|  |  |
| --- | --- |
|  |  |

UNIVERSIDADE ESTADUAL PAULISTA “JÚLIO DE MESQUITA FILHO” – UNESP

INSTITUTO DE GEOCIÊNCIAS E CIÊNCIAS EXATAS

**Dalton Varussa de Oliveira Lima - 141152257**

**Lucas Augusto Pinheiro - 141150688**

**Stefany Lacroux - 141150955**

**Tadeu Martines - 141152214**

**Victor Henrique Ribeiro - 141153237**

TRABALHO DE ARQUITETURA

PARSER MIPS ASSEMBLY LANGUAGE

Rio Claro

2015

**Dalton Varussa de Oliveira Lima**

TRABALHO DE ARQUITETURA

PARSER MIPS ASSEMBLY LANGUAGE

**Dalton Varussa de Oliveira Lima - 141152257**

**Lucas Augusto Pinheiro – 141150688**

**Stefany Lacroux - 141150955**

**Tadeu Martines - 141152214**

**Victor Henrique Ribeiro - 141153237**

**Lucas Augusto Pinheiro**

**Victor Henrique Ribeiro**

**Stefany Lacroux**

**Tadeu Martines**

Trabalho apresentado como requisito parcial para aprovação na disciplina Arquitetura de Computadores do curso de Ciência da Computação, UNESP.

Prof. Dr. Maurício Acconcia Dias

Rio Claro

2015

SUMÁRIO

|  |
| --- |
|  |
| 1 introdução 4  2 desenvolvimento 4  3 funcionamento do programa 5  4 Tratamento de labels 6  5 Conclusão 7  6 Detalhes técnicos 7  7 Bibliografia 8 |

1 - INTRODUÇÃO

Um parser é um instrumento de ampla utilização na ciência da computação, sendo responsável por analisar sintaticamente uma sequência de entrada, determinando sua estrutura de acordo com regras previamente estabelecidas. Essa análise faz parte de um processo, conhecido como compilação, onde o computador transforma em linguagem legível para si (linguagem de máquina), uma determinada sequência de instruções feitas por um programador, em linguagem de alto nível.

O objetivo deste trabalho é implementar um parser em linguagem C, capaz de interpretar um arquivo de entrada em linguagem Assembly, com comandos de um processador MIPS, e gerar um arquivo de saída em binário com esses comandos.

**2 - DESENVOLVIMENTO**

O trabalho foi desenvolvido com o auxílio de um sistema de controle de versão, o GIT. O trabalho, portanto, pode ser encontrado, com todos os seus detalhes, e diferentes versões no seguinte endereço: https://github.com/daltonbr/MIPS.

Seguindo as orientações do professor, e visando a entrega do trabalho em tempo útil, foram criadas algumas restrições para o funcionamento do programa. Tais restrições são:

* O arquivo de entrada em Assembly não pode conter comentários.
* O arquivo de entrada deve ser formatado conforme especificado na seção “***3 – Funcionamento do programa***”.
* O arquivo que será usado para conversão deverá estar na mesma pasta do arquivo executável, mais detalhes em “***3 – Funcionamento do programa”***.
* Os *Labels* presentes no arquivo Assembly e no Binary serão tratados da forma mais próxima ao funcionamento do processador MIPS, maiores detalhes sobre esse tratamento se encontram na seção referente a *Labels.*

**3 - FUNCIONAMENTO DO PROGRAMA**

O programa recebe como parâmetro de entrada dois arquivos de texto (.*txt*). Um em linguagem Assembly e outro em binário (ambos contendo instruções de um processador MIPS). Ambos os arquivos devem estar na mesma pasta do arquivo executável (main.exe):

* Insira o arquivo .*txt* nodiretório criado*,* renomeando-os para *assembly.txt, e binary.txt* **obrigatoriamente.**

Após a execução do arquivo main.exe, dois arquivos de saída serão gerados na mesma pasta do arquivo principal: outputBinary.txt e outputAssembly.txt.

Além disso, o arquivo assembly.*txt* deve seguir uma formatação pré-definida, especificada a seguir:

* Comentários não são aceitos.
* Um comando Assembly deve ser procedido de um único espaço em branco. Por exemplo:

*add $t0,$s0,$s1*

* Registradores e constantes devem ser separados somente por vírgulas. Nenhum espaço adicional deve ser colocado após vírgulas.
* Cada instrução Assembly deverá estar em uma linha separada.
* Linhas vazias não são aceitas.

Caso a formatação acima não seja seguida, o programa pode não funcionar da maneira correta.

O arquivo gerado a partir do binário (*binary.txt)* irá “traduzir” somente os valores numéricos destes labels. Isto se dá pela natureza com que estes arquivos são originalmente construídos, já que nesta parte faremos uma *engenharia reversa* nos comandos *MIPS*. Este tratamento de *labels* é explicado em maiores detalhes na próxima seção.

**4 – TRATAMENTO DE LABELS**

Os *labels* foram implementados tentando ao máximo emular o tratamento dado pelo processador *MIPS*. Faremos algumas considerações sobre eles:

* *Labels* devem ser únicos (ao menos na declaração)
* Uma linha com *label* deve conter somente ele (instruções devem ser colocadas na linha de seguinte).
* No início do algoritmo, temos uma função ( ***fillJumpAdressTable()*** ) que “varre” o arquivo *assembly.txt* em busca de todos os *labels* do arquivo, e a partir deles criam uma tabela (matriz de *structs*) que guarda o nome de todos os *labels* presentes no arquivo de entrada, bem como suas respectivas linhas de origem. Posteriormente estes valores são usados para calcular a quantidade de linhas que deverão ser “puladas” ou retrocedidas para calcular os desvios de rotina.
* As chamadas de *label* dentro de funções não podem conter espaços vazios entre as vírgulas.
* As funções ***bne*** e ***beq*** quando convertidas para binário, exibem no ***immediate*** a quantidade linhas que devem ser “puladas” ou retrocedidas para encontrar o *label* de destino. Para tanto o algoritmo calcula a diferença de entre o ***Program Counter*** (**PC**, que basicamente armazena a linha da instrução atual) e a linha efetiva do destino do *label*.
* As funções ***j*** e ***jal*** por outro lado simplesmente exibem no campo ***address*** o valor absoluto da linha onde se encontra o *label* de destino.

5 - CONCLUSÃO

Esta primeira parte do trabalho é fundamental para os futuros desenvolvimentos deste simulador de processador MIPS.

O trabalho em grupo também só foi possível graças ao uso da plataforma git / github, que facilitou muito o trabalho em grupo, descentralizando as atividades.  
Implementar este programa exigiu conhecimentos aprofundados em linguagem C, bem como um conhecimento das instruções básicas do MIPS. Foi fundamental entender como estas são projetadas, bem como os desvios de rotina (labels) e a forma que os registradores operam neste conjunto. Domínio de sistemas de bases numéricos como binária, hexadecimal e decimal são fundamentais para o desenvolvimento e compreensão deste projeto.

6 – DETALHES TÉCNICOS

Os arquivos fonte principais são: *main.c*, *mips.h* e *filemanager.h,.*Existem arquivos secundários de documentação nas demais pastas internas.   
Todos estes arquivos podem ser encontrados e baixados gratuitamente em:

<https://github.com/daltonbr/MIPS>

Para compilar utilizamos e testamos em dois compiladores diferentes:

* Microsoft C/C++ Optimizing Compiler Version 19.00.22816 for x86  
  Encontrado junto ao Microsoft Visual Studio 2015.
* MinGW 4.9.2, baseado no GCC  
  http://www.mingw.org

7 - BIBLIOGRAFIA

Patterson, David A.; Hennessy, John L. Organização e Projeto de Computadores. 3 ed. Rio de Janeiro: ELSEVIER, 2006.