Relatório do Trabalho 1- Codificação e Decodificação *Lempel-Ziv (Versão LZ78)*

Bruno Andreghetti Dantas
Universidade de Brasília
brunoandreghetti@gmail.com

Sara Gomes Cardoso
Universidade de Brasília
saragcard@gmail.com

Abstract

This report discusses the implementation of the Lempel-Ziv compres- sion routine, version 78, implemented using Assembly MIPS in the MIPS Assembly and Runtime Simulator (MARS).

1. Objetivos

Este trabalho tem como objetivo introduzir o contato com a linguagem Assembly MIPS e familiarizar a aplicação eficiente e otimizada de metodologias de projetos [1]. Pretende-se utilizar os princípios de projeto da arquitetura MIPS, que envolvem a simplicidade como meio de favorecer a regularidade; a redução de tamanho implicando em aumento de velocidade e a agilização de casos comuns [2].

2. Introdução

A arquitetura MIPS é baseada na filosofia RISC (Reduced Instruction Set Computer) e seu modelo de programação é similar ao de outras arquiteturas desenvolvidas por volta dos anos 80 [2].

Os principais objetivos de projeto dessa arquitetura são a maximização do desempenho, a minimização do custo e a redução do tempo de projeto.

A fim de cumprir-se esses objetivos, a arquitetura MIPS tem 3 princípios:

- Simplicidade favorece regularidade.

  Para isto, a aritmética MIPS consiste em instruções de, na maioria dos casos, 3 operandos com ordem fixa, o que simplifica o hardware.

- Menor significa mais rápido.

  Uma grande quantidade de registradores pode aumentar o tempo de ciclo de clock devido ao aumento da distância física entre sinais eletrônicos. Por esse motivo, o MIPS possui apenas 32 registradores, cada um com 32 bits.

- Agilize o caso mais comum.

  As funções nativas suportam operações aritméticas simples com registradores e imediatos, constantes declaradas sem a necessidade de um registrador para serem utilizadas na instrução em questão.

A compressão de dados baseada em dicionário consiste na substituição da ocorrência de uma sequência de símbolos específica em um arquivo pela referência à sua ocorrência anterior [4].

Abraham Lempel e Jacob Ziv desenvolveram os algoritmos de compressão LZ77 e LZ78 em 1977 e 1978 respectivamente. Na compressão de arquivos moderna, eles se enquadram nas duas classes principais de algoritmos de compressão baseados em dicionário [3].

O algoritmo LZ78 contém inicialmente um dicionário vazio, que deve ser construído ao longo da compressão. A medida que é lido caractere por caracter do arquivo, cada nova sequência de caracteres é incorporada ao dicionário e atribuída um código. As sequências que já estão presentes no dicionário são substituídas pelo seu código no dicionário. Uma de suas características é o lento crescimento do dicionário, tornando-o realmente útil apenas após uma grande leitura de informação [3].

O problema de algoritmos baseados em dicionários é a memória requisitada, pois a demanda de memória é dependente da quantidade de dados lidos do arquivo e das sequências de armazenamento no dicionário. Além disso, há o problema de espaço de endereçamento, já que os códigos de entrada no dicionário crescem juntamente a ele. Para resolver esse problema, pode-se congelar o dicionário; esvaziá-lo, separando o arquivo em blocos comprimidos separadamente ou usar uma política de esvaziamento comum ao compressor e ao descompressor [3].

3. Materiais e Métodos

Para este trabalho, utilizou-se a IDE Mars (MIPS Assembler and Runtime Simulator), desenvolvida pela Universidade de Missouri, para implementar e simular os códigos do compressor e descompressor de arquivos. Essa plataforma permitiu também a comparação de eficiência dos procedimentos para diferentes arquivos testes.

3.1. Compressão

O algoritmo implementado para compressão consiste em uma leitura de um caractere por vez no arquivo original, seguido dos seguintes passos:

- Armazena-se o primeiro caractere lido a um registrador e supõe-se que o índice dele seja 0, criando-se assim um par índice-caractere temporário.
- Compara-se o par temporário aos pares existentes no dicionário, que é inicializado sem conteúdo. A comparação é realizada primeiramente para o índice e em seguida, o caractere.
- Caso seja encontrado um par igual ao par temporário, substitui-se o índice deste pelo índice do par encontrado no dicionário e o próximo caractere do arquivo substitui o caractere anterior.
- O processo de comparação continua para o novo par temporário a partir do índice em que estava e continua até o fim do dicionário.
- Ao fim do dicionário, o par temporário é acrescentado ao mesmo e escrito no arquivo comprimido.
- Esse procedimento é realizado até que se termine a leitura do arquivo.

Dessa forma, para o exemplo:

$$aaababbbaaabaaaaaaabaabb$$

A compressão ficaria:

$0 < a > 1 < a > 0 < b > 1 < b > 3 < b > 2 < a > 3 < a > 6 < a > 2 < b > 9 < b >$

Sendo que os valores entre ¡¿ são caracteres representados com 8 bits e, na nossa implemenatação, os índices são representados também com 8 bits (mais sobre isso na seção 3.3).

O dicionário ficaria:

0: 0,
1: 0, a
2: 1, a
3: 0, b
4: 1, b
5: 3, b
6: 2, a
7: 3, a
8: 6, a
9: 2, b
10: 9, b

O arquivo gerado pela compressão é denominado ”out.lzw”e o gerado pelo dicionário, ”dict.txt”. Eles ficam em uma pasta ”out”no mesmo diretório do simulador MARS.

3.2. Descompressão

O algoritmo implementado para descompressão consiste em uma leitura de um símbolo por vez no arquivo comprimido, seguido dos seguintes passos:

- Armazena-se o primeiro símbolo lido a um vetor de índices;
- Armazena-se o segundo símbolo lido a um vetor de caracteres;
  - Imprime-se o caractere;
- Caso o próximo símbolo lido seja zero, o procedimento continua o mesmo.
- Caso contrário, armazena-se os símbolos lidos em seus respectivos vetores e adiciona-se um novo vetor para armazenar uma sequência de caracteres que será formada.
  - Armazena-se o último símbolo lido ao vetor de sequência de caracteres.
  - O último índice obtido corresponde a posição no vetor de índices do próximo índice a ser analisado. Caso seja zero, basta armazenar o caractere correspondente do vetor de caracteres no vetor de sequência de caracteres e imprimir o vetor de trás para frente.
  - Caso contrário, a busca permanece.

  O procedimento é repetido enquanto o arquivo não acabar.

Ao final do procedimento, é gerado um arquivo texto ”text.txt”na pasta ”out”que encontra-se no mesmo diretório do simulador MARS.

3.3. Dicionário

Restringiu-se o dicionário para comportar apenas 255 índices de 1 byte cada a fim de simplificar a implementação do algoritmo, que originalmente gera um desalinhamento dos bytes dos índices na compressão de arquivos. Esse congelamento do dicionário é responsável também por aumentar a velocidade de execução do programa pois mantém constante o custo da busca no dicionário a partir do momento em que ele atinge seu limite.

Os efeitos esperados por essa simplificação envolvem uma perda de desempenho na compressão de pequenos arquivos e também na de arquivos muito grandes, em que não há repetição das sequências iniciais nas porções do fim do arquivo.

O programa de compressão exporta junto ao arquivo comprimido um dicionário dict.txt, onde é possível observar os pares índice-valor utilizados para a compressão

4. Executando o programa

Para execução dos procedimentos de compressão e descompressão, deve-se realizar os seguintes passos:

- Para abrir o simulador MARS por linha de comando no terminal, deve-se usar o seguinte comando

```
java -jar mars.jar
```

- Para comprimir um arquivo

```
java -jar mars.jar lz78-compress.asm
```

- Para descomprimir um arquivo

```
java -jar mars.jar lz78-decompress.asm
```

- Após escolha da operação, o programa perguntará o nome do arquivo a ser comprimido ou descomprimido. Deve-se inserir por linha de comando o nome completo do arquivo (nome do arquivo mais o diretório em que ele se encontra), caso o arquivo tenha diretório comum ao simulador, basta inserir o nome do arquivo. Em seguida, deve-se pressionar a tecla ”Enter”do teclado.

5. Resultados

Uma vez que a execução do algoritmo é bastante demorada, foram utilizados arquivos de teste anexados junto ao trabalho na pasta ”data”. Os resultados obtidos na compressão foram dispostos na Tabela 1.

Arquivo	log1.txt	log2.txt	log3.txt	log4.txt
Tam. original	10934	15226	49231	11330
Tam. comprimido	8054	11328	40050	8950
Variação (%)	26,34	25,6	18,64	21

Tabela 1: Comparativo de tamanho em bytes dos arquivos originais e comprimidos

Utilizando-se a ferramenta Instructions Statistics do MARS (mostrada na Figura 1), verificou-se os percentuais de utilização das instruções do tipo ALU, Jump, Branch, Memory e Other durante as chamadas dos procedimentos de compressão e descompressão. Para diversos arquivos diferentes, os valores obtidos foram os mesmos, registrados na Tabela 2:

6. Discussão e Conclusões

O algoritmo de Lempel-Ziv se mostrou bastante competente. Apesar das simplificações feitas na implementação, o algoritmo continuou obtendo taxas de compressão razoáveis. Sua principal falha está no tempo de execução,

Parâmetro	Compressão	Descompressão
ALU	34%	55%
Jump	10%	5%
Branch	32%	10%
Memory	13%	24%
Other	11%	7%

Tabela 2: Percentual de utilização das instruções tipo ALU, Jump, Branch, Memory e Other

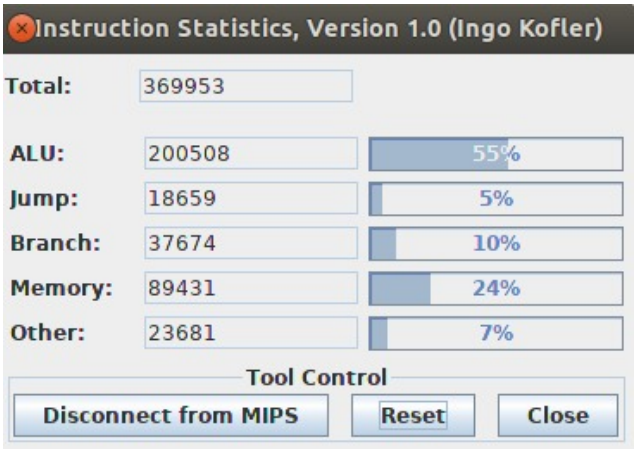

Figura 1: Exemplo de percentual de utilização das instruções tipo ALU, Jump, Branch, Memory e Other para o procedimento de decodificação

que deixa bastante a desejar. Além disso, o algoritmo necessita de bastante memória RAM, o que pode ser uma limitação grave em máquinas com poucos recursos.

A compressão de arquivos com mais cadeias de repetições de um mesmo caractere torna o procedimento mais eficiente, uma vez que cada item escrito no arquivo comprimido contenha mais informação sobre o arquivo original.

Para arquivos pequenos, o algoritmo se mostrou ineficiente, muitas vezes até mesmo expandindo o arquivo. Isso se deve ao fato de que a implementação feita utiliza índices de tamanho fixo e, portanto, desperdiça espaço em disco na representação dos primeiros índices do dicionário.

De um modo geral, embora a implementação realizada tenha sido sub-ótima devido às questões referidas acima, pode-se dizer que o objetivo do trabalho foi cumprido no que diz respeito ao aprendizado do Assembly MIPS e da familiarização com a filosofia RISC, tendo sido feitas até mesmo pequenas bibliotecas com *macros* para facilitar o trabalho de futuras turmas da disciplina.

Referências

[1] F. de Barros Vidal. Laboratório 1, 2018. Roteiro do Laboratório.

[2] F. de Barros Vidal. Linguagem de máquina, 2018. Slide de aula 5.

[3] A. Lempel and J. Ziv. Compression of individual sequences via variable-rate coding, 1978. IEEE Transactions on Information Theory.

[4] S. SC. Data compression. LZ78.