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

Juntamente com o conhecimento adquirido durante o primeiro trabalho de programação em *assembly*, nesta atividade, adicionamos uma nova maneira de manipular e de “expandir” a nossa capacidade de armazenamento em registradores.

Utilizando a estrutura de dados denominada Pilha, podemos tornar uma tarefa que seria impossível de fazer com poucos registradores, em algo possível, armazenando todos os dados importantes e retirando quando necessário. Através do registrador **$sp** (*Stack Pointer*), podemos acessar a memória principal, expandindo nossas capacidades de armazenamento, sempre seguindo as regras da estrutura de empilhamento (**LIFO).**

1. ENUNCIADO DO PROGRAMA

Utilizando a linguagem de montagem do MIPS, implemente um programa sobre dados geográficos de países que realize a classificação dos mesmos sob os diferentes indicadores (População, PIB per capita, e IDH) utilizando o algoritmo de ordenação *bubblesort* implementado na forma de um procedimento.

2. CÓDIGO EM C++

Abaixo iremos mostrar um exemplo do código que foi feito em linguagem de montagem do MIPS na linguagem C++.

```
#include <iostream>

using namespace std;


int main()
{
    int ddi[6], pop[6], idh[6], pib[6], lista[6]={0};

    int tamanho, organizador, maior,i=0,j, testeddi=0;

    cout << "Digite a quantidade de paises: ";
    cin >> tamanho;

    while (tamanho > 6 || tamanho < 2){
        cout << "\n\nvalor invalido, digite novamente o valor desejado entre 2 e 6: ";
        cin >>tamanho;
    }

    i=0;

    while (i<tamanho){
        cout << "\nDDI [" << i << "]: ";
        cin >> ddi[i];
        testeddi=0;
        j=0;
        while(testeddi==0){
            if(ddi[i] == ddi[j] && i!=j){
                cout << "\nValor ja existente, digite denovo: ";
                cin >> ddi[i];
                j=0;
            }else{
                j++;
            }
            if(j>=tamanho && testeddi==0)
                testeddi = 1;
        }
        i++;
    }
```

```
i=0;
while (i<tamanho){
    cout << "\POP [" << i << "]: ";
    cin >> pop[i];
    i++;
}
i=0;
while (i<tamanho){
    cout << "\IDH [" << i << "]: ";
    cin >> idh[i];
    i++;
}
i=0;
while (i<tamanho){
    cout << "\nPIB [" << i << "]: ";
    cin >> pib[i];
    i++;
}

cout << "\n\nDigite o organizador sendo 1=pop, 2=idh, 3=pib: ";
cin >> organizador;



if (organizador == 1){

    for (j=0; j<tamanho; j++){
        maior=0;
        i=0;
        while (i<tamanho){
            if (pop[maior] < pop[i]){
                maior=i;
            }
            i++;
        }
        pop[maior]*=-1;
        lista[j]=maior;
    }

    for(i=0; i<tamanho; i++)
    pop[i]*=-1;


    for(i=0; i<tamanho; i++){
        cout << "\nºDDI: " << ddi[lista[i]] << "  ºPOP: " << pop[lista[i]] << "  ºIDH: " << idh[lista[i]] << "  ºPIB: " << pib[lista[i]];
    }

}



if (organizador == 2){

    for (j=0; j<tamanho; j++){
        maior=0;
        i=0;
        while (i<tamanho){
            if (idh[maior] < idh[i]){
                maior=i;
            }
            i++;
        }
        idh[maior]*=-1;
        lista[j]=maior;
    }

    for(i=0; i<tamanho; i++)
    idh[i]*=-1;


    for(i=0; i<tamanho; i++){
        cout << "\nºDDI: " << ddi[lista[i]] << "  ºPOP: " << pop[lista[i]] << "  ºIDH: " << idh[lista[i]] << "  ºPIB: " << pib[lista[i]];
    }

}
```

```
    if (organizador == 3){

        for (j=0; j<tamanho; j++){
            maior=0;
            i=0;
            while (i<tamanho){
                if (pib[maior] < pib[i]){
                    maior=i;
                }
                i++;
            }
            pib[maior]*=-1;
            lista[j]=maior;
        }

        for(i=0; i<tamanho; i++)
        pib[i]*=-1;


        for(i=0; i<tamanho; i++){
            cout << "\nºDDI: " << ddi[lista[i]] << "  ºPOP: " << pop[lista[i]] << "  ºIDH: " << idh[lista[i]] << "  ºPIB: " << pib[lista[i]];
        }

    }


}
```

3. CÓDIGO EM LINGUAGEM DE MONTAGEM MIPS

A seguir mostraremos o nosso código em linguagem de montagem MIPS, e iremos explicar cada uma das partes do código.

```
.data

#################################################
Vetor_DDI:      .word 0,0,0,0,0,0
Vetor_POP:      .word 0,0,0,0,0,0
Vetor_PIB:      .word 0,0,0,0,0,0
Vetor_IDH:      .word 0,0,0,0,0,0
```

Primeiramente inicializamos os *arrays* necessários para armazenar todos os dados que o usuário irá digitar, todos com seis posições iniciadas em zero, no segmento de dados.

```
NumeroPaises:    .asciiz "\nDigite a quantidade de paises (Maximo 6, Minimo 2): "
Erro:            .asciiz "\nValor invalido!!!"
ErroIgual:       .asciiz "\nValor digitado ja existe!!!"
VetorDDI:        .asciiz "\nDDI["
VetorPOP:        .asciiz "\nPOP["
VetorPIB:        .asciiz "\nPIB["
VetorIDH:        .asciiz "\nIDH["
FechaColchete:   .asciiz "]= "
Indicador:       .asciiz "\nSelecione o Indicador para ser Utilizado na Classificacao dos Paises"
Opcao1:          .asciiz "\n- 1. Populacao"
Opcao2:          .asciiz "\n- 2. PIB per Capita"
Opcao3:          .asciiz "\n- 3. IDH"
SelecionaOpcao:  .asciiz "\nOpcao: "
Classificacao:   .asciiz "\nClassificacao dos Paises para o Criterio Selecionado: "
POPClass:        .asciiz "Populacao"
PIBClass:        .asciiz "PIB per Capita"
IDHClass:        .asciiz "IDH"
POPRes:          .asciiz ", Populacao = "
PIBRes:          .asciiz ", PIB = "
IDHRes:          .asciiz ", IDH = "
Placar:          .asciiz "o Lugar - DDI =  "
Espaco:          .asciiz "\n\n"
```

Ainda no segmento de dados, declaramos todas as mensagens para interação via console com o usuário através do *syscall.*

```
.text

j Main
```

Assim que iniciamos o segmento de código, fazemos um *jump* para o nosso *main.*

```
Main:   # No rótulo MAIN do código, primeiramente, iremos analisar se a quantidade de paises digitada pelo usuário é válida
        # Inicializando dois registradores, um em 2 e outro em 6, para verificar o tamanho digitado pelo usuário
        addi $s6, $0, 2
        addi $s7, $0, 6
        # Pedindo a quantidade de paises para o usuario (SYSCALL)
        addi $v0, $0, 4
        la $a0, NumeroPaises
        syscall
        # Lendo o numero digitado pelo usuario e armazenando o mesmo em $s0 (SYSCALL)
        addi $v0, $0, 5
        syscall
        addi $s0, $v0, 0
        # Testando se o valor que o usuario digitou é valido
        bgt $s0, $s7, ValorInvalido                    # if ($a0 > 6) então o valor é inválido
        blt $s0, $s6, ValorInvalido                    # if ($a0 < 2) então o valor é inválido
```

A primeira parte do problema no qual o *main* lida é com a obtenção da quantidade de países do usuário. Utilizando os registradores $s6 e $s7, armazenamos,

respectivamente, dois e sete em cada um, para mais adiante utilizarmos os mesmos como critério para o numero digitado pelo usuário.

Se o número digitado pelo usuário for maior que seis ou menor do que dois, o programa apresentará uma mensagem de erro e em seguida pedira novamente um valor.

```
ValorInvalido:
        # Mostrando a mensagem de erro caso o valor digitado pelo usuario seja inválido (SYSCALL)
        addi $v0, $0, 4
        la $a0, Erro
        syscall
        j Main
```

Acima mostramos o procedimento que é chamado caso o valor digitado pelo usuário seja inválido.

Agora, considerando que o valor que o usuário foi válido, o programa continuará a percorrer o *main* normalmente.

```
        # Zerando os registradores $s6 e $s7 para poder reutiliza-los na função de verificação
        add $s6, $0, $0
        add $s7, $0, $0
        # Chamando a função para preencher o vetor de DDI's
        jal SalvaRaMain
```

Na figura acima temos a continuação do *main*, onde zeramos os registradores utilizados anteriormente para comparação, para podermos utiliza-los novamente mais á frente. Na outra linha temos a instrução *jal* que nos levará para o procedimento responsável pelo preenchimento do vetor DDI pelo usuário. Utilizamos o *jal* pelo fato de podermos continuar no *main* de onde paramos ao finalizar o procedimento.

```
SalvaRaMain: # Rótulo com o intuito de armazenar o RA para poder voltar ao main
         # Gravando o RA para o main na Pilha
         addi $sp, $sp, -4
         sw $ra, 0($sp)
         j VetDDI
```

Pelo fato do nosso procedimento de preencher o DDI não ser um procedimento folha, utilizamos esse procedimento “SalvaRaMain” para podermos voltar para o *main*,

salvando o $ra dele. Ao final, finalmente chamamos o procedimento “VetDDI” que irá realizar o preenchimento do vetor DDI.

```
VetDDI: # Rótulo para preencher o vetor DDI dos paises
        # Zerando o nosso $s5 (J) para podermos testar todo o vetor novamente desde o inicio
        add $s5, $0, $0
        # Pedindo os valores DDI para o usuario
        li $v0, 4
        la $a0, VetorDDI
        syscall
        # Printando o valor de i entre colchetes
        li $v0, 1
        addi $a0, $s1, 0
        syscall
        # Printando o fecha colchetes
        li $v0, 4
        la $a0, FechaColchete
        syscall
        # Colocando o endereço base do vetor DDI em $s3
        la $s3, Vetor_DDI
        # Ler o valor que o usuario vai digitar e armazena-lo em $s2
        li $v0, 5
        syscall
        addi $s2, $v0, 0
        # Chamada do procedimento de teste para verificar se o valor DDI digitado pelo usuário já exite
        jal VerificaDDI

      # Deslocamento no vetor DDI (4 em 4)
      add $s4, $s1, $s1
      add $s4, $s4, $s4
      add $s4, $s4, $s3
      # Armazenar o valor digitado pelo usuario
      sw $s2, 0($s4)
      # Incrementando o contador (i++)
      addi $s1, $s1, 1
      # Verificando a quantidade de valores já digitados, se for menor ou igual a quantidade de paises ele vai para o VetDDI novamente
      blt $s1, $s0, VetDDI
      # Load do RA do main da pilha
      lw $ra, 0($sp)
      addi $sp, $sp, 4
      # Empilhando endereço base de ddi
      addi $sp, $sp, -16
      sw $s3, 12 ($sp)
      # Jr para continuar de onde parou no MAIN
      jr $ra
```

Neste procedimento acima, é onde preenchemos o vetor DDI, nele podemos observar diversas chamadas *syscall*, o que é comum se tratando de coleta de dados via console. O DDI do país é o que o identifica, como se fosse um nome, portanto, nesse procedimento é feita uma verificação também, que a cada valor digitado pelo usuário, o procedimento “VerificaDDI”, percorre todo o vetor procurando por elementos repetidos. (Por isso salvamos o $ra do *main* anteriormente, pois chamamos outro procedimento com *jal* aqui, e perderíamos a referência para voltar ao *main*).

```
VerificaDDI: # Procedimento para verificar se o DDI digitado pelo usuário já se encontra no vetor
        # Deslocamento no vetor DDI (4 em 4), $s5 é o nosso J
        add $s6, $s5, $s5
        add $s6, $s6, $s6
        add $s6, $s6, $s3
        # Load no valor da posição do vetor para $s7 para verificar se é igual ao digitado pelo usuario
        lw $s7, 0($s6)
        # Verficar se $s0 é igual a vet[j]
        beq $s7, $s2, ValorIgual
        # Contador J é incrementado (J++)
        addi $s5, $s5, 1
        # Verificar se J é menor ou igual a I, para poder analizar I vezes o vetor, ou seja, analisar apenas o que o usuario já digitou até o momento
        ble $s5, $s1, VerificaDDI
        jr $ra
```

Este é o verificador de valores repetido no vetor DDI, caso haja algum (beq) ele irá para o rótulo com um *syscall* informando que o valor do DDI já existe. O ble na última linha serve para o programa sempre verificar todos os valores do vetor, com o valor que o usuário recém digitou.

```
ValorIgual:
        # Mostrando a mensagem de erro caso o valor digitado pelo usuario já exista no vetor DDI
        addi $v0, $0, 4
        la $a0, ErroIgual
        syscall
        # Zerando o nosso $s5 (J) para podermos testar todo o vetor novamente desde o inicio
        add $s5, $0, $0
        j VetDDI
```

Rótulo que printa a mensagem de que o valor já existe, e zera o contador que percorre o vetor para poder verificar todos novamente com o próximo número que o usuário digitar.

```
        # Deslocamento no vetor DDI (4 em 4)
        add $s4, $s1, $s1
        add $s4, $s4, $s4
        add $s4, $s4, $s3
        # Armazenar o valor digitado pelo usuario
        sw $s2, 0($s4)
        # Incrementando o contador (i++)
        addi $s1, $s1, 1
        # Verificando a quantidade de valores já digitados, se for menor ou igual a quantidade de paises ele vai para o VetDDI novamente
        blt $s1, $s0, VetDDI
        # Load do RA do main da pilha
        lw $ra, 0($sp)
        addi $sp, $sp, 4
        # Empilhando endereço base de ddi
        addi $sp, $sp, -16
        sw $s3, 12 ($sp)
        # Jr para continuar de onde parou no MAIN
        jr $ra
```

Este pedaço restante do procedimento Verifica DDI serve apenas para caminhar pelo vetor e ir armazenando nas suas respectivas posições.

Caso já tenha colocado todos os valores correspondentes ao tamanho que o usuário escolheu, ele irá realizar um POP na pilha, para retornar o endereço do *main*

para o $ra. Abaixo iniciamos uma nova pilha, que será abordada mais tarde, por enquanto, apenas precisamos saber que ela está com o endereço base do vetor DDI salvo.

```
        # zerando registradores usados anteriormente
        jal Zerador
        # POPULACAO
        jal VetPOP
        # zerando registradores usados anteriormente
        jal Zerador
        # PIB
        jal VetPIB
        # Zerando registradores usados anteriormente
        jal Zerador
        # IDH
        jal VetIDH
        # Zerando registradores usados anteriormente
        jal Zerador
```

De volta ao *main*, continuamos de onde paramos graças ao *jal*. Como podemos observar, tudo o que foi feito agora, foi zerar os registrador que foram utilizados anteriormente para usa-los novamente em cada um dos procedimentos de preencher vetores. Foram feitos vários procedimentos, um para cada vetor. Todos com a mesmo método de funcionamento mudando apenas as mensagens do *syscall* por motivos estéticos para o console.

```
VetPOP: # Colocar valores dentro do vetor pop referente a cada pais no ddi
        # Pedindo os valores POP para o usuario
        li $v0, 4
        la $a0, VetorPOP
        syscall
        # Printando o valor de i entre colchetes
        li $v0, 1
        addi $a0, $s1, 0
        syscall
        # Printando o fecha colchetes
        li $v0, 4
        la $a0, FechaColchete
        syscall
        # Colocando o endereço base do vetor POP em $s3
        la $s3, Vetor_POP
        # Ler o valor que o usuario vai digitar e armazena-lo em $s2
        li $v0, 5
        syscall
        addi $s2, $v0, 0
        # Deslocamento no vetor POP (4 em 4)
        add $s4, $s1, $s1
        add $s4, $s4, $s4
        add $s4, $s4, $s3
        # Armazenar o valor digitado pelo usuario
        sw $s2, 0($s4)
        # Incrementando o contador (i++)
        addi $s1, $s1, 1
        # Verificando a quantidade de valores já digitados, se for maior ou igual a quantidade de paises ele vai para o zerador
        blt $s1, $s0, VetPOP
        # Empilhando endereço base de POP
        sw $s3, 8 ($sp)
```

```
        # Jr para continuar de onde parou no MAIN
        jr $ra
```

Acima temos o procedimento de preenchimento do vetor população. Nada nele é novidade, exceto o penúltimo comando. Lembra da pilha que inicializamos no procedimento do preenchimento do vetor DDI? Sim! Exatamente !!! Iremos colocar o endereço base do nosso vetor população na pilha juntamente com ele. Faremos isso com todos os nossos vetores de dados.

```
VetPIB: # Colocar valores dentro do vetor pop referente a cada pais no PIB
        # Pedindo os valores PIB para o usuario
        li $v0, 4
        la $a0, VetorPIB
        syscall
        # Printando o valor de i entre colchetes
        li $v0, 1
        addi $a0, $s1, 0
        syscall
        # Printando o fecha colchetes
        li $v0, 4
        la $a0, FechaColchete
        syscall
        # Colocando o endereço base do vetor PIB em $s3
        la $s3, Vetor_PIB
        # Ler o valor que o usuario vai digitar e armazena-lo em $s2
        li $v0, 5
        syscall
        addi $s2, $v0, 0
        # Deslocamento no vetor PIB (4 em 4)
        add $s4, $s1, $s1
        add $s4, $s4, $s4
        add $s4, $s4, $s3
        # Armazenar o valor digitado pelo usuario
        sw $s2, 0($s4)
        # Incrementando o contador (i++)
        addi $s1, $s1, 1
        # Verificando a quantidade de valores já digitados, se for maior ou igual a quantidade de paises ele vai para o zerador
        blt $s1, $s0, VetPIB
        # Empilhando endereço base de PIB
        sw $s3, 4 ($sp)


        # Jr para continuar de onde parou no MAIN
        jr $ra
```

```
VetIDH: # Colocar valores dentro do vetor pop referente a cada pais no IDH
        # Pedindo os valores IDH para o usuario
        li $v0, 4
        la $a0, VetorIDH
        syscall
        # Printando o valor de i entre colchetes
        li $v0, 1
        addi $a0, $s1, 0
        syscall
        # Printando o fecha colchetes
        li $v0, 4
        la $a0, FechaColchete
        syscall
        # Colocando o endereço base do vetor IDH em $s3
        la $s3, Vetor_IDH
        # Ler o valor que o usuario vai digitar e armazena-lo em $s2
        li $v0, 5
        syscall
        addi $s2, $v0, 0
        # Deslocamento no vetor IDH (4 em 4)
        add $s4, $s1, $s1
        add $s4, $s4, $s4
        add $s4, $s4, $s3
        # Armazenar o valor digitado pelo usuario
        sw $s2, 0($s4)
        # Incrementando o contador (i++)
        addi $s1, $s1, 1
        # Verificando a quantidade de valores já digitados, se for maior ou igual a quantidade de paises ele vai para o zerador
        blt $s1, $s0, VetIDH
        # Empilhando endereço base de IDH
        sw $s3, 0 ($sp)
```

```
        # Jr para continuar de onde parou no MAIN
        jr $ra
```

No final de todos esses procedimentos a nossa pilha ficou da seguinte forma:

Memória	Visualmente
End. DDI	End. IDH
End. POP	End. PIB
End. PIB	End. POP
End. IDH	End. DDI

Após todas estas inserções, e empilhamentos, voltaremos de onde paramos no *main*.

```
        # Desempilhando os endereços bases
        jal Desempilha
```

Iremos desempilhar todos os endereços base dos vetores.

```
Desempilha: # Função que coloca o end. inicial de cada vetor nos registradores correspondentes
        # Desempilhando os endereços bases
        lw $s4, 0($sp)
        lw $s3, 4($sp)
        lw $s2, 8($sp)
        lw $s1, 12($sp)
        addi $sp, $sp, 16
        jr $ra
```

Com todos os endereços desempilhados e em registradores acessíveis, iremos continuar no *main*.

```
        # Agora o usuário irá escolher o critério de ordenação dos paises
        jal EscolhaOrdenacao
```

Vamos chamar o procedimento no qual o usuário irá escolher o vetor índice de ordenação dos países.

```
EscolhaOrdenacao: # Aqui o usuario ira digitar o valor da ordenacao escolhida
        # Mensagem para o usuario escolher a ordem
        li $v0, 4
        la $a0, Indicador
        syscall
        # opções
        li $v0, 4
        la $a0, Opcao1
        syscall
        li $v0, 4
        la $a0, Opcao2
        syscall
        li $v0, 4
        la $a0, Opcao3
        syscall
        li $v0, 4
        la $a0, SelecionaOpcao
        syscall
        #lendo valor
        li $v0, 5
        syscall
        addi $s7, $v0, 0
        # Empilhar o indicador utilizado pelo usuário ($s7)
        addi $sp, $sp -4
        sw $s7, 0($sp)
        #voltando para o main
        jr $ra
```

Neste procedimento de escolha da ordenação iremos mostrar todas as opções para o usuário via console (*syscall*) e armazenar o valor correspondente ao indicador

escolhido em $s7. Iremos armazenar o indicador em uma pilha (sim, novamente uma pilha) para, mais à frente, utilizar como parâmetro para qual mensagem ele irá mostrar no placar final. Com o indicador selecionado voltaremos ao famoso *main*.

```
        # Se for por POP
        addi $s6, $0, 1
        beq $s7, $s6, ParametroPOP
        # Se for por PIB
        addi $s6, $0, 2
        beq $s7, $s6, ParametroPIB
        # Se for por IDH
        addi $s6, $0, 3
        beq $s7, $s6, ParametroIDH
```

No *main* iremos analisar o indicador escolhido pelo usuário. Se ele foi igual a um, o usuário escolheu ordenar por população, então, iremos para o rótulo “ParametroPOP”, se o indicador for dois, iremos para o PIB e se for três iremos para o IDH.

Abaixo mostraremos esses rótulos:

```
ParametroPOP: # Função para colocar os dados da opção selecionada nos registradores de argumento antes de chamar o bubble
        addi $a0, $s0, -1
        add $a1, $s2, $0
        add $a2, $s1, $0
        # Zerar os registradores inuteis antes de ir para o BubbleSort
        jal Zerador
        j BubbleSort

ParametroPIB: # Função para colocar os dados da opção selecionada nos registradores de argumento antes de chamar o bubble
        addi $a0, $s0, -1
        add $a1, $s3, $0
        add $a2, $s1, $0
        # Zerar os registradores inuteis antes de ir para o BubbleSort
        jal Zerador
        j BubbleSort

ParametroIDH: # Função para colocar os dados da opção selecionada nos registradores de argumento antes de chamar o bubble
        addi $a0, $s0, -1
        add $a1, $s4, $0
        add $a2, $s1, $0
        # Zerar os registradores inuteis antes de ir para o BubbleSort
        jal Zerador
        j BubbleSort
```

Com essa imagem acima tudo fará mais sentido. Lembra dos endereços de cada um dos vetores que empilhamos e desempilhamos? aqui eles serão utilizados. Caso o usuário tenha escolhido população, ele colocará em $a1 o endereço base do vetor população que foi desempilhado e em $a2 o endereço base do vetor DDI. Caso foi PIB, $a1 recebe o endereço base do vetor PIB que também estava na pilha e foi desempilhado anteriormente, mesma coisa para o IDH...

Após colocar todos os endereços e valores necessários nos registradores de argumentos iremos para o grande protagonista do nosso código, o *BubbleSort*.

```
BubbleSort: # Função que irá ordenador os valores dentro do vetor escolhido como parametro de ordenação
        bge $s5, $a0, UltimoZerador  # While(N_trocou != tamanho do vetor)
        addi $s5, $0, 0
        addi $sl, $0,0
        jal TrocaTroca
        j BubbleSort
```

Nesta primeira etapa do *BubbleSort* zeramos alguns registradores e criamos a condição de parada, se $s5 for maior ou igual a $a0, que é o tamanho do vetor.

Em seguida damos um *jal* para o famoso troca troca:

```
TrocaTroca: # Função para verificar se é necessaria a troca
         # Constante de parada
         bge $sl, $a0, BubbleSort
         # Caminhamento para i gravado em $s6
         add $s6, $sl, $sl
         add $s6, $s6, $s6
         add $s6, $s6, $al
         ##########################################################################
         addi $sl, $sl,1                          #i+1                          ##
         ##########################################################################
         # Caminhamento para i+1 gravado em $s7
         add $s7, $sl, $sl
         add $s7, $s7, $s7
         add $s7, $s7, $al
         ##########################################################################
         subi $sl, $sl, 1                         #i-1 para voltar ao normal ##
         ##########################################################################
         # Load Words
         lw $s3, 0($s6)
         lw $s4, 0($s7)
         # Verifica se é menor
         blt $s3, $s4, FazTrocaVet
         # Incrementa i++ e n_Trocas++
         addi $sl, $sl, 1
         addi $s5, $s5, 1
         # If ($s1 < $a0) se o contador 'i" for menor que a quantidade de valores no vetor
         blt $sl, $a0, TrocaTroca
         jr $ra
```

O procedimento troca troca é responsável por verificar se o valor na posição i do vetor é menor do que o valor na posição i+1 do vetor. Note que utilizamos um addi para conseguirmos acessar o i+1 e logo em seguida subtraímos 1 novamente para não

prejudicar o restante do código. Caso o valor de i seja menor de i+1 então iremos para o rótulo que fará realmente a troca dos valores:

```
FazTrocaVet: # Função para trocar os valores
        addi $a3, $s3, 0                        # Auxiliar = Vet[i]
        sw $s4, 0($s6)                          # Fazendo a troca, Vet[i] = Vet[i+1]
        sw $a3, 0($s7)                          # vet[i+1] = Auxiliar
        j FazTrocaDDI
```

Acima temos uma clássica troca de valores do vetor com a ajuda de um aulixiar. Ao finalizar a troca iremos para o outro passo.

Pelo fato do DDI do pais representa-lo, todas as alterações que são feitas no vetor indicador, também devem ser aplicadas no vetor DDI, por isso, chamamos um outro rótulo “FazTrocaDDI”.

```
FazTrocaDDI: # Função para trocar os DDI's
        # Pela falta de registradores para fazer o processo de troca no DDI seguindo o vetor indicador, iremos jogar alguns registradores
        # na pilha, são eles: $s3, $s4, Ss6 e $s7
        # Empilhando os registradores
        addi $sp, $sp, -16
        sw $s7, 12($sp)
        sw $s6, 8($sp)
        sw $s4, 4($sp)
        sw $s3, 0($sp)
        # Caminhamento para i gravado em $s6
        add $s6, $s1, $s1
        add $s6, $s6, $s6
        add $s6, $s6, $a2
        ################################################################
        addi $s1, $s1,1                         #i+1                    ##
        ################################################################
        # Caminhamento para i+1 gravado em $s7
        add $s7, $s1, $s1
        add $s7, $s7, $s7
        add $s7, $s7, $a2
        ################################################################
        subi $s1, $s1, 1                        #i-1 para voltar ao normal ##
        ################################################################
        # Load Words
        lw $s3, 0($s6)
        lw $s4, 0($s7)
        # Trocando i pelo i+1
        addi $a3, $s3, 0                        # Auxiliar = Vet[i]
        sw $s4, 0($s6)                          # Fazendo a troca, Vet[i] = Vet[i+1]
        sw $a3, 0($s7)                          # vet[i+1] = Auxiliar


        # Desimpilhando os valores originais dos registradores
        lw $s3, 0($sp)
        lw $s4, 4($sp)
        lw $s6, 8($sp)
        lw $s7, 12($sp)
        addi $sp, $sp 16
        # i++
        addi $s1, $s1, 1
        j TrocaTroca
```

Ele apenas replica todos as trocas que ocorrem no vetor indicar nele mesmo, a única diferença nele é a implementação de uma pilha. Por conta da falta de

registradores, utilizamos os mesmo que são usados no procedimento do vetor indicar, note que salvamos todos os dados na pilha no começo e no final retornamos.

Após todo o *BubbleSort* ser finalizado iremos para outro rótulo zerador:

```
UltimoZerador:
        addi $s1, $0, 1
        addi $s2, $0, 0
        addi $s5, $0, 0
        addi $s6, $0, 0
        addi $s7, $0, 0
        # Recuperar o indicador selecionado pelo usuário
        lw $s4, 0($sp)
        addi $sp, $sp, 4
        j Colocacao
```

Lembra do número indicador que empilhamos anteriormente lá no procedimento de escolher o indicador? Iremos recupera-lo aqui com um POP utiliza-lo no próximo passo do código, que é mostrar o placar final, ou seja, a colocação.

```
Colocacao: # Mostra os paises na ordem de suas respectivas colocações
        # Enquanto for diferente do tamanho do vetor, executará o programa
        bgt  $s1, $s0, Exit
        # Syscall frufru :3
        li $v0, 4
        la $a0, Espaco
        syscall
        # Chamando o Syscall para imprimir a posição, (famoso I)
        li $v0, 1
        addi $a0, $s1, 0
        syscall
        # Lugar print
        li $v0, 4
        la $a0, Placar
        syscall
        # Caminhando no vetor DDI
        add $s2, $s1, $s1
        add $s2, $s2, $s2
        add $s2, $s2, $a2
        lw $s3, -4($s2)
        # Printando o valor de DDI
        li $v0, 1
        addi $a0, $s3, 0
        syscall
```

```
# Testes para POP
addi $s2, $0, 1
beq $s2, $s4, ImprimePOP
# Testes para PIB
addi $s2, $0, 2
beq $s2, $s4, ImprimePIB
# Testes para IDH
addi $s2, $0, 3
beq $s2, $s4, ImprimeIDH
```

Esse rótulo irá mostrar uma parte da classificação final no console para o usuário que é igual para todos, a seguir utilizamos três beq's para verificar qual foi o indicador escolhido pelo usuário (desempilhado anteriormente), e com base nesse indicador, será chamado um imprime que mostrará uma mensagem diferente para cada caso. Os três imprimes são iguais, mudando apenas a mensagem que é chamada pelo *syscall*.

```
ImprimePOP: # Funcao para imrpimir o vetor pop
        li $v0, 4
        la $a0, POPRes
        syscall
        # Caminha no vetor POP
        add $s2, $s1, $s1
        add $s2, $s2, $s2
        add $s2, $s2, $a1
        lw $s3, -4($s2)
        # Printa valor do vetor POP
        li $v0, 1
        addi $a0, $s3, 0
        syscall
        # I++
        addi $s1, $s1, 1
        j Colocacao
```

```
ImprimePIB: # Funcao para imrpimir o vetor PIB
        li $v0, 4
        la $a0, PIBRes
        syscall
        # Caminha no vetor PIB
        add $s2, $s1, $s1
        add $s2, $s2, $s2
        add $s2, $s2, $a1
        lw $s3, -4($s2)
        # Printa valor do vetor PIB
        li $v0, 1
        addi $a0, $s3, 0
        syscall
        # I++
        addi $s1, $s1, 1
        j Colocacao
```

```
ImprimeIDH: # Funcao para imrpimir o vetor IDH
        li $v0, 4
        la $a0, IDHRes
        syscall
        # Caminha no vetor IDH
        add $s2, $s1, $s1
        add $s2, $s2, $s2
        add $s2, $s2, $a1
        lw $s3, -4($s2)
        # Printa valor do vetor IDH
        li $v0, 1
        addi $a0, $s3, 0
        syscall
        # I++
        addi $s1, $s1, 1
        j Colocacao
```

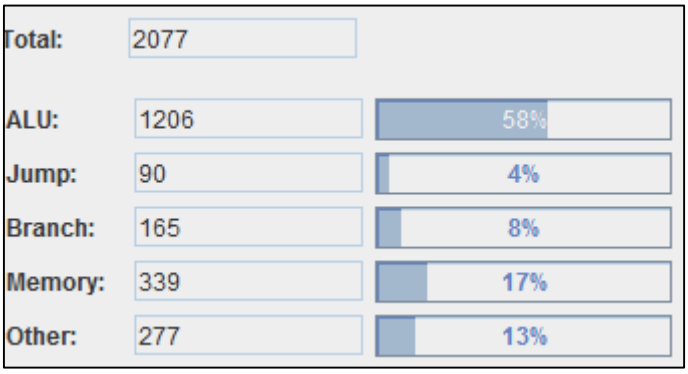
Como utilizamos o i como índice de caminhamento no vetor e como a colocação na classificação, quando damos o lw utilizamos -4, pois ele é iniciado em 1, e precisamos começar do 0 para caminhar corretamente pelo vetor, por isso, é sempre i-1.

Após imprimir todos os dados do vetor índice, o programa irá finalizar no *exit* pelo “colocação”.

```
Exit:
        nop
```

4. ESTATÍSTICAS DE INSTRUÇÕES

As estatísticas a seguir foram tiradas após uma simulação completa, com o tamanho máximo dos vetores.



5. CONCLUSÃO

Com esta atividade conseguimos entender a importância da aplicação da pilha em códigos de montagem e como ela pode ajudar a simplificar diversos procedimentos nos quais forem necessários mais registradores do que o processador pode proporcionar, já que, se tratam de arquiteturas limitadas e nós como programadores *assembly*, precisamos saber utilizar de recursos da forma mais correta possível, para o que código se torne elegante, porém, funcional.