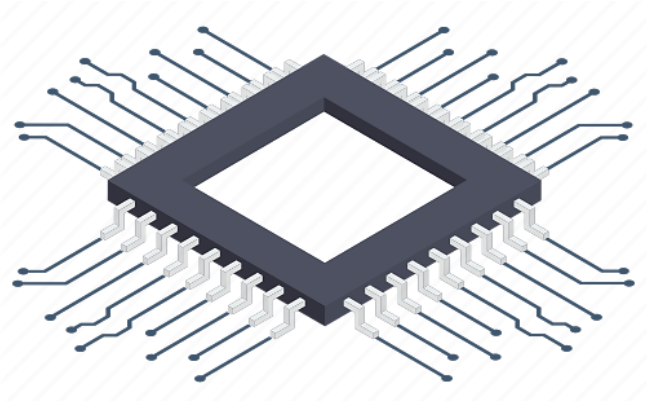
C208 – Arquitetura de Computadores

Capítulo 1 – Introdução à Arquitetura de Computadores

(parte 4)



Prof. Yvo Marcelo Chiaradia Masselli

Arquitetura MIPS

Melhoria de Desempenho do Processador

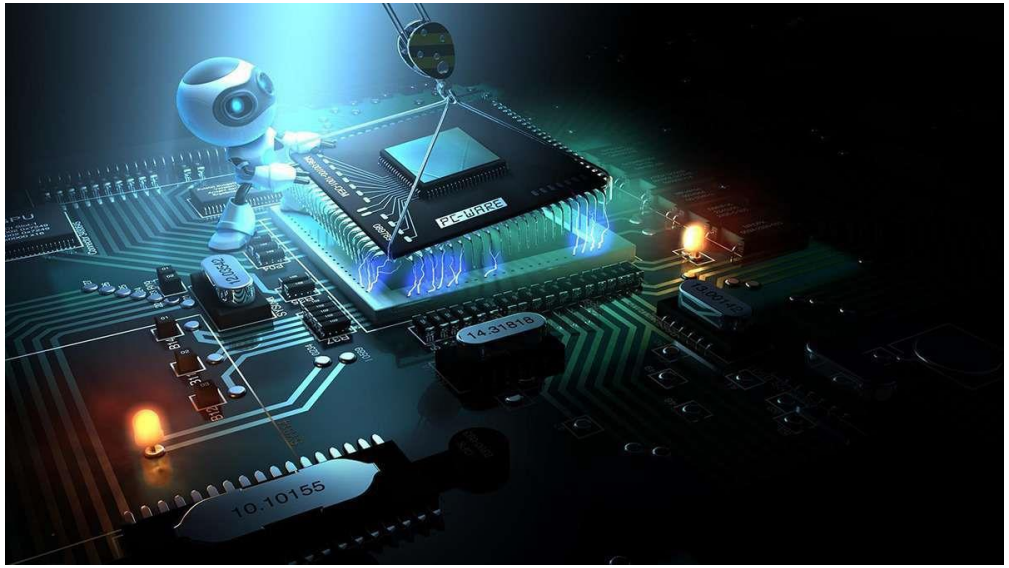
Paralelismo a Nível de Instrução - Pipeline

Melhoria de Desempenho de Processadores

Há duas maneiras de aumentar o desempenho de uma CPU:

- Aperfeiçoar o hardware com circuitos mais rápidos;

- Organizar o hardware de maneira que mais de uma operação possa ser executada ao mesmo tempo.



Melhoria de Desempenho de Processadores

O projeto do processador MIPS visto anteriormente é conhecido como MONOCICLO.

Neste modelo, cada instrução é executada dentro de um único ciclo de *clock.*

- *O problema com este projeto é sua* ineficiência no tempo de processamento.

- Todas as instruções têm o mesmo tamanho de ciclo de instrução, ou seja, um período de *clock.*

- *O tamanho do* ciclo deve ser grande o suficiente para suportar o maior atraso relativo dos componentes utilizados por uma determinada instrução (em geral, as instruções de acesso à memória possuem o maior tempo de execução).

Melhoria de Desempenho de Processadores

Na implementação do processador usando o modelo MONOCICLO, as instruções (Tipo-R, Tipo-I e Tipo-J) mesmo tendo acessos a dispositivos diferentes (Memória de programa, Registradores e Memória de Dados), todas elas precisam ser executadas dentro do período de um ciclo de *clock* e isto afeta o desempenho.

Para execução de uma instrução, dependendo do seu tipo, são realizadas várias operações como:

- Busca (leitura de memória de programa)
- Acesso a Registradores
- Acesso a operações na ALU
- Acesso a memória de dados

Melhoria de Desempenho de Processadores

Considerando, a título de exemplo, que cada operação seja realizada em **1ns**, temos:

Classe	Unidades funcionais utilizadas					Tempo
Tipo R	Busca (2)	Acesso a reg (1)	ALU (2)	Acesso a reg (1)		6 ns
Load word	Busca (2)	Acesso a reg (1)	ALU (2)	Acesso a mem (2)	Acesso a reg (1)	8 ns
Store word	Busca (2)	Acesso a reg (1)	ALU (2)	Acesso a mem (2)		7 ns
Branch	Busca (2)	Acesso a reg (1)	ALU (2)			5 ns
Jump	Busca (2)					2 ns

Observações:

- Neste caso, o ciclo tem tamanho de 8ns, logo o período de *clock* deve ter 8ns, o que significa uma frequência máxima de *clock* de 125MHz.

- Toda instrução deve então usar 8ns para sua execução (ciclo)

- No caso da instrução *jump*, que gasta *somente 2 ns, são desperdiçados 6 ns.*

Melhoria de Desempenho de Processadores

Exercício:

Considere as seguintes unidades funcionais e seus respectivos tempos de atraso para a implementação monociclo.

Busca = 2 ns, ALU = 2 ns, Acesso a registradores = 1ns, PC + 4 = *X ns,* Acesso à memória = 2 ns e Somador do desvio condicional = *Y ns.*

Qual seria o período de *clock* caso*:*

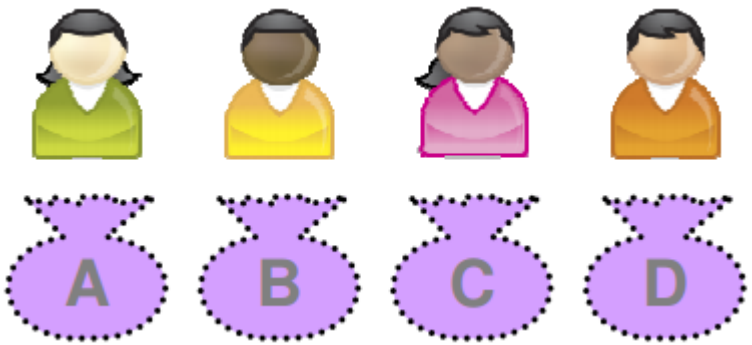
a) X = 3 e Y = 5;
b) X = 5 e Y = 5;
c) X = 1 e Y = 8;
d) X = 2 e Y = 3.

Melhoria de Desempenho de Processadores

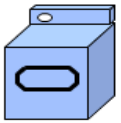
PIPELINE:

A técnica de *pipelining* é usada em processadores atuais para melhorar o desempenho pela superposição da execução de instruções.

Uma analogia: 4 pessoas (A, B, C, D) possuem sacolas de roupa para lavar, secar e dobrar.



Lavar leva 30 minutos



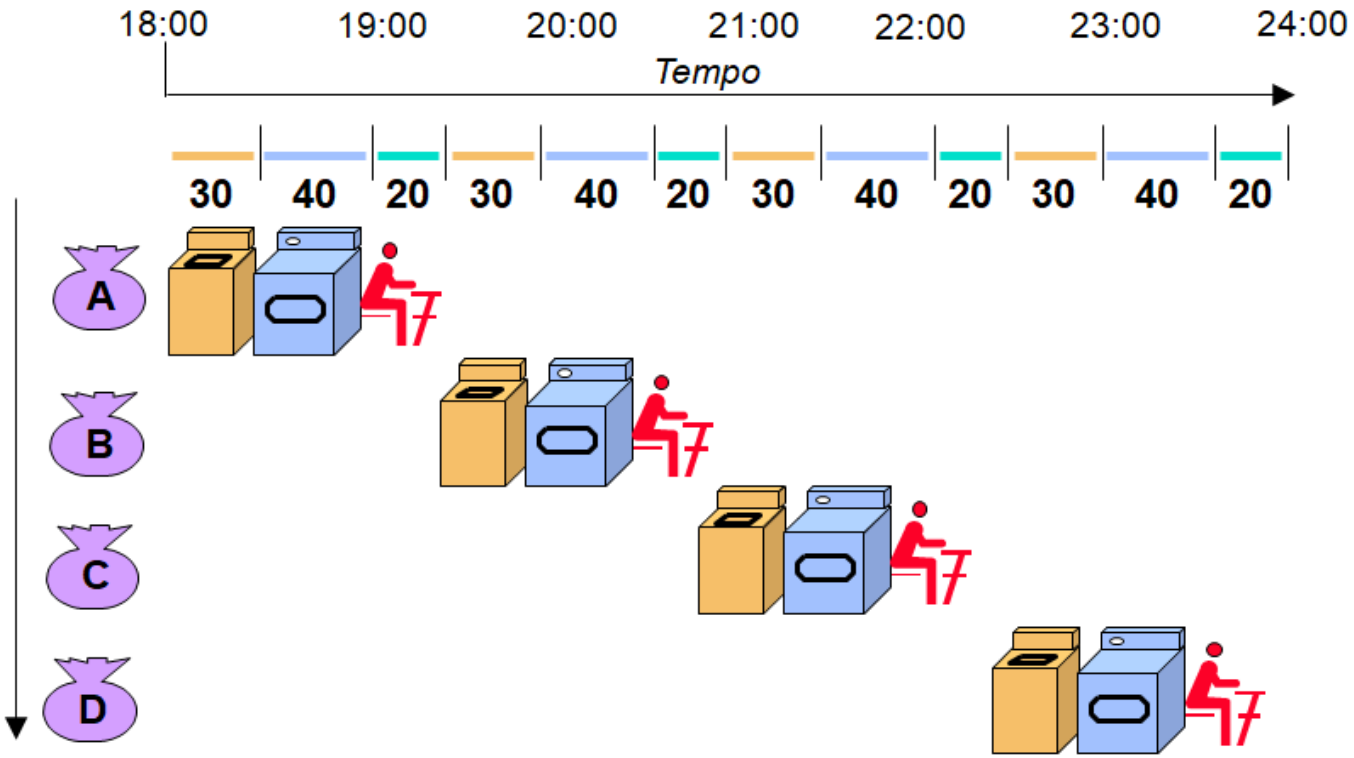
Secar leva 40 minutos

Dobrar leva 20 minutos

Melhoria de Desempenho de Processadores

PIPELINE:

Lavanderia sem *Pipeline* (sequencial - monociclo):

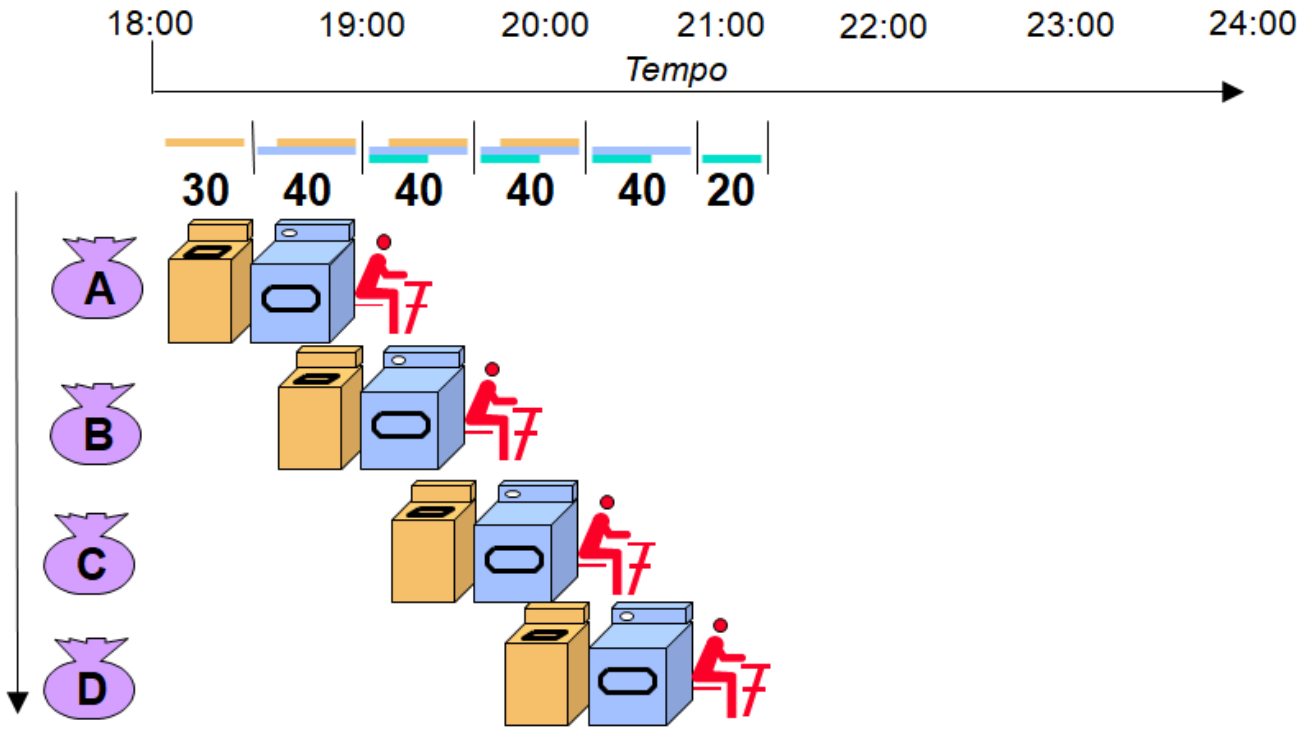


Total de tempo: 6 horas

Melhoria de Desempenho de Processadores

PIPELINE:

Lavanderia com *Pipeline*:



Total de tempo: 3,5 horas

Melhoria de Desempenho de Processadores

Observações sobre o PIPELINE:

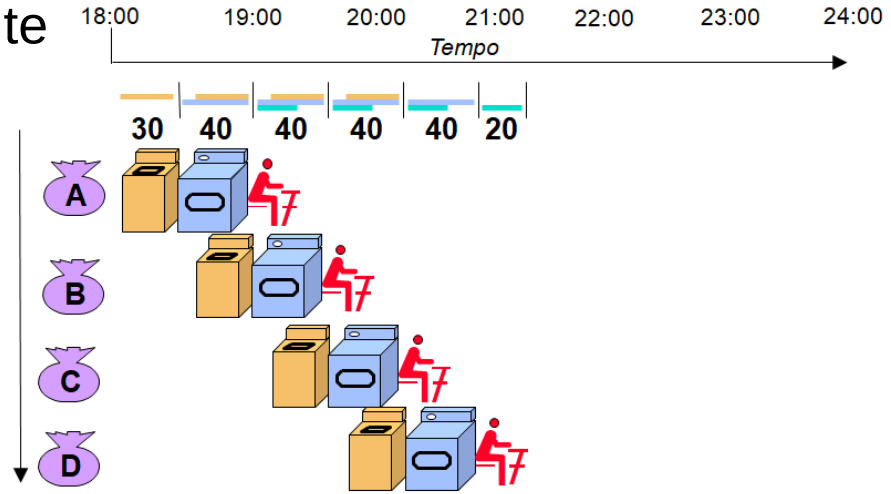
• *Pipeline* não melhora a latência de uma única tarefa, mas melhora o *throughput* de todo trabalho.

• Tempo de execução de uma tarefa é o mesmo, com ou sem *pipelining*.

• O ganho começa a existir a partir da segunda tarefa.

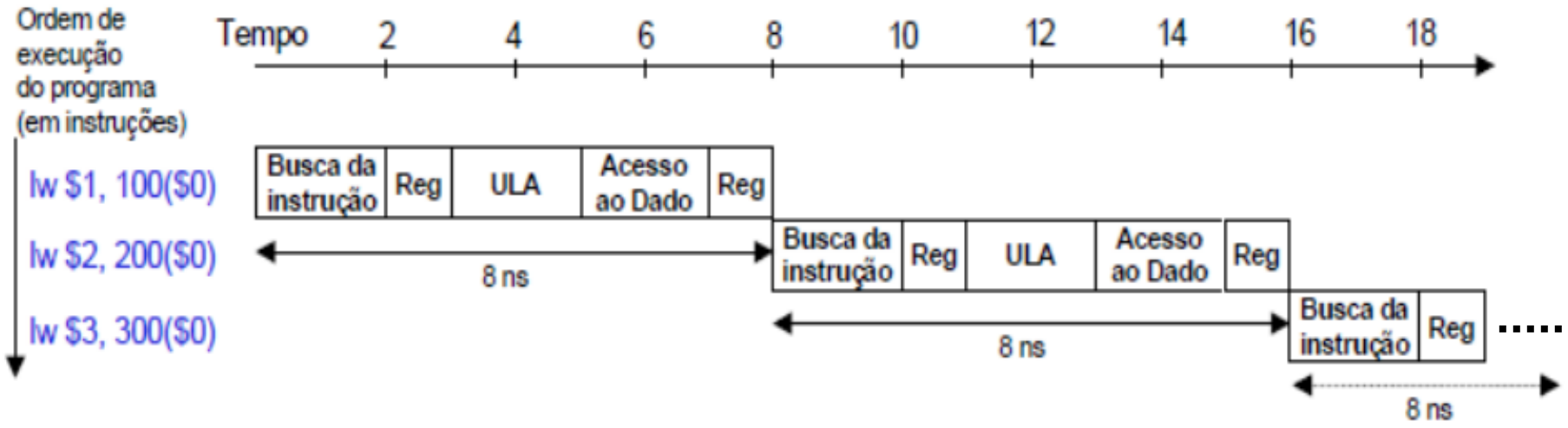
• Várias tarefas executam simultaneamente usando recursos diferentes

Latência *define-se como a diferença de tempo entre o início de um evento e o momento em que seus efeitos tornam-se perceptíveis. 2-* ***Throughput*** *é a quantidade de dados transferidos de um lugar a outro, ou a quantidade de dados processados em um determinado espaço de tempo*



Melhoria de Desempenho de Processadores

Em Instruções MIPS **sem o PIPELINE:**

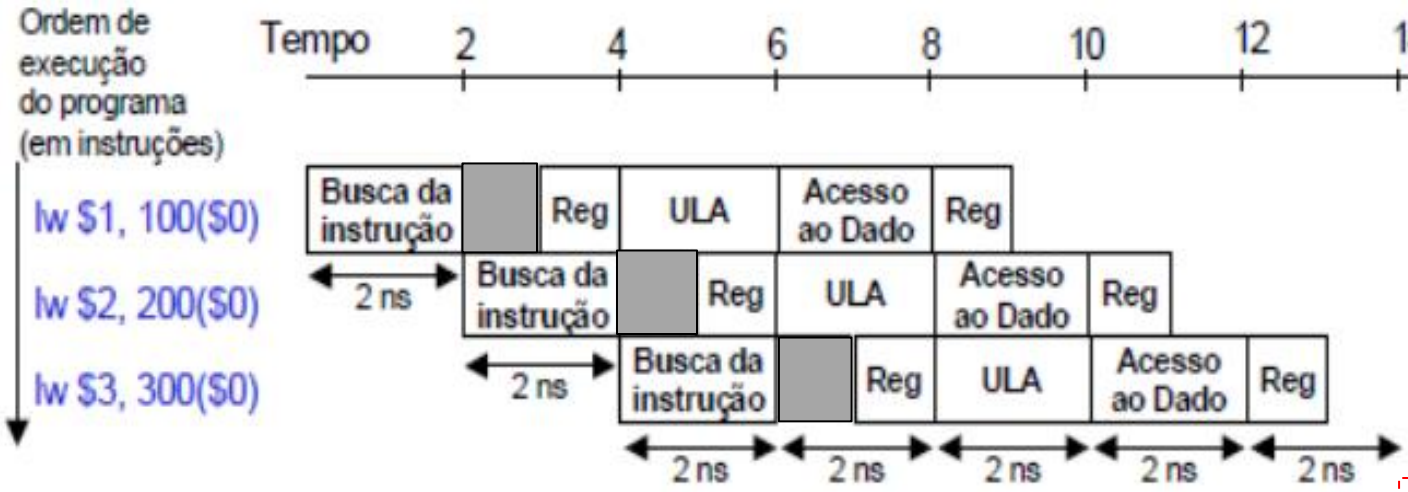


Tempo total de execução: 24ns

Tempo total entre os inícios da primeira quarta instruções = 3×8ns = 24ns.

Melhoria de Desempenho de Processadores

Em Instruções MIPS **com o PIPELINE:**



O ciclo precisa acomodar o estágio mais lento, neste caso os estágios com 2ns

Tempo total de execução: 14ns

Tempo total entre os inícios da primeira quarta instruções = 3×2ns = 6ns

Melhoria de Desempenho de Processadores

Exercício:

Como ficaria no *pipeline* a seguinte sequência de instruções:

```
lw  $t0, 8($t1)
add $a0, $t0,$t5
sw $a0,12($t0)
beq $t0,$a0, Label
```

Melhoria de Desempenho de Processadores

PIPELINE - Desempenho

Considerando:

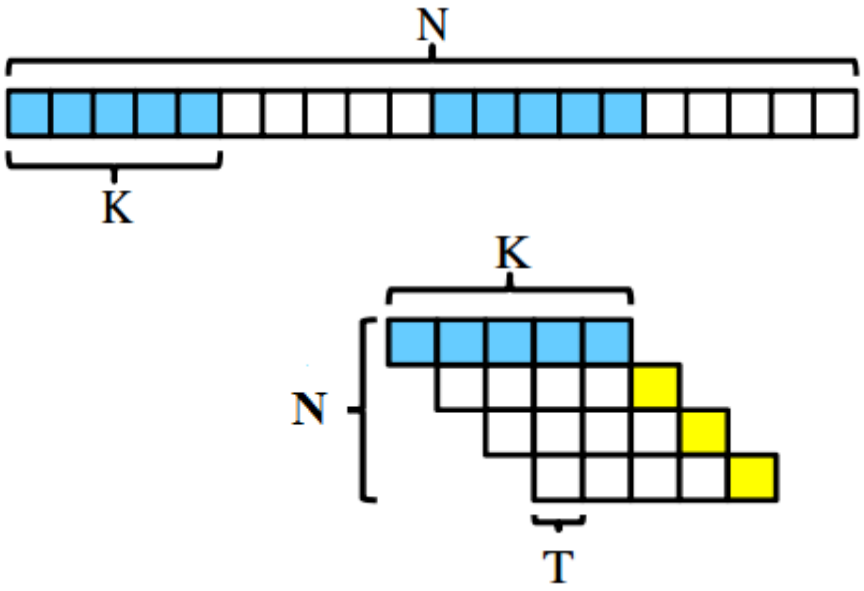
– N instruções
– K estágios de duração T

• Tempo de execução sem pipelining:

N × K × T

• Tempo de execução com pipelining:

K × T + ((N-1) × T)



Melhoria de Desempenho de Processadores

Exercício:

Calcule o tempo de execução das instruções abaixo, sem e com *pipeline*. Considere a arquitetura MIPS com 5 estágios e duração de 5ns por estágio.

lw $t0,50($t1)

add $t5, $t0, $t1

sw $t5, 10($t6)

j LABEL

Melhoria de Desempenho de Processadores

Com os estágios do *pipeline* **balanceados** (todos os estágios levam o mesmo tempo para serem concluídos), o ganho máximo teórico do *pipeline* é:

$$G = (N \times K \times T) \ / \ ((K \times T) + (N-1) \times T))$$

Onde: N = número de instruções, K = número de estágios e T = período do estágio

Considerando muitas instruções a serem executadas, o ganho máximo teórico será de aproximadamente o número de estágios do *pipeline*.

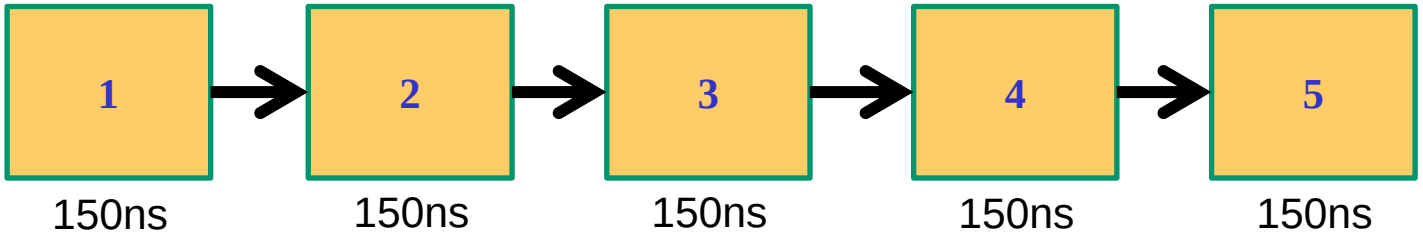
$$G = K \quad \text{ou} \quad G = (K-1)*100\ \%$$

Se não estiverem balanceados e um dos estágios for mais lento que os outros, a vazão do *pipeline* será afetada como um todo.

Melhoria de Desempenho de Processadores

Exercício:

Uma certa CPU utiliza a tecnologia de *pipeline* e possui cinco estágios. Sabe-se que cada estágio consome 150ns, conforme ilustrado a seguir:



Pede-se:

a) Qual o ganho máximo teórico deste *Pipeline* ?
b) Qual o ganho caso execute apenas 1 instrução?
c) Qual o ganho caso execute apenas 2 instruções?
d) Qual o ganho caso execute 1000 instruções?

Melhoria de Desempenho de Processadores

Exercício - Respostas

a) Qual o ganho máximo teórico deste Pipeline ?

G = K estágios .... G = 5 ou G = 400%

b) Qual o ganho caso execute apenas 1 instrução?

G = (N x K x T) / ((K x T) + (N-1) x T))
G = (1*5*150ns) / ((5*150ns) + (1-1)*150ns)) = 750 / 750 = 1 ou (1-1)*100% = 0%

c) Qual o ganho caso execute apenas 2 instruções?

G = (2*5*150ns)/((5*150ns)+(2-1)*150ns)) = 1500/900 = 1,66 ou 66,66%

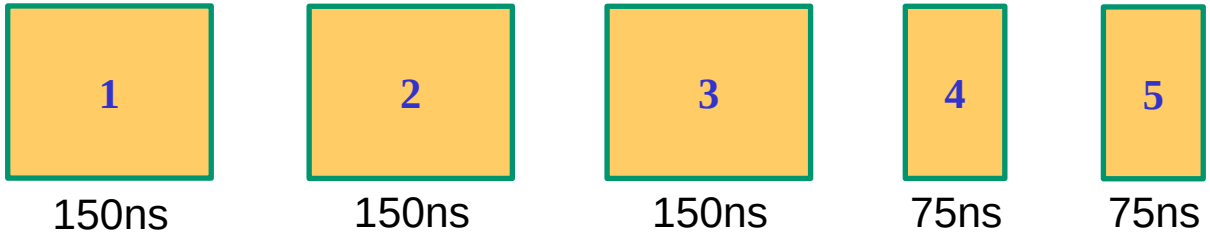
d) Qual o ganho caso execute 1000 instruções?

G = (1000*5*150ns)/((5*150ns)+(1000-1)*150ns)) = 750000/150600 = 4,98 = 398%

Melhoria de Desempenho de Processadores

Exercícios:

Uma certa CPU utiliza a tecnologia de *pipeline* e possui cinco estágios. Sabe-se que três estágios consomem 150ns cada um e os outros dois estágios consomem 75ns cada um (estágios desbalanceados), conforme ilustrado a seguir:



Pede-se:

a) Qual o ganho máximo teórico deste *Pipeline* ?
b) Qual o ganho caso execute apenas 1 instrução?
c) Qual o ganho caso execute apenas 5 instruções?
d) Qual o ganho caso execute 1000 instruções?

Melhoria de Desempenho de Processadores

Exercícios:

Na tabela abaixo, se o tempo para a operação da ALU puder ser reduzido em 25%, responda:

a) Isso afetará o ganho de velocidade obtido pela técnica de *pipelining*? Nesse caso, em quanto? Caso contrário, por quê?

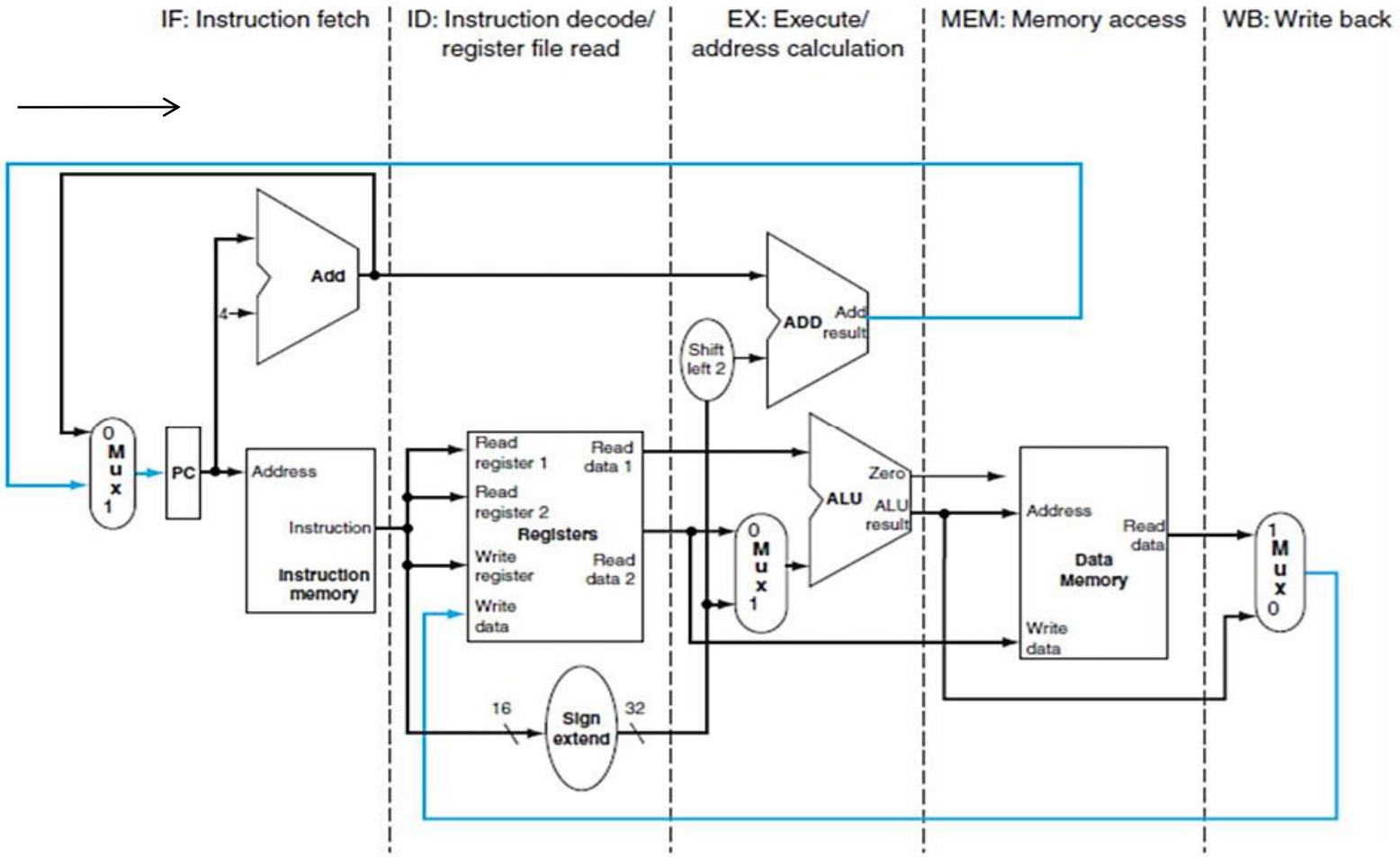
b) E se a operação da ALU exigir a mais 25% do tempo?

Instruction class	Instruction fetch	Register read	ALU operation	Data access	Register write	Total time
Load word (lw)	200 ps	100 ps	200 ps	200 ps	100 ps	800 ps
Store word (sw)	200 ps	100 ps	200 ps	200 ps		700 ps
R-format (add, sub, and, or, slt)	200 ps	100 ps	200 ps		100 ps	600 ps
Branch (beq)	200 ps	100 ps	200 ps			500 ps

Melhoria de Desempenho de Processadores

Implementação de PIPELINE de 5 estágios na Arquitetura MIPS:

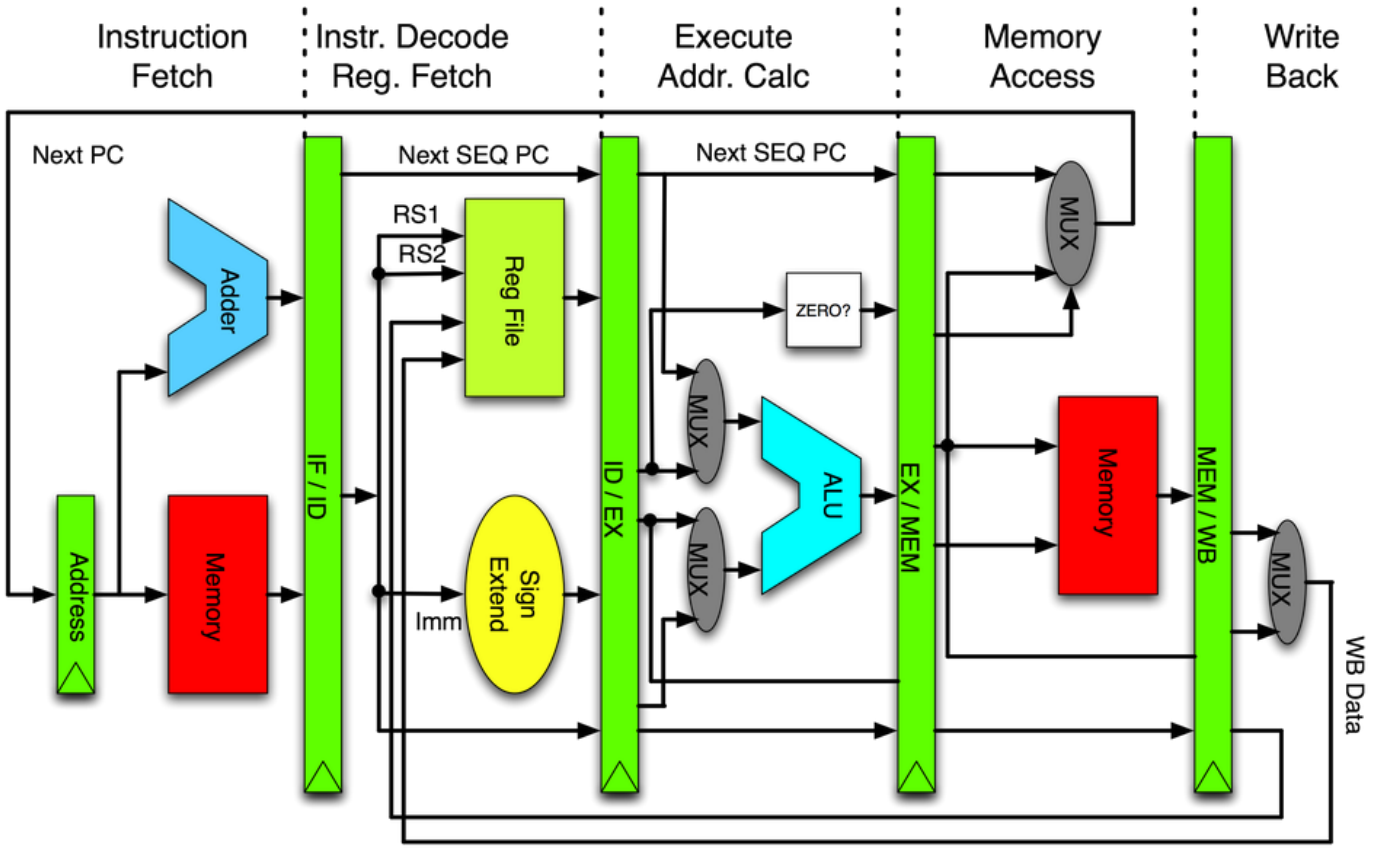
Divisão dos Estágios De Pipeline →



Melhoria de Desempenho de Processadores

Implementação de PIPELINE de 5 estágios na Arquitetura MIPS:

Cada divisão entre etapas deve conter registradores para reter os resultados de um ciclo enquanto processam o ciclo seguinte.



Conflitos de Pipeline (Pipeline “Hazards”)

➢ *Hazards* são situações em que a próxima instrução não pode ser executada no ciclo de *clock* seguinte.

➢ Pode acontecer uma dependência entre os estágios das instruções, gerando um atraso, conhecido como parada ou “*stall*”;

➢Existem 3 tipos:

- Hazard de Dados
- Hazard de Instruções (ou controle)
- Hazard Estrutural

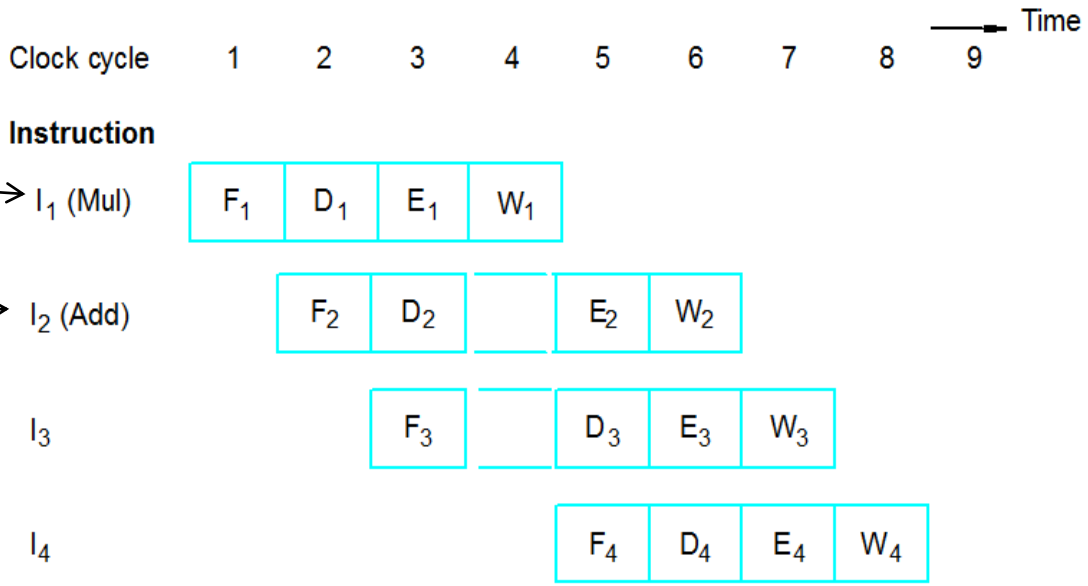
Conflitos de Pipeline (Pipeline "Hazards")

Hazard de dados – qualquer condição na qual a origem ou o destino dos operandos de uma instrução não estão disponíveis. Assim, alguma operação deverá ser atrasada e o *pipeline* será paralisado (*stall*);

Exemplo:

Mul R4, R3, R2 #(R4 = R2 x R3) → I_1 (Mul)

Add R6, R4, R5 #(R6 = R5 + R4) → I_2 (Add)



F: busca, D: decodificação, E: execução, W: gravação

Conflitos de Pipeline (Pipeline "Hazards")

Hazard de dados

É possível contornar o *hazard* utilizando a opção de adiantamento (*forwarding*) de operando:

- Ao invés de um ler um registro, a 2ª instrução pode utilizar o dado diretamente da saída da ALU antes que a instrução anterior seja completada (hardware);

- Os compiladores detectam o *hazard* e ajustam o programa para que não sejam gerados erros (software):

```
I1: Mul  R4, R3, R2
    NOP
    NOP
I2: Add  R6, R4, R5
```

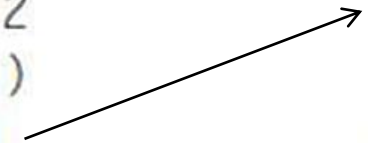
❖ Perde-se desempenho pois são inseridas instruções (NOP) para atrasar o pipeline (o compilador pode reordenar as instruções para evitar a inclusão de NOPs).

Conflitos de Pipeline (Pipeline “Hazards”)

Hazard de dados

Reordenação de instruções pelo compilador para resolver o *Hazard* da dados:

```
lw    $t1, 0($t0)          lw    $t1, 0($t0)
lw    $t2, 4($t0)          lw    $t2, 4($t0)
add   $t3, $t1,$t2         lw    $t4, 8($t0)
sw    $t3, 12($t0)         add   $t3, $t1,$t2
lw    $t4, 8($t0)          sw    $t3, 12($t0)
add   $t5, $t1,$t4         add   $t5, $t1,$t4
sw    $t5, 16($t0)         sw    $t5, 16($t0)
```

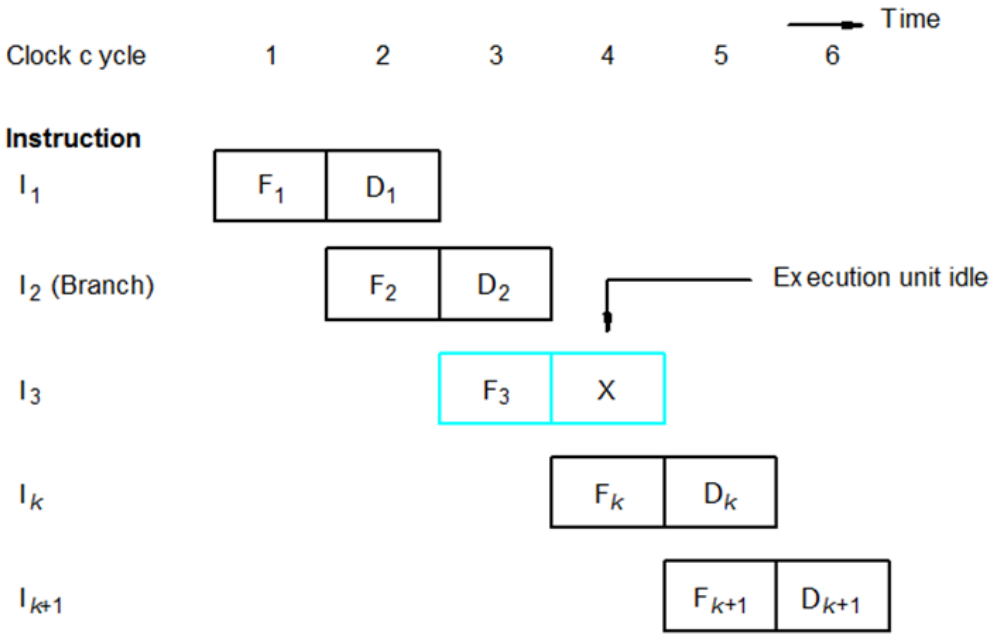


Conflitos de Pipeline (Pipeline "Hazards")

Hazard de Instruções (ou controle) – um atraso na disponibilidade de uma instrução causa uma parada do pipeline (stall).

Exemplo:

A instrução I2 é um salto (branch): durante o ciclo de decodificação, verifica-se que a instrução I3, que teve início, deve ser abortada e tem início a instrução Ik.



F: busca, D: decodificação, E: execução, W: gravação

Conflitos de Pipeline (Pipeline "Hazards")

Hazard de Instruções (ou controle)

- Um salto condicional (branch) introduz um hazard adicional causado pela dependência da condição do salto do resultado da instrução;

- A decisão do salto não pode ser feita enquanto a execução da instrução não for completada;

- As instruções de salto representam cerca de 20% das instruções executadas na maior parte dos programas;

    - Por isso, a forma de se tratar os saltos é um dos fatores mais importantes no desempenho final do pipeline.

Conflitos de Pipeline (Pipeline "Hazards")

Hazard estrutural – situação quando duas instruções requerem o uso simultâneo de um mesmo recurso de hardware

Exemplo:
A 1ª. Instrução faz uso do recurso de memória, enquanto que a instrução i+3 estaria lendo a memória para buscar a nova instrução.

	Clock cycle number								
Instruction	1	2	3	4	5	6	7	8	9
Load instruction	IF	ID	EX	MEM	WB				
Instruction *i* + 1		IF	ID	EX	MEM	WB			
Instruction *i* + 2			IF	ID	EX	MEM	WB		
Instruction *i* + 3				IF	ID	EX	MEM	WB	
Instruction *i* + 4					IF	ID	EX	MEM	WB
Instruction *i* + 5						IF	ID	EX	MEM
Instruction *i* + 6							IF	ID	EX

- IF: Instruction Fetch – busca
- ID: Instruction Decode – decodificação
- EX: Execution – execução da instrução (ALU e LD/ST)
- MEM: Memory – acesso a memória
- WB: Write Back – armazenamento (load) de registrador por resultado da ALU ou operação de load.

Conflitos de Pipeline (Pipeline "Hazards")

Hazard estrutural

Resolvendo com o uso de ciclo de STALL

	Clock cycle number									
Instruction	1	2	3	4	5	6	7	8	9	10
Load instruction	IF	ID	EX	MEM	WB					
Instruction $i+1$		IF	ID	EX	MEM	WB				
Instruction $i+2$			IF	ID	EX	MEM	WB			
Instruction $i+3$				stall	IF	ID	EX	MEM	WB	
Instruction $i+4$						IF	ID	EX	MEM	WB
Instruction $i+5$							IF	ID	EX	MEM
Instruction $i+6$								IF	ID	EX

- IF: Instruction Fetch – busca
- ID: Instruction Decode – decodificação
- EX: Execution – execução da instrução (ALU e LD/ST)
- MEM: Memory – acesso a memória
- WB: Write Back – armazenamento (load) de registrador por resultado da ALU ou operação de load.

Conflitos de Pipeline (Pipeline "Hazards")

Exercício:

Para cada sequência de código abaixo, indique se deverá haver "stall", se o *hazard* pode ser resolvido somente com *forwarding* ou se não há *hazard* algum.

Sequence 1	Sequence 2	Sequence 3

`lw $t0,0($t0)`
`add $t1,$t0,$t0`
	`add $t1,$t0,$t0`
	`addi $t2,$t0,#5`
	`addi $t4,$t1,#5`
		`addi $t1,$t0,#1`
		`addi $t2,$t0,#2`
		`addi $t3,$t0,#2`
		`addi $t3,$t0,#4`
		`addi $t5,$t0,#5`