DESENVOLVIMENTO DE APLICAÇÕES EM ASSEMBLY MIPS

Lucas Villela de Souza Lacerda*, Victor Fabro Neri - Lider†

*12/0126648

†12/0153939

Emails: lucasvillela.unb@gmail.com, victor.fneri@gmail.com

Disciplina: Organização e Arquitetura de Computadores

Turma: A

Professor: Flávio Vidal

Semestre: 1º/2017

23/04/2017

1 Objetivos

Permitir que o aluno(a) se familiarize com a linguagem Assembly MIPS e metodologias de aplicações eficientes e otimizadas. Esta atividade tem como intuito formar espírito crítico de avaliação a respeito do desempenho real provido pelo sistema computacional, proporcionando assim melhorias na compreensão do funcionamento destes tipos de sistemas.

O projeto desta disciplina é uma atividade planejada de forma a complementar e reforçar o conteúdo programárico da disciplina Organização e Arquitetura de Computadores. Espera-se qua nas atividades de projeto os alunos desenvolvam sua capacidade de observação, análise e compreensão das metodologias de OAC.

Desta forma cabe ao aluno(a) ou grupo, partindo da premissa que possui os requisitos para o curso, juntamente com o conteúdo adquirido nas aulas teóricas, desenvolver todas as etapas da implementação solicitada.

2 Introdução

Para controlar o hardware de um computador é preciso falar sua linguagem. As palavras da linguagem de um computador são chamadas instruções e seu vocabulário é denominado conjunto de instruções.

Em uma análise superficial, pode-se pensar que as linguagens dos computadores são tão diversificadas quanto as linguagens humanas, mas, na realidade, as linguagens de computador são muito semehlante entre si e mais parecidas com dialetos regionais do que idiomas independentes. Assim, ao aprender uma das linguagens, aprender as outras se torna fácil. Essa semelhança ocorre porque todos os computadores são construídos a partir de tecnologias de hardware baseadas em princípios básicos semelhantes e porque existem algumas operações básicas que todos os computadores precisam oferecer. Além do mais, os projetistas de computadores possuem um objetivo comum: Encontrar ma linguagem que facilite o projeto do hardware e do compilador enquanto maximiza o desempenho e minimiza o custo.

O conjunto de instruções escolhido vem da MIPS Technology, que é um exemplo elegante dos conjuntos de instruções criados desde a década de 1980.

Como primeiro passo na eliminação, é necessário multiplicar a primeira linha da matriz [A] da figura 1, pelo fator f 21 = (a21/a11), em seguida, subtrair os resultados na segunda linha da matriz [A], eliminando o termo a21. Em seguida, repete-se o primeiro passo, porém agora multiplicando a primeira linha pelo fator f 31 = (a31/a11) e subtrair na terceira linha da matriz [A]. Por último (visto que é uma matriz 3x3), a segunda linha foi

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

Figura 1: Matriz A

multiplicada pelo fator f32 = (a'32/a'22) e subtrair na terceira linha da matriz [A], obtendo a matriz [U] (figura 2)

$$U = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ 0 & a'_{22} & a'_{23} \\ 0 & 0 & a'_{33} \end{bmatrix}$$

Figura 2: Matriz U

Bastando agora, apenas montar a matriz triangular inferior [L]. Para isso, é necessário apenas colocar 1?s na diagonal principal e os fatores f 21, f 31 e f 32, calculados previamente, nas posições restantes.

$$L = \begin{bmatrix} 1 & 0 & 0 \\ f_{21} & 1 & 0 \\ f_{31} & f_{32} & 1 \end{bmatrix}$$

Figura 3: Matriz L

Ao multiplicarmos as matrizes [L] e [U], encontramos a matriz [A].

Por exemplo, dada uma matriz [A]:

1	2	3
4	5	6
7	8	9

Figura 4: Matriz A

As matrizes [L] e [U] serão respectivamente:

1	0	0
4	1	0
7	2	1

Figura 5: Matriz L

1	2	3
0	-3	-6
0	0	0

Figura 6: Matriz U

Já o método de Crout, apesar de possuir um mesmo princípio, utiliza um algoritmo diferente para se encontrar as matrizes [L] e [U].

A11	A21	A31
A21	A22	A32
A31	A32	A33

Figura 7: Matriz A do método de Crout

Como principal diferença para o método de eliminação de Gauss, é matriz [U] que possui 1's como elementos da diagonal principal.

L11	0	0
L21	L22	0
L31	L32	L33

Figura 8: Matriz L do método de Crout

1	U12	U13
0	1	U23
0	0	1

Figura 9: Matriz U do método de Crout

Sabendo que LU = A, é possível seguir para o primeiro passo do método: multiplica-se as linhas da matriz [L] pela primeira coluna da matriz [U]. Assim obtém-se: l11 = a11, l21 = a21 e l31 = a31, ou seja, li1 = ai1, para i = 1, 2, 3, ..., n

Em seguida, é multiplicada a primeira linha da matriz [L] pelas colunas da matriz [U]. Sendo obtidos: l11 = a11, l11u12 = a12, l11u13 = a13. Como l11 = a11 já havia sido definido no passo anterior, os demais podem ser generalizados como: u1j = (a1j/ l11), para j = 2, 3, ..., n

Como terceiro passo, multiplica-se as linhas 2 e 3 da matriz [L] pela segunda coluna de [U]. Assim: l21u12 + l22 = a22, l31u12 + l32 = a32. Resolvendo as equações e generalizando: li2 = ai2 ? li1u12, para i = 2, 3, ..., n

Por último, multiplica-se a segunda linha de [L] pela terceira coluna de [U]. Assim: l21u31 + l22u23 = a23; generalizando: u2j = (a2j ? l21u1j)/l22, para j = 3, ..., n.

Pode-se também, repetir tal processo para os demais elementos das matrizes: u3j = (a3j ? l31u1j - l32u2j)/l33, para j = 4, ..., n; li3 = a13 ? li1u13 ? li2u23, para i = 3, ..., n.

Já em relação a linguagem utilizada, o Assembly MIPS é uma linguagem de programação de baixo nível baseada na arquitetura de processador RISK MIPS. Neste tipo de arquitetura, o processador consiste basicamente em uma unidade central que opera sobre números inteiros e dois co-processadores auxiliares: co-processador 0, encarregado de realizar desvios, chamadas de sistema, exceções, etc. e o co-processador 1 que da suporte ao tratamento de dados em ponto flutuante. Dessa forma, temos na unidade central 32 registradores, que são os componentes capazes de armazenar números inteiros que representam os dados, e na unidade de ponto flutuante temos outros 32 registradores, que armazenam números em ponto flutuante. A capacidade de cada registrador (tanto inteiro como ponto flutuante) é de 32 bits. Dessa forma, o assembly mips trabalha sobre estes registradores.

Um programa em Assembly segue certo padrão, a fim de tornar o projeto de hardware mais eficiente. Só é possível executar uma instrução por linha de código. As instruções só operam sobre registradores, dessa forma, dados, endereços, e instruções são armazenados e processados nos registradores. Na maioria dos casos as operações são realizados sobre exatamente dois registradores, retornando o resultado para um outro registrador. As instruções são endereçadas de modo que o código pode navegar sobre suas linhas, fazendo controle de fluxo, branchs, jumps etc. Por fim, muitas vezes o numero de variáveis envolvidas no nosso programa é maior que o numero de registradores, e por isso, temos à disposição a memória RAM, possibilitando ao Assembly acessar e escrever na memória, empilhando dados e carregando instruções.

3 Materiais e Métodos

Para a realização do experimento foi utilizado apenas um computador e o software de simulação MARS. Tal simulador permite que sejam criados programas em linguagem Assembly MIPS.

Com isso, foi então criado o programa que lê uma matriz 3x3 e entrega as matrizes LU.

Após aberto o MARS, foi criado um novo arquivo e salvo com extensão .asm. Em seguida, foi criada uma diretiva .data (os itens subsequentes são armazenador em um segmento Data no próximo endereço disponível).

Dentro dela, foram criados rótulos com diretivas próprias. As diretorias utilizadas foram: .asciiz (armazena uma string em um segmento Data e adiciona um terminador nulo, onde o que vai ser armazenado deve estar dentro de aspas); .double (armazena os valores listados como ponto flutuante de precisão dupla); .space (reversa o espaço determinado de bytes no segmento Data).

```
1  .data
2  tab:     .asciiz "\t"
3  linha:   .asciiz "\n"
4  zerod:   .double 0.0
5  umd: .double 1.0
6  A:       .space 72
7  matriz: .asciiz "Digite os termos de sua
8   matriz (aperte <ENTER> apos cada termo): \n"
9  sua_matriz:.asciiz "Sua matriz e: \n"
10 matriz_l:.asciiz "A matriz L e: \n"
11 matriz_u:.asciiz "A matriz U e: \n"
```

Figura 10: Area .Data

Ao analisar as linhas 2 e 3, é possível perceber que rótulos tab e linha foram definidos como strings e possuem dentro de si ”contrabarra t”e ”contrabarra n”; onde ”contrabarra t”e ”contrabarra n”não aparece nada, dando apenas um <tab> na saída e pulando uma linha, respectivamente.

Depois, foi criada a diretiva .text (os subsequentes itens ? instruções ? armazenadas em um segmento Text no próximo endereço disponível).

Dentro dele foi então definido um rótulo início como .globl (o que torna o rótulo global, permitindo-o ser referenciado em outros arquivos).

Foram, então, criadas as instruções para, enfim, criar o programa.

```
11  .text
12
13          .globl inicio
14
15          inicio:
```

Figura 11: Diretivas .text e .globl

Como primeira parte do código, foi zerado o registrador t5 com a instrução addi (soma com um imediato ? 0 ? com o registrador zero e colocado em t5) para uso futuro. Além disso, utilizando a instrução ldc1 (coloca no registrador o valor presente no rótulo), foi zerado o registrador f20 e colocado o valor 1 no registrador f22.

Em seguida, foi colocado no registrador v0 o valor 4 (através da instrução li ? load immediate) e no registrador a0 o que estava no primeiro endereço do rótulo matriz (através da instrução la ? load address) para, em seguida, fazer uma chamada ao sistema (quando v0 = 4, ele imprime a string que está em a0).

```
15          inicio:
16                  ldc1 $f20, zerod
17                  ldc1 $f22, umd
18                  addi $t5, $zero, 0
19                  li $v0, 4
20                  la $a0, matriz
21                  syscall
```

Figura 12: imprime a string dentro do rótulo "matriz"

Depois, o código vai para a função ler.matriz, através da instrução jal (jump and link ? pula até a função ler.matriz e ?deixa? no registrador ra o endereço da linha 22. Dentro de tal função foi zerado o registrador t2 (através da instrução addi ? somando o registrador zero com o valor 0) para ser o contador de um loop.

Para se manter no loop existe um condicional (enquanto t2 for diferente de 72 ele continua, porém quando t2 for igual a 72, o código pula para a função voltar ? isso foi feito através da instrução beq (brench if equal)). Já dentro do loop, foi colocado no registrador v0 o valor 7 (através da instrução li ? load immediate) e feita uma chamada ao sistema (syscall ? como v0 = 7, a chamada lê um double e coloca o valor lido dentro de f0). Em seguida, foi colocado o que está no registrador f0 na posição t2 do espaço reservado para o rótulo A. Por último, foi somado o atual valor de t2 mais 8 (como um ponto flutuante de precisão dupla ocupa o espaço de duas words, que por sua vez ocupa 4 bytes ? dando o total de 8 que deve ser somado a t2), voltando em seguida para o início do loop (através da instrução j - jump).

Ao finalmente sair do loop (indo para a função voltar), a instrução jr (jump register unconditionally) faz o código ir para o endereço do registrador ra mais 4 (ou seja, volta para a linha 23 do código).

```
22                  jal ler_matriz
23                  li $v0, 4
```

Figura 13: Vai para a função Ler.Matriz

```
37  ler_matriz:
38          addi $t2, $zero, 0
39          While:
40                  beq $t2, 72, voltar
41                  li $v0, 7
42                  syscall
43                  sdc1 $f0, A($t2)
44                  addi $t2, $t2, 8
45                  j while
46          voltar:
47                  jr $ra
```

Figura 14: Função Ler.matriz

Na linha 23, é colocado o valor 4 no registrador v0 (através da instrução li) e a string sua.matriz no registrador a0, em seguida sendo feita uma camada ao sistema (como v0 = 4, o sistema imprime o que está em a0), indo em seguida para a função imprime.double.

Dentro da função imprime.double (imprime a matriz de entrada), inicialmente é zerado o registrador de contagem t4 (através da instrução addi, somando o registrador zero com o valor 0). Em seguida, entramos na função While2 (loop que imprime a matriz), nela é feita uma comparação (beq

```
23                  li $v0, 4
24                  la $a0, sua_matriz
25                  syscall
26                  jal imprime_double
```

Figura 15: Imprime a string sua.matriz e vai para a função imprime.double

t4, 72, fim2), onde quando t4 é igual a 72, o código vai à função fim2, senão continua.

Passada a comparação, é colocado em v0 o valor 3 (para quando feita a chamada ao sistema, imprime um double ? que está armazenado dentro de f12), em seguida é salvo (através da instrução l.d ? load floating point Double precision) em f12 o que está presente na posição t4 do rótulo A.

```
82  imprime_double:
83          addi $t4, $zero, 0
84          While2:
85                  beq $t4, 72, fim2
86                  li $v0, 3
87                  l.d $f12,A($t4)
88                  syscall
```

Figura 16: Imprime a matriz A

Ainda dentro da função imprime.matriz, passada a função While2, é entrada na função da.tab (função para dar tab entre dois elementos de uma mesma linha da matriz). Onde enquanto t5 for diferente de 3, vai para a linha de baixo, senão vai para a função pula.linha. Sendo diferente, coloca o valor 4 em v0 (através da instrução li); coloca o que está no endereço reservado a tab em a0 e faz uma chamada ao sistema (como v0 = 4, a chamada imprime uma string ? que está em a0). E por último soma o atual valor de t5 mais 1 e coloca em t5 (através da instrução addi).

```
90  da_tab:
91          condicao:
92                  beq $t5, 3, pula_linha
93                  li $v0, 4
94                  la $a0, tab
95                  syscall
96                  addi $t5, $t5, 1
```

Figura 17: Função de dar um tab

Abaixo da função da.tab, está a função pula.linha (função para pular uma linha após o terceiro elemento de cada linha da matriz). Onde quando t5 for igual de 3, vai para a linha de baixo, senão vai para a função continua. Sendo igual, coloca o valor 4 em v0 (através da instrução li); coloca o que está no endereço reservado a linha em a0 e faz uma chamada ao sistema (como v0 = 4, a chamada imprime uma string ? que está em a0). E por último zera o valor de t5 somando zero e 0 e colocando em t5 (através da instrução addi).

```
97  pula_linha:
98          cond2:
99                  bne $t5, 3, continua
100                 li, $v0, 4
101                 la $a0, linha
102                 syscall
103                 addi $t5, $zero, 0
```

Figura 18: Função para pular uma linha

Em seguida, o código vai para a função continua, que apenas soma o atual valor de t4 com 8 e coloca em t4 (através de addi), fazendo com que t4 passe para o próximo termo da matriz A. Logo depois voltando para função While2 (através da instrução j).

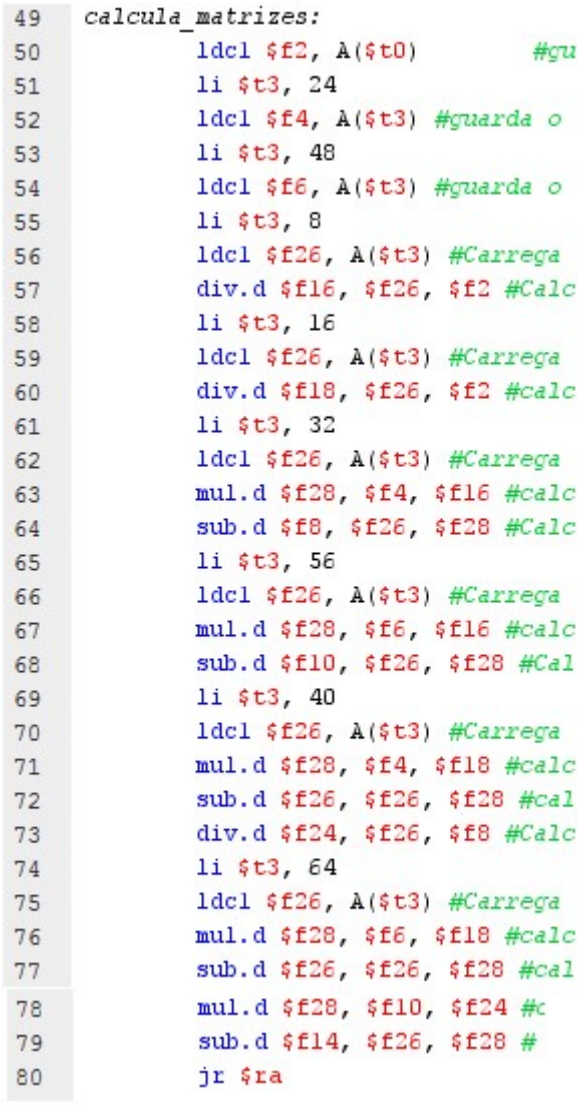
Finalmente saindo do loop da função imprime.double, o programa cai na função fim2 que, através da instrução jr faz o programa voltar para a última posição de um registrador ar mais 4 (ou seja, voltando para a linha 27).

```
104 continua:
105         addi $t4, $t4, 8
106         j While2
107 fim2:
108         jr $ra
```

Figura 19: Soma ao registrador contador e volta a linha 27

Na linha 27, a instrução executada é um jal para a função calcula.matrizes:

Esta função foi implementada para fazer o a decomposição das matrizes L e U. O método usado para tal foi o método de Crout, que está descrito na Introdução deste documento. A fim de minimizar o acesso à memoria, os elementos das matrizes foram armazenados no co-processador 1, ocupando dois registrados f's cada, uma vez que são doublés. Dessa forma, o mapeamento dos dados ficou assim: L11 = f2; L21 = f4; L31 = f6; L22 = f8; L32 = f10; L33 = f14; L12 = L13 = L23 = f20 (registrador que contém o zero em double); U11 = U22 = U33 = f22 (registrador que contém o um em double); U12 = f16; U13 = f18; U23 = f24; U21 = U31 = U32 = f20. Além disso, usamos



```
49  calcula_matrizes:
50          ldcl $f2, A($t0)        #gu
51          li $t3, 24
52          ldcl $f4, A($t3) #guarda o
53          li $t3, 48
54          ldcl $f6, A($t3) #guarda o
55          li $t3, 8
56          ldcl $f26, A($t3) #Carrega
57          div.d $f16, $f26, $f2 #Calc
58          li $t3, 16
59          ldcl $f26, A($t3) #Carrega
60          div.d $f18, $f26, $f2 #calc
61          li $t3, 32
62          ldcl $f26, A($t3) #Carrega
63          mul.d $f28, $f4, $f16 #calc
64          sub.d $f8, $f26, $f28 #Calc
65          li $t3, 56
66          ldcl $f26, A($t3) #Carrega
67          mul.d $f28, $f6, $f16 #calc
68          sub.d $f10, $f26, $f28 #Cal
69          li $t3, 40
70          ldcl $f26, A($t3) #Carrega
71          mul.d $f28, $f4, $f18 #calc
72          sub.d $f26, $f26, $f28 #cal
73          div.d $f24, $f26, $f8 #Calc
74          li $t3, 64
75          ldcl $f26, A($t3) #Carrega
76          mul.d $f28, $f6, $f18 #calc
77          sub.d $f26, $f26, $f28 #cal
78          mul.d $f28, $f10, $f24 #c
79          sub.d $f14, $f26, $f28 #
80          jr $ra
```

Figura 20: Função que calcula as matrizes L eU

os registradores f26 e f28 como temporários para auxiliar nos cálculos dos elementos.

No primeiro momento, nas linhas 50-54, queremos carregar os elementos A11, A21 e A31 para os elementos L11, L21 e L31, respectivamente. Obedecendo a ordem em que os elementos da matriz A foram armazenados na leitura de dados, usamos o registrador temporário t3 para localiza-los (Exceto no caso de A11, que está localizado no inicio da memoria reservada para o rotulo A. Neste caso, usamos o registrador zero que contem 0.), e através da instrução ldc1 carregou-se para os registradores dos elementos L11, L21 e L31, os elementos A11, A21 e A31, respectivamente, conforme citado acima.

Para calcular U12 = A12/L11, carregamos o valor A12 em f26 (linhas 55 e 56) e fizemos a divisão por L11 (f2) através da instrução div.d, guardando o resultado no registrador de U12 (linha 57). Semelhantemente, carregamos o valor de A13 em f26 (linhas 58 e 59) e dividimos por L11, guardando o resultado no registrador do U13 (linha 60).

Nas linhas 61-64, carregamos o elemento A22 no registrador f26 e fizemos a multiplicação, através do comando mul.d, entre L21 e U12 e guardamos o resultado em f18. Por fim, calculamos a diferença entre A22 (f26) e L21U12 (f28) e guardamos o resultado em L22 (f8). Nas linhas 65-68 o mesmo foi feito para o campo L32, carregamos o elemento A32 em f26, guardamos o resultado da multiplicação entre L13 e U12 em f28 e salvamos a diferença entre f26 e f28 (A32 ? L13U12) em L32 (f10).

Nas linhas 69-73 queremos calcular e armazenar o valor de U23 (U23 = (A23-L21U13) /L22)). Assim, carregou-se o elemento A23 em f26 (linhas 69 e 70), guardou o resultado da multiplicação entre L21 e U13 em f28 (linha 71), guardou o resultado da subtração entre A23 e L21U13 em f26 (linha 72) e, por fim, guardou o resultado da divisão entre A23-L21U13 e L22 em U23, f24 (linha 73).

Por fim, nas linhas 74-79, calculou-se o elemento L33 = A13 ? L31U13 ? L32U23. Primeiro, carregou-se o elemento A13 no registrador f26 (linhas 74 e 75). Depois, guardou o resultado da multiplicação entre L31 e U13 em f28 (linha 76). Fez-se a subtração entre A13 e L31U13 e guardou-se o resultado novamente em f26 (linha 77). Fez-se a multiplicação entre L32 e U23, guardando o resultado em f28 (linha 78). Finalmente, fez-se a subtração entre A13-L31U13 (f26) e L32U23 (f28), guardando o resultado em L33 (f14).

Com isso, todos os elementos das matrizes L e U foram calculados, conforme método de Crout. Os elementos que contem 0 ou 1 serão apenas exibidos, encurtando o numero de instruções. Na linha 80 a instrução retoma o programa para a linha 28.

```
28  li $v0, 4
29  la $a0, matriz_l
30  syscall
31  jal imprime_l
```

Figura 21: Imprime a Matriz L

Agora, com v0 em 4 e a0 apontado para matriz.l, a chamada ao sistema imprime na tela a string ”Sua matriz L e :contrabarra n”. E depois, pula para o código da função imprime l.

A função imprime.l, ao contrario do que e feito na função imprime.double, ela não realiza um controle de fluxo, pela dificuldade de percorrer os re-

gistradores, mas realiza sucessivas chamadas ao sistema a fim de imprimir a matriz L, separando os elementos de uma mesma linha por<tab> (através do rotulo tab) e fazendo a quebra de linhas através do rotulo linha. Dessa forma, atribuímos a v0 o valor 3, quando a impressão for um double, ou 4, quando a impressão for string. Quando formos imprimir double, o sistema vai imprimir o valor configurado no registrador f12, portanto, atribuímos a este registrador todos o elementos da matriz. Após imprimir todos os elementos conforme o layout descrito, a função retorna o programa para a linha 32, através da instrução jr ra.

```
32  li $v0, 4
33  la $a0, matriz_u
34  syscall
35  jal imprime_u
36  j fim
```

Figura 22: Imprime na tela a Matriz U

Semelhantemente ao caso acima, o sistema imprime na tela, agora, a string "Sua matriz U e contrabarra n ". E faz o jal para a função imprime.u. A função imprime.u foi implementado da mesma maneira que a função imprime.l, de modo a imprimir na tela a matriz u separando os elementos de uma mesma linha por <tab> e quebrando as linhas pelo rotulo linha. Vale salientar que para os casos em que os elementos correspondem a zero ou um, fizemos a impressão da tela dos registradores f20 e f22, respectivamente.

Por fim, a função imprime.u retorna o código para a linha 36. E então o programa vai para função fim:

```
226  fim:
227          li $v0, 10
228          syscall
```

Figura 23: Finaliza a execução do programa

Onde atribui-se o valor 10 ao registrador v0, dizendo ao sistema que encerre a execução, e, com a chamada ao sistema, a execução do programa termina.

4 Resultados

Ao compilar o código, utilizando a ferramenta Instruction Statistics fornecida pelo MARS, foi montada a tabela 1 com as porcentagens de utilização das instruções utilizadas (de um total de 472) em cada procedimento, onde cada procedimento é uma função chamada.

Parâmetros	Procedimento 0	Procedimento 1	Procedimento 2
ALU	60%	44%	58%
Jump	7%	13%	5%
Branch	7%	11%	15%
Memory	0%	0%	0%
Other	27%	34%	22%
Total de Instruções	15	80	186

Figura 24: Tabela 1

Parâmetros	Procedimento 3	Procedimento 4	Procedimento 5
ALU	34%	57%	56%
Jump	4%	3%	4%
Branch	0%	0%	0%
Memory	0%	0%	0%
Other	62%	41%	40%
Total de Instruções	50	69	72

Figura 25: Tabela 2

O Procedimento 0 é o pedaço de código do início até a impressão de "Digite os termos de sua matriz (aperte <ENTER> apos cada termo):"; O Procedimento 1 é o pedaço de código que lê a matriz de entrada A; O Procedimento 2 é o pedaço de código que imprime a matriz A, com três elementos em cada linha (separados por um tab); O Procedimento 3 calcula as matrizes L e U; os Procedimentos 4 e 5 imprimem as matrizes L e U, respectivamente.

As porcentagens dos parâmetros em todos os procedimentos fazem sentido, visto que, por exemplo, em todos os procedimentos a porcentagem do parâmetro Memory é sempre zero (o código não utilizou pilha em nenhum momento). Além disso, grande parte das instruções é alocada no parâmetro ALU (utilizam a ALU), o que faz sentido, uma vez que a maior parte das instruções utilizadas no programa são do tipo R, lw (load word) ou sw (save word).

5 Discussão e Conclusões

Contudo, tal código pode ser mais eficiente (realizar os cálculos em menos tempo). Para isso, algumas funções poderiam utilizar o conceito de pilha, ou mesmo, usar alguns loops para o calculo e impressão das matrizes L e U.

Além disso, visando uma estética melhor, poderiam ser implementadas algumas instruções para que a leitura da matriz possa ser realizada já de maneira normalizada (seja possível ler três termos distintos por linha).

O código desenvolvido atinge os objetivos desejados, pois ele consegue decompor uma matriz quadrada 3x3 e mostra na saída, as matrizes triangular inferior L e triangular superior U, além ser necessário o conhecimento da linguagem Assembly MIPS (para que o código pudesse ser feito de maneira correta).

Dito isso, conclui-se como funcionam dois métodos de decomposição LU de matrizes quadradas (eliminação de Gauss e método de Crout), além do funcionamento de algumas instruções e conceitos presentes na linguagem de programação Assembly Mips, para a implementação de tal programa.

Bibliografia

http://coimbra.lip.pt/ rui/Lu.pdf

http://www.ebah.com.br/content/ABAAABOeoAD/metodos-decomposicao-lu