Parte IV: Parser

Nesta quarta parte do nosso projeto você deve implementar o parser para a linguagem de montagem. O parser é usado para quebrar uma instrução da linguagem de montagem em suas partes constituintes: o rótulo, o operador e os operandos que compõem a expressão. Para completar esta etapa do projeto, você deve estar totalmente familiarizado com a linguagem de montagem (MACAL), apresentada na apostila feita pelo professor Fernando Mário de Oliveira, disponibilizada no paca.

Os arquivos mencionados abaixo podem ser baixados aqui. Você também vai precisar do buffer da parte II e da tabela de símbolos da Parte III do projeto, bem como do módulo error. ## Tipos de dados básicos

O arquivo asmtypes.h contém as declarações de tipos de dados que serão usados pelo nosso programa, a saber:

- Operator: representa um operador da linguagem de montagem; contém o nome do operador, seu opcode e os tipos de seus operandos.
- OperandValue: o valor de um operando, que pode ser um número, um rótulo, uma string, ou o número de um registrador.
- Operand: um operando, composto de seu tipo e valor.
- Instruction: uma instrução após o parsing; contém sua posição (que será determinada durante a montagem do programa), o número da linha na qual ocorre no arquivo de entrada, seu rótulo, operador e operandos.

O arquivo asmtypes.c implementa funções para criar e destruir os tipos acima. As implementações estão completas e vocês não precisam fazer nada aqui.

Tabela de operadores

O arquivo opcodes.h contém macros com os opcodes de todas as instruções do MACAL. Os arquivos optable.h e optable.c implementam uma tabela de símbolos que associa operadores (instâncias de Operator) a strings; a tabela é usada para devolver um operador dado seu nome.

Novo header para a tabela de símbolos

O header para a tabela de símbolos, stable.h, mudou um pouquinho: alguns campos novos foram adicionados à union EntryData. Vocês não precisarão alterar a suas implementações feitas na Parte III.

O parser

Seu maior trabalho nesta parte do projeto é escrever o arquivo parser.c que implementa a função:

```
int parse(const char *s, SymbolTable alias_table, Instruction **instr,
        const char **errptr);
```

A função recebe os seguintes argumentos:

- s: uma string contendo uma linha de código de montagem que será analisada.
- alias_table: uma tabela de símbolos que associa operandos (instâncias de Operand) a strings. O programador pode usar o pseudo-operador IS para associar operandos a rótulos. Por exemplo, a instrução a IS $0 associa o registrador $0 ao rótulo a. Esta tabela é usada para guardar essas associações.
- instr: aponta para o início da lista ligada de instruções (instâncias de Instruction). Cada nó da lista contém:
    - rótulo (=*label) associado à instrução;
    - operador (=*op) da instrução e
    - os seus operandos (=opds)

Se a linha de código sendo analisada contém uma instrução válida, essa deverá ser inserida como um novo nó da lista ligada.

Se a instrução é válida então parse() devolve um valor não-zero. Em caso contrário, devolve zero e *errptr é a posição de s em que um erro ocorreu e a função set_error_msg() deve guardar uma mensagem de erro apropriada.

Exemplo

Suponha que alias_table tenha o registrador $0 associado à string a. Considere a chamada:

```
const char  *errptr;
Instruction *instr;
parse("loop  MUL  a,a,2  * Faz multiplicacao\n", alias_table,
      &instr, &errptr);
```

Após o retorno, *instr deve contém:

- label == "loop";
- op é o operador que corresponde a MUL, devolvido pela função optable_find();
- opds são os operandos: os dois primeiros são o registrador $0, com tipo REGISTER, o terceiro é o número 2, com tipo NUMBER_TYPE e
- pos = 0 e lineno = 0.

O operador MUL espera três operadores. Os primeiros dois devem ser registradores; o terceiro é um IMMEDIATE, ou seja, pode ser um registrador ou um número de um byte (veja o arquivo optable.c para entender o que está acontecendo). Se há discrepância entre os operadores esperados e os fornecidos, então a instrução é inválida e parse() acusa erro com mensagem apropriada.

Note nesse exemplo como o rótulo a foi substituído pelo registrador $0, a ele associado na alias_table. Observe ainda que, caso não houvesse rótulo na linha dada, teríamos label == NULL após o retorno. Finalmente, note que o comentário, que começa no *, é ignorado.

Programa de teste

Além de escrever a função parse() você deve escrever um programa para testá-la. Seu programa, chamado parse_test.c, deve receber na linha de comando o nome de um

arquivo. Ele deve ler cada linha do arquivo e fornecê-la à função parse(). Se a função parse() retorna com sucesso, seu programa deve imprimir a linha de código e o conteúdo da estrutura instr devolvido pela função. Se ocorreu um erro, seu programa deve mostrar onde e imprimir uma mensagem de erro apropriada.

Quando a instrução é o pseudo-operador IS o programa deve associar o valor do operando ao rótulo passado, inserindo o par correspondente na alias_table. Se o rótulo já foi associado, você deve produzir uma mensagem de erro. Essa é a única instrução que deve ser interpretada pelo parse_test.

Considere por exemplo o seguinte arquivo de entrada:

```
* Teste
a      IS      $0
start  ADD     a,a,1
       MUL     a,$2,$3     * Multiplica.
       JMP     start
       DIV     a,2
```

A saída do seu programa deve ser algo como:

```
line     = a      IS      $0
label    = "a"
operator = IS
operands = Register(0)

line     = start  ADD     a,a,1
label    = "start"
operator = ADD
operands = Register(0), Register(0), Number(1)

line     =        MUL     a,$2,$3
label    = n/a
operator = MUL
operands = Register(0), Register(2), Register(3)

line     =        JMP     start
label    = n/a
operator = JMP
operands = Label("start")

line 6:       DIV     a,2
                      ^
expected operand
```

Rótulos válidos

Em seu programa você vai precisar decidir se um rótulo dado é válido ou não. Lembre-se da definição: um rótulo válido começa com uma letra ou um underscore e pode conter apenas letras, underscores e números. Note também que o nome de um operador não é um rótulo válido!

Arquivos a serem baixados

Para está parte vocês precisarão baixar os arquivos:

- asmtypes.h e asmtypes.c (nada a fazer);
- optable.h e optable.c (nada a fazer);
- opcodes.h (nada a fazer);

- parser.h (nada a fazer);
- stable.h qu econtém atualizações;
- error.h e error.c (nada a fazer);
- mactypes.h

Baixe-os daqui. Você deverá fazer o parser.c e parser_test.c.