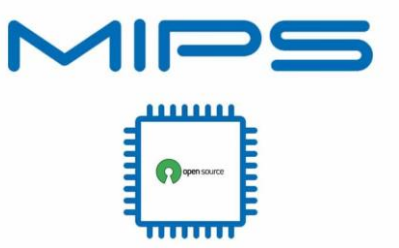
Faculdade de Computação

Arquitetura e Organização de Computadores 1

Laboratório de Programação Assembly 3

Prof. Cláudio C. Rodrigues

Programando a Arquitetura MIPS



MIPS, acrônimo para **Microprocessor without Interlocked Pipeline Stages** (microprocessador sem estágios intertravados de pipeline), é uma arquitetura de microprocessadores RISC desenvolvida pela MIPS Computer Systems.

Arquitetura

O MIPS é uma arquitetura baseada em registrador, ou seja, a CPU usa apenas registradores para realizar as suas operações aritméticas e lógicas. Existem outros tipos de processadores, tais como processadores baseados em pilha e processadores baseados em acumuladores. Processadores baseados no conjunto de instruções do MIPS estão em produção desde 1988. Ao longo do tempo foram feitas várias melhorias do conjunto de instruções. As diferentes revisões que foram introduzidas são MIPS I, MIPS II, MIPS III, MIPS IV e MIPS V. Cada revisão é um super conjunto de seus antecessores. Quando a MIPS Technologies saiu da Silicon Graphics em 1998, a definição da arquitetura foi alterada para definir um conjunto de instruções MIPS32 de 32 bits e um MIPS64 de 64 bits.

Projetos MIPS são atualmente bastante usados em muitos sistemas embarcados como dispositivos Windows CE, roteadores Cisco e consoles de games como: Nintendo 64, Playstation, Playstation 2 e Playstation Portable.

A Filosofia do Projeto do MIPS:

- A simplicidade favorece a regularidade.
  O menor é (quase sempre) mais rápido. Levando em conta que uma corrente elétrica se propaga cerca de um palmo por nanosegundo, circuitos simples são menores e portanto, mais rápidos.
- Um bom projeto demanda compromissos.
- O caso comum DEVE ser mais rápido.
  É muito melhor tornar uma instrução que é usada 90% do tempo 10% mais rápida do que fazer com que uma instrução usada em 10% das vezes torne-se 90% mais rápida. Esta regra é baseada na "Lei de Amdahl".

Instruções:

I. Apresentar as soluções usando a linguagem de montagem da Arquitetura de processadores MIPS.
II. O trabalho deve ser desenvolvido em grupo composto de 1 até 5 (um até cinco) integrantes e qualquer identificação de plágio sofrerá penalização;
III. Entrega dos resultados deverá ser feita por envio de arquivo zipado com os seguintes artefatos de software: Memorial descritivo das soluções em pdf; arquivo em txt com os códigos que solucionam os problemas propostos escritos em MIPS assembly.
IV. Submeter os documentos na plataforma MS Teams, dentro do prazo definido para a atividade.

Desafios de Programação MIPS:

P1. Escreva um programa em MIPS assembly, que implemente a função ***isdigit(c)*** para verificar se o caractere passado como parâmetro é um dígito ou não. A função deve retornar o valor inteiro **1** se for um dígito, caso contrário, ele retorna **0**.

A função ***isdigit(c)*** deve ser declarada no programa MIPS assembly, verificando se o caractere inserido é um caractere numérico [0 – 9] ou não.

O caractere de teste deve ser obtido pela função ***getchar()*** definida abaixo, que emprega a técnica E/S programada com dispositivo mapeado em memória (**MMIO**) emulado no simulador MARS. A resposta da função deve ser escrita como *string*, utilizando a função ***print_string()***, definida abaixo.

```
getchar: # retorna  char em $v0
     lw   $v0, 0xffff0000
     andi $v0, $v0, 0x01
     beq  $v0, $zero, getchar
     lw   $v0, 0xffff0004
     jr   $ra
```

```
print_string: # $a0: & da string
     li   $t0, 0xffff00008
     j    ps_cond
ps_loop:
     lw   $v0, ($t0)
     andi $v0, $v0, 0x01
     beq  $v0, $zero, ps_loop
     sw   $t1, 4($t0)
ps_cond:
     lbu  $t1, ($a0)
     addi $a0, $a0, 1
     bne  $t1, $zero, ps_loop
     jr   $ra
```

O algoritmo deve estar contido no arquivo "**isdigit.asm**"

P2. Converta o fragmento de código abaixo, escrito em linguagem C, para a linguagem do MIPS assembly.

```
/** Converts the string S to lowercase */
void string_to_lowercase(char *s) {
        for(char c = *s; (c=*s) != '\0'; s++) {
                if(c >= 'A' && c <= 'Z') {
                        *s += 'a' - 'A';
                }
        }
}
```

O algoritmo deve estar contido no arquivo "**tolower.asm**"

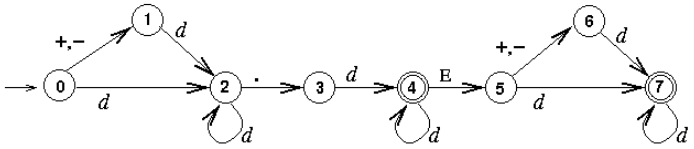
P3. Escreva um procedimento ***atoi*** *(ASCII to int)*, na linguagem de montagem do MIPS, que converta uma *string* decimal em ASCII para um número inteiro. Seu programa deve considerar que o registrador **$a0** tem endereço do caractere null, usado para marcar o fim da *string* que contém alguma combinação dos dígitos de 0 a 9. Seu programa deve calcular o valor inteiro correspondente a essa *string* de bits, colocando seu valor no registrador **$v0**. Seu programa não deve preocupar-se com números negativos. Se for identificado no *string* um caractere que não represente um dígito, seu programa deve retornar o valor -1 no registrador **$v0**. Por exemplo, se o registrador **$a0** aponta para uma sequência de três bytes: 50_{10}, $52_{10,}$ 0_{10} , seguida do caractere null = 24, então, quando o programa termina, o registrador **$v0** deve ter o valor 24_{10} armazenado nele. (O subscrito "10" significa base 10.)

O algoritmo deve estar contido no arquivo "**atoi.asm**"

P4. Escreva um programa em MIPS assembly, que implemente a função ***floatValidate(string).*** A função deve receber um *string* (por exemplo, 3526.123) que representa um valor real, realiza a verificação seguindo o autômato ilustrado abaixo e retorna um valor inteiro: **1** se o *string* representa um valor real; ou **0** se o *string* não representa um valor real.

A ***expresão regular*** da representação de um número real pode ser definida como:

`[-+]?([0-9]*[.])?[0-9]+([eE][-+]?\d+)?`



^ **Beginning.** Matches the beginning of the string, or the beginning of a line if the multiline flag (**m**) is enabled.

[**Character set.** Match any character in the set.

- **Character.** Matches a "-" character (char code 45).

+ **Character.** Matches a "+" character (char code 43).

]

? **Quantifier.** Match between 0 and 1 of the preceding token.

(**Capturing group #1.** Groups multiple tokens together and creates a capture group for extracting a substring or using a backreference.

[**Character set.** Match any character in the set.

0-9 Range. Matches a character in the range "0" to "9" (char code 48 to 57). Case sensitive.

]

* **Quantifier.** Match 0 or more of the preceding token.

[**Character set.** Match any character in the set.

. **Character.** Matches a "." character (char code 46).

]

)

? **Quantifier.** Match between 0 and 1 of the preceding token.

[**Character set.** Match any character in the set.

0-9 Range. Matches a character in the range "0" to "9" (char code 48 to 57). Case sensitive.

]

+ **Quantifier.** Match 1 or more of the preceding token.

(**Capturing group #2.** Groups multiple tokens together and creates a capture group for extracting a substring or using a backreference.

[**Character set.** Match any character in the set.

e **Character.** Matches a "e" character (char code 101). Case sensitive.

E **Character.** Matches a "E" character (char code 69). Case sensitive.

]

[**Character set.** Match any character in the set.

- **Character.** Matches a "-" character (char code 45).

+ **Character.** Matches a "+" character (char code 43).

]

? **Quantifier.** Match between 0 and 1 of the preceding token.

? **Quantifier.** Match between 0 and 1 of the preceding token.

\d **Digit.** Matches any digit character (0-9).

+ **Quantifier.** Match 1 or more of the preceding token.

)

? **Quantifier.** Match between 0 and 1 of the preceding token.

O algoritmo deve estar contido no arquivo "**valida_float.asm**"

P5. Escreva um programa em linguagem *MIPS Assembly* que receba como entrada os vértices de definição de dois retângulos no plano cartesiano O programa deve determinar se eles se interceptam ou não, caso haja interseção mostrar as coordenadas do **retângulo de interseção** resultante. A entrada contém um único conjunto de testes, que deve ser lido do dispositivo padrão de entrada (teclado). Cada caso de teste contém duas linhas. Cada linha contém quatro inteiros (x0, y0, x1, y1, sendo $0 \leq x0 < x1 \leq 1.000.000$ e $0 \leq y0 < y1 \leq 1.000.000$) separados por um espaço em branco representando um retângulo. Os lados do retângulo são sempre paralelos aos eixos x e y.

Entrada: o par x1, y1 representa a coordenada do canto superior esquerdo e o par x2, y2 representa a coordenada do canto inferior direito de um retângulo.

Exemplos de entrada:
1 5 6 3
2 4 7 1
Saída: Para cada conjunto de teste da entrada seu programa deve produzir as coordenadas do retângulo de interseção encontrado pelo seu programa, no mesmo formato utilizado na entrada. Caso a interseção seja vazia, deve conter a expressão "não há interseção".

Exemplos de saída:

Retângulo interseção:

2 4 6 3

O algoritmo deve estar contido no arquivo "**retangulos.asm**"