UNIVERSIDADE DO VALE DO ITAJAÍ - UNIVALI

GUSTAVO COPINI DECOL
JOÃO PAULO ROSLINDO

RELATÓRIO SOBRE ATIVIDADES DE PROGRAMAÇÃO MIPS

Itajaí
2017

GUSTAVO COPINI DECOL

JOÃO PAULO ROSLINDO

RELATÓRIO SOBRE ATIVIDADES DE PROGRAMAÇÃO MIPS

Relatório apresentado ao curso de Engenharia de Computação na Universidade do Vale do Itajaí - UNIVALI, na disciplina de Arquitetura de Computadores, como requisito parcial para obtenção de nota.

Professor: Douglas Rossi de Melo

Itajaí

2017

INTRODUÇÃO

Com o desenvolvimento cada vez maior na área da computação, especificamente na dos sistemas embarcados, a importância do engenheiro da computação tem sido cada vez maior. Por isso, o conhecimento da linguagem de montagem (*Assembly)*, é um pré requisito de grande necessidade para quem procura se especializar na área.

Neste relatório serão apresentados dois programas feitos no software *Mars*, para um processador MIPS (*Microprocessor without interlocked pipeline stage*), como exercícios de fixação de comandos e lógicas de programação em uma linguagem de baixo nível.

PROGRAMA 1

Implemente um programa que leia um vetor via console, carregue todos os elementos do vetor na memória e, após, percorra todo o vetor realizando o somatório de todos os elementos.

Abaixo, na figura 1, vemos um exemplo do código requisitado na linguagem C++,

```
int main()
{
    int vetor_A[8]={0};
    int tamanho_vetor=0,i=0,soma=0;

    while(tamanho_vetor<=1 || tamanho_vetor>8){
        cout<<"Entre com o tamanho do vetor (max. 8): ";
        cin>>tamanho_vetor;
        if(tamanho_vetor<=1 || tamanho_vetor>8){
            cout<<"Valor invalido!!!"<<endl;
            system("pause");
            system("cls");
        }
    }
    cout<<"\n";

    while(i<tamanho_vetor){
        cout<<"Vetor_A["<<i<<"]: ";
        cin>>vetor_A[i];
        i++;
    }
    i=0;
    while(i<tamanho_vetor){
        soma+=vetor_A[i];
        i++;
    }
    cout<<"\n-------------------------------------------------";
    cout<<"\nA soma de todos os elementos do vetor e "<<soma;
    cout<<"\n-------------------------------------------------";
}
```

Figura 1 - Código do programa 1 em C++

Após essa prévia do código em C++, vamos apresentar o código em linguagem de montagem do MIPS, explicando cada parte do mesmo.

```
        .data # Segmento de Dados

Vetor_A:        .word 0, 0, 0, 0, 0, 0, 0, 0
Mensagem1:      .asciiz "\nEntre com o tamanho do vetor (máx. = 8, min = 2): "
Erro:           .asciiz "\nValor inválido!"
Mensagem2:      .asciiz "\nVetor_A["
Mensagem3:      .asciiz "] = "
Mensagem4:      .asciiz "\nA soma de todos os elementos do vetor é "
```

Figura 2 - Código em linguagem de montagem MIPS

Na parte inicial do código, figura 2, temos o segmento de dados *".data*" temos a declaração do nosso vetor "A" com oito posições iniciadas em 0, no qual serão armazenados os valores digitados pelo usuário para a realização do somatório do mesmo. Em seguida são criadas todas as mensagens que serão utilizadas para a interação via console com o usuário.

```
        .text # Segmento de Codigo
main:
        la $s6, Vetor_A
        addi $s5, $0, 0
        addi $s4, $0, 1
        addi $s1, $0, 0
        addi $s2, $0, 0
```

Figura 3 - Código em linguagem de montagem MIPS

Em seguida, figura 3, é iniciado o segmento de código ".*text"*, que é onde começa o programa em si. É criado o primeiro rótulo denominado "*main"* no qual são inicializados alguns registradores essenciais. Cada registrador será definido na tabela abaixo:

REGISTRADOR	***DEFINIÇÃO***
$s6	Registrador com endereço base da primeira posição do vetor A.
$s5	Registrador usado como "I"
$s4	Registrador usado como "J"
$s1	Registrador usado como "K"
$s2	Registrador que armazenará o soma dos valores do vetor A.

```
loop1:
# SOLICITANDO AO USUARIO O NUMERO DE ELEMENTOS NO VETOR A
        li $v0, 4
        la $a0, Mensagem1
        syscall
# LENDO O NUMERO DIGITADO PELO USUARIO
        li $v0, 5
        syscall
        add $s0, $v0, $0
# VERIFICAÇÃO SE O NUMERO DIGITADO PELO USUARIO É VALIDO
        slti $t0, $s0, 2
        beq $t0, $0, teste2
        j errou
```

Figura 4 - Código em linguagem de montagem MIPS

Na figura 4, é apresentado o primeiro *loop* que foi utilizado no código. Pelo fato de existirem duas condições para o tamanho do vetor, foram feitos dois testes separados, assim, se o número passar no primeiro teste, ele irá para o segundo teste, e se passar nele também, será valido, caso não passe, será invalido.

Voltando para a explicação no contexto da imagem, temos as primeiras linhas de código que são apenas chamadas do *syscall* para a interação via console com o usuário, as mensagens são explicadas e mostradas na figura 2. Nas últimas três linhas é que o corre o teste do valor digitado pelo usuário. A tabela abaixo mostrará cada linha com a sua descrição.

CÓDIGO	**DESCRIÇÃO**
Slti $t0, $s0, 2	Se o número digitado pelo usuário (armazenado em $s0) for menor do que 2, o registrador temporário $t0, recebera 1.
Beq $t0, $0, teste2	Se a temporária $t0, for igual a 0, ou seja, o usuário digitou um número maior ou igual a 2, ele passou no primeiro teste e irá para o segundo.
J errou	Caso o usuário tenho digitado um número menor que 2, o *jump* o levará para a mensagem de erro.

```
teste2:
# SEGUNDA VERIFICAÇÃO SE O NUMERO É VALIDO OU NAO
        slti $t0, $s0, 9
        addi $s7, $0, 1
        beq $t0, $s7, loop2
        j errou
errou:
# PRINT DO AVISO DE VALOR INVALIDO
        li $v0, 4
        la $a0, Erro
        syscall
        j loop1
```

Figura 5 - Código em linguagem de montagem MIPS

Na figura 5, temos o segundo teste, onde é feita a segunda verificação do número digitado pelo usuário. Cada linha será explicada abaixo:

COMANDO	DESCRIÇÃO
Slti $t0, $s0, 9	Se o número digitado pelo usuário for menor que 9, ou seja, menor ou igual a 8, a temporária $t0, recebe 1.
Addi $s7, $0, 1	Registrador $s7 inicializado em 1 para comparar com $t0 caso seja válido.
Beq $t0, $s7, loop2	Se a temporária $t0, for igual ao registrador $s7 (1=1), o valor foi válido e passou no segundo teste, indo para o próximo *loop* do programa.
J errou	Caso não tenha passado no segundo teste, irá para o rótulo de erro.

Abaixo do teste2, temos o rótulo do erro, onde são apenas chamadas de mensagens pelo já conhecido *syscall*, e em seguida retornando ao loop1, onde será solicitado novamente um novo valor ao usuário e todos os testes serão feitos novamente, até que um valor válido seja passado.

```
Entre com o tamanho do vetor (máx. = 8, min = 2): 1

Valor inválido!
Entre com o tamanho do vetor (máx. = 8, min = 2): 9

Valor inválido!
Entre com o tamanho do vetor (máx. = 8, min = 2): 5

Vetor_A[0] =
```

Figura 6 - Console com as mensagens para o usuário

A figura 6, mostra o console de interação com o usuário, as suas mensagens e erros.

```
loop2:
# SOLICITA AO USUARIO OS VALORES A SEREM COLOCADOS NO VETOR A
        li $v0, 4
        la $a0, Mensagem2
        syscall
        li  $v0, 1
        add $a0, $0, $s5
        syscall
        li $v0, 4
        la $a0, Mensagem3
        syscall
        add $t1, $s5, $s5
        add $t1, $t1, $t1
        add $t1, $t1, $s6
        li  $v0, 5
        syscall
        add $s3, $v0, $0
        sw $s3, 0($t1)
        slt $t7, $s4, $s0
        beq $t7, $0, loop3
        addi $s5, $s5, 1
        addi $s4, $s4, 1
        j loop2
```

Figura 7 - Código em linguagem de montagem MIPS

Na figura 7 está o segundo *loop*, que será responsável pelo armazenamento dos valores digitados pelo usuário no *array* do vetor “A”. As primeiras linhas são grande parte para impressão de mensagem no console, o resto são linhas de código para caminhar no vetor e ir armazenando os valores.

CODIGO	DESCRIÇÃO

`add $t1, $s5, $s5`
`add $t1, $t1, $t1`
`add $t1, $t1, $s6`
	Este pedaço do código é responsável por fazer o "caminhamento" no vetor. Como as posições dentro da memória funcionam de 4 em 4, este pedaço do código é necessário para fazer com que os valorem sejam armazenados nas suas devidas posições.
	Na primeira linha o registrador temporário $t1 recebe o registrador $s5, mais conhecido como "I", duas vezes, ou seja, $t1 = 2.i.
	Na segunda linha o $t1 recebe duas vezes ele mesmo, ou seja, $t1 = 4.i.
	Finalmente na última linha, o $t1 recebe ele mesmo, mais a posição base do primeiro elemento do vetor A, que está armazenado em $s6(figura 3).

`add $s3, $v0, $0`
`sw $s3, 0($t1)`
`slt $t7, $s4, $s0`
`beq $t7, $0, loop3`
`addi $s5, $s5, 1`
`addi $s4, $s4, 1`
`j loop2`
	Essa parte final do código é onde o valor digitado pelo usuário é armazenado no vetor.
	Primeiramente o número é colocado no registrador $s3.
	Em seguida ele é armazenado no endereço de memória que está no registrador temporário $t1.
	Na terceira linha, é feito um teste, se j ($s4) for menor que $s0 (número de valores que serão armazenados no vetor), então o registrador temporário $t7 recebe 1.
	Na quarta linha, caso $t7 for igual a 0, ou seja, a quantidade de valores que o usuário digitou já foi preenchida, ele irá para o próximo *loop*.
	As outras 3 linhas são um i++ e um j++, para ir percorrendo o vetor.
	O *jump* irá retornar ao início desse *loop* até que o número de valores no vetor, seja igual ao tamanho do vetor escolhido pelo usuário.

```
loop3:
# LOOP QUE IRA  REALIZAR O SOMATORIO DO VETOR A
        beq $s1, $s0 soma
        add $t1, $s1, $s1
        add $t1, $t1, $t1
        add $t1, $t1, $s6
        lw $t5, 0($t1)
        add $s2, $s2, $t5
        slt $t4, $s1, $s0
        beq $t4, $0, soma
        addi $s1, $s1, 1
        j loop3
```

Figura 8 - Código em linguagem de montagem MIPS

Na figura 8, temos a parte responsável por percorrer as posições do vetor e realizar o somatório dos seus valores armazenados.

CÓDIGO	DESCRIÇÃO

`beq $s1, $s0 soma`
`add $t1, $s1, $s1`
`add $t1, $t1, $t1`
`add $t1, $t1, $s6`
`lw $t5, 0($t1)`
	A primeira linha é um teste, para ver se o registrador $s1, mais conhecido como “k”, for igual a $s0(quantidade de valores do vetor), caso ele seja, irá para o próximo rotulo.
	As próximas linhas funcionam como “caminhamento” descrito anteriormente, porém, invés de ser para armazenamento, é para a leitura dos valores de cada posição do vetor:
	$t1 = 2.k.
	$t1 = 4.k.
	$t1 = end.base + 4.k
	O valor de cada posição vai sendo armazenado no registrador temporário $t5.

`add $s2, $s2, $t5`
`slt $t4, $s1, $s0`
`beq $t4, $0, soma`
`addi $s1, $s1, 1`
`j loop3`
	Essa parte do código faz o somatório das posições.
	A primeira linha vai pegando o valor de cada posição do vetor e vai somando em $s2.
	Na segunda linha é feito um teste, se $s1(k)<$s0 (quantidade de valores digitado pelo usuário), então, a temporária $t4=1.
	O beq, na terceira linha, verifica se $t4 possui 0 ou 1, caso seja 0, quer dizer que $s1 é igual a $s0, ou seja, todos os valores já foram somados e o código já pode ir para o próximo rotulo da soma.

	Na penúltima linha é feito o incremento no k (k++0, e por fim, um *jump*, para o começo do *loop* caso ainda haja valores para serem somados.

Pode-se notar que existem dois “beq” neste trecho de código, um no começo e outro no final. O “beq” no começo foi implementado para evitar que fosse somado um valor a mais na soma do vetor, já que o teste do mesmo ocorreria apenas no final, e a soma é feita antes desse último teste.

```
soma:
#PRINTAR O RESULTADO DO SOMATORIO DO VETOR A
        li  $v0, 4
        la  $a0, Mensagem4
        syscall
        li  $v0, 1
        add $a0, $0, $s2
        syscall
        j exit
exit:
#FINALIZA PROGRAMA
        li $v0, 10
        syscall
```

Figura 9 - Código em linguagem de montagem MIPS

Na figura 9 é mostrado a parte final do código, onde é feito o “*print”* do somatório do vetor que na sua última linha faz um *jump* para o exit, que finaliza o código.

Classe	**No de execuções**	**Porcentual**
Aritmética e lógica (ALU)	58	63%
Desvio incondicional (*Jump*)	4	4%
Desvio condicional (*Branch*)	9	10%
Acesso à memória (*Memory*)	4	4%
Outras (*Other*)	17	18%

A análise do código foi feita com um valor passado pelo usuário igual a 2.

PROGRAMA 2

Implemente um programa que leia dois vetores via console e, após a leitura dos vetores, produza um terceiro vetor em que cada elemento seja o maior elemento de mesmo incide dos outros dois vetores. Por fim, o programa deve imprimir esse novo vetor na tela.

```
int main()
{
    int vetor_A[8], vetor_B[8], vetor_C[8]={0};

    int tamanho, contador=0;

    cout << "Digite o tamanho do vetor desejado, entre 2 e 8: ";
    cin >> tamanho;

    while (tamanho <= 1 || tamanho > 8){
        cout << "Valor invalido, digite denovo: ";
        cin >> tamanho;

    }

    while (contador < tamanho){
        cout << "Digite os valores para o vetor A[" << contador << "] = ";
        cin >> vetor_A [contador];
        contador ++;
    }

    contador=0;
    while (contador < tamanho){
        cout << "Digite os valores para o vetor B[" << contador << "] = ";
        cin >> vetor_B [contador];
        contador ++;
    }


    contador =0;
    while (contador < tamanho){
        if (vetor_A [contador] >= vetor_B [contador])
            vetor_C [contador] = vetor_A[contador];
        else
            vetor_C [contador] = vetor_B [contador];

        contador ++;
    }
    contador=0;
    cout << "Vetor resultante: " << endl;
    while (contador < tamanho){
        cout << vetor_C [contador] << "\t";
        contador ++;
    }

}
```

Figura 10 - Código do programa 2 em C++

Após essa prévia do segundo programa, vamos apresentar o mesmo, porém, em linguagem de montagem MIPS.

```
        .data # Segmento de Dados

Vetor_A:        .word 0, 0, 0, 0, 0, 0, 0, 0
Vetor_B:        .word 0, 0, 0, 0, 0, 0, 0, 0
Vetor_C:        .word 0, 0, 0, 0, 0, 0, 0, 0
Mensagem1:      .asciiz "\nEntre com o tamanho do vetor (máx. = 8, min = 2): "
Erro:           .asciiz "\nValor inválido!\n"
Mensagem2:      .asciiz "\nVetor_A["
Mensagem3:      .asciiz "\nVetor_B["
Mensagem4:      .asciiz "\nVetor_C["
Mensagem5:      .asciiz "] = "
```

Figura 11 - Código em linguagem de montagem MIPS

Assim como no primeiro programa, nesse trecho “.data” são declarado os vetores A, B e o C que recebera o maior valor entre os dois. Também são declaradas as mensagens que serão utilizadas para interação via console com o usuário.

```
        .text #segmento de codigo

loadAdress:
        la $s5, Vetor_A
        la $s6, Vetor_B
        la $s7, Vetor_C

testeTamanho:
        addi $t0, $0, 8
        addi $t1, $0, 2
        #print msg 1
        addi $v0, $0, 4
        la $a0, Mensagem1
        syscall

        #lendo nº usuario
        addi $v0, $0, 5
        syscall
        addi $s0, $v0, 0

        #testando
        bgt $s0, $t0, erro
        blt $s0, $t1, erro
        j zerador
```

Figura 12 - Código em linguagem de montagem MIPS

Na figura 12 acima, temos a primeira parte do código com dois rótulos, o *load address* e o teste tamanho. O *load address* nada mais é do que o armazenamento das posições inicias de cada vetor do programa.

No rotulo seguinte é onde ocorre a leitura e o teste do valor do vetor digitado pelo usuário, assim como no programa anterior. Caso o número digitado pelo usuário fosse

maior do que oito ou menor do que dois, a mensagem de erro aparecerá, se não, irá para o próxima etapa do programa.

```
erro:
        addi $v0, $0, 4
        la $a0, Erro
        syscall
        j testeTamanho

zerador:
        addi $t0, $0, 0
        addi $t1, $0, 0
        j inserir_A

zerador2:
        addi $t0, $0, 0
        addi $t1, $0, 0
        addi $t5, $0, 0
        addi $t6, $0, 0
        addi $s1, $0, 0
        j inserir_B

zerador3:
        addi $t0, $0,0
        j comparador
```

Figura 13 - Código em linguagem de montagem MIPS

Na figura 13, temos o rotulo erro, que apenas imprime no console caso o usuário passe algum valor inválido. Em relação aos rótulos zeradores, foi uma necessidade que sentimos durante o programa anterior deste trabalho e então decidimos aplicar neste código. Eles servem apenas para “limpar” alguns registrador já utilizados, para que pudessem ser usados novamente, sem criar um grande “emaranhado” de registradores diferentes no decorrer do programa.

```
inserir_A:
        beq $t0, $s0, zerador2
        #print msg 1
        addi $v0, $0, 4
        la $a0, Mensagem2
        syscall

        #print int [i]
        addi $v0, $0, 1
        add  $a0, $t0, $0
        syscall

        #print msg 5 "]="
        addi $v0, $0, 4
        la $a0, Mensagem5
        syscall

        #posi do vetor para t5
        add $t5, $t0, $t0
        add $t5, $t5, $t5
        add $t5, $t5, $s5

        #read valor
        addi $v0, $0, 5
        syscall
        addi $s1, $v0, 0

        sw $s1, 0($t5)
        addi $t0, $t0,1
        j inserir_A
```

Figura 14 - Código em linguagem de montagem MIPS

Neste trecho de código apresentado na figura 14, temos a parte responsável por preencher o primeiro vetor do usuário. Antes de tudo é feito um teste na primeira linha do código para verificar se todas as posições já foram preenchidas, caso sim, ele irá para o zerador que já foi explicado anteriormente. Os próximos três pequenos trechos de códigos são apenas *prints* do *syscall* para a interação via console com o usuário que será mostrada em breve.

Após a marcação em cinza, temos a parte do código que será responsável pelo armazenamento dos valores no vetor nas suas posições da memória (assim como já foi explicado no programa 1, aqui foi utilizado o mesmo método no qual 2.i, 4.i...., etc). Temos o incrementador do nosso $t0 e o retorno para o início do *loop* para que todas as outras posições também possam ser preenchidas.

```
inserir_B:
        beq $t0, $s0, zerador3
        #print msg 1
        addi $v0, $0, 4
        la $a0, Mensagem3
        syscall

        #print int [i]
        addi $v0, $0, 1
        add  $a0, $t0, $0
        syscall

        #print msg 5 "]="
        addi $v0, $0, 4
        la $a0, Mensagem5
        syscall

        #posi do vetor para t5
        add $t6, $t0, $t0
        add $t6, $t6, $t6
        add $t6, $t6, $s6

        #read valor
        addi $v0, $0, 5
        syscall
        addi $s1, $v0, 0

        sw $s1, 0($t6)
        addi $t0, $t0,1
        j inserir_B
```

Figura 15 - Código em linguagem de montagem MIPS

Aqui (figura 15) temos um trecho idêntico ao apresentado na figura 14, porém, a inserção está acontecendo no *array* do vetor B, portanto, considere as mesmas descrições feitas anteriormente para este rótulo do código.

```
Entre com o tamanho do vetor (máx. = 8, min = 2): 2

Vetor_A[0] = 1

Vetor_A[1] = 1

Vetor_B[0] = 1

Vetor_B[1] = 1
```

Figura 16 - Interação via console com o usuario

```
comparador:
        beq $t0, $s0, exit
        #print msg 4
        addi $v0, $0, 4
        la $a0, Mensagem4
        syscall
        #print int [i]
        addi $v0, $0, 1
        add  $a0, $t0, $0
        syscall
        #print msg 5 "]="
        addi $v0, $0, 4
        la $a0, Mensagem5
        syscall

        add $t5, $t0, $t0
        add $t5, $t5, $t5
        add $t5, $t5, $s5
        lw  $s1, 0($t5)
        add $t6, $t0, $t0
        add $t6, $t6, $t6
        add $t6, $t6, $s6
        lw  $s2, 0($t6)
        add $t7, $t0, $t0
        add $t7, $t7, $t7
        add $t7, $t7, $s7

        addi $t0, $t0,1
        bge $s1, $s2, bota_A_em_C
        j bota_B_em_C
```

Figura 17 - Código em linguagem de montagem MIPS

Na figura 16, temos o principal rótulo do programa dois, o comparador, que será descrito passo a passo na tabela abaixo:

CÓDIGO	DESCRIÇÃO

`beq $t0, $s0, exit`
`#print msg 4`
`addi $v0, $0, 4`
`la $a0, Mensagem4`
`syscall`
`#print int [i]`
`addi $v0, $0, 1`
`add  $a0, $t0, $0`
`syscall`
`#print msg 5 "]="`
`addi $v0, $0, 4`
`la $a0, Mensagem5`
`syscall`
	Neste pedaço do código é feita primeira uma comparação entre o registrador temporário $t0 for igual ao $s0 que é o número que representa o tamanho do vetor digitado pelo usuário, quer dizer que todos as posições dos vetores foram comparadas e o programa irá terminar, caso seja falso, continuará o código abaixo.
	As demais linhas são mensagens de console para o usuário.

`add $t5, $t0, $t0`
`add $t5, $t5, $t5`
`add $t5, $t5, $s5`
`lw $s1, 0($t5)`
`add $t6, $t0, $t0`
`add $t6, $t6, $t6`
`add $t6, $t6, $s6`
`lw $s2, 0($t6)`
`add $t7, $t0, $t0`
`add $t7, $t7, $t7`
`add $t7, $t7, $s7`
 	
`addi $t0, $t0,1`
`bge $s1, $s2, bota_A_em_C`
`j bota_B_em_C`
	Este pedaço do código já está bem familiarizado. É responsável por caminhar nos vetores A e B.
	No final é feito um teste, e caso o valor da posição i do vetor A ($s1) for maior ou igual ao valor da mesma posição no vetor B ($s2), irá chamar o rótulo para colocar o valor de A em C, caso seja o contrário, irá para a próxima linha onde irá colocar o B em C pelo *jump*.

`bota_A_em_C:`
`sw $s1, 0($t7)`
`#print int [i]`
`addi $v0, $0, 1`
`add $a0, $s1, $0`
`syscall`
`j comparador`
 	
`bota_B_em_C:`
`sw $s2, 0($t7)`
`#print int [i]`
`addi $v0, $0, 1`
`add $a0, $s2, $0`
`syscall`
`j comparador`
	Aqui são mostradas os dois rótulos com os códigos que armazenam o maior valor entre os vetores no terceiro vetor C ($t7).
	Após ser armazenado é mandado um *print* ao usuário, mostrando qual foi o maior valor entre os dois vetores e em qual posição do vetor C ele foi armazenado. Em seguida ambos os *jumps* retornam ao comparador que irá realizar todo o processo novamente até que o rotulo *exit* seja solicitado.

```
Entre com o tamanho do vetor (máx. = 8, min = 2): 2

Vetor_A[0] = 1

Vetor_A[1] = 2

Vetor_B[0] = 3

Vetor_B[1] = 1

Vetor_C[0] = 3
Vetor_C[1] = 2
```

Figura 18 - Console de interação com o usuário

Na figura 18 temos um exemplo final de como fica a console de interação com o usuário e, abaixo, uma análise das instruções do código.

Classe	**No de execuções**	**Porcentual**
Aritmética e lógica (ALU)	118	65%
Desvio incondicional (*Jump*)	11	6%
Desvio condicional (*Branch*)	13	7%
Acesso à memória (*Memory*)	10	5%
Outras (*Other*)	30	16%

CONCLUSÃO

O processador MIPS é um bom método de iniciar neste mundo da programação em baixo nível. Sua simplicidade e ao mesmo tempo suas ferramentas em conjunto com o MARS formam um bom ambiente de trabalho e aprendizado.

Ambos os programas foram de ótima aplicação e serviram de uma boa base para o primeiro contato com a linguagem de montagem, ainda mais na nossa área da engenharia.