

Relatório - Laboratório 1

Lempel-Ziv

Lucas da Silva Moutinho - 15/0015747

¹Universidade de Brasília - Instituto de Ciências Exatas
Departamento de Ciência da Computação - CIC 116394
Organização e Arquitetura de Computadores
2018.1 - Turma D - Professor Flávio Vidal
Prédio CIC/EST - Campus Universitário Darcy Ribeiro
Asa Norte 70919-970 Brasília, DF

lucas.silvamoutinho@gmail.com

Resumo. Neste relatório tem-se por objetivo discutir e analisar criticamente um programa de codificação e decodificação construído a partir da linguagem Assembly MIPS, com o intuito de formar um espírito crítico de avaliação a respeito do desempenho real provido pelo sistema computacional. Para tal, fora desenvolvido o programa utilizando o algoritmo de compressão de dados Lempel-Ziv LZ78 por meio do simulador MARS. Benchmarks para testes do algoritmo e análises estatísticas foram utilizadas neste programa para um melhor embasamento na discussão pretendida. **Palavras-chave:** OAC. Arquitetura de Computadores. Lempel-Ziv. Compactação. UnB.

1. Objetivos

Obter conhecimentos relacionados à linguagem Assembly MIPS e sua utilização por meio da implementação do algoritmo de Lempel-Ziv para compressão de dados em arquivos texto e analisá-lo criticamente em relação a performance e otimização, permitindo, assim, uma melhor compreensão de um sistema computacional.

2. Introdução

O algoritmo LZ78 consiste em um algoritmo de compressão sem perda de dados, isto é, uma compressão que permite que os dados originais sejam perfeitamente reconstruídos a partir da descompressão dos dados comprimidos.

Conhecido como uma das versões do algoritmo de Lempel-Ziv, fora publicado em artigos por Abraham Lempel e Jacob Ziv em 1978; e por isto recebeu o nome LZ78.

O Algoritmo LZ78 utiliza de um dicionário para armazenar sequência de caracteres encontradas no arquivo original. Por meio deste dicionário, permite a compressão substituindo ocorrência repetida de dados no arquivo por códigos que referenciam a posição correta da sequência de caracteres no dicionário.

Em termos matemáticos, pode-se dizer que o Dicionário pode ser representado por D , tal que:

$$D = [I, SC] \tag{1}$$

onde, I é o índice da linha do dicionário, e SC é uma sequência de caracteres.

Por outro lado, cada código no arquivo comprimido é um par (I, c) , onde I faz referência ao índice da linha do dicionário onde se encontra a sequência de caracteres que estaria naquela posição no arquivo original; enquanto que c é um caractere adicional após a sequência de caracteres.

Este algoritmo apresenta-se como uma ótima oportunidade, e desafio, para implementação de um código em baixo nível, permitindo a análise crítica de desempenho e otimização das instruções para a compressão de dados, principalmente no quesito de grandes arquivos.

Foi construído o algoritmo utilizando o simulador MARS em sua versão 4.5, permitindo assim, a utilização de estatísticas para analisar a quantidade de instruções no código e a influência destes para o processamento.

Ressalta-se, também, que foram disponibilizados benchmarks para análise de desempenho do algoritmo, estes que consistem em arquivos com uma quantidade grande de dados e que demandam de mais tempo para a compressão total.

Será, portanto, detalhado a implementação do algoritmo e debatida a sua performance em uma série de testes de desempenho, além, claro, da própria demonstração da compressão e descompressão.

3. Materiais e Métodos

- Mars 4.5
- Benchmarks data.txt disponibilizados pelo professor via plataforma moodle
- Livro de Organização e Arquitetura de Computadores Patterson
- Editor de texto Sublime

Será detalhado, primeiramente, a estratégia utilizada para a construção do algoritmo em Assembly MIPS.

O programa desenvolvido apresenta um menu simples de escolha como entrada para o usuário, permitindo, por meio de um input numérico, que este selecione entre utilizar o compressor ou fechar o programa.

Caso o usuário escolha utilizar o compressor, será requisitado um input com o nome do arquivo a ser lido. Foi construída uma lógica tal que por meio da sequência de caracteres escritas, o programa consiga ler a extensão e determinar se será realizada uma compressão ou descompressão. Caso o input tenha a extensão *.txt*, o programa criará um arquivo com o mesmo nome e a extensão *.lzw* que receberá os dados comprimidos. De maneira similar, caso o arquivo tenha a extensão *.lzw*, o programa criará um arquivo *.txt* para armazenar os dados descomprimidos. Tal lógica foi implementada por meio das chamadas de sistema *syscalls* do MIPS, tanto para receber o inputs quanto para a abertura, leitura e escrita dos arquivos de texto.

Explicando de maneira sucinta a lógica da compressão, esta foi implementada utilizando um buffer para armazenar uma sequência de caracteres lida do arquivo original. Caso a sequência de caracteres no buffer consista em uma ainda não presente no dicionário, o buffer é descarregado e salvo em uma nova linha do dicionário, ao mesmo tempo em que é escrito no arquivo de saída, o par comprimido, composto pelo último caractere lido juntamente do índice da sequência $SC = SC_{atual} - c$, ou seja, o índice de uma sequência de caracteres igual a sequência atual do buffer sem o último caractere. Para separar os pares escritos na saída, foi escolhido, arbitrariamente, o caractere 96 da tabela ASCII, que corresponde ao símbolo `'`. Desta forma, é necessário considerar, principalmente para arquivos maiores, que a complexidade O do algoritmo cresce exponencialmente de acordo com o tamanho das sequências de caracteres e do número de linhas do dicionário. Foi escolhido para este programa que cada linha do dicionário teria capacidade de 32 words, 128 bytes, o que equivale a uma sequência de 128 caracteres em cada linha; além de 20000 linhas no total, o que implica na utilização de 640000 bytes de memória para o armazenamento do dicionário, o que resulta, assim, na utilização da memória do *.data* e do *.heap* do MARS. Optou-se pela utilização de mais linhas com menos words, para diminuir espaços inutilizados de caracteres nas sequências de linhas do dicionário na memória, evitando a fragmentação desta.

A descompressão, por sua vez, utiliza de maneira similar, um buffer de entrada para ler os pares do arquivo de entrada. Ressalta-se que não é necessário que o dicionário fique armazenado na memória, pois seguindo a lógica da descompressão, este dicionário é recriado e o arquivo original restaurado. Assim, cada leitura de um novo par implica na construção de uma nova linha no dicionário, contendo o caractere *c* e a sequência de caracteres do índice *I* no par (caso o índice seja 0, não existe uma sequência anterior no

dicionário). A cada leitura de par, é também recuperado os dados e escritos no arquivo de saída descomprimido, recuperando, assim, os dados originais.

Tendo atestado o funcionamento do programa, seguiu-se para a elaboração de uma lógica para os testes de desempenho, funcionamento e otimização a serem aplicados, este que será explicado a seguir.

Para averiguar o funcionamento, criou-se um data0.txt com diversos palíndromos, sendo este, o primeiro benchmark, com 26 Kbytes, tamanho consideravelmente menor do que os outros benchmarks. Por meio deste, é averiguado a correta compressão e descompressão dos dados, além de alguns testes iniciais de desempenho. Fora, também, comparado o tamanho do arquivo comprimido com o descomprimido, juntamente com outro benchmark, o data.txt.

Com o foco maior na análise de estatísticas das instruções, utilizando o *instruction statistics* do MARS, foram utilizados os outros arquivos textos, benchmarks, fornecidos pelo professor, com uma quantidade de dados bastante superior. Nestes o foco da discussão é mais a otimização e o tempo de execução do algoritmo. Para cada benchmark averiguado, fora colado as imagens do *statistics*.

4. Resultados

Apresentando os resultados do primeiro benchmark, este que fora utilizado para averiguar o desempenho e o correto funcionamento do algoritmo. Este arquivo contém uma série de palíndromos à serem comprimidos, seguem imagens de exemplo:

```
1 A base do teto desaba.
2 A cara rajada da jararaca.
3 Acuda cadela da Leda caduca.
4 A dama admirou o rim da amada.
5 A Daniela ama o lei? Nada!
6 Adias a data da saída.
7 A diva em Argel alegra-me a vida.
8 A droga do dote é todo da gorda.
9 A gorda ama a droga.
10 A grama é amarga.
11 Ai, Lima falou: "Olá, família!".
12 Ajudem Edu, já!
13 A lupa pula.
14 A mãe te ama.
15 A mala nada na lama.
16 Ame o poema.
17 A miss é péssima!
18 Amo Omã, Se Roma me tem amores, amo Omã!
19 Anotaram a data da maratona.
20 A pateta ama até tapa.
21 Após a sopa.
22 Arara rara.
23 A Rita, sítira!
24 A Rita, sobre vovô, versos atira.
25 A rua Laura.
26 Assim é aia ia à missa.
27 Ato idiota.
28 A torre da derrota.
29 E até o Papa poeta é.
30 Irene ri.
31 Laço bacana para panaca boçal.
32 Lá vou eu em meu eu oval.
33 Luz Rocelino, a namorada do Manuel, leu na moda da romana: "anil é cor azul".
34 Luz azul.
35 Mega bobagem.
36 Me vê se a panela da moça é de aço, Madalena Paes, e vem.
37 Missa é assim.
38 O céu sueco.
39 O galo ama o lago.
40 Olá, galo!
41 Olé! Maracujá, caiu caramelo.
```

Figura 1. Arquivo com primeiras linhas do gerado data0.txt

```

1 0A 0 0b 0a 05 0e 2d 0a 2f 0f 8 0d 65 4b 4 0
2 1 0c 4r 4 0r 4j 4d 20d 20j 19a 21a 18a 0 10a 18u 12a 2c 23c 0l 24a 2l 6d 20c 23u 28 30
3 17d 47g 24d 70 0l 6 73c 90 120 52 810 21d 90a 7g 8r 32 4m 62d 92a 56a 104r 107a 2A 0c 61a 21g
103R 74 37l 54b 27f 4l 0u 0 23 0c 0e 00 35f 0 0 121a 54f 74l 68l 0a 120 119j 48d 6m 2e 12u 13l
0j 73j 0l 42l 48p 20p 48l 103 133f 96f 96a 120 42m 122a 2n 55 0n 20l 112 30m 960 2p 8e 155
17m 465 69A 114 0p 97s 5l 120l 1630 20 152 25 96R 8m 20m 153e 79a 540 21e 5 610 177m 73f 146
1m 0l 20m 07d 4t 71a 20l 0n 151p 196e 95a 115 106A 171l 4p 103p 73j 69a 25 0p 103r 26
27r 90a 12b 0R 46t 4 210A 130t 46r 4l 42R 218a 143s 8b 18e
880 88A 0 143v 6r 50 209t 222a 110 21u 20L 4u 27 30s 174m 87a 68
245A 9 2m 109s 103l 11l 12l 193a 237l 105v 96d 24e 21t 253 16E 53t 97
11P 206a 1650 10a 113c 56l 106m 96r 46 16l 4A 0f 11b 4c 58a 165a 27 172a 160a 28
30 73s 122 272A 130 229u 2e 48 140 86u 289u 49v 285
0L 462 20R 0c 82l 28l 87m 1070 27d 93 0H 500 02 2l 0u 150a 240a 106d 20r 181a 201 2 58l 35
262c 105 4z 150 56l 297 322u 35 10M 84a 2b 227a 81e 54 328e 2v 73p 210e 07p 58e 60
106m 8A 274a 267 12e 53A 2740 143M 55l 6n 20P 4e 187 96v 140 328l 55 20A 171a 357l 333
129 18A 114u 210u 0c 0 160 184a 350 2030 309a 10l 368l 145 369l 8l 374A 114l 2M 26c 48j 375
28j 48 33a 27a 1020 500 3090 283 112 2750 370 360 47a 580 53c 106a 188r 13 36m 4s 115d 403
1880 120 399c 200 311m 1050 389 95e 290d 48e 950 56A 0 95l 185 220 293 2q 415
8e 49m 2180 389t 11c 101e 312c 193A 0 212t 47t 1530 21t 267
217l 2l 40 3r 6v 354c 21b 11r 222 16R 315 203a 185r 508 449m 183m 400 165 273 387
11t 460 289 8l 416 306a 211 403 456e 180 7e 50a 77a 143c 8l 2990 1580 330 370
28v 68f 461d 299e 5 4560 468r 387 86 365b 46
1600 113 160l 30 60e 79m 19r 474s 502 262d 90L 51a 143R 48a 37a 48r 219
54l 319e 469z 237b 403c 94
413t 11d 13a 30 237c 213r 22a 313a 2j 26r 276a 110c 139a 386d 82a 100L 38a 529u 41
91a 407a 45l 47u 441r 5l 160a 120d 1510 310e 340u 932
63l 07 65a 32l 67l 405a 52t 71s 457d 151d 47l 291A 116e 319a 63g 85e 154 88l 75
91r 8g 305d 193e 344t 0d 513a 1040 102a 237g 105d 44m 108r 503a 571r 392A 359m 19g 117 143L 500
0r 1220 45 125c 1270 129l 143f 107A 134l 2230 137 30j 139e 79e 142 520A 130l 147u 200
172u 60 156A 0E 183 162
168a 340n 159a 161a 167A 86 11p 166m 151m 249 262p 173s 500l 176
120m 190 179c 20 453e 9e 104m 105e 353a 105 622A 009l 30m 253r 107
70a 100m 1980 281 42p 201t 573a 261A 205a 280 30p 208s 07s 211a 110r 517a 241
73c 625l 202 220l 419r 223
11R 225 2180 442e 2350 230 232e 21c 425 196l 653A 50u 239a 504a 110s 243
44l 20l 358 248l 357a 110t 251d 460t 151t 255e 100d 233r 259
0E 6440 441P 264 1720 266 97 16l 269e 50l 324a 2840 330a 28n 149a 279p 277c 20b 342s 295l 145
20a 210 2010 711b 200a 88a 737

```

Figura 2. Arquivo com primeiras linhas do gerado data0.lzw

```

1 1A
2 2
3 3b
4 4a
5 5s
6 6a
7 7 d
8 8o
9 9 t
10 10et
11 11o
12 12d
13 13es
14 14ab
15 15a
16 16
17
18 17A
19 18c
20 19ar
21 20a
22 21r
23 22aj
24 23ad
25 24e d
26 25a l
27 26ara
28 27ra
29 28co
30 29
31 30
32 A

```

Figura 3. Arquivo com primeiras linhas do gerado Dicionario.txt

Fora comprimido o data0.txt utilizando o programa, este resultou em um data0.lzw codificado. Para averiguar o correto funcionamento, utilizou-se a descompressão no data0.lzw gerado. Verificou-se que o novo data0.txt gerado pela descompressão, de fato, continha os dados do arquivo original, e que estes foram recuperados corretamente.

A fim de se comparar o poder de compressão. Tabelou-se o tamanho em bytes do arquivo comprimido e descomprimido do data0. Para comparativos, fez-se o mesmo com um novo arquivo, o data.txt, que continha originalmente 40,1 Kbytes. Resultados encontram-se abaixo, juntamente com o tempo de execução para o término da compressão.

Quadro 1. Tabela de Arquivos e Tamanhos

Título	Tam. original do .txt	Tam. comprimido do .lzw	T. de Execução
data0.txt	26 kB	27,8 kB	9,37min
data.txt	40,1 kB	42,2 kB	15,12min

Observa-se que, mesmo a compressão e descompressão tendo ocorrido corretamente. O arquivo comprimido apresenta tamanho maior do que o original. Isto deve-se porque a quantidade de caracteres adicionais para o índice no arquivo comprimido e

também para o caractere de espaçamento entre os pares de saída não compensam, em média, a quantidade da sequência de caracteres que fora suprimida do arquivo original. Para arquivos maiores, o arquivo original aproxima-se do comprimido, o que pode implicar que para arquivos muito extensos, o número de caracteres substituídos compensa a compressão. Para melhorar esta compressão, poderia-se, também, pensar em outra forma de separar os pares de saída no arquivo comprimido, evitando assim a utilização de um byte para separar os pares.

Segue-se também estatísticas iniciais do *instruction statistics* do MARS para ambos arquivos. Tais estatísticas serão mais discutidas com os próximos benchmarks.

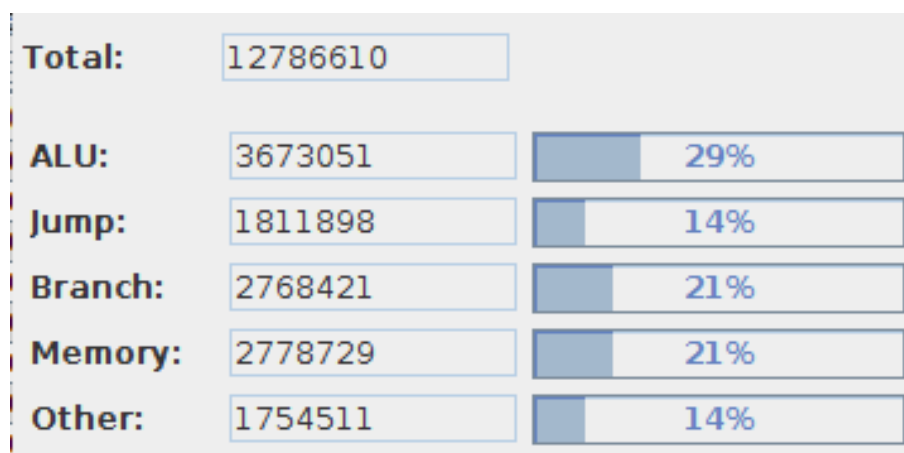


Figura 4. Tabela de Instruções para data0.txt

Por sua vez, agora foram utilizados os arquivos benchmarks providos pelo professor. Como estes possuem uma quantidade bem maior de bytes, o tempo de execução aumentou exponencialmente, impossibilitando a visualização do término destes. Segue-se as imagens do statistics enquanto o algoritmo estava sendo executado e havia estabilizado.

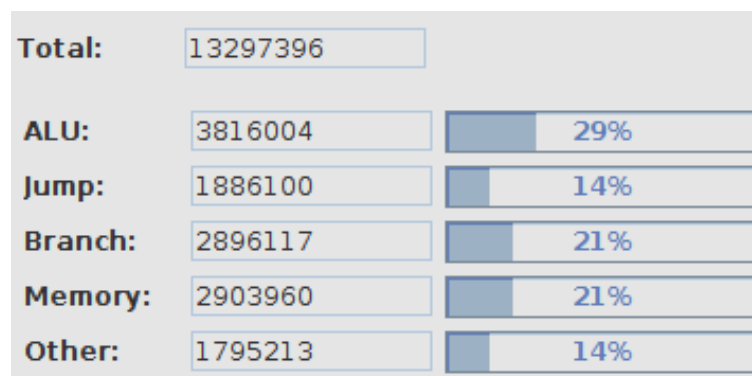


Figura 5. Tabela de Instruções para data1.txt

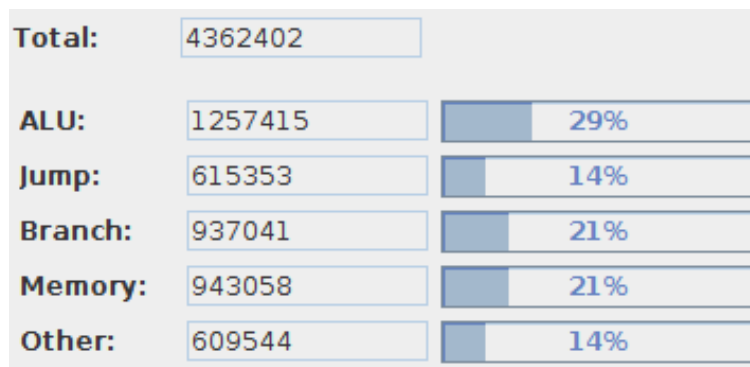


Figura 6. Tabela de Instruções para data2.txt

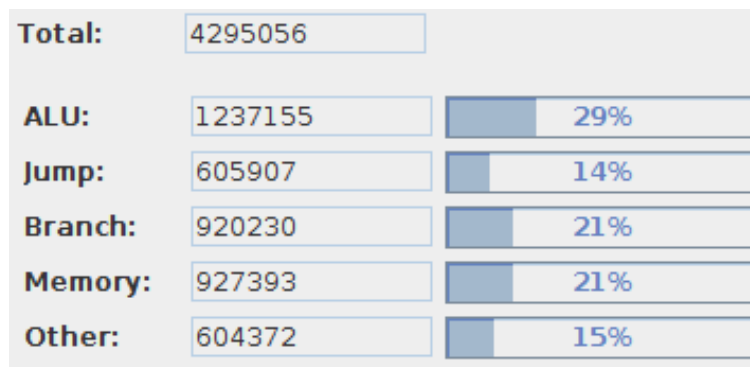


Figura 7. Tabela de Instruções para data3.txt

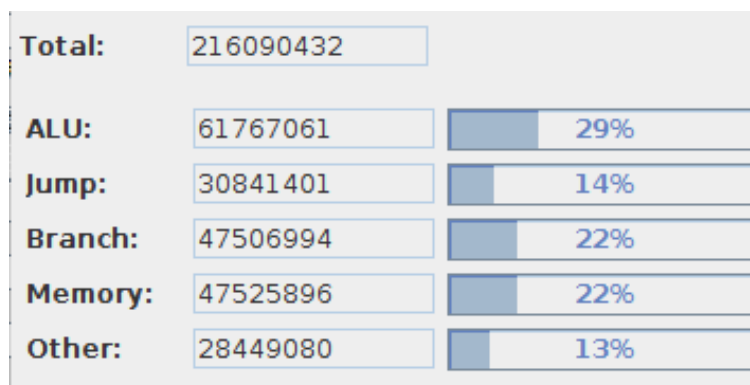


Figura 8. Tabela de Instruções para data4.txt

Pode-se observar que em média, o percentual de uso de cada uma das instruções aparentemente está equilibrado. Observa-se que operações aritméticas na ULA são as que tem um maior percentual de uso; provavelmente devido a constante necessidade dos off-sets para navegar entre os dicionários e os buffers, além de outras operações aritméticas exigidas também pelas outras instruções. Este pode ser considerado um resultado esperado dado que todas as instruções, exceto alguns casos do jump, exigem algum tipo de

acesso ao processador para a realização de um cálculo. Observa-se, também, o alto acesso à memória para resgatar dados do dicionário, o que é um dos maiores motivos para o alto tempo de execução e um desempenho não tanto otimizado. Outro motivo para queda de desempenho seriam as instruções *Other*, que incluem os *syscalls*. Como acessos ao disco rígido são ainda mais impactantes para o tempo de execução, este provavelmente tem grande influência no desempenho.

Observa-se pela análise destas estatísticas que um provável código mais otimizado exigiria uma lógica para diminuir o número de acessos ao dicionário, memória, e coletas de bytes do arquivo texto por meio do acesso ao disco rígido. Uma estratégia poderia considerar ler mais bytes por meio de uma única leitura ao disco rígido, salvando-os em um buffer intermediário. Fica claro a necessidade de se pensar nestas otimizações, principalmente para arquivos mais extensos, e também, o poder de se trabalhar em baixo nível para a utilização de estratégias para aumentar o desempenho, principalmente diminuindo o acesso à memória e ao disco.

5. Discussão e Conclusões

Fora corretamente desenvolvido o programa de compressão e descompressão Lempel-Ziv na linguagem MIPS. Por meio do desenvolvimento deste programa, pode-se treinar e compreender diversos dos aspectos de um programa desenvolvido em baixo nível, permitindo a familiarização, principalmente, com a linguagem MIPS e com o simulador MARS.

A lógica do algoritmo fora corretamente implementada e, de fato, permite a compressão e descompressão de dados. Entretanto, por meio de testes comparativos dos tamanhos de arquivos comprimidos e descomprimidos, testes de tempo de execução, benchmarks para avaliação de desempenho e estatísticas do MARS, observou-se diversos motivos para concluir que o código não estava otimizado e poderia ter um melhor desempenho. Fica claro a necessidade de se avaliar estratégias para a construção de algoritmos e a possibilidade de melhoras de desempenho disponibilizadas por uma linguagem de baixo nível.

Referências

David A Patterson and John L Hennessy. Computer organization and design. *zadnje izdanje*, 1994.

Jacob Ziv and Abraham Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on information theory*, 23(3):337–343, 1977.

Wikipedia. Lz78, a. URL <https://pt.wikipedia.org/wiki/LZ78>.

Wikipedia. Lossless compression, b. URL https://en.wikipedia.org/wiki/Lossless_compression.

Wikipedia. Lz77 and lz78, c. URL https://en.wikipedia.org/wiki/LZ77_and_LZ78.

University Academy. Lempel-ziv coding. URL <https://www.youtube.com/watch?v=EgreLYa-81Y>.