Desenvolvimento de Aplicações em Assembly MIPS
Organização e Arquitetura de Computadores
Turma B Data: 06/05/2018

Cristóvão Bartholo Gomes
Matrícula: 14/0135081
UnB - Universidade de Brasília
Campus Darcy Ribeiro, Brasília-DF
cristovao@live.com

Guilherme de Castro Ribeiro
Matrícula: 14/0142151
UnB - Universidade de Brasília
Campus Darcy Ribeiro, Brasília-DF
guilherme2260@hotmail.com

Objetivos

Este relatório visa abordar uma avaliação da implementação e desempenho de um software operando sobre uma arquitetura de ISA do tipo Mips. A aplicação é uma implementação do algorítmo de codificação e decodificação Lampel-Ziv (Versão LZ-78), que é um método de compressão de dados sem perdas. E através de ferramentas de verificação do comportamento de execução, realizar um mensuramento da eficiência do modelo aplicado. Este trabalho foi desenvolvido na primeira metade do semestre de 2018 para complementar os estudos do curso de Organização e Arquitetura de Computadores da Universidade de Brasília.

1. Introdução

Em ciência da computação, a compressão ou compactação de dados é extremamente valorizada, baseando-se na codificação da informação de modo que ela possa ser representada em um número menor de bits do que a representação original, na compressão sem perdas isso é feito através da redução de redundâncias. A compressão é útil pois permite que se use menos recursos para gravação e transmissão de dados, representando uma redução em custos de recursos para armazenação, como a otimização do uso da capacidade de HD's, e possibilitando maior velocidade de transferência de dados, otimizando serviços de streaming, por exemplo. Porém, em compensação, exige-se um esforço computacional na compressão e descompressão da informação. [2]

O algorítmo Lampel-Ziv78 provê a compressão sem perdas removendo redundâncias através da substituição de dados recorrentes por referências a um dicionário criado baseado no fluxo de dados providos pela informação que se deseja comprimir[3] [1].

1.1. Algorítmo

Um pseudo-código[3] representando o algoritmo de codificação está a seguir:

```
cadeia :=   #inicializa cadeia de tamanho 0
D.insere(cadeia)
enquanto c := leia.novo.caractere()
  se D contém cadeia concatenada com c
    cadeia := cadeia concatenada com c
  senão
    print.na.saida([D.código(cadeia),c])
    D.insere(cadeia concatenada com c)
    cadeia =”
  fim se
fim enquanto
print.na.saida([D.código(cadeia),”])
```

Nele, é utilizado um dicionário (chamado de D) para arquivar as sequências de caracteres presentes no arquivo. Ao ler um caractere do arquivo, o mesmo é procurado em D para verificar se já está presente. Caso esteja, o próximo caractere será lido e concatenado com o anterior, e novamente será realizado a verificação de existência em D, mantendo esse processo até que a sequência concatenada não esteja presente no dicionário. Quando a nova sequência é encontrada, é gerado um par com o código da sequência e o seu último caractere e é introduzido ao dicionário.

Para a decodificação, segue-se uma lógica similar. Mantendo o algoritmo de atribuição de códigos às sequências do processo de codificação, é iniciado o procedimento com o dicionário apenas com o código para a cadeia vazia, e prossegue-se lendo os pares de código e último caractere concatenado. Para cada par lido, é emitido na saída a sequência presente no dicionário, concatenada com o caractere salvo.

Seguindo essa lógica como modelo, o programa em As-

sembly MIPS foi desenvolvido para alcançar-se os objetivos do projeto.

2. Materiais e Métodos

A implementação foi feita na plataforma Mars, que é implementada em Java. A número da versão do software é 4.5, e a versão do Java utilizada é 1.8.0.151. O programa foi implementado e testado nos sistemas operacionais Ubuntu 14.04.5 LTS (Linux) e Windows 7 Ultimate.

2.1. Dicionário

A implementação e a analáse dos dados foi feita de modo a contornar o problema do crescimento rápido do dicionário.

A implementação do algorítmo foi feito de modo que o dicionário impeça a inserção de novos valores a medida que novos dados vão sendo adquiridos, portanto o dicionário é fixo a partir do momento em que ele atinge esse máximo. Isso supõe que é mais vantajoso utilizar a redundância global ao invés da local, portanto preve-se que o desempenho em arquivos com dados de redundância homogênea seja melhor do que aqueles cujo tipo redundância varie ao longo da informação. O tamanho máximo do dicionário escolhido foi de 12800 bytes, como cada elemento (código ou caractere) comporta 32 bits (uma word), o dicionário comporta 400 elementos.

Esse valor foi escolhido como forma de ponderar o tempo de execução e a qualidade da compressão, entretanto esse valor poderia ser alterado, supondo que o foco do software desenvolvido seja para arquivos pequenos de até aproximadamente 100Kb (cem KiloBytes) por exemplo, provavelmente seria mais vantajoso aumentar o limite do dicionário já que menos dados serão processados.

2.2. Abordagem dos dados

Uma abordagem de buffers foi utilizada de modo a lidar com arquivos arbitrariamente grandes. Na compactação temos a leitura, byte a byte, do arquivo a ser compactado e a execução do algorítmo de compactação. A medida que os dados são lidos e processados a saída é colocada em um buffer até que o número de bits presentes nele seja múltiplo de oito (conjunto de bytes), então os dados do buffer de saída são adicionados ao arquivo de saída, o buffer de saída é limpo e o processo se repete. O processo inverso ocorre na descompactação.

2.3. Procedimentos

A compressão de dados é feita em duas etapas. A primeira etapa é responsável pela execução do algorítmo e a segunda etapa é responsável pela formatação dos dados de saída.

Na primeira etapa a saída gerada é salva no arquivo "lzw.temp", cada código e cada caractere apresenta 32 bits (uma word). Ao terminar-se a primeira etapa, calcula-se a quantidade mínima necessária para a codificação dos índices dos elementos da tabela e na segunda etapa o arquivo é reescrito na saída.

Agora formatado com o número correto de bits para representar a tabela e oito bits para os caracteres, ao final os dados são então salvos em um arquivo do tipo ".lzw"

O processo de descompressão é feito em uma etapa, na qual se realiza o procedimento análogo porém inverso ao de compressão.

2.4. Dados

Os dados utilizados para o procedimento de análise de desempenho foram providos pelo professor do curso. São eles: `data1.txt`, `data2.txt`, `data3.txt`, `data4.txt`.

Tendo em vista que constituem arquivos extensos, para a realização de testes iniciais, foram utilizados, também, arquivos menores, constituidos por frases, com o objetivo de que todo o processo fosse feito de forma mais rápida no decorrer dos estágios iniciais do programa. Economizando tempo e possibilitando a detecção de problemas mais fácilmente.

3. Resultados

Após a realização de todas as etapas de codificação e decodificação dos diversos exemplos de arquivo de texto, por meio dos arquivos data disponibilizados pelo professor e também por meio de exemplos de arquivos texto menores, foi possível validar a funcionalidade do algoritmo criado por completo, e, também, verificar suas estatísticas de execução, como número de instruções, através de ferramentas inerentes do software Mars 4.5 (Instruction Counter e Instruction Statistics).

A partir disso, foi obtido resultados satisfatorios quanto a aplicabilidade do programa construido.

Ilustrando um caso prático simples e de fácil visualização, foi utilizado um arquivo com o seguinte texto: `a asa da casa`.

Após o processo de codificação, obteve-se o seguinte dicionário: `| a | | as | a | d | a c | asa`.

Com isso, percebe-se que o comportamento do algoritmo, durante o processo de concatenações, foi como esperado. E para que seja possível uma análise mais estatística de seu funcionamento, foi verificado sua utilização de instruções em cada procedimento, sendo ilustrado nas tabelas 1 e 2, construídas após a realização de testes com cada arquivo e sendo feita uma média dos resultados de cada medição, tendo em vista que seus valores podem variar de acordo com o tamanho e conteúdo do arquivo.

Após cada verificação estatística, foi constatado que a quantidade de instruções utilizadas para codificação de um

Parâmetros	Codificação	Decodificação
ALU	41%	43%
Jump	16%	24%
Branch	15%	9%
Memory	8%	12%
Others	20%	11%

Tabela 1. Estatísticas de intruções

Parâmetro	Codificação	Decodificação
Tipo R	34%	28%
Tipo I	57%	55%
Tipo J	8%	15%

Tabela 2. Contagem de instruções

arquivo pode variar consideravelmente dependendo do conteúdo do mesmo. Foram verificados diferenças de cerca de 7% de um arquivo para outro no parâmetro de instruções ALU, por exemplo.

Porém, não se encontrou divergências desse nível para o procedimento de decodificação. Nesse processo, houve um funcionamento consideravelmente parecido para todos os arquivos testados, já que apresentaram porcentagem de utilização de instruções em um nível muito similar.

Considerando a compressão como um parâmetro de eficiência do algoritmo, temos que, novamente, esse nível de funcionabilidade depende do conteúdo apresentado no arquivo. Para o arquivo `data2.txt`, que possui originalmente um tamanho de 365KB, obteve-se uma compressão de cerca de 30% do seu tamanho, alcançando-se um arquivo comprimido de 258,3KB. Para o arquivo `data3.txt`, que possui um tamanho original de 262KB, obteve-se um arquivo comprimido de 231,5KB, que representa uma redução de 12% de tamanho.

Porém, percebe-se que mesmo com essa eficiência apresentada, foi constatado a necessidade de um grande espaço de tempo para a conclusão de cada processo na plataforma Mars.

4. Discussão e Conclusões

Portanto, ao longo de todo o projeto, foi possível verificar a extrema utilidade que um processo de codificação e descodificação tem. Sendo que agrega atributos muito valorizados em qualquer projeto que está envolvido com armazenamento e transferência de dados, possuir um tamanho diminuto.

O algoritmo proposto como modelo foi trabalhado e implementado em linguagem Assembly MIPS. Seu ideal baseado em remoção de redundâncias através da substituição de dados recorrentes por referências à um dicionário possibilita, de forma efetiva, o tratamento dos arquivos com o propósito de alcançar sua compressão e descompressão.

Com isso, tem-se que o objetivo proposto foi atingido. Por meio de todos os testes, foi validado o comportamento do algoritmo e sua funcionalidade através de cada teste no software Mars. Apresentando resultados como esperado e aplicabilidade satisfatória.

Poder trabalhar com essa aplicação através de uma linguagem de baixo nível como o Assembly MIPS proporciona aos alunos excelente visão do que pode ser feito, possibilitando um controle de cada operação realizada pelo sistema, o que, consequentemente, deixa o programador responsável pelo tratamento de cada erro ou divergência que possa ocorrer durante a execução do programa. E assim, permite que se tenha noção, ou torne possivel um entendimento, de como realmente funciona um computador, coordenando-o, bit à bit, através de uma linguagem de máquina.

Referências

[1] U. F. Ellen Chang and J. Hu. Data compression. `https://cs.stanford.edu/people/eroberts/courses/soco/projects/data-compression/lossless/lz78/index.htm`. Acesso: 20/04/2018.

[2] M. A. M. Omar Adil Mahdi and A. J. Mohamed. Implementing a novel approach an convert audio compression to text coding via hybrid technique. `http://ijcsi.org/papers/IJCSI-9-6-3-53-59.pdf`. Página: 53. Acesso: 28/04/2018.

[3] Wikipédia. Lz78. `https://pt.wikipedia.org/wiki/LZ78`. Acesso: 20/04/2018.