanto assíncrona, incluindo Bi-Sinc e SDLC.

3.3. HI-TECH C Z80 COMPILER

3.3.1. Visão geral do processo de compilação

A compilação de um programa é executada por diversos aplicativos separados cujas operações são controladas ou pelo *driver* de linha de comando (CLD) ou pelo *driver* HPD (no caso do compilador para o Z80, este *driver* é chamado HPDZ). Em cada caso, o driver toma as opções especificadas pelo programador (opções do menu no HPDZ ou argumentos da linha de comando para o CLD) e determina quais aplicativos internos precisam ser executados e quais opções devem ser enviadas para cada. Quando o termo *compilador* é usado, quer-se denotar a coleção inteira de aplicativos e o *driver* envolvidos

no processo. Do mesmo modo, *compilação* refere-se à completa transformação da entrada para a saída pelo compilador.

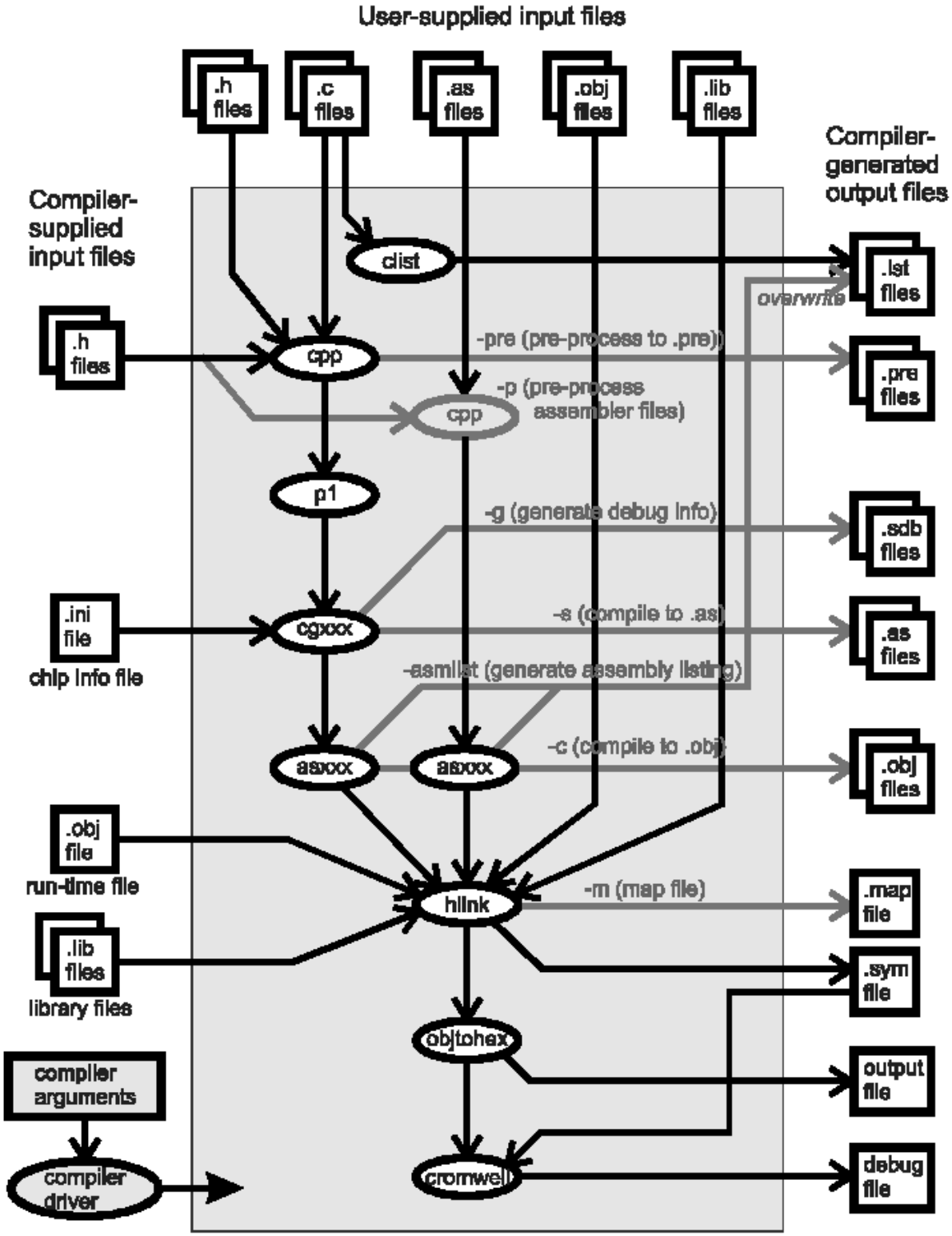

Figura 4: Visão geral do processo de compilação

O processo de compilação é discutido a seguir e pode ser ilustrado pela Figura 4 que mostra o diagrama de blocos dos estágios internos do compilador HI-TECH.

O usuário pode fornecer ao compilador as opções e diversos tipos de arquivos de entrada a fim de criar um programa. Na maioria dos projetos, no entanto, os únicos arquivos fornecidos são arquivos-fonte C (*C source files*), possivelmente acompanhados de arquivos de cabeçalho (*header files*). Um *header file* (também conhecido como *include file*) é um arquivo com informação relacionada ao programa e que geralmente inclui declarações de funções e tipos de dados, mas não código executável.

Arquivos assembler (*assembler files*) contêm mnemônicos que são específicos ao processador para o qual o programa está sendo compilado. Estes arquivos podem ser derivados de arquivos C previamente compilados ou desenvolvidos diretamente pelo programador. A dependência do processador, porém, os torna pouco portáveis, razão pela qual arquivos C que possam realizar as mesmas tarefas são geralmente preferidos.

Arquivos-fonte já pré-compilados (*object files*) ou que contenham rotinas comumente usadas (*library files*) também podem ser passados como entrada ao compilador.

De todos os diferentes tipos de arquivos que podem ser aceitos pelo compilador, são os arquivos-fonte C que requerem maior processamento. Para cada arquivo C é produzido um *C listing file,* que contém cada linha do programa precedida de um número. Os arquivos-fonte C passam também por um *pré-processador CPP*, que prepara os arquivos para posterior interpretação. As tarefas do CPP incluem remoção de comentários e espaços múltiplos, substituição de macros pelo respectivo texto (por ex. diretivas **#define**), inclusão condicional de código (por ex. diretivas **#if**, **#ifdef** etc.) e inserção de *header files*.

A saída do pré-procesador é similar mas diferente do código do arquivo C original, sendo normalmente chamada de *módulo*. O módulo é passado ao *parser*, que inicia a tarefa de transformar a descrição de um programa em C numa seqüência executável de instruções em assembler. É o *parser* que é responsável por detectar a maior parte dos erros no programa, principalmente erros de sintaxe.

Em seguida, o gerador de código (*code generator*) converte a saída do *parser* em mnemônicos em assembler. Este é o primeiro passo do processo de compilação que é específico ao processador utilizado (no caso, o Z80). O gerador de código pode produzir opcionalmente um arquivo .sdb para cada módulo contendo informação relativa aos símbolos definidos e que pode ser usado por programas de debug. Outra tarefa opcional é a otimização do código.

Os arquivos em *assembly* resultantes são os primeiros no processo de compilação que fazem referência às *psects*, ou seções de programa (program sections), que serão tratadas mais adiante pelo *linker*. Estes arquivos são passados ao *assembler*, que

converte a representação ASCII das instruções (mnemônicos) em código binário específico ao processador utilizado.

A saída do *assembler* é um *object file* em código binário e pode vir em dois tipos: relocáveis e absolutos. Arquivos relocáveis não têm seus endereços de memória definidos em valores absolutos.

A série de passos acima (pré-processamento, *parsing*, geração de código e *assembly*) é executada para cada arquivo C fornecido em seqüência. Erros são reportados à medida em que ocorrem no programa.

Assembler files passados ao compilador não requerem todo o processamento de arquivos C, mas passem pelo *assembler*. Object e library files já estão compilados e não são usados até o estágio de linkagem.

3.3.2. Linker

A tarefa fundamental do *linker* é combinar diversos object files relocáveis em um só arquivo. Os object files são ditos relocáveis desde que eles contenham informação suficiente para que quaisquer referências a endereços de programa ou de dados (por ex. o endereço de uma função) possam ser ajustados de acordo com onde o arquivo será localizado em memória depois do processo de linkagem. Relocação pode tomar duas formas básicas: relocação por nome, isto é, pelo valor final de um símbolo global, ou relocação por *psect*, isto é, pelo endereço-base de uma seção em particular do código.

Os object files operados pelo *linker* incluem os resultantes da compilação de arquivos C e em *assembly*. O *linker* performa o agrupamento e a relocação das *psects* contidas nos arquivos e substitui os nomes dos símbolos por endereços absolutos, uma vez que a relocação tornou possível saber onde os objetos serão armazenados em ROM ou RAM.

O *linker* também pode gerar um *map file* com a posição de todas as psects e os endereços absolutos dos símbolos, ou um *symbol file* apenas com os endereços dos símbolos.

Embora a saída do *linker* contenha toda a informação necessária para rodar o programa, ela ainda deve ser transferida do computador para o hardware final. Diversos formatos são disponíveis para isto, sendo o *Motorola* HEX e o *Intel* HEX os mais comuns. Para o compilador HI-TECH, o aplicativo OBJTOHEX é responsável por esta tarefa.

3.3.3. Psects

O gerador de código gera um *assembler* file para cada arquivo-fonte C compilado. O conteúdo destes arquivos em *assembler* inclui diferentes seções: alguma contêm instruções que representam o programa em C; outras contêm diretivas em *assembler*

que reservam espaço em RAM; outras contém constantes definidas em C que serão armazenadas em ROM; e outras contêm dados para objetos especiais como variáveis a serem armazenadas em memória não-volátil, vetores de interrupção e palavras de configuração usadas pelo processador. Uma vez que pode haver mais de um arquivo de entrada, existirão seções similares de código espalhadas em múltiplos arquivos em assembler que precisam ser agrupados juntos depois que toda a geração final de código é concluída.

Faz sentido ter todos os valores de dados inicializados juntos em blocos contíguos para que eles possam ser copiados em RAM de uma vez só em vez de tê-los divididos entre seções de código; o mesmo se aplica a todos os objetos globais não inicializados que precisam ter um espaço alocado em memória antes que o programa comece; algumas linhas de código ou objetos precisam ser posicionados em certas áreas de memória devido a requerimentos da capacidade de endereçamento do processador; e, às vezes, o usuário precisa ser capaz de posicionar o código ou os dados em endereços específicos para atender a requerimentos especiais de software (como no caso da implementação de um programa monitor). O gerador de código deve, portanto, incluir informação que indique como as diferentes seções em *assembler* devem ser manipuladas e posicionadas pelo *linker* mais adiante no processo de compilação.

O método usado pelo compilador HI-TECH para agrupar e posicionar diferentes partes de um programa é colocar todas as instruções em assembler em seções individuais e relocáveis. Estas seções de programa são conhecidas como *psects* (***p****rogram* ***sect****ions*). Diversas opções são então passadas ao linker que indicam a memória disponível no sistema-alvo e como todas as *psects* devem ser posicionadas neste espaço de memória.

Uma *psect* é definida através de uma diretiva PSECT em assembler. Estas diretivas são geradas automaticamente na compilação de um arquivo em C ou incluídas manualmente em *assembly* files:

PSECT *name*,*option*,*option*...

Esta diretiva consiste da palavra-chave PSECT seguida pelo nome pelo qual a *psect* será referida e por opções. Mais de uma *psect* pode ter o mesmo nome, o que implica que o seu conteúdo é similar e que elas devem ser agrupadas juntas pelo *linker*. Isto permite posicionar objetos juntos na memória mesmo se eles são definidos em arquivos diferentes.

As opções são instruções para o *linker* que descrevem como a *psect* deve ser agrupada e relocada no *object file* final. Estas opções incluem os endereços de *link* e *load* da *psect*. Genericamente falando, o endereço de *link* é o endereço onde a *psect* será acessada em tempo de execução. O endereço de *load*, que pode ou não ser o mesmo que o de *link*, é o endereço onde a *psect* se inicia no arquivo de saída (HEX, binário, etc.).

Um exemplo onde os endereços de *link* e *load* são diferentes é o de uma psect com dados inicializados, que é copiada da ROM para a RAM no carregamento do programa, de modo que ela possa ser modificada em tempo de execução.

3.3.4. Tipos de psects

Embora um número arbitrário de *psects* possa ser utilizado, o compilador já utiliza um conjunto de *psects* pré-definidas, cujas principais são:

vectors Contém o vetor de reset o vetor NMI. Esta *psect* é linkada normalmente no endereço 0 em ROM para que o Z80 busque o conteúdo do vetor de reset do zero na inicialização. O código nesta psect vem do módulo de *run-time startup*.

lowtext Usada opcionalmente para o código que deve ir na área comum zero como, por exemplo, o código de *startup*.

text Usada para todo o código executável. Sub-rotinas em assembler escritas pelo usuário, código em C e funções de atendimento a interrupções são colocadas nesta *psect*.

strings Usada para todas as *strings* constantes não nomeadas, como os argumentos das funções *printf()* e *puts()*. Esta *psect* é linkada em ROM, uma vez que ela não precisa ser modificável.

const Usada para todas as constantes inicializadas da classe *const*. Por exemplo:

```
const char masks[] = { 1,2,4,8,16,32,64,128 };
```

Esta *psect* é linkada em ROM, uma vez que ela não precisa ser modificável.

im2vecs Quando usando o modo de interrupção 2, é necessário ter um bloco de memória alocado para a tabela de vetores de interrupção modo 2. Esta tabela deve começar numa fronteira de 256 bytes e pode se estender até 256 bytes de comprimento. Esta *psect* representa o bloco de memória. Ela é linkada em ROM depois de todas as outras *psects*.

ramstart Possui de comprimento zero e especifica o início da área de RAM.

data Contém todos os dados estaticamente inicializados, exceto os da classe *code* e *const*. É linkada em ROM, pois é os dados possuem valores iniciais, e é copiada em RAM no *startup* para poder ser modificada em tempo de execução.

bss Usada para todos os dados não inicializados e variáveis externas residentes em RAM. É completamente limpa pela rotina de *startup* antes da invocação do código principal do programa.

stack Possui comprimento zero e especifica o topo da memória RAM. O endereço-base desta *psect* é carregado no *stack pointer* no código de *startup*.

nvram Usada para armazenar variáveis *persistent* (não-voláteis). Não é limpa ou modificada no *startup*.

O módulo de *runtime startup* citado acima é um módulo específico para cada modelo de processador e fornecido pelo compilador HI-TECH. Como o nome diz, ele é executado na inicialização do programa e performa algumas operações essenciais, como declarar a tabela de vetores de interrupção, limpar a área de memória da *psect bss*, copiar a *psect data* da ROM para a RAM e chamar a rotina *main()*. Pode-se modificar o módulo para incluir, por exemplo, rotinas de inicialização da CPU ou de seus periféricos.

3.3.5. Lucifer

Lucifer é um *debugger* remoto de nível de programa para uso com o compilador HI-TECH. Consiste de um programa que roda num computador (*host*) e se comunica com um sistema baseado em microprocessador Z80, Z180, 64180 ou NSC800 via conexão serial. O programa no *host* fornece a interface ao usuário, incluindo apresentação das linhas do código, *disassembly*, amostragem da memória, etc. O sistema-alvo (*target*) deve ter lógica para ler e escrever na memória e nos registradores, além de implementar *single stepping*. Com o *Lucifer* é fornecido um pequeno programa (*monitor*) que pode ser compilado e colocado em ROM no sistema-alvo para implementar estas funções.

Tabela 1: Lista de comandos do *debugger Lucifer*

Command	Meaning
B *[line|addr]*	Set or display breakpoints
C	Display instruction at PC
D *[addr [addr]]*	Display memory contents
E *fn|file*	Examine C source code
G *[addr]*	Commence execution
I	Toggle instruction trace mode
L *file*	Load a HEX file
M *addr val1 [val2 ...]*	Modify memory
P	Toggle input prompting mode
Q	Exit to operating system
R *[breakpnt]*	Remove breakpoints
S	Step one line
T	Trace one instruction
U *[addr]*	Disassemble machine instructions
W *file addr length*	Upload binary
X *[reg1 val1 [reg2 val2 ...]]*	Examine or change registers
@ *type [indirection]expr*	Display C variables
. *[breakpoint]*	Set a breakpoint and go
; *[line]*	Display from a source line
=	Display next page of source
-	Display previous page of source
/ *[string]*	Search source file for a string
! *command*	Execute a DOS command

Para o uso do monitor, é necessário compilá-lo e gravá-lo em ROM juntamente com o programa do *target*. Antes da compilação, é necessário fazer modificações sobre o código do monitor como, por exemplo, sobre o a rotina de *single stepping* e os *drivers* da porta serial.

Lucifer reconhece uma série de comandos, como mostrado na tabela 1.

3.4. UNIROM

3.4.1. Visão geral

Emuladores de EPROM não apenas eliminam a necessidade de queimar ROMs, mas também adicionam recursos avançados que aceleram o desenvolvimento de *firmware*. UniROM é um emulador avançado de memória da TechTools com "*Live Memory Access*" e impacto zero sobre o sistema-alvo. UniROM permite acesso simultâneo de leitura e escrita ao espaço de memória emulado para ambos PC e *target*. UniROM também inclui uma "UART virtual" integrada que pode ser mapeada na memória emulada e fornecer assim ao *target* uma porta serial adicional para fins de *debugging*.

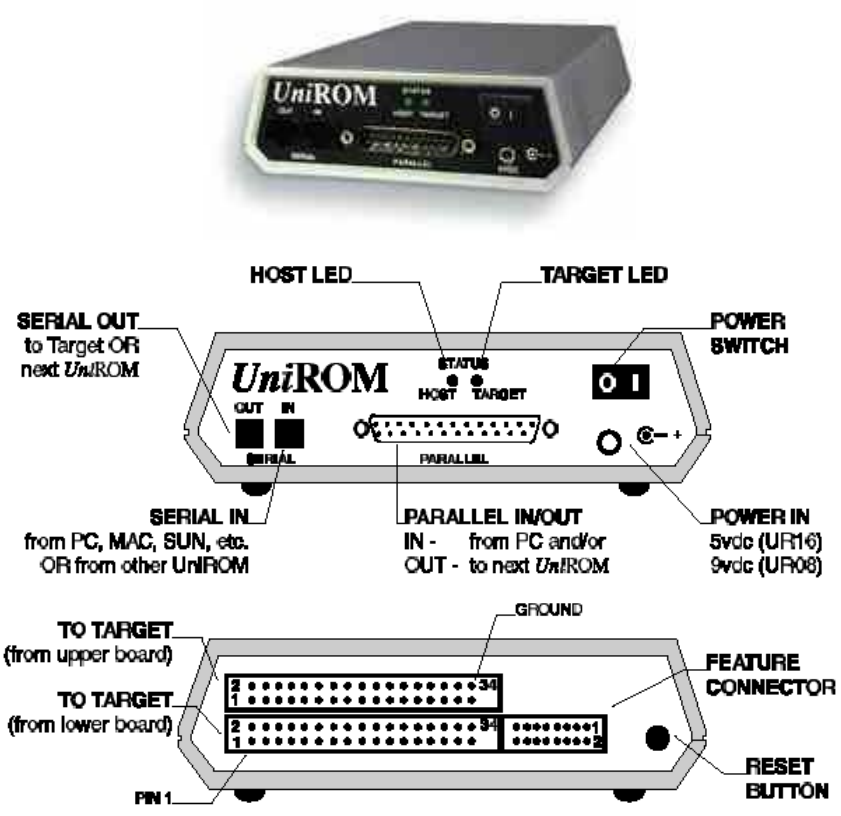

Figura 5: Vista do UniROM e detalhes de arquitetura

3.4.2. Detalhes da arquitetura

- PARALLEL IN/OUT

A porta paralela pode ser conectada à porta de impressora em um PC. URLOAD, URTERM e as bibliotecas do UniROM podem carregar, contorolar e verificar o status do UniROM através desta conexão.

- SERIAL IN

A porta SERIAL IN serve a vários propósitos. Pode ser usada para carregar, controlar e verificar o status do UniROM. Pode também ser usada para permitir *software debuggers* ou programas de emulação de terminal a se comunicarem com o *target* através de um dos caminhos de comunicação mapeados em memória ou da porta SERIAL OUT.

- SERIAL OUT

O principal propósito da porta SERIAL OUT é permitir a conexão em cadeia de múltiplos UniROMs. Um simples cabo pode conectar a porta SERIAL OUT de um UniROM à porta SERIAL IN do UniROM seguinte. A porta OUT também pode ser conectada à porta serial do *target*. Se o alvo possui uma porta serial livre, este console permite a interface ao *kernel* do *target* ou monitor sem modificações. Isto é particularmente útil se o *kernel* já está pronto e funcionando através da porta serial. O UniROM pode simplesmente interceptar a linha serial e o *debugging* continua normalmente.

- RESET

A saída de reset é compatível com TTL/CMOS e pode ser configurada com bipolar ou tri-state. Deve ser conectada ao reset do circuito do *target*.

- INTERRUPT

A saída de interrupção é compatível com TTL/CMOS e pode ser configurada como bipolar ou tri-state. É normalmente conectada ao sistema de interrupção do *target* quando usando URCOM ou VCOM para comunicação mapeada em memória. URCOM e VCOM podem gerar interrupções automaticamente na recepção, na transmissão ou em ambos.

- CONTROL LINES

UniROM suporta quatro linhas para controle do usuário. Estas podem ser individualmente setadas ou resetadas conforme comandado, para estímulo ou controle do *target*.

- STATUS LINES

UniROM permite ao usuário monitorar quatro diferentes entradas do target. Estas linhas de status podem ser conectadas a quaisquer sinais TTL ou CMOS.

- DUAL-PORT MEMORY

UniROM incorpora uma arquitetura de memória "*dual-ported*". Isto permite acesso completo à memória emulada sem perturbar o *target*. Este recurso é usado para permitir observação da memória em tempo real e comunicação mapeada em memória através da VCOM, além de leitura e escrita em memória em tempo real enquanto o *target* está executando fora deste mesmo espaço de memória.

- VCOM

A CPU no UniROM gerencia uma UART virtual chamada VCOM, que formaliza o uso da memória *dual-ported* para comunicação entre o *target* e o *host*. VCOM parece muito como um UART padrão, mapeada em memória para o *target*. Isto torna fácil configurar a maioria dos monitores remotos ou *debuggers* através do UniROM em vez de por uma porta serial, eliminando a necessidade de dedicar recursos do *target* para *debugging*.

3.4.3. UART virtual - VCOM

A UART virtual é basicamente uma UART padrão que é mapeada no espaço em memória do *target*. O programa no *target* interage com este tipo de UART quase como com qualquer outra UART. A diferença principal é que ela também ocupa uma pequena parte do espaço de código. A UART virtual integrada ao UniROM é chamada VCOM. A VCOM usa 4 bytes de memória compartilhada para facilitar as comunicações. O *target* envia um caractere ao *host* escrevendo este caractere numa posição específica do espaço de memória (*TX BUFF*) em EPROM ou FLASH. Ele então seta um bit em outra posição de memória (*flag TX FULL*) para informar o UniROM que o buffer de transmissão está cheio. O UniROM ecoa este caractere na sua porta serial e reseta a *flag TX FULL*. Da mesma maneira, quando o UniROM recebe um caractere do *host*, ele escreve este dado numa posição específica de memória (*RX BUFF*), seta a *flag RX FULL* e gera uma interrupção no *target*. O *target* pode sondar a *flag RX FULL* para determinar se há um dado disponível ou pode executar uma rotina de interrupção. O *target* então lê o byte recebido e reseta a *flag RX FULL* para indicar que este byte foi recebido. Estas etapas podem ser ilustradas pelo fragmento de código abaixo, que mostram as modificações necessárias nos comandos I/O de baixo nível do *kernel*:

```
// assume-se que tx_full_flag, rx_full_flag, rx_buff e tx_buff são ponteiros unsigned char
// que foram inicializados para apontar as posições apropriadas de memória para a VCOM

void put_char(txdata)
{
        while (*tx_full_flag)    // espera o buffer de transmissão esvaziar
        continue;
        *tx_buff = txdata;       // preenche o buffer de transmissão
```

```
        *tx_full_flag = 0x01;    // seta a flag TX FULL
}

unsigned char get_char(void)
{
        unsigned char ch;
        while (*rx_full_flag == 0)        // espera um caractere chegar
        continue;
        ch = *rx_buffer;                        // busca o caractere
        *rx_full_flag = 0;                // reseta a flag RX FULL
        return(ch);
}
```

Como pode-se ver, a VCOM opera quase como qualquer outra UART. A única diferença é que o software do target é responsável por setar a *flag TX FULL* e resetar a *flag RX FULL*. Com a porta VCOM, elimina-se a necessidade de uma porta serial dedicada no *target* para *debugging*.

4. ATIVIDADES REALIZADAS

4.1. PLANO DE TRABALHO

As principais atividades executadas no estágio na NBN foram a atualização do software da CPU da base do Water Mixer e a implementação de um programa monitor junto com o mesmo software para fins de debug. Estas atividades estão compreendidas dentro de um plano maior de trabalho, que deve incluir posteriormente a validação da situação atual do Water Mixer e do DAP frente às normas, além da implementação de novas funções no firmware.

O período inicial do estágio consistiu no estudo das ferramentas necessárias para o trabalho e do código-fonte. Uma vez obtido um nível de familiarização adequado, o estagiário pôde iniciar as tarefas propriamente ditas.

4.2. ATUALIZAÇÕES NO SOFTWARE DA CPU – WM BASE

O software da CPU da base do Water Mixer é composto por diversos módulos separados conforme sua finalidade, cada um dedicado a implementar um grupo de funções similares. A grande maioria destes módulos foi desenvolvida em C, e inclui quase todas as operações em curso durante o funcionamento normal do Water Mixer. Tarefas como controle da abertura das válvulas, regulagem de parâmetros, programação da base e das remotas e comunicação com o sistema Smarttan são incluídas aqui.

Uma pequena parte dos módulos, porém, é escrita diretamente em *assembler*. O desenvolvimento de código em C provê uma maior portabilidade ao programa mas, justamente por isso, as funções que lidam mais diretamente com o hardware envolvido precisam ser escritas em *assembler*. Estas funções incluem entre outras:

- a rotina de inicialização básica e definição de vetores de interrupção (rotina de startup, comentada nas informações preliminares).
- a inicialização do display, da PIO, do relógio, do watchdog e dos flags da NMI
- a programação do sistema básico de I/O e do CTC, incluindo o atendimento às rotinas de interrupção
- as rotinas de comunicação básica (com a SIO)
- as funções de controle do SmartWatch

O código-fonte do Water Mixer foi desenvolvido para servir também aos produtos Reciclo e DAP da NBN. Devido à semelhança do funcionamento geral dos três produtos, optou-se por fazer um código comum, onde as seções específicas a cada produto são separadas por diretivas do tipo **#ifdef** PRODUTO... **#endif**, onde PRODUTO é uma constante fornecida no momento da compilação. Assim, atualizações e correções em funções básicas servem aos três produtos igualmente.

O programa da CPU do Water Mixer já possui vários anos e passou por várias atualizações, encontrando-se hoje na versão 9.7. O motivo para uma atualização pode ser o acréscimo de uma nova função, a correção de um *bug* ou a adaptação a um hardware ou compilador novo. No presente caso, os motivos foram a adaptação à última versão do compilador C HI-TECH juntamente com a correção de um *bug*.

Idealmente, cada versão nova de um compilador deve ser capaz de compilar sem problemas qualquer código já compilado pelas versões anteriores. Mas, algumas vezes, isso não acontece, pois os desenvolvedores do compilador podem ter decidido de mudar as etapas envolvidas na transformação de uma determinada função em C em mnemônicos *assembler*. O resultado é que *psects* novas podem ser usadas ou o código final pode ficar maior, o que é bastante problemático quando se tem uma disponibilidade pequena de espaço em memória. Verificou-se que as duas situações ocorreram na compilação do software do Water Mixer com o compilador novo da HI-TECH.

O primeiro problema foi resolvido com uma pequena alteração na rotina de *startup* do programa para incluir a nova *psect*. O problema de espaço pode ser rapidamente percebido no exame do mapa do espaço em memória resultante da compilação. Por restrições de projeto, a memória ROM (para *psects* da classe CODE, ou seja, código) ocupa o espaço de 0000h a C000h, enquanto que o espaço restante até FFFFh é reservado para RAM (*psects* da classe DATA, ou seja, dados). Tais restrições são informadas ao *linker*, que reúne todas as *psects* no arquivo final. Porém, na nova versão do compilador, o código resultante passou a exceder o espaço determinado e a sobrescrever o espaço em RAM, o que evidentemente impedia o programa final de funcionar.

O uso de alguns recursos de otimização do compilador reduziu o tamanho do código até que ele coubesse no espaço de ROM. O uso de rotinas de otimização, no entanto, pode produzir erros inesperados devido à relocação de linhas de programa em pontos críticos, então deve ser feito com cautela, e o resultado final deve ser testado.

O programa otimizado passou a rodar, mas com uma falha manifestada na seqüência de *polling* das remotas pela base. Apenas uma das remotas estava sendo sondada entre as três que podem ser conectadas. Este defeito não acontecia na compilação com o compilador antigo e, apesar de ser um defeito aparentemente simples, exigiu bastante tempo para ser solucionado.

Inicialmente partiu-se por experimentar diversas opções globais de compilação e depois passou-se a experimentá-las separadamente sobre cada módulo do projeto. Nisto, foi feita inclusive a linkagem de módulos compilados pelas versões antiga e nova do compilador. Verificou-se enfim que um em especial – que definia o conjunto de variáveis não voláteis – produzia a manifestação da falha com o compilador novo, mas não com o antigo. Examinando-se os mnemônicos em *assembler* resultantes da compilação das duas versões, pôde-se ver que eles diferiam apenas na ordem da definição das variáveis.

Dessa forma, identificou-se que o problema deveria estar na gravação errônea das variáveis na memória não-volátil. Uma vez que nem o compilador não é obrigado a manter a ordem das variáveis declaradas em C após a compilação (a não ser que elas estejam dentro de uma *struct*), nem o programa deveria sofrer alterações no funcionamento pela mudança das posições das variáveis (desde que, é claro, os símbolos identificadores dos endereços de cada variável estejam apontando para os lugares corretos), suspeitou-se que deveria haver um *bug*, uma falha visível somente quando as variáveis são gravadas na memória em uma determinada ordem.

Depois de uma investigação rigorosa, verificou-se que realmente havia uma falha. Um certo conjunto de variáveis era gravado em bloco na memória não-volátil por endereçamento indexado e um erro na formulação dos índices fazia que algumas posições na memória eram sobrescritas umas sobre as outras. Também foi localizada uma outra falha causada pela mesma razão, mas que não havia sido ainda constatada como erro visível.

Resolvidos estes dois problemas, o programa passou a funcionar corretamente.

4.3. IMPLEMENTAÇÃO DO PROGRAMA MONITOR

A atualização do software da CPU da base do Water Mixer foi um passo necessário antes da tarefa seguinte, que era a implementação de um programa monitor junto com o programa principal para auxiliar a depuração de erros e o teste de novas funções a serem adicionadas.

Muito do tempo gasto no desenvolvimento de um projeto se dá no teste do protótipo e na busca de erros. Uma vez que um produto nunca está verdadeiramente "pronto", principalmente no que diz respeito ao software, o projetista sempre terá que passar bastante tempo em sucessivas depurações sobre o mesmo projeto. Neste sentido, quaisquer ferramentas que venham a acelerar estas depurações são bem-vindas.

O UniROM é uma destas ferramentas. Como dito, nas informações preliminares, o UniROM é um emulador de EPROM com vários recursos. Além de evitar o trabalho de gravar e regravar EPROMs, ele fornece uma porta serial extra – através da UART virtual VCOM – que é bastante útil para debug.

O UniROM pode se conectar ao computador (*host*) por via paralela ou serial. Como será demonstrado adiante, as duas conexões foram necessárias para o uso do monitor.

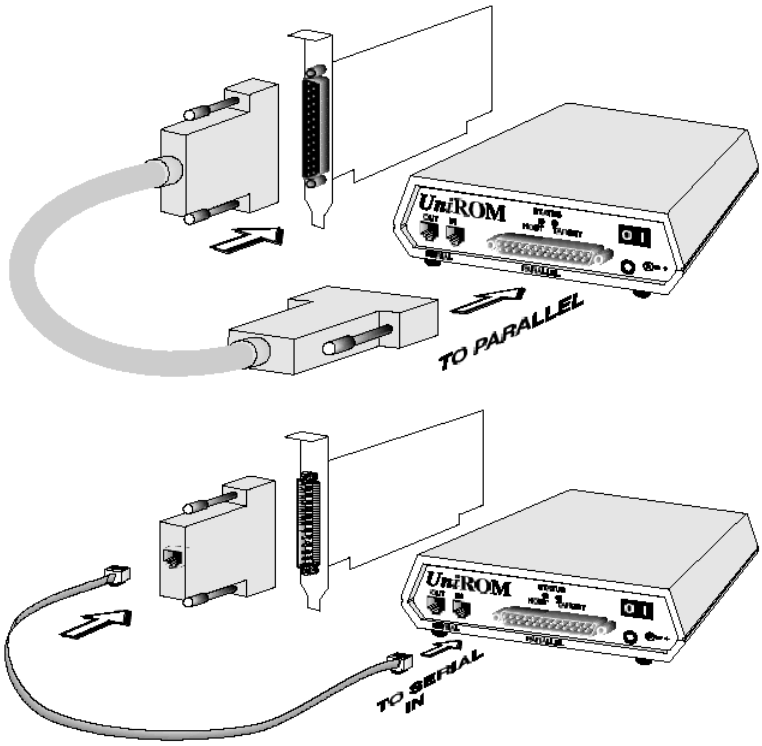

Figura 6: Conexão paralela e serial do UniROM ao *host*

O *debugger* usado foi o *Lucifer*, fornecido pela HI-TECH, que pode se comunicar ao sistema de teste via conexão serial. Para o uso do *Lucifer*, é necessário gravar junto com o programa principal um monitor que implementa as funções necessárias ao *Lucifer* e faz a comunicação com o *debugger*.

A Figura 7 ilustra simplificadamente o processo de implementação do programa monitor. Este deve ocupar o início do espaço de memória (página 0) de modo que ele seja executado no reset do sistema. O programa principal, por sua vez, deve ser totalmente relocável, de modo que possa ser movido para um espaço diferente de memória e continuar funcionando corretamente. Por relocável, quer-se dizer que não devem ser feitas referência a endereços absolutos (**JP** 1000, **ORG** 40, etc.), mas sim a posições referenciadas por símbolos.

O programa do Water Mixer foi concebido para ser o mais relocável possível, de modo que as principais correções necessárias no seu código referiam-se à tabela de vetores de interrupção do modo 2. Os endereços do vetores em si continuaram os mesmos – o endereço é montado com o valor colocado no barramento de dados pelo periférico interruptor durante o período de reconhecimento da interrupção – mas as rotinas de atendimento a que eles se referem foram relocadas.

O monitor pode permanecer na EPROM mesmo depois da fase de depuração. Uma diretiva **#ifdef** pode ser usada para mudar o ponto de entrada na inicialização, se no início do programa principal ou no monitor.

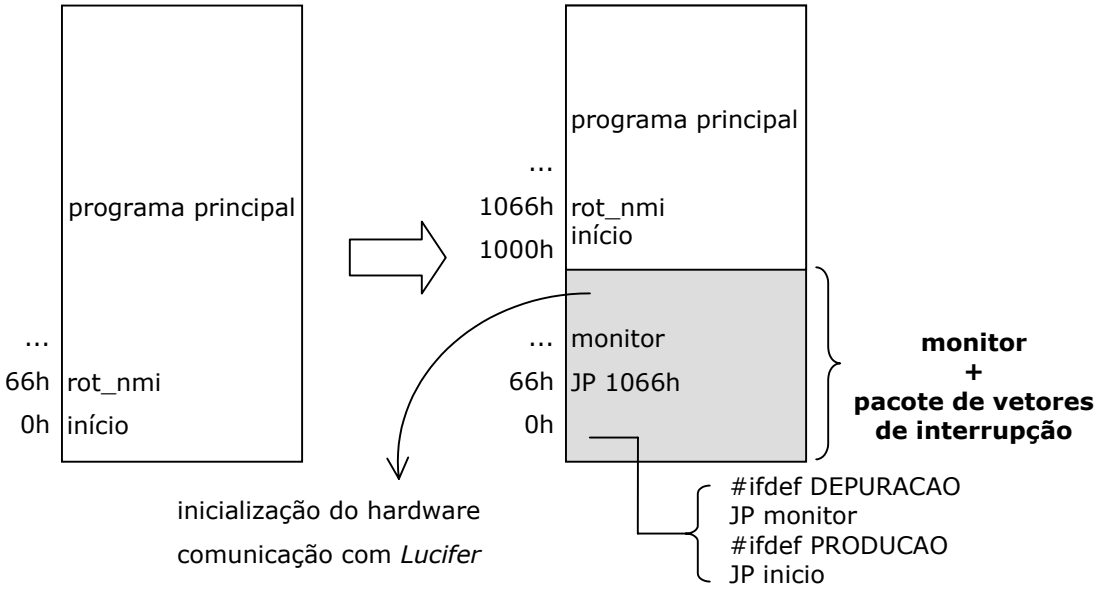

Figura 7: Ilustração da implementação de um programa monitor

O programa monitor fornecido pela HI-TECH era bastante genérico: supunha comunicação serial com o *host* por uma porta SIO e tinha suas próprias rotinas de atendimento às interrupções geradas pelos periféricos (PIO, SIO e CTC). Através do uso do UniROM, não foi necessário ocupar uma porta SIO unicamente para a comunicação com o *debugger*. Em vez disso, fez-se uso da UART virtual VCOM que exigiu algumas modificações nas rotinas I/O de baixo nível do monitor, conforme o modelo mostrado nas informações preliminares.

Assim, a comunicação do *target* com o *debugger* pôde ser feita através da porta serial do UniROM. Como os próprios aplicativos do UniROM (URLOAD e UREDIT) para o carregamento e o acompanhamento do software de teste exigiam para si uma via de comunicação, fez-se uso também da porta paralela.

A Figura 8 mostra um esquema simplificado da memória emulada do *target* e sua interação com o UniROM e o *Lucifer*. Devido ao grau de complexidade e à grande possibilidade de erros no processo, as etapas de implementação do monitor foram feitas aos poucos. No seu funcionamento mínimo, o monitor deveria incluir as rotinas mínimas de inicialização do sistema do Water Mixer, como a inicialização da PIO, dos vetores de interrupção e o desligamento do display, entre outros. Nesta situação, o *debugger* não poderia responder a uma interrupção comandada pelo *host*; permitindo o uso das interrupções do CTC pelo monitor, porém, ele se torna capaz de interromper o sistema em seu funcionamento normal a qualquer momento para execução de um comando do *debugger*. Nas informações preliminares, está a lista de comandos aceitos pelo *Lucifer*.

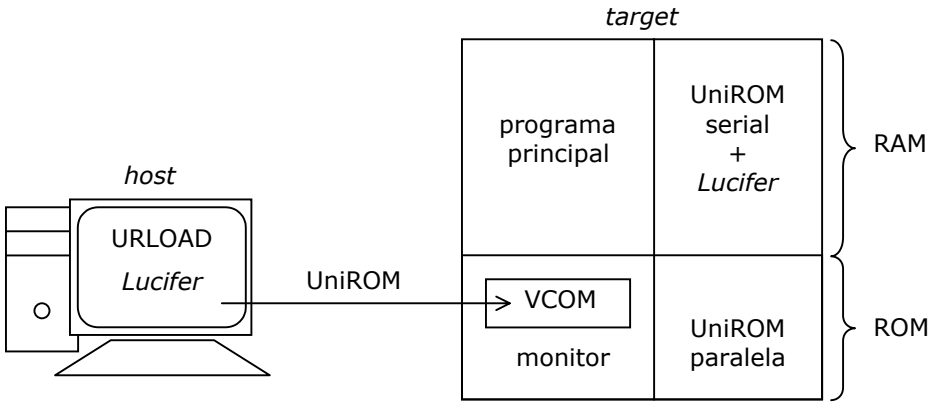

Figura 8: Emulação de memória com o monitor e o UniROM

Após a conclusão da implementação do monitor, quaisquer alterações poderão ser feitas com maior facilidade. Com a experiência obtida, pode-se planejar também o uso de programas monitores e *debuggers* nos demais produtos da empresa.

5. CONCLUSÕES

A experiência de estágio na NBN Automação Industrial foi bastante proveitosa, já que o ambiente de trabalho privilegia o aprendizado do estagiário e sua interação com os demais funcionários e com os outros setores da empresa. Novos conhecimentos foram adquiridos e outros foram aprimorados, principalmente os relacionados ao uso de microprocessadores e à programação C e *assembler*.

Neste ponto, fica registrada uma deficiência do curso no que se refere à programação em linguagem C. Apesar de ser universalmente difundida e amplamente utilizada no projeto de qualquer circuito microprocessado, não há disciplina no curso que enfoque a programação em C, ou mesmo que estimule o seu uso. Assim, o estudante é obrigado freqüentemente a corrigir por conta própria a sua deficiência e a sua falta de experiência por exigência do seu estágio. Assim, seria de grande valia a existência de algum projeto no curso em que o aluno desenvolvesse algumas rotinas em C para uso em microprocessadores.

Por fim, através desta experiência, pode-se dizer porque o Estágio Supervisionado é uma cadeira obrigatória e de suma importância para a graduação do curso de Engenharia Elétrica. Ele vem colocar à prova os conhecimentos e a formação adquirida ao longo do curso e, principalmente, vem possibilitar maiores chances para o futuro profissional no cada vez mais exigido e concorrido mercado de trabalho. É importante destacar também a flexibilidade permitida ao estagiário, flexibilidade esta proveniente de sua condição de aprendiz. O estudante pode ser mais ousado e tomar maiores riscos, sendo o estágio é a oportunidade que ele tem de experimentar e contribuir definitivamente na formação do seu caráter profissional futuro.

REFERÊNCIAS BIBLIOGRÁFICAS

1. *Communicating through Virtual UARTs*. TechTools. EUA, 1996.
2. *HI-TECH C Z80 Compiler User's Guide*. HI-TECH Software. Australia, 2001.
3. *UniROM User's Manual*. TechTools. EUA, 1997.
4. *Z80 Family CPU User's Manual*. Zilog Inc. EUA, 2002.