"CROSS ASSEMBLERS" PARA MICROPROCESSADORES-GERAÇÃO AUTOMÁTICA ATRAVÉS DO SPD

JOÃO JOSÉ NETO

ESCOLA POLITÉCNICA DA UNIVERSIDADE DE SÃO PAULO
LABORATÓRIO DE SISTEMAS DIGITAIS
CAIXA POSTAL 11455
CEP 05508 - SÃO PAULO - SP

RESUMO: Atualmente, os microprocessadores definiram-se incontestavelmente como os componentes mais importantes da arquitetura de computadores. A construção de software básico para tais microprocessadores e para outros sistemas novos mostra-se uma tarefa repetitiva, frequente, tediosa e demorada. A semelhança encontrada entre os montadores sugere automatizar-se sua geração, com a finalidade de baratear seu desenvolvimento. O SPD, sistema inicialmente voltado à construção automática de simuladores a partir de descrições de máquinas digitais, foi recentemente ampliado em suas capacidades pela incorporação de novas ferramentas dirigidas à definição de montadores e programas similares. O cross-software assim definido pode operar em conjunto com os simuladores anteriormente mencionados, permitindo ao usuário final montar e executar seus programas no computador hospedeiro. Alterar um montador existente, ou construir um outro totalmente novo, é uma mera questão de modificação do texto de entrada do SPD que define tais programas, seguida de uma re-submissão da descrição ao sistema. Este texto faz uma ligeira coleta das necessidades mais importantes do programador de um montador e descreve as principais ferramentas incorporadas ao SPD para suprir tais necessidades.

ABSTRACT: Presently, microprocessors have undoubtly defined their place as the most important components of computer architecture. Building system programs for these microprocessors and for other new systems became a very frequent cumbersome and time-consuming repetitive task. The similarity presented by assemblers suggests the automation of their generation, in order to decrease their cost of development. SPD, a system initially intended to support the automatic building of simulators from digital machine descriptions, had its power recently improved by means of new tools which allow the user to define assemblers and similar programs in addition to simulators. The generated cross-software may operate in a cooperative fashion, allowing the final user to assemble and execute his programs on the host machine. Modifying such an assembler, or building a new one is done by simply altering the input definition program and re-submitting it to SPD. The present text shortly surveys the main assembler-writer needs, and describes the most important assembler-generating description tools just included in the SPD system.

1. INTRODUÇÃO

Durante muito tempo, a construção de módulos de software básico foi considerada como um trabalho artesanal obrigatório da fase de desenvolvimento de um sistema digital. Hoje, uma década após o aparecimento dos microprocessadores no mercado de componentes eletrônicos, ainda existe um imenso desperdício de mão de obra técnica especializada, devido à aplicação de milhares de homens-hora técnicos anuais na tarefa de construir, repetidas e repetidas vezes, dezenas de versões de "cross-assemblers" para algumas poucas variedades dos microprocessadores disponíveis na praça. Quem ainda não construiu o seu próprio "cross-assembler", para o 8080, ou Z-80, ou outro microprocessador qualquer?... E, voltadas para uma mesma máquina hospedeira, quantas versões do mesmo "cross-assembler" existem, criados por programadores diferentes, muitos, quem sabe, nem sequer divulgados? A pesar de esforços desta natureza levarem, muitas vezes, à disseminação e à criação de uma capacitação profissional que outrora foi privilégio de uma pequena minoria de especialistas, a comunidade, como um todo, acaba lucrando muito pouco, uma vez que, destes programas, raríssimos são colocados à disposição dos prováveis usuários, e, dos que chegam a sê-lo, muitas vezes só o são para grupos restritos, dificultando o intercâmbio de programas entre os usuários, devido à proliferação de dialetos, muitas vezes não total

mente compatíveis, da mesma linguagem.

Com a finalidade de reduzir, por um lado, o número de novos desenvolvimentos de tais programas, e, por outro, de permitir a sua obtenção metódica e confiável a custo reduzido e com grande facilidade de manutenção e alteração do produto, desenvolveu-se a possibilidade de gerar automaticamente processadores de linguagens de baixo nível como parte dos recursos de automação oferecidos pelo sistema SPD - sistema automático de apoio ao projeto e desenvolvimento de sistemas digitais - desenvolvido no Laboratório de Sistemas Digitais da Escola Politécnica da USP. Neste sistema dispõe-se de uma linguagem voltada para a especificação de tais programas a qual faz parte da linguagem LD (linguagem de definição). [1]

2. CARACTERÍSTICAS DOS MONTADORES

Os montadores [2] em geral, apresentam entre si grandes semelhanças estruturais, devido à similaridade exibida por suas linguagens de entrada e por suas funções internas. Desta maneira, é possível, através do levantamento destas características, estudar os requisitos de um sistema para geração automática de montadores, através especialmente da definição dos recursos de que deverá ser dotada uma meta-linguagem por meio da qual o sistema automático é alimentado com as especificações do montador particular desejado.

2.1 - Considerações Sintáticas

Um texto em linguagem simbólica apresenta-se usualmente na forma de sequência de sentenças básicas, que assumem as seguintes conotações:

a) Instruções de máquina, representando em notação simbólica os códigos de linguagem básica da máquina.

b) Pseudo instruções, com formato semelhante ao das instruções de máquina, mas que são comandos para o montador, e cuja operação é pré-definida.

c) Definições de macros, delimitadas por pseudo instruções especiais, e compostas de textos simbólicos quaisquer.

d) Chamadas de macro-instruções, que funcionam como pseudo-instruções, mas cuja operação é definida, através do uso de pseudo-instruções especiais, pelo próprio programador através de definições de macros declaradas no texto.

e) Comentários, que permitem ao programador documentar seu programa.

A partir do texto simbólico de entrada, o montador gera uma série de textos de saída, tais como:

a) listagens do programa, representando:
   - cópia do texto fonte
   - código de máquina gerado, e seus endereços em formato legível
   - mensagens de erro
   - numeração das linhas do texto
   - formatação automática (por exemplo, tabulação dos campos)
   - tabelas de símbolos
   - tabelas de referências cruzadas
   - impressão do texto resultante da expansão das macro-instruções
   - mapas de memória, com informações sobre a alocação de memória realizada.

b) arquivos contendo o texto fonte expandido, sem macros.

c) arquivos contendo o texto objeto apresentando:
   - identificação
   - código de máquina, com endereço e informações sobre relocação
   - dicionários de símbolos
   - redundâncias para aumentar a confiabilidade

Algumas considerações extra podem ser feitas no sentido de detalhar um pouco o estudo da sintaxe das linguagens de entrada dos montadores:

a) Há montadores cuja linguagem de entrada tem formato livre, e outros em que este formato é fixo. No primeiro caso, separadores especiais delimitam os vários campos, que não têm posição física fixa. No segundo, uma tubulação é exigida, e o posicionamento físico do campo é que determina a sua interpretação.

b) Na maioria absoluta das linguagens de entrada dos montadores, uma sentença básica é caracterizada por uma estrutura típica em que constam os seguintes campos:

   - rótulo: (normalmente opcional), para fins de identificação e referência. No caso geral, alguns montadores permitem qualquer número de rótulos para a mesma sentença. Em alguns casos, os rótulos devem obrigatoriamente comparecer em posição fixa do registro físico que contém a sentença. Em outros casos, os rótulos são identificados através de delimitadores especiais, o que permite seu uso em qualquer posição física do texto.

   - mnemônico: (normalmente obrigatório), que caracteriza o tipo de sentença que se está considerando, na maioria das linguagens. Montadores para microcódigos permitem muitas vezes um número variável de mnemônicos, cada qual responsável pelo preenchimento de determinados campos do código de máquina. Muitos mnemônicos determinam o comportamento do montador em relação aos demais campos da sentença, sendo mais comum o caso de o mnemônico impor restrições ou condições sobre os tipos de operandos da sentença.

- operando, nem sempre obrigatório, mas cuja função é a de complementar a sentença introduzida pelo mnemônico. Dependendo do caso, há mnemônicos que exigem operandos de um determinado tipo, por exemplo, numérico, ou simbólico; outros permitem expressões como operandos, sendo suas exigências em alguns casos limitadas ao tipo do resultado da avaliação das expressões. Assim, há mnemônicos cujos operandos podem ser expressões cujo resultado seja um número, ou então um endereço absoluto, ou ainda um endereço relativo. Montadores há que permitem apenas formas simples de expressões, outros em que estas podem assumir as formas mais complexas, dispondo inclusive de pequenas bibliotecas de funções pré-definidas à disposição do programador. Em alguns casos há a necessidade de mais de um operando, usualmente separados entre si por meio de delimitadores.

- comentário: (normalmente opcional), que é muitas vezes separado do campo anterior por meio de um delimitador especial, usualmente inutiliza o restante do registro físico em que a sentença corrente está contida, para efeito de análise. Outras vezes, apresenta-se contido entre dois delimitadores especiais. Neste caso, somente o texto contido entre os dois delimitadores é ignorado pelo montador.

- delimitadores: em uma sentença, os vários campos são usualmente identificados posicionalmente. Em montadores cujo formato de entrada é livre, há uma certa dificuldade de interpretação dos campos quando ocorre a omissão de um ou mais campos opcionais. Por isso, existem montadores que impõem a utilização de separadores específicos para delimitação dos campos da sentença: apesar de os campos serem opcionais, os delimitadores não o são, neste caso. Outros montadores resolvem o problema introduzindo os diversos campos por meio de símbolos específicos. Outros ainda recorrem ao formato fixo, em que à posição física do campo no registro é associada a conotação apropriada. Outro tipo de delimitador é aquele que separa uma sentença da outra. Na grande maioria dos montadores, cada registro físico pode conter uma única sentença. Este caso, o final do registro opera como separador. Em alguns outros casos, delimitadores especiais se prestam à tarefa de marcar o início ou o final das várias sentenças. Neste caso, o mesmo registro físico pode comportar mais de uma sentença.

c) Muitos montadores oferecem ao programador o recurso da utilização de abreviações do texto fonte, através das macro instruções (ou simplesmente macros). Neste caso, são denominados macro-montadores. Em alguns casos, há macro montadores em que exclusivamente um conjunto de macros pré-definidas são disponíveis ao programador. Tais montadores tratam as chamadas destas macro como se fossem pseudo instruções especiais. Em muitos casos o programador pode definir suas próprias macros, ou utilizar macros pré-definidas. Nestas circunstâncias, pode-se identificar as macros de biblioteca e as macros locais. Macros locais são definidas e utilizadas, em um texto, e só têm sentido neste contexto. As macro de biblioteca são permanentes, e conforme sua origem, são intrínsecas do sistema, ou então definidas pelos usuários, que desta maneira são capazes de criar, manter e explorar suas bibliotecas particulares. Naturalmente, macro-montadores que permitem este tipo de flexibilidade interagem fortemente com o sistema operacional, uma vez que exploram diretamente os recursos do sistema de gerenciamento de dados do sistema.
Em sua maioria, os macro-montadores permitem que no texto fonte fornecido pelo programador, compareçam definições e utilizações de macros. A definição de uma macro do usuário é normalmente introduzida através de uma pseudo instrução especial, onde são declarados o nome da macro, e seus (eventuais) parâmetros formais. O macro-montador extrai, do texto desta pseudo, as informações mencionadas, armazenando-as para uso quando da utilização da macro, e também quando do tratamento do corpo da macro, onde referências aos parâmetros formais devem ser identificadas. Após o cabeçalho da macro assim declarado, vem normalmente o texto que compõem o corpo da macro e que se compõem de uma sequência qualquer de sentenças na linguagem de entrada. O final deste texto é marcado pela ocorrência de uma nova pseudo instrução especial que indica o encerramento da definição da macro. Alguns macro montadores permitem que, em qualquer ponto do corpo da macro, ocorram outras definições completas de macros, cujo tratamento é realizado, neste caso, apenas na ocasião da utilização da macro principal.

. Depreende-se que, em tais montadores, o tratamento sintático da linguagem de entrada seja um pouco mais rebuscado, já que esta exibe construções aninhadas ("self-embedded").
Uma vez disponível (pré definida ou declarada pelo usuário), a macro pode ser utilizada pelo programador na forma de chamadas de macros. Usualmente as chamadas de macro são identificadas pelo aparecimento do nome desta macro no campo de mnemônicos de alguma sentença e seus parâmetros de chamada figuram no campo dos operandos, separados entre si de modo semelhante ao utilizado nas instruções comuns para a separação de operandos. Alguns macro-montadores permitem que um dos parâmetros figure no campo de rótulo. Outros exigem que sejam demarcados os vários parâmetros, através de delimitadores. Estes delimitadores em alguns casos podem ser dinamicamente determinados pelo próprio programador, que assim tem a possibilidade de escolher o melhor delimitador para cada caso. Muitos macro montadores permitem que se utilize, como parâmetro das macros, textos gerais, de formato e comprimento quaisquer, que pode inclusive conter definições e chamadas de macros.
Outros limitam o formato dos parâmetros a textos simples, outros a expressões, e outros ainda, a variáveis ou constantes.

d) Quanto às pseudo instruções, pode-se afirmar que a grande maioria dos montadores apresenta um conjunto relativamente padrão. Com estas pseudos, o programador define o nome, o tipo do programa, a origem do código, informações de relocação, valores iniciais e variáveis, constantes, reservas de área de memória, utilização de registradores para o endereçamento relativo, sinônimos simbólicos (equivalências); delimita as definições de macros, define variáveis de tempo de montagem, altera essas variáveis de montagem, testa seus valores, efetua desvios condicionais e incondicionais de montagem, redefine os mnemônicos, indica o endereço de execução de um programa, indica os pontos de acesso de um programa, e os símbolos externos referenciados em um módulo, marca o final físico do texto fonte, controla as saídas do montador (listagem, código objeto, tabelas de símbolos, tabelas de referências cruzadas), etc..
Além destas pseudos, alguns montadores exibem outras, características da arquitetura particular da máquina para a qual o programa se destina. No entanto, apesar da sua grande variedade, todas elas apresentam funções muito semelhantes às das pseudos usuais.
Em relação às saídas de montadores, é possível afirmar de modo relativamente geral que o texto objeto é também formado de sequências de bloco de dados, cujo conteúdo reflete o significado do texto fonte em termos da linguagem de máquina. A grande maioria dos montadores produz textos objeto relocáveis ou absolutos, conforme o caso. A maior porção do texto objeto é formada de uma sequência de agrupamentos de dados representando a sequência de instruções em linguagem de máquina, usualmente acompanhadas de informações sobre relocação. Eventualmente este código contém indicações simbólicas acerca da utilização de referências a endereços pertencentes a outros módulos separadamente desenvolvidos. Para permitir que outros módulos referenciem endereços internos ao módulo corrente, um dicionário de pontos de acesso ("entry-points") deve constar do texto objeto. Completando o código objeto gerado pelos montadores, blocos de identificação, contendo informações gerais sobre o programa, e blocos delimitadores indicando o final físico do texto objeto fornecem dados complementares acerca do programa. Do ponto de vista sintático, embora haja muitas outras variantes, a sequência usual destes blocos é a seguinte: bloco de identificação - dicionário de pontos de acesso - dicionário de referências a símbolos externos - sequência de blocos de dados - bloco delimitador final. Um bloco de texto objeto tem por sua vez uma estrutura em que constam a seguinte sequência de informações: comprimento do bloco - tipo do bloco - dados - redundância. Em um bloco de dados em particular, os dados correspondem a grupos de informações em que constam o tipo de relocação a ser efetuada, o dado propriamente dito e eventualmente, uma referência a um símbolo externo.

2.2 - Considerações Semânticas

Um montador é um tradutor cuja missão é converter texto fonte em linguagem simbólica em textos objeto equivalente, em linguagem de máquina.

Do ponto de vista de implementação, os montadores exibem via de regra as seguintes partes funcionais:

a) Processamento de macros - encarregado de eliminar definições e referências às macros do texto fonte. Em alguns casos, isto é feito juntamente com as outras duas funções. Em outros o processamento de macros é realizado como um pré-processamento de texto fonte, separado portanto das atividades de montagem propriamente ditas.

b) Resolução de referências simbólicas - encarregada de converter rótulos e operandos simbólicos em endereços. Esta função é executada em alguns montadores como uma tarefa independente, correspondendo esta configuração à montagem em dois passos. A resolução dos endereços simbólicos é, no caso, efetuada no primeiro passo de montagem. Em outros montadores, ditos de um passo, a resolução de endereços acompanha a geração de código.

c) Tradução - encarrega-se de efetuar a conversão final das sentenças do programa fonte em código de máquina. Em montadores de dois passos, a tradução é feita como tarefa independente, e corresponde à função do segundo passo de montagem.

Em montadores de um passo, a tradução é feita, na medida do possível, para as sentenças que apresentem referências simbólicas resolvidas. As referências não resolvidas são neste caso armazenadas, sendo o código correspondente gerado na ocasião em que os símbolos indefinidos de que dependem forem totalmente resolvidos.

As atividades semânticas ligadas às funções do montador são, como se pode depreender, bastante simples, e a sua grande maioria consta de ações ligadas à pesquisa e manutenção de tabelas. Pode-se citar como principais atividades básicas da semântica dos montadores:

a) construção, manutenção e acesso à tabela de símbolos.

b) construção de arquivos de definições de macros.

c) expansão de macros referenciadas

d) construção e manutenção das tabelas de referências cruzadas.

e) acesso às tabelas de mnemônicos.

f) acesso às tabelas de tradução.

g) geração do código objeto.

h) interpretação de pseudo-instruções.

i) suporte de montagem condicional.

2.3 - Semelhanças com Outros Módulos de Software Básico

Foram detalhadas nos itens anteriores características sintáticas e semânticas dos montadores, e mencionados programas congêneres. Entre os programas de sistema, os ligadores e os relocadores ("linkers","locators", "linking loaders", "binders", etc.) são os módulos cujas estruturas mais se aproximam da dos montadores, sendo marcantes as similaridades seguintes:

a) os montadores tratam os símbolos do campo de rótulos ("labels") da mesma forma que estes módulos aos símbolos marcados como pontos de acesso ("entry-points").

b) símbolos que figuram como operandos para o montador são tratados analogamente aos símbolos marcados como externos nos ligadores.

c) tabelas de símbolos do montador têm seu análogo nas tabelas de resolução de referências externas nos ligadores.

d) devido a esta semelhança das linguagens de entrada, os programas têm consequentemente uma semelhança de lógica e de estrutura, permitindo a existência de ligadores e relocadores de um ou dois passos, como no caso dos montadores.

e) além da analogia sintática das linguagens de entrada, ocorre também uma analogia correspondente nas linguagens de saída, que no caso podem eventualmente ser as mesmas para os montadores, ligadores e relocadores.

Além dos ligadores e relocadores, outros módulos há que compartilham parcialmente com os ligadores e relocadores algumas semelhanças estruturais ou funcionais com os montadores, divergindo em outros aspectos. Entre elas pode-se citar: os editores, os carregadores, alguns interpretadores de comandos, alguns compiladores de linguagens simples, e muitas linguagens de entrada para programas de aplicação.

Para todos estes casos, uma ferramenta que venha a automatizar a tarefa de geração dos programas em questão mostra-se extremamente útil, e por esta razão, procurou-se criar tal ferramenta de modo tal que venha a simplificar a tarefa de produzir especificações para a geração automática daqueles módulos.

3. UMA LINGUAGEM PARA A ESPECIFICAÇÃO DE MONTADORES

Um sistema de auxílio à obtenção de montadores e programas similares deve, antes de mais nada, oferecer ao projetista meios através dos quais este possa definir formalmente, de maneira confortável, o comportamento daqueles módulos. O SPD dispõe, para isso, de uma meta linguagem apropriada, que foi criada como parte da linguagem LD. Esta linguagem permite ao projetista fornecer ao sistema uma definição funcional de um sistema digital completo (por exemplo, de um microprocessador), a partir da qual o SPD constrói automaticamente um programa simulador do sistema digital em questão. Através de uma descrição adicional análoga do módulo montador desta máquina, este módulo também é produzido pelo SPD, dando assim ao usuário final a possibilidade de programar sua máquina em linguagem de montagem ao invés de em linguagem de máquina. Se o montador for especificado adequadamente, o programa objeto produzido pode ser diretamente armazenado na memória simulada do processador, permitindo operar em modo "load and go", integrando desta maneira as operações do montador com as do simulador.

3.1 - Requisitos da Linguagem

Do estudo da sintaxe e da semântica dos montadores em geral, é possível estabelecer alguns requisitos sobre a linguagem através da qual estes podem ser especificados. Esta linguagem deverá apresentar algumas características essenciais, que ofereçam uma certa comodidade para:

a) manipulação de símbolos (identificadores), para a criação e manutenção de tabelas de símbolos e tabelas de atributos; para a manipulação dos mnemônicos da linguagem, através de consultas a conjuntos simbólicos e a conversão dos símbolos em códigos numéricos quando da tradução dos mnemônicos para códigos de máquina; para a criação e manutenção de conjuntos simbólicos que representam os nomes e parâmetros das macros do usuário.

b) conversão numérica, permitindo a interpretação de cadeias de caracteres que representam números em várias bases de numeração

e a respectiva conversão para os códigos binários correspondentes.

c) definição da sintaxe de entrada, através de construções que permitam especificar leituras de um novo registro, tabulações, saltos de campos, posicionamento em um dado campo, posicionamento em um dado registro (para facilitar a especificação de montadores de mais de um passo), mudanças de arquivos de entrada (para facilitar a definição de expansões de macros, ou operações com textos de entrada provenientes de mais de um arquivo).

d) definição da sintaxe de saída, através de construções que permitam definir as seqüências que geram listagens e códigos objeto: mudança de registro, tabulações, saltos de campos, geração de saídas numéricas e alfanuméricas, mudanças de arquivo, e preenchimento de memórias simuladas (para montadores "load and go").

e) definição de leis de formação repetitivas, para a especificação sintática de seqüências de construções do mesmo tipo, tanto nas entradas como nas saídas.

f) definição de leis de formação alternativas para a especificação sintática de construções de formatos diversos que podem ser escolhidas em determinados pontos das sentenças de entrada.

g) especificação de formas tentativas, com a finalidade de especificar construções cuja presença não seja obrigatória.

h) especificação de ações (semânticas) a serem executadas na ocasião em que uma dada construção sintática for encontrada no texto fonte, ou no caso em que tal construção não for encontrada (tratamento de omissões).

i) execução de ações semânticas incondicionalmente em pontos determinados pelo projetista.

A linguagem assim especificada deverá ser parte da LD, e os comandos por ela introduzidos serão utilizáveis em qualquer circunstância no texto LD. Optou-se por uma linguagem compacta, que lembra as especificações de formatos de entrada/saída das linguagens de programação usuais. Entretanto, em contraste com estas, a meta-linguagem adotada permite especificar simultaneamente a sintaxe e a semântica dos textos que define. Integrada na LD, esta linguagem permite a utilização, na especificação de montadores, de todos os mecanismos disponíveis para a definição de simuladores, o que torna simples a geração de montadores e simuladores que se complementem, e que ofereçam desta maneira ao usuário final um sistema de desenvolvimento para o (micro) processador desejado.

3.2 - Apresentação dos Recursos da Linguagem

Para dar suporte à especificação de montadores e programas congêneres, a LD foi ampliada através de uma série de declarações e comandos específicos:

a) declaração de tabelas - através destas declarações, são definidas as tabelas que o montador irá utilizar. São permitidas tabelas multidimensionais, cujos elementos são estruturados em campos. O comprimento dos campos é especificado pelo usuário, que pode declará-los como campos binários ou alfanuméricos, cujo comprimento é dado em bits ou caracteres, respectivamente. O aspecto de uma declaração de tabela é o seguinte:

```
TABLES: SYMBOLTABLE [1:100]
        (SYMB < 6 CHARACTERES>,
         ADDR < 16 BITS>,
         DEF  < 01 BITS>), ...
```

Nesta declaração, uma tabela de 100 elementos é especificada, cada qual exibindo um campo alfanumérico de 6 caracteres, um campo binário de 16 bits, e outro campo binário de um só bit. Tal declaração poderia ser utilizada para definir uma tabela de símbolos simplificada. A referência, no texto LD, ao campo ADDR do terceiro elemento da tabela seria feita através da especificação do campo, da tabela e do índice:

```
ADDR OF SYMBOLTABLE [3]
```

b) comandos de manipulação de tabelas - para criar e manter as tabelas declaradas, a LD dispõe de quatro comandos:

- Comando de inserção de elemento na tabela

```
INSERT dado IN campo OF tabela
```

- Comando de busca de um dado na tabela

```
FIND dado IN campo OF tabela
```

- Comando de remoção de um elemento de tabela

```
REMOVE dado FROM campo OF tabela
```

- Comando de alteração de um campo de tabela

```
REPLACE campo OF tabela BY dado
```

O comando INSERT cria um novo elemento na tabela, com um campo preenchido com um dado fornecido. Buscas em tabela não efetuadas através do comando FIND, que pesquisa um campo da tabela à procura de um elemento em que este campo tem um valor especificado. O resultado da busca retorna em um indicador pré declarado (DONE) que pode assumir valores TRUE (se a busca foi bem sucedida) ou FALSE (caso contrário). A posição da tabela em que o dado foi encontrado é acessível ao usuário através da palavra reservada FOUNDELEMENT, que deve substituir a lista de índices na referência ao elemento em questão da tabela. O comando REMOVE efetua também uma pesquisa, sendo que o elemento da tabela, cujo campo pesquisado for encontrado, é removido da tabela, o que também é informado ao usuário através do indicador DONE.

O comando REPLACE se destina a efetuar a manutenção propriamente dita da tabela efetuando alterações ou preenchimentos de campos especificados da tabela.

c) declarações de conjuntos - com a finalidade de especificar conjuntos simbólicos ou numéricos que não se alteram durante o processamento, estas declarações permitem ao usuário associar valores numéricos a cadeias de caracteres (permitindo por exemplo a especificação de regras de mapeamento de mnemônicos em códigos numéricos), e também especificar gamas de variação permitidas para números (para que seja possível restringir os valores assumidos por operandos numéricos, por exemplo).
Por exemplo:

```
SETS: MNEMONICOS / 0 ("LDA"),3("STA"),.../,
      INTEIROS / -128: +127 /, ...
```

Esta seria uma forma de declarar um conjunto de mnemônicos, simbólico, em que o mnemônico LDA tem associado o código numérico 0, STA, o valor 3, etc., e um conjunto numérico representando números de oito bits, que assumem valores entre -128 e +127.

d) comandos de acesso aos conjuntos - para que seja efetuada uma verificação de pertinência de um dado a um conjunto, é usado o comando FIND na forma seguinte:

```
FIND dado IN conjunto
```

Neste caso, o indicador DONE indica o resultado de busca, como no caso de pesquisas em tabelas, e o valor numérico correspondente ao dado pesquisado é colocado à disposição do usuário através de um símbolo pré declarado, FOUNDVALUE.

e) declaração de arquivos - utilizada para indicar ao SPD os nomes dos arquivos a serem utilizados para a entrada e saída do montador. Sua forma é muito simples, já que são deixados para os comandos de controle do sistema operacional as declarações dos atributos dos arquivos, irrelevantes no nível de referência em que estas declarações ocorrem. Distinguem-se arquivos exclusivamente de entrada, exclusivamente de saída ou de entrada e saída, através de construções do tipo seguinte:

```
INPUT FILES: F1, F2, ...;
OUTPUT FILES: F3, F4, ...;
IO FILES: F5, F6, ...;
```

f) declaração de especificações de entrada - através desta declaração, é possível fornecer à LD a especificação propriamente dita da sintaxe e da semântica da linguagem de entrada do montador que se deseja obter. Para tanto, esta declaração se apresenta com um formato relativamente elaborado em comparação com os das declarações citadas anteriormente. Cada especificação de entrada é uma sequência de especificadores mais elementares, cada qual podendo por sua vez vir ou não acompanhada de um modificador, que impõe, caso esteja presente, restrições à aplicação do especificador de entrada correspondente. Os modificadores, que precedem os especificadores de entrada são os seguintes:

- teste de condição - este modificador permite que uma variável ou um procedimento booleano seja testado, processando o especificador de entrada correspondente caso o teste tenha êxito. Assume a forma ? identificador booleano:

- salto - este modificador indica que o respectivo especificador de entrada deverá ser interpretado como especificação do texto a ser ignorado no programa fonte de entrada do montador. Caso seja necessário, pode figurar opcionalmente uma indicação do número mínimo e máximo de vezes que o salto deve ser realizado. A forma deste modificador é:

```
& (mínimo : máximo)
```

- repetição - permite estabelecer que o especificador de entrada associado deve ser interpretado várias vezes seguidas; o número de vezes também pode ser indicado caso seja necessário: * (mínimo : máximo):

- tentativa - textos opcionais na linguagem de entrada podem ser descritos através de especificadores precedidos deste modificador. Um identificador de procedimento pode ser utilizado opcionalmente para indicar uma ação a ser executada caso não seja encontrado o texto em questão no programa fonte fornecido como entrada. Formato: # procedimento opcional :

Os especificadores de entrada disponíveis na LD são os seguintes:

- Caracteres alfanuméricos quaisquer (podendo-se indicar o número de caracteres):

```
C , número
```

- Um caracter alfabético: A
- Um dígito decimal: D
- Um rótulo (identificador): L
- Um elemento pertencente a um conjunto simbólico declarado na LD: S . conjunto
- Um número inteiro qualquer: N
- Um número inteiro pertencente a um conjunto numérico declarado na LD:

```
N. conjunto
```

- Uma cadeia de caracteres: "cadeia"

Qualquer destes especificadores de entrada quando seguido de uma vírgula e do nome de um procedimento indica que ao ser encontrada, no texto analisado, a construção descrita, o procedimento indicado deverá ser executado como ação semântica.

Por exemplo: N . INTEIROS, GUARDA

Este especificador indica que a rotina GUARDA deve ser executada se for encontrado no texto fonte um número que pertença ao conjunto INTEIROS, declarado anteriormente.

O conjunto seguinte de especificadores de entrada designam ações de controle de operação do montador definido, podendo inclusive em alguns casos ser utilizados em especificações de saídas:

- mudança para o registro seguinte do arquivo: R
- execução incondicional de uma rotina semântica: E . rotina
- indicação do tipo de código de entrada/

saída: K . ASCII ou K . EBCDIC

- indicação da base de numeração a ser usada para inteiros: B. base
- indicação de um novo arquivo de entrada/saída: I . arquivo ou O . arquivo
- chamada de uma outra especificação de entrada/saída: F . identificador
- tabulação em uma dada coluna do texto: T . número
- posicionamento em um dado registro do arquivo: P . número

A partir dos especificadores elementares relacionados até aqui, é possível montar especificadores de entrada compostos:

- sequências: <especificadores; especificadores; ... >
- alternativas: [especificador | especificador | ...]

Com as definições acima, é possível fornecer um exemplo do uso da declaração de especificações de entrada:

```
INPUT SPECIFICATIONS: COMENT (T1;"*";R),
                      INSTR (T1;#:L;&:" ";
                      S.MNEMONICOS;
                      &:" ";[L/N]; R),
                      PROG (*:[ F.COMENT |
                      F.INSTR] );
```

Neste exemplo, não estão indicadas as ações semânticas, apenas a sintaxe de um montador rudimentar, em que o texto fonte é definido na especificação PROG como uma repetição indefinida de construções que assumem uma das formas descritas pelas especificações COMENT ou INSTR. No primeiro caso, trata-se de comentários, introduzidos por um asterisco na coluna 1, o que força a leitura de um novo registro. No segundo caso, trata-se de uma instrução. Um rótulo é opcional na coluna 1. Brancos são ignorados, e a seguir, um elemento do conjunto MNEMONICOS anteriormente declarado é obrigatório. Novamente brancos são ignorados, seguindo-se um operando simples em que é facultativo ao usuário o uso de um rótulo ou de um número. A seguir é lido um novo registro.

g) declaração de especificações de saída - análoga em formato à de especificações de entrada, esta declaração apresenta sequências de especificadores de saída, eventualmente precedidos de modificadores. Estes podem assumir as formas seguintes:

- repetição indefinida: *:
- repetição limitada: * (número):
- test de condição:
  ? identificador booleano

As conotações destes modificadores são análogas às dos correspondentes de entrada.

Os especificadores de saída disponíveis na LD são os seguintes:

- sequência de caracteres alfanuméricos contidos em um acumulador de entrada/saída pré declarado na LD:
  C . número de caracteres
- identificador contido no acumulador de entrada/saída: L
- número inteiro contido no acumulador de entrada/saída: N
- cadeia de caracteres: "cadeia"

Como no caso das entradas, é possível associar a cada uma destas saídas elementares uma ação semântica. Por exemplo:

```
N, SEMAN
```

efetua a saída de um número inteiro, executando a seguir o procedimento SEMAN. Além destas saídas elementares, são tratados como especificadores de saída os casos mencionados anteriormente como especificadores de entrada com funções de controle do montador definido.

É possível neste caso ainda criar saídas compostas, apenas na forma de sequências similares às entradas compostas correspondentes. Um exemplo desta declaração pode ser dado por:

```
OUTPUT SPECIFICATIONS: LISTA (E.INCREMENTA;
        T1; N; T10; C.80; R);
```

esta especificação de saída, quando chamada, lista uma linha de oitenta caracteres a partir da décima coluna da impressora, numerando-a nas primeiras colunas. O número de linha poderia ser incrementado a cada uso da especificação pela chamada incondicional da rotina INCREMENTA. Ao final da sua operação, é feita uma mudança de registro de saída. Naturalmente, estão omitidos no exemplo os mecanismos de colocação dos dados para a saída no acumulador de entrada/saída, o que deve ser feito através de rotinas semânticas.

h) comandos para disparar a interpretação das especificações de entrada e saída - através destes comandos é possível iniciar a execução das operações especificadas nas declarações em questão, dando assim início ao processo de montagem propriamente dito. Estes comandos assumem a forma seguinte:

```
PROCESS arquivo WITH especificação
```

O texto contido no arquivo será interpretado de acordo com as regras fornecidas pelo programador na especificação. Para disparar o montador rudimentar do exemplo anteriormente fornecido, sobre dados contidos no arquivo F1, o comando assumiria a forma:

```
PROCESS F1 WITH PROG
```

Em resumo, estas são os principais recursos oferecidos pela LD com a finalidade de facilitar a especificação de montadores. Todas as demais construções da LD estão disponíveis ao usuário, que pode utilizá-las em conjunto com as novas declarações e comandos, para uma especificação cômoda dos seus montadores, carregadores, ligadores, relocadores, editores ou outros programas estruturalmente semelhantes, no SPD.

4. CONCLUSÕES

Este artigo corresponde a uma primeira publicação da ampliação implementada do SPD no sentido de dar ferramentas de suporte à construção automática do software básico, apresentando algumas alterações em relação à proposta original [1], e correspondendo a uma descrição funcional do sistema que foi implementado, naturalmente sem detalhes de formalização da linguagem. Procurou-se, através de um estudo inicial dos montadores em geral, indicar a linha de raciocínio que levou à idealização do sistema tal como se apresenta, passando-se em seguida à apresentação da linguagem implementada. Cumpre citar que esforços desta mesma natureza constam na literatura [3,4], através de técnicas diversas das utilizadas neste projeto. Da parte do SPD, as alterações necessárias à incorporação dos novos recursos estão totalmente projetados, estando o sistema em fase avançada de implementação, com a maior parte dos recursos aqui descritos em funcionamento, alguns poucos em fase final de depuração, restando por implementar as declarações e manipulações de tabelas, o que foi postergado pela possibilidade de utilização de outros recursos alternativos já implementados no SPD. Prevê-se para breve a conclusão do trabalho, com o que o sistema estará disponível para ser utilizado pelos interessados.

5. REFERÊNCIAS

[1] - José Neto, J.
"SPD - Um Sistema Automático de Apoio ao Projeto e Desenvolvimento de Sistemas Digitais"
Tese de doutoramento-EPUSP-São Paulo, 1980

[2] - Donovan, J.J.
"Systems Programming"
Mc Graw-Hill, 1972

[3] - Mueller, R.A. and G.Johnson
"A Generator for Multiprocessor Assemblers and Simulators"
Proc. IEEE 64,6 - June 1976

[4] - Heath, J.R. and Patel, S.M.
"How to Write a Universal Cross Assembler"
IEEE Micro - August 1981