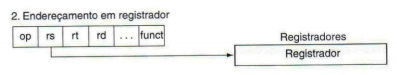
**RESUMO DOS MODOS DE ENDEREÇAMENTO NO MIPS**

Existem múltiplas formas de se endereçar um dado, geralmente essas formas são denominadas **modos de endereçamento.** Os modos de endereçamento do MIPS são os seguintes:

1. **Endereçamento em registrador,** onde o operando é um registrador.

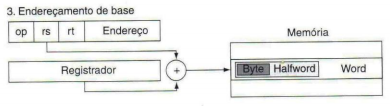
Ex.: *add $s1, $s2, $s3*



1. **Endereçamento de base ou deslocamento,** onde o operando está no local da memória cujo endereço é a soma de um registrador e uma constante na instrução.

0

Ex.: *lw $s1, offset($s2)*

**

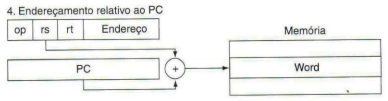
1. **Endereçamento imediato,** onde o operando é uma constante dentro da própria instrução.

Ex.: *addi $s1, $s2, value*

**

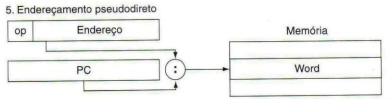
1. **Endereçamento relativo ao PC:** em uma operação de desvio condicional, é necessário especificar dois operandos além do valor de destino, logo, sobram apenas 16 bits para o endereço. Verificando que geralmente esses desvios referenciam locais próximos a ele, podemos calcular o endereço tomando como base o endereço da instrução atual (PC, mas na verdade é utilizado o endereço da próxima instrução: PC + 4) variando 216 bits para mais ou para menos.

Ex.: *beq $s1, $s2, value*

**

1. **Endereçamento pseudodireto,** onde o endereço de jump são os 26 bits da instrução concatenados com os bits mais altos do PC. Nesse caso, ocorre a seguinte operação: os 26 bits do jump se referem a quantidade de words que deveriam ser “puladas”, porém, é necessário fazer essa referência em bytes, uma vez que a memória é assim referenciada. Para isso, multiplicamos o endereço especificado nos 26 bits do jump por 4, que é a mesma coisa de deslocar dois bits à direita. Após isso, os 4 bits mais significativos do PC são concatenados com os 28 bits obtidos na operação de deslocamento, dando origem ao endereço de 32 bits da próxima instrução a ser executada.

Ex.: *j 1000*



**Arquitetura RISC(MIPS)**

A arquitetura RISC é constituída por um pequeno conjunto de instruções simples que são executadas diretamente pelo hardware, onde não há a intervenção de um interpretador (microcódigo), o que significa que as instruções são executadas em apenas uma microinstrução(de uma única forma e seguindo um mesmo padrão). As máquinas RISC só se tornaram viáveis devido aos avanços de software otimizado para essa arquitetura, através da utilização de compiladores otimizados e que compensam a simplicidade dessa arquitetura. Existe um conjunto de características que permite uma definição de arquitetura básica RISC, são elas:

* Utilização de apenas uma instrução por ciclo do datapath (ULA, registradores e os barramentos que fazem sua conexão);
* **O processo de carregar/armazenar, ou seja, as referências à memória são feitas por instruções especiais de load/store;**
* Inexistência de microcódigo, fazendo com que a complexidade esteja no compilador;
* **Instruções de formato fixo;**
* **Conjunto reduzido de instruções, facilitando a organização da Unidade de controle de modo que esta tenha uma interpretação simples e rápida;**
* Utilização de pipeline ( é uma técnica de dividir a execução de uma instrução em fases ou estágios, abrindo espaço para execução simultânea de múltiplas instruções);
* **Utilização de múltiplos conjuntos de registradores**.

Ao contrário dos complexos CISC, os processadores RISC são capazes de executar apenas poucas instruções simples, e justamente por isso que os chips baseados nesta arquitetura são mais simples e muito mais baratos. Uma outra vantagem dos processadores que utilizam essa arquitetura é o

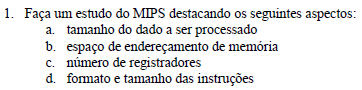
fato de terem um menor número de circuitos internos, permite que se trabalhe com frequências mais altas de clocks. Um bom exemplo são os processadores Alpha, que em 97 já operavam a 600 MHz.

<http://www.sistemasembarcados.org/2015/11/15/processadores-arquitetura-risc-e-cisc/>

**LISTA COMEÇA NA PRÓXIMA PÁGINA**

**Lista de Exercícios - Infra de Hardware**

1ª unidade



**Resposta:**

MIPS é uma arquitetura de processadores em que o **tamanho do dado a ser processado** é de 32 bits (também possui a versão 64 bits, o MIPS64) para inteiros, 32 e 64 bits para pontos flutuantes (onde há a combinação de words), 8 bits para caracteres e 8 bits para dados lógicos.

O **espaço de endereçamento** da memória tem 232 bytes (aproximadamente 4 GB), para conferir essa informação, basta atentarmos ao registrador contador de programa, o qual possui 32 bits. Naturalmente, podemos perceber que a memória é referenciada por byte, ou seja, cada byte possui um endereço diferente.

Por que o espaço de endereçamento não são de 32 words, visto que seria possível endereçar um espaço muito maior? Os vários tipos de dados possuem tamanhos diferentes, logo, um caractere - que possui 8 bits - ocuparia apenas um dos 4 bytes da word que foi reservada, ocasionando o desperdício dos outros 3 bytes.

Esse tipo de endereçamento (por bytes), traz um problema: como armazenar uma word, que possui 32 bits, na memória? Na arquitetura MIPS, utiliza-se o método de ordenação de dado onde o byte mais significativo ocupa o primeiro byte da memória reservado para aquela word, o segundo byte mais significativo ocupa o próximo byte reservado e assim por diante, até o fim da word. Esse método de representação do dado é chamado de *Big Endian*.

A arquitetura MIPS possui **32 registradores** de 32 bits cada, visto que esse é o tamanho do dado adotado pela arquitetura.

Nesse contexto, todas as instruções possuem **tamanho único**: **32 bits**. Isso facilita na implementação do hardware. Cada instrução possui um formato, sendo este definido pelo dado contido na instrução. O MIPS possui três formatos:

* **Formato R**: instruções que envolvem operações com registradores. Possuem a seguinte configuração,



Onde o campo *op* se refere ao código da operação a ser realizada, *rs, rt e rd* se referem ao número de um determinado registrador a ser utilizado na operação, *shamt,* à quantidade de deslocamento a ser efetuado por operações como shift left ou shift right logical e o campo *funct* que representa qual das instruções de um subconjunto vai ser realizada.

Instruções aritméticas, lógicas e de deslocamento utilizam esse formato.

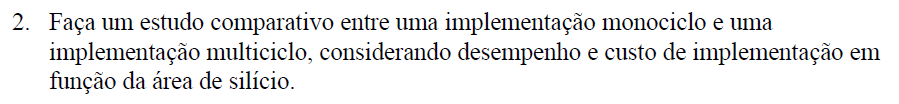
* **Formato I:** instruções de informação. Possuem a seguinte configuração,



Ao invés de se ter os campos *rd, shamt* e *funct* como nas instruções de formato R, aqui existe um campo de endereço destinado para constantes ou endereços a serem utilizados por esse tipo de instrução.

Instruções de acesso à memória, com constante ou desvios condicionais possuem esse formato.

* **Formato J:** desvio incondicional. Nesse formato, os 6 bits mais significativos representam o código da operação e os 26 bits restantes são utilizados para formação do endereço para qual a execução será desviada.



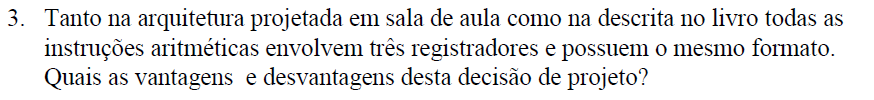
**Resposta:**

**Monociclo -** Toda instrução é executada em um ciclo de clock.

* Vantagens: apenas a facilidade de implementação do controle.
* Desvantagens: a instrução que demora mais tempo determina o tempo de clock. Para que cada instrução seja capaz de ser executada em um ciclo de clock, é necessário ter algumas unidades duplicadas (duas memórias, 1 ALU, dois somadores).

**Multiciclo -** Cada etapa da instrução ocorre em um ciclo de clock.

* Vantagens: clock definido pela etapa mais lenta. Precisa de componentes mais simples, como multiplexadores, sem a necessidade de duplicar componentes mais complexos.
* Desvantagens: necessita de mais registradores para guardar os resultados obtidos na execução da etapa da instrução realizada naquele ciclo de clock. Além disso, a implementação do controle fica bem mais complexa, uma vez que se torna necessário construir uma máquina de estados.



**Resposta:**

Vantagens

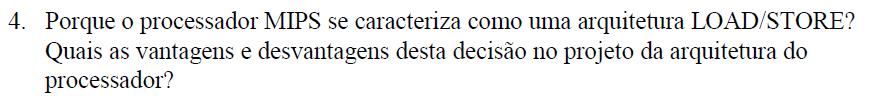
- Velocidade em se ler um array de registradores (32 registradores), ao invés da memória, um vetor de 2^32 bytes. A tecnologia adotada na construção de registradores também é de leitura mais rápida que a leitura da memória.

- A quantidade específica de 3 registradores ocasiona a preservação dos operandos, não obtida com instruções aritméticas de 2 registradores, por exemplo. Diminui também o número de acessos à memória.

- Facilidade de implementação de obter instruções de um formato fixo.

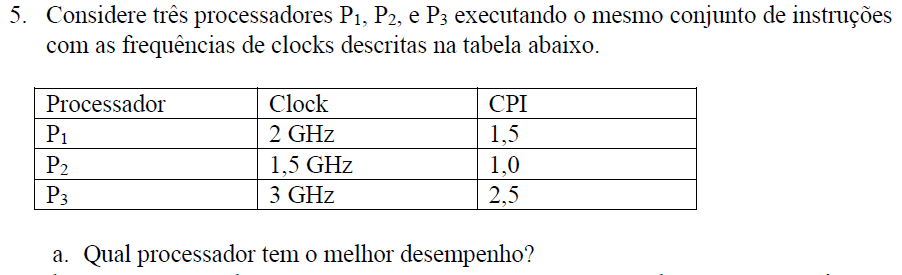
Desvantagens

- Como desvantagem, pela necessidade de carregar as variáveis nos registradores, mais instruções são necessárias, aumentando o código. O processador também precisa ter mais registradores.



**Resposta:**

O MIPS se caracteriza dessa forma por só acessar a memória com as instruções *load* e *store*. Se só essas duas acessam a memória, as outras utilizam registradores e/ou constantes, o que deixa o processamento muito mais rápido, uma vez que a tecnologia presente nos registradores é diferente da memória, favorecendo-o.

0

**Resposta:**

**Suponhamos que o nº de instruções é igual a I.**

Tempo de execuçãoP1 = (Nº de instruções x CPIP1)/Velocidade de clockP1

Tempo de execuçãoP1 = (I . 1,5)/2.109 = **0,75.10-9.I s**

Tempo de execuçãoP2 = (Nº de instruções x CPIP2)/Velocidade de clockP2

Tempo de execuçãoP2 = (I . 1,0)/1,5.109 = **0,67.10-9.I s**

Tempo de execuçãoP3 = (Nº de instruções x CPIP3)/Velocidade de clockP3

Tempo de execuçãoP2 = (I . 2,5)/3,0.109 = **0,83.10-9.I s**

Como o desempenho é o inverso do tempo de execução, chegamos a conclusão de que o processador P2 é o que possui melhor desempenho (possui o menor tempo de execução).



**Resposta:**

**P1**

Tempo de execuçãoP1 = (Nº de instruçõesP1 x CPIP1)/Velocidade de clockP1

10 = (IP1 x 1,5)/(2.109)

2.1010 =(IP1 x 1,5)

(2.1010/1,5)=IP1

**IP1 = 1,3.1010**

Nº de ciclosP1 = Nº de instruçõesP1 x CPIP1

Nº de ciclosP1 = 1,3.1010 x 1,5

**Nº de ciclosP1 = 1,95.1010**

**P2**

Tempo de execuçãoP2 = (Nº de instruçõesP2 x CPIP2)/Velocidade de clockP2

10 = (IP2 x 1,0)/(1,5.109)

**IP2 = 1,5.1010**

Nº de ciclosP2 = Nº de instruçõesP2 x CPIP2

Nº de ciclosP2 = 1,5.1010 x 1,0

**Nº de ciclosP1 = 1,5.1010**

**P3**

Tempo de execuçãoP3 = (Nº de instruçõesP3 x CPIP3)/Velocidade de clockP3

10 = (IP3 x 2,5)/(3,0.109)

3.1010 =(IP3 x 2,5)

**IP3 = 1,2.1010**

Nº de ciclosP1 = Nº de instruçõesP1 x CPIP1

Nº de ciclosP1 = 1,2.1010 x 2,5

**Nº de ciclosP1 = 3,0.1010**



Pela equação

*Tempo de execuçãoP1 = (Nº de instruçõesP1 x CPIP1)/Taxa de clockP1*

percebemos que se a quantidade de CPI aumenta e o tempo de execução diminui, naturalmente a velocidade do clock também tem que aumentar.

Consideremos

*Taxa de clock= (Nº de instruções x CPI)/Tempo de execução*

Taxa de clocknova = (Nº de instruções x CPInovo)/Tempo de execuçãonovo

Taxa de clocknova = (Nº de instruções x 1,2.CPI)/0,7\*Tempo de execução

Taxa de clocknova = 1,2\*(Nº de instruções x CPI)/0,7\*Tempo de execução

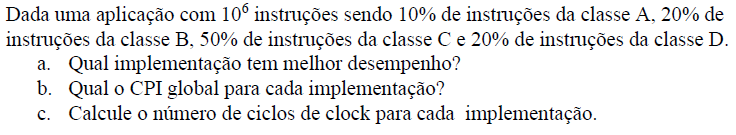
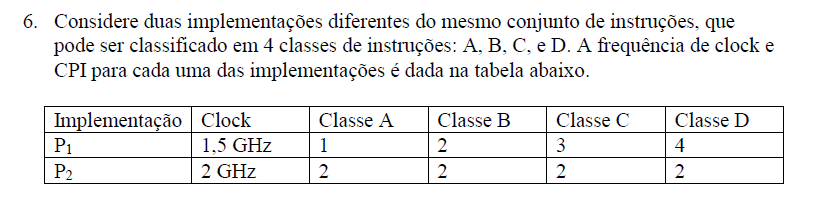
Taxa de clocknova = (1,2/0,7)(Nº de instruções x CPI)/Tempo de execução

Taxa de clocknova = 1,71(Taxa de clockantiga)

Taxa de clockP1 = 1,71\*2\*109 = 3,42\*109

Taxa de clockP2 = 1,71\*1,5\*109 = 2,57\*109

Taxa de clockP3 = 1,71\*3\*109 = 5,13\*109

Sabemos que o CPI é a média da quantidade de clocks de todas as instruções, logo

CPI = ((Nº de instruçõesclasse Ax CPIclasse A)+(Nº de instruçõesclasse Bx CPIclasse B)+(Nº de instruçõesclasse Cx CPIclasse C)+(Nº de instruçõesclasse Dx CPIclasse D))/ Nº de instruçõestotal

CPIP1 = ((105.1) + (2.105.2) + (5.105.3) + (2.105.4))/106

CPIP1 = 2,8 = CPImédio

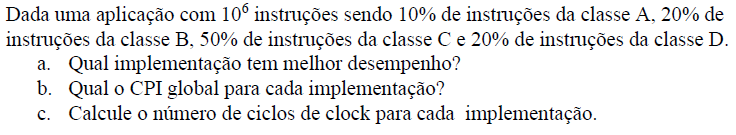
Tempo de execuçãoP1 = (CPImédioxNº de instruções)/Taxa de clockP1

**Tempo de execuçãoP1 = (2,8.106)/1,5.109 = 1,87.10-3s**

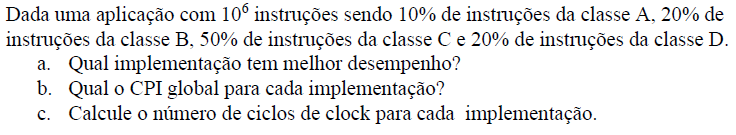
CPIP2 = (2.106)/106 = 2

Tempo de execuçãoP2 = (CPImédioxNº de instruções)/Taxa de clockP2

**Tempo de execuçãoP2 = (2.106)/2.109 = 10-3s**

Concluímos então que o processador 2 possui um desempenho melhor do que o processador 1.

Para a implementação P1 é de 2,8 e para a implementação P2 é 2.



Nº de ciclo de clocks = CPI x Nº de instruções

Nº de ciclo de clocksP1 = 2,8 x 106 = 2,8.106 ciclos

Nº de ciclo de clocksP2 = 2 x 106 = 2.106 ciclos

**7 -**