MAC-221 — Construção de Montadores
MAC-222 — Sistemas de Programação

1998 — Especificação da segunda fase do Montador p/ MIPS

Idéias gerais

Na segunda fase do montador para o processador RISC MIPS R2000 deve-se implementar um analisador sintático. Não haverá preocupação em montar os 'opcodes' das instruções (que deverá ser feito numa terceira fase). Como idéia geral, desejamos ser capazes de reconhecer como sintaticamente correto, qualquer arquivo assembly gerado pelo "cross-compiler" como será descrito oportunamente.

O analisador léxico

Nesta segunda fase deverá ser utilizado o analisador léxico gerado na primeira fase. Poucas alterações poderão ser necessárias. Os módulos já definidos, incluindo-se aquele gerado pelo flex e que implementa a função `yylex()` e o módulo necessário à manipulação da tabela de hashing serão usados como antes. O módulo `testlex` será omitido no analisador sintático, mas pode ser utilizado nalgum outro executável de forma a se obter uma listagem dos ítens léxicos que estão sendo sucessivamente remetidos ao analisador sintático.

Recuperação e relatório de erros

O programa deve ser capaz de se recuperar de um erro sintático, prosseguindo a análise sintática nas linhas subsequentes. Quanto à função `yyerror()` deve dar informações sobre onde ocorreu o erro sintático. Ao menos o número da linha em que foi detectado o erro (basta que `yylex()` conte os caracteres de mudança de linha) e o valor de `yytext` do último token lido deve ser impresso.

De fato, o arquivo fornecido `testsint.c` já possui uma implementação de uma tal função `yyerror()` e também tem uma função `main()` que inicializa a tabela de símbolos, lê os parâmetros da linha de comando e chama `yyparse()`. Este arquivo, como todos os demais que foram fornecidos, pode ser alterado.

Arquivos fornecidos

O diretórios (que são idênticos no momento)

`<http://www.ime.usp.br/~alair/mac221-98/SegundaFase/>`
`<http://www.ime.usp.br/~alair/mac222-98/SegundaFase/>`

contêm a especificação da segunda fase e arquivos auxiliares à confecção do analisador sintático. Um arquivo com todo o conteúdo do diretório pode ser encontrado em

`<http://www.ime.usp.br/~alair/mac221-98/SegundaFase.tar.gz>`
`<http://www.ime.usp.br/~alair/mac222-98/SegundaFase.tar.gz>`

Para ser extraído, basta usar o comando

```
tar xvzf SegundaFase.tar.gz
```

no linux[1].

Nesses diretórios são fornecidos os arquivos:

- `Tokens-2.5` com a especificação dos tokens;
- `Observacoes.txt` com várias observações[2] sobre o projeto.

Estes dois arquivos *fazem parte da especificação do projeto* e poderão ser alterados de forma a dar mais esclarecimentos porventura necessários.

São fornecidos ainda vários arquivos que *podem* ser usados na implementação do projeto:

- `symrec.h` com a especificação de uma tabela de símbolos;
- `symrechash.c` com uma implementação da tabela de símbolos através de uma tabela de hashing (com uma função de hashing simples);
- `initsymrec.c` com uma função que inicializa a tabela de hashing;
- `token2hash` um script AWK que processa o arquivo `Tokens-2.5` e gera um vetor de inicialização com as palavras reservadas a serem inseridas inicialmente na tabela de Hashing;
- `testsint.c` com a função `yyerror()` e a função `main()`;
- `token2bison` um script AWK que processa o arquivo `Tokens-2.5` e gera definições como `%token` e `%union` para o bison;
- `Makefile` um arquivo descritor das dependências entre os diferentes arquivos do projeto;
- outros arquivos/scripts AWK relacionados à primeira fase do projeto;
- `exemplos` um diretório que será comentado mais tarde.

A análise sintática

Todas as *instruções*, *pseudoinstruções* e *diretivas* deverão ser reconhecidas corretamente.

No caso das instruções, o reconhecimento sintático deve ser *estrito*, ou seja, todas as possíveis combinações de operandos devem ser reconhecíveis e nenhuma a mais. Todas as diretivas `.text`, `.data`, `.sdata`, `.rdata`, `.bss`, `.sbbs`, `.comm`, `.lcomm`, `.word`, `.byte`, `.half`, `.double`, `.ascii` e `.asciiz` também devem ter reconhecimento sintático *estrito*, bem como as pseudo-instruções `la` (load address), `li` (load immediate) e `move`.

Todas as demais pseudoinstruções e diretivas podem ter um reconhecimento sintático *relaxado*, ou seja, todas as combinações corretas de operandos devem ser reconhecíveis mas pode-se reconhecer como sintaticamente corretos operan-

[1] Quem usa o `DJGPP` pode fazer o mesmo já que o `DJGPP` possui o `tar` e o `gzip`.

[2] Algumas destas observações serão mais úteis quando for feita a geração de código, na terceira fase.

dos que de fato não o são. Na terceira fase, isto não trará problemas para estas pseudoinstruções e diretivas pois não serão implementadas a princípio.

Instruções com duas sintaxes possíveis

Várias instruções dos tipos `R3` e `J` podem ter uma segunda sintaxe. Alguns montadores MIPS admitem que um nome de instrução como `add` admita duas sintaxes possíveis. Isto inclui o montador GNU as.

```
add reg,reg,reg
add reg,reg,imm
```

No primeiro caso a instrução seria montada com um opcode (correspondente a `ADD_OP`) e na segunda com outro opcode (correspondente a `ADDI_OP`).

Num montador mais simples, estas instruções como `add` poderiam aceitar somente a primeira sintaxe e caso se quizesse a segunda sintaxe, deveríamos explicitamente escrever

```
addi reg,reg,imm
```

Neste projeto, para que se possa reconhecer como sintaticamente corretos quaisquer arquivos assembly gerados pelo cross-compiler, estas segundas sintaxes devem ser reconhecidas como corretas. Isto pode complicar um pouco a maneira de se fazer a análise sintática de instruções dos tipos `R3` e `J`. Mais detalhes devem ser vistos no ítem 3 do arquivo `Observacoes.txt`.

Documentação fornecida

Está à disposição no Xerox o apêndice A da primeira edição do excelente livro “Computer Organization & Design: The Hardware / Software Interface” de autoria de Patterson & Hennessy. Este apêndice contém os ‘opcodes’ das instruções do MIPS 2000 bem como sua sintaxe e *DEVE* ser consultado. Há alguns erros no apêndice e o arquivo `Observacoes.txt` e `Tokens-2.5` têm informações mais confiáveis quando houver discrepância. Uma terceira fonte de informação é o manual do SPIM que também está no xerox. Já no apêndice A há alguma informação referente ao SPIM.

Usando os arquivos fornecidos

Com os arquivos fornecidos, basta escrever os arquivos `analex.l`, `anasintdef.y` e `anasintregras.y` compatíveis. Rodando `make`, o `Makefile` fornecido é analisado e é gerado o executável `./testaanasint`. Se só estes arquivos forem implementados, basta entregar um arquivo-pacote `.tar`[3] (ou um `.zip`) dos três arquivos. Se os arquivos fornecidos forem alterados ou outros arquivos forem acrescentados, todos estes arquivos devem ser entregues neste pacote. Com o pacote entregue e extraído sobre um diretório contendo previamente os mesmos arquivos entregues na especificação da SegundaFase, deve-se ser capaz de gerar o executável, de nome `testaanasint` simplesmente rodando `make`.

Diretório ***`exemplos`***

[3]Um comando `tar cvzf meutar.tgz analex.l anasintdef.y anasintregras.y` monta o pacote.

Este diretório possui vários arquivos assembly (`.s`) para o MIPS. Possui também alguns arquivos C (`.c`) que podem ser cross-compilados de forma que se gerem os correspondentes arquivos `.s`. O arquivo `Makefile` presente neste diretório oferece regras de como fazer esta cross-compilação.

O "cross-compiler", disponível apenas em linux, pode ser encontrado em

`<ftp://ftp.linux.ime.usp.br/pub/alair/>`

juntamente com o simulador `spim`. (Já estão instalados na rede linux do IME.) Há também uma versão do simulador `spim` que roda em DOS.

Entrega

A entrega deste pacote deve ser feita de forma eletrônica em

`<http://panda.ime.usp.br>`

até o dia 19 de outubro.