



# *Arquitetura de Alto Desempenho*

*Paralelismo em nível de instrução (princípios)*

António Rui Borges

# Resumo

- O que é pipeline? •

*Paralelismo em nível de instrução* •

*Conjunto de instruções RISC simplificado*

- *Implementação sem pipeline do conjunto de instruções RISC* •

*Pipeline clássico de 5 estágios para um processador RISC* •

*Principais obstáculos de pipeline* –

*Desempenho da pipeline com travamento* –

*Riscos estruturais* –

– *Perigos de dados*

– *Controlar perigos*

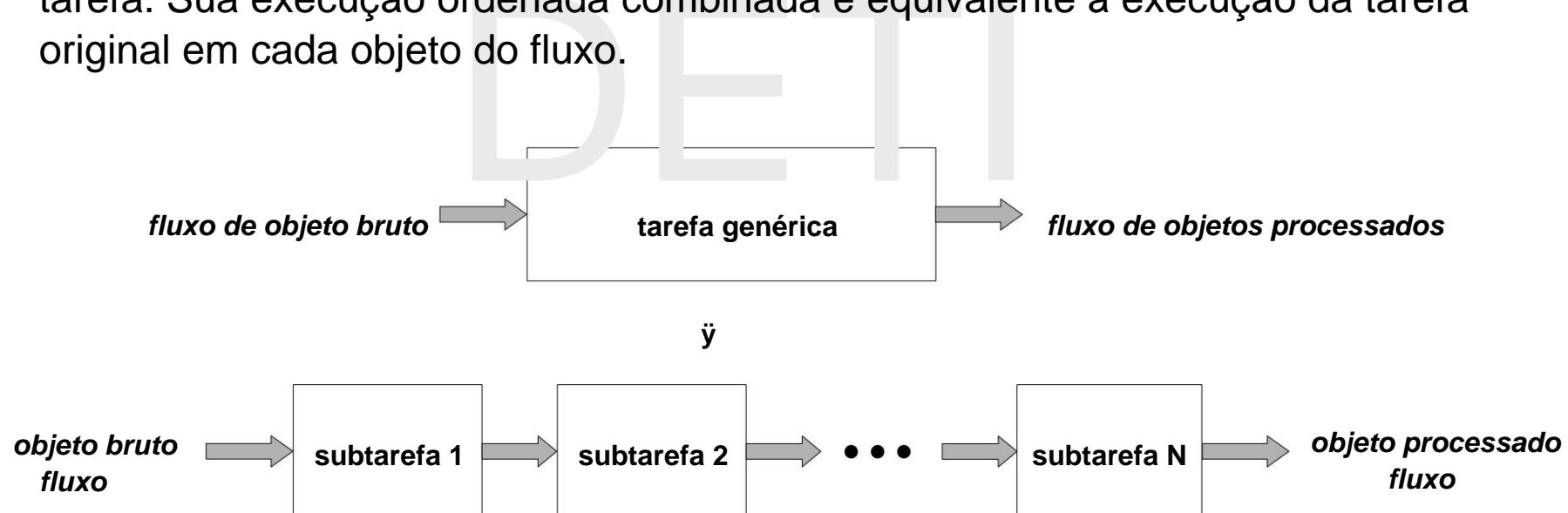
- *Implementação de pipeline clássico de 5 estágios* •

*Exceções* • *Operações multiciclo em pipeline clássico de 5 estágios* • *Pipeline*

*MIPS R4000* • *Leitura sugerida*

## O que é pipeline? - 1

*Pipelining* é uma técnica de implementação onde a execução de uma tarefa genérica nos objetos de um fluxo é convertida em um conjunto de subtarefas independentes que operam simultaneamente em objetos sucessivos do fluxo. Cada uma das subtarefas individuais, chamadas de *estágios* ou *segmentos de tubo*, é executada em sequência e representa uma fração definida de toda a tarefa. Sua execução ordenada combinada é equivalente à execução da tarefa original em cada objeto do fluxo.



## ***O que é pipeline? - 2***

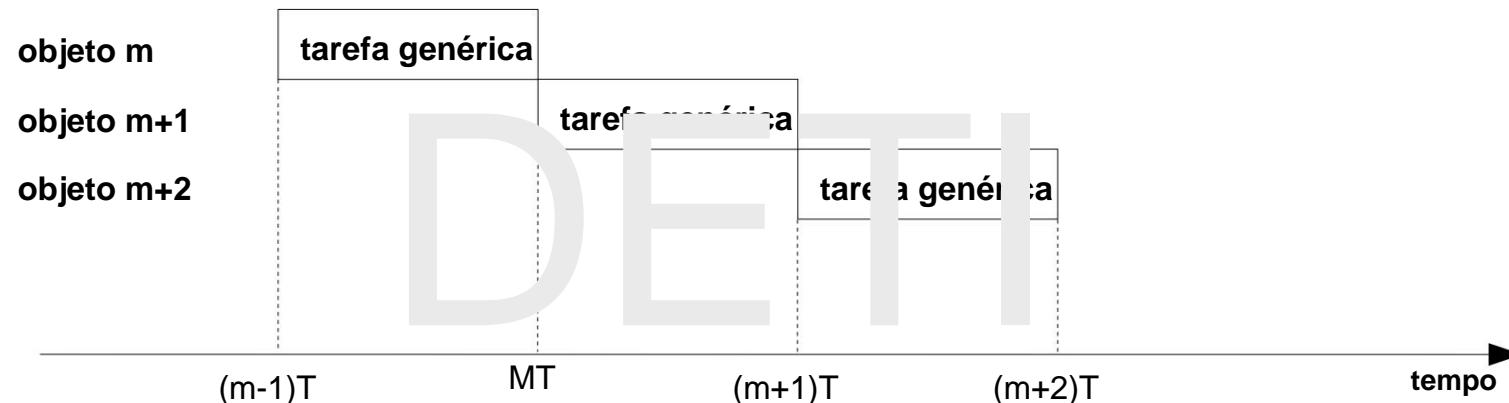
O conceito de pipelining foi inventado no contexto da indústria automobilística pela Ford Motor Company no início dos anos 1900, inspirado nos procedimentos operacionais dos matadouros de gado de Chicago. A primeira fábrica que introduziu uma linha de montagem móvel totalmente desenvolvida iniciou suas operações no final de 1913 em Detroit para a produção do Ford Modelo T.

O objetivo de uma linha de montagem móvel era agilizar o processo de construção de um carro a partir de suas peças e, consequentemente, baixar seu preço de mercado e disponibilizá-lo para um público consumidor muito mais amplo. Na verdade, o tempo de montagem de um carro passou de cerca de 12h e 30m, antes da introdução da linha de montagem móvel, para apenas 1h e 33m depois.

O impacto da movimentação das linhas de montagem foi tão grande para a indústria automobilística como um todo que a Ford Motor Company produzia 50% de todos os carros vendidos nos EUA e cerca de 40% dos carros vendidos na Comunidade Britânica em 1920.

## O que é pipeline? -3

### versão sem pipeline

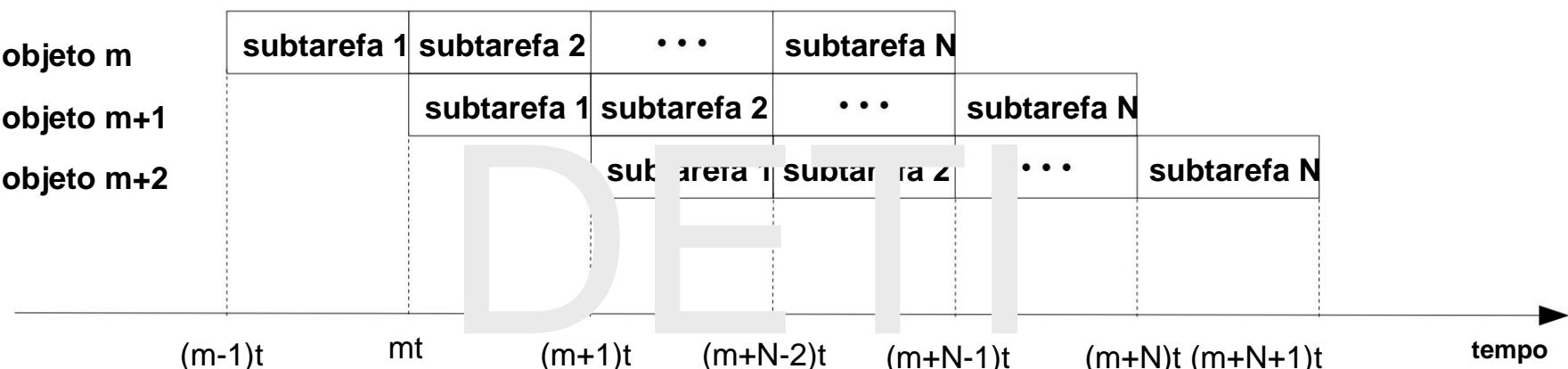


$$\text{rendimento} = \frac{1}{T}$$

tempo de execução para um fluxo de objetos M =  $M \cdot T$

# O que é pipeline? - 4

Versão em pipeline de N estágios



$t_n$ , com  $n = 1, 2, \dots, N$  - tempo de execução para subtarefa  $n$

$$\text{rendimento} = \frac{1}{t}, \text{ onde } t = \max(t_1, t_2, \dots, t_N)$$

tempo de execução para um fluxo de objetos  $M = (N + 1 + M)t$

## O que é pipeline? - 5

A aceleração obtida com a execução de uma implementação de pipeline de N estágios sobre a execução sem pipeline da mesma tarefa é expressa por

$$\begin{aligned}
 & \text{acelerar o pipeline de } N \text{ estágios} = \frac{\text{tempo de execução da implementação sem pipeline}}{\text{tempo de execução da implementação em pipeline de } N \text{ estágios}} = \\
 & = \frac{\frac{M}{N+1} + M \cdot \frac{t^*}{N}}{M \cdot \frac{t^*}{N+1} + M \cdot \frac{t^*}{N}} = \frac{M \cdot N \cdot \frac{t^*}{N+1} + M \cdot N \cdot \frac{t^*}{N}}{(N+1) \cdot M \cdot \frac{t^*}{N+1} + N \cdot M \cdot \frac{t^*}{N}} = \frac{t^*}{t} ,
 \end{aligned}$$

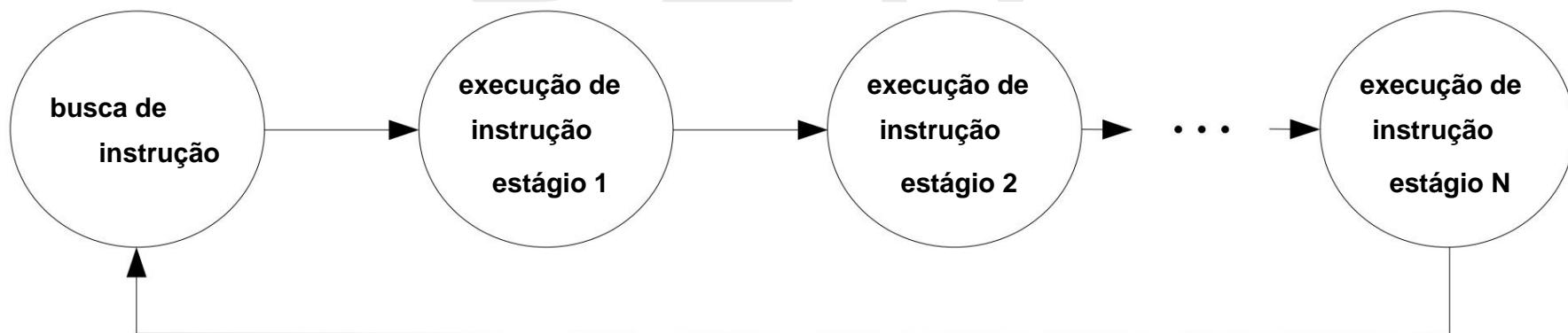
onde  $T = M \cdot t^*$  e  $N \ll M$ .

Idealmente, se os estágios do tubo estiverem quase perfeitamente balanceados e a sobrecarga envolvida no pipelining for insignificante, então a razão  $t^*/t$  se aproxima da unidade e a aceleração é aproximadamente igual ao número de estágios usados na partição da tarefa original em subtarefas.

## Paralelismo em nível de instrução

Todos os processadores desde 1985 adotaram o pipelining como meio de sobrepor a execução de instruções e melhorar o desempenho. Essa potencial sobreposição de instruções durante sua própria execução é chamada de *paralelismo em nível de instrução* (ILP), porque as instruções são, de certa forma, processadas em paralelo por meio da decomposição da atividade que ocorre.

Uma abordagem comum para fazer isso é decompor a execução da instrução fase do ciclo do processador em vários estágios.



## Conjunto de instruções RISC simplificado - 1

Parte da arquitetura central de um RISC (Computador com conjunto de instruções reduzido) típico, MIPS64 será usado para ilustrar os conceitos básicos de pipeline.

As arquiteturas RISC são caracterizadas por algumas propriedades principais, que simplificam dramaticamente sua implementação

- as únicas operações que afetam a memória são as instruções *de carregamento e armazenamento* que movem dados da memória para um registrador do banco de registradores, ou de um registrador do banco de registradores para a memória; instruções que transferem menos do que o conteúdo de um registro completo também estão frequentemente disponíveis
- as únicas operações sobre dados são as instruções *aritméticas e lógicas* que se aplicam aos dados nos registros do banco de registros; eles mudam em princípio todo o registro
- os formatos de instrução são poucos e normalmente têm o mesmo tamanho.

Estas propriedades também levam a simplificações dramáticas na implementação de pipelining, que é uma das razões pelas quais os conjuntos de instruções são tão projetados.

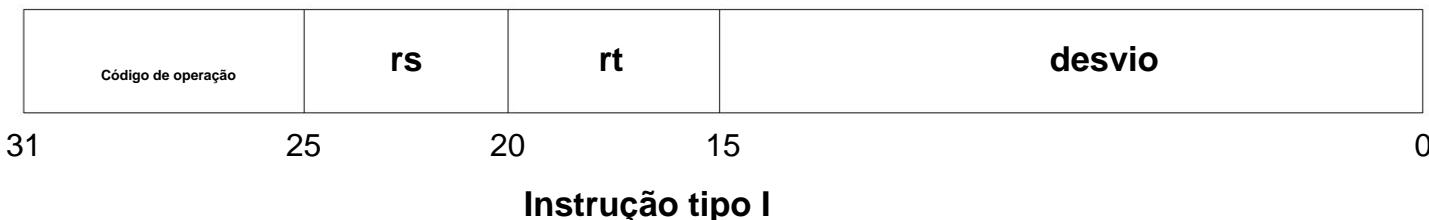
## Conjunto de instruções RISC simplificado - 2

Segue uma breve descrição da versão de 64 bits do conjunto de instruções MIPS. Todas as instruções têm 32 bits de comprimento. Os tipos de dados consistem em bytes de 8 bits , *meias palavras* de 16 bits , *palavras* de 32 bits e *palavras duplas* de 64 bits. As instruções estendidas de 64 bits são geralmente indicadas por terem um D no início ou no final do mnemônico.

O banco de registradores de uso geral contém 32 registradores de 64 bits, dos quais *registradores 0* é somente leitura e tem valor zero.

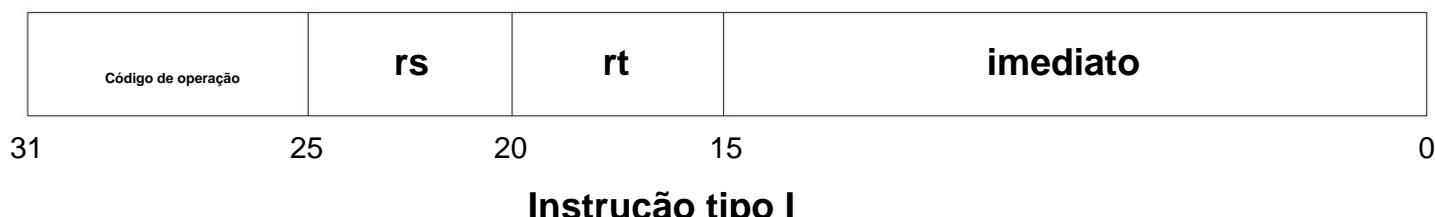
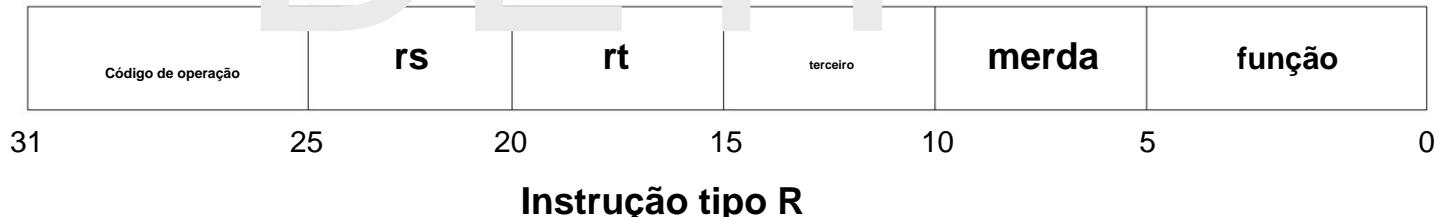
O conjunto de instruções principal possui três classes de instruções

- *instruções de carga e armazenamento* – essas instruções tomam como operandos um registrador fonte, o *registrator base*, e um campo imediato de 16 bits, o *offset*; a soma do conteúdo do registrador base e do deslocamento de sinal estendido é usada como endereço de memória, o *endereço efetivo*; as instruções LD e SD carregam ou armazenam todo o conteúdo do registrador de 64 bits



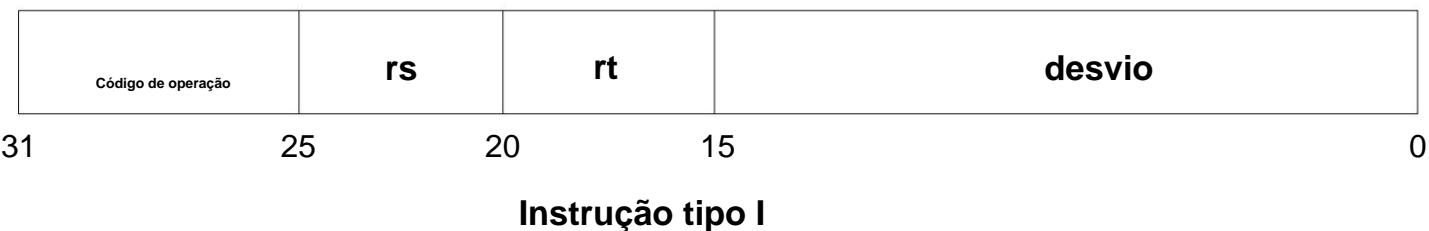
## Conjunto de instruções RISC simplificado - 3

- *instruções aritméticas/lógicas* – estas instruções pegam dois registradores ou um registrador e um valor imediato com sinal estendido, operam sobre eles e armazenam o resultado em outro registrador; instruções *aritméticas* típicas incluem adição (DADD) e subtração (DSUB) para extensões de 64 bits; as instruções *lógicas*, entretanto, não diferenciam entre extensões de 32 e 64 bits; as versões *imediatas* usam os mesmos mnemônicos com sufixo I; existem formas assinadas e não assinadas das instruções *aritméticas*: as formas não assinadas não geram exceções de overflow e possuem um sufixo U no final (DADDU, DSUBU, DADDIU)



## Conjunto de instruções RISC simplificado - 4

- *instruções de desvio* – estas instruções são modificações condicionais da estrita sequencialidade de execução das instruções; a *condição de ramificação* é normalmente especificada por um conjunto de *bits de condição*, também chamado de *código de condição*, um conjunto limitado de comparações entre um par de registradores (*maior, maior ou igual, menor, menor ou igual, igual e diferente*) ou entre um registrador e zero (*igual e não igual*); o *destino* da ramificação é obtido adicionando um deslocamento de sinal estendido, deslocado para a esquerda em dois bits para torná-lo um deslocamento de palavra, ao conteúdo atual do *contador de programa*



## Implementação sem pipeline de um conjunto de instruções RISC - 1

Uma implementação simples sem pipeline de parte do conjunto de instruções principais do MIPS64 é apresentada e será usada como uma estrutura para compreender completamente o funcionamento interno do pipeline. Não é a implementação mais econômica ou de maior desempenho, mas foi projetada para levar naturalmente ao pipeline.

A implementação do conjunto de instruções requer a introdução de vários registradores temporários que não fazem parte da arquitetura; eles são introduzidos apenas para simplificar o pipeline. Esta implementação se concentrará apenas no subconjunto inteiro do conjunto de instruções principais do MIPS64 que consiste nas operações de carregamento/armazenamento, ALU inteiro e ramificação. Eles serão implementados em no máximo 5 ciclos de clock.

A operação em cada ciclo de clock é a seguinte

### 1. Ciclo de busca de instrução (IF)

O conteúdo do contador de programa (PC) é usado como endereço para buscar a próxima instrução na memória, seu valor é armazenado no registrador de instruções (IR). O PC é atualizado para apontar a próxima instrução adicionando 4 ao conteúdo atual.

## **Implementação sem pipeline de um conjunto de instruções RISC - 2**

### **2. Ciclo de decodificação de instruções / busca de registro (ID)**

A instrução é decodificada levando às operações

- o conteúdo dos registros correspondentes aos especificadores de origem do registro é lido do banco de registros
- o teste de igualdade é realizado no conteúdo dos registros à medida que são lidos, para serem posteriormente utilizados em uma possível ramificação
- o campo de deslocamento é estendido com sinal caso seja necessário
- o possível endereço de destino da ramificação é calculado adicionando o conteúdo incrementado do PC ao campo de deslocamento de sinal estendido de 2 bits deslocado para a esquerda.

A instrução *de desvio* é concluída no final deste estágio, armazenando o endereço de destino do desvio no PC se a condição de teste for verdadeira, ou deixando-o inalterado se a condição de teste for falsa.

## ***Implementação sem pipeline de um conjunto de instruções RISC - 3***

### **3. Ciclo de execução / endereço efetivo (EX)**

A ULA pega os operandos computados no ciclo anterior e, dependendo do tipo de instrução, executa uma das ações abaixo da *referência de memória* – o

- registrador base e o offset são somados para formar o endereço efetivo
- *instrução ALU registro a registro* – a operação especificada pelo opcode é realizada nos valores lidos do banco de registros
- *instrução ALU com registro imediato* – a operação especificada pelo opcode é realizada no primeiro valor lido do banco de registros e no valor imediato com sinal estendido.

## ***Implementação sem pipeline de um conjunto de instruções RISC - 4***

### ***4. Acesso à memória (MEM)***

Quando a instrução é uma *carga*, o endereço efetivo calculado no ciclo anterior é o endereço da localização da memória cujos dados devem ser lidos; quando a instrução é uma *loja*, o segundo valor lido do banco de registradores deve ser escrito no local de memória cujo endereço é o endereço efetivo.

A instrução de *armazenamento* é concluída no final desta etapa.

### ***5. Ciclo de write-back (WB)***

O resultado de uma instrução de um dos tipos *registrador a registrador*, *registrator imediato* ou *carga*, é gravado em um registrador do banco de registradores.

## Implementação sem pipeline de um conjunto de instruções RISC - 5

Nesta implementação, as instruções *de desvio* são executadas em 2 ciclos de clock, as instruções *de armazenamento* em 4 ciclos de clock e todas as outras instruções em 5 ciclos de clock. Supondo que um determinado benchmark tenha uma frequência de ramificação de 12% e uma frequência de armazenamento de 10%, qual é o CPI médio para executar esse aplicativo na implementação sem pipeline que acabamos de descrever?

$$\text{IPC geral} = \bar{y} = \frac{\text{contagem de instruções}}{\text{contagem de instruções}} \bar{y}\text{IPC} = 0,12\bar{y}2 + 0,10\bar{y}4 + 0,78\bar{y}5 = 4,54.$$

## Pipeline clássico de 5 estágios para um processador RISC - 1

A implementação sem pipeline pode ser convertida em uma implementação com pipeline quase sem alterações estruturais, buscando uma nova instrução a cada ciclo de clock.

Instrução número	Número do relógio								
	2	3	4	5	6	7	8	9	
eu	SE	ID EX MEM WB							
m+1	SE	ID EX MEM WB							
m+2		SE	ID EX MEM WB						
m+3			SE	ID EX MEM WB					
m+4				SE	ID EX MEM WB				

Embora cada instrução leve 5 ciclos de clock para ser concluída, durante cada ciclo o hardware processará parte de cinco instruções consecutivas.

## ***Pipeline clássico de 5 estágios para um processador RISC - 2***

No entanto, para que esta transformação funcione, é preciso determinar o que acontece em cada ciclo de clock do processador e garantir que não há duas operações usando o mesmo recurso de caminho de dados ao mesmo tempo, ou seja, é preciso verificar cuidadosamente se a sobreposição de instruções no pipeline não estão causando tal conflito.

O número de detalhes a serem considerados é diretamente proporcional ao número de estágios do pipeline e ao número de instruções que estão em execução sobreposta. Portanto, é importante elaborar uma representação gráfica da execução que deixe claro o que está acontecendo.

Uma maneira popular de fazer isso é descrever o fluxo de operações como uma série de os recursos do caminho de dados em uso nos diferentes estágios do pipeline.

## **Pipeline clássico de 5 estágios para um processador RISC - 3**

**Pipeline visto como uma série de recursos de caminhos de dados deslocados no tempo**

Fonte: Arquitetura de Computadores: Uma Abordagem Quantitativa



CC – ciclo de clock

DM – memória de dados

ALU – unidade aritmética/lógica

IM – memória de instruções

Reg – registrar banco

## Pipeline clássico de 5 estágios para um processador RISC - 4

Algumas observações são necessárias

- há uma necessidade de *memórias de instruções* e de dados separadas, o que significa a implementação de caches de instruções e de dados separados para eliminar o conflito de uma única memória que surgiria entre a *busca de instrução* (IF) e estágios de acesso à *memória de dados* (MEM)
- o *banco de registradores* é acessado em duas etapas: para leitura, na etapa de *decodificação / busca de registradores* (ID), e para escrita, na etapa de *write-back* (WB); é necessário realizar duas leituras e uma gravação a cada ciclo de clock – para lidar com a leitura e a escrita no mesmo registrador, a gravação do registrador é realizada na primeira metade do ciclo de clock e a leitura do registrador na segunda metade
- para buscar uma instrução a cada ciclo de clock, o *contador de programa* (PC) deve ser atualizado a cada ciclo de clock no estágio de *busca de instrução* (IF) em preparação para a próxima instrução; além disso, o potencial endereço alvo da ramificação deve ser calculado durante o estágio de *decodificação da instrução / busca de registro* (ID); ainda assim, como a ramificação, quando a condição de teste é verdadeira, também altera o PC no estágio ID, há um conflito que será tratado posteriormente.

## **Pipeline clássico de 5 estágios para um processador RISC - 5**

É preciso também garantir que as instruções em diferentes estágios do pipeline não interfiram uma nas outras. A solução é adicionar registros para armazenamento temporário de dados relevantes entre estágios sucessivos, de modo que no início de um ciclo de clock todos os resultados de um determinado estágio sejam armazenados nos registros que serão usados como entradas para o próximo estágio. Os registros são nomeados de acordo com os dois estágios separados por esse registro: por exemplo, os registros entre os estágios IF e ID são chamados de IF/ID.

Observe que nenhum registro é necessário no final do estágio WB. Todas as instruções devem atualizar de alguma maneira particular o estado do computador (o banco de registradores, a memória ou o contador do programa), de modo que um registrador separado no final do pipeline seja redundante para o estado que está sendo atualizado.

O próprio contador do programa pode ser considerado um registrador de pipeline: um que alimenta o estágio IF do pipeline. Ao contrário dos outros registos, contudo, o contador do programa faz parte do estado arquitectónico visível; seu conteúdo deve ser salvo quando uma exceção é atendida, enquanto o conteúdo dos outros registradores do pipeline é descartado.

## ***Pipeline clássico de 5 estágios para um processador RISC - 6***

### **Pipeline com registros entre etapas sucessivas para armazenamento temporário**

Fonte: Adaptado de Arquitetura de Computadores: Uma Abordagem Quantitativa



DETI

## Pipeline clássico de 5 estágios para um processador RISC - 7

O pipeline aumenta o rendimento da instrução, mas não reduz o tempo de execução de uma instrução individual. Na verdade, aumenta ligeiramente devido à sobrecarga necessária para controlar o pipeline. Observe que o aumento no rendimento das instruções significa que um programa é executado mais rapidamente e tem um tempo de execução menor, mesmo que nenhuma instrução seja executada mais rapidamente.

Considere o processador sem pipeline descrito anteriormente. Suponha que ele tenha um ciclo de clock de 1ns e execute um benchmark com frequência de ramificação de 20% e frequência de armazenamento de 10%. Suponha que, devido à sobrecarga para controlar o pipeline, o ciclo de clock da versão do processador em pipeline seja de 1,2 ns. Qual é a velocidade ideal na taxa de execução de instruções obtida com o pipelining?

$$\begin{aligned}\text{média de inst exec timenonpipelined} &= \text{CPI}_{\text{nonpipelined}} \times \text{tempo de ciclo de clock geral} = \\ &= (0,2 \times 2 + 0,1 \times 4 + 0,7 \times 5) \times 1,0 = 4,3 \text{ ns}\end{aligned}$$

$$\text{acelerar} = \frac{\text{tempo médio de execução instantânea sem pipeline}}{\text{timepipelined médio do inst exec}} = \frac{4.3}{1.2} = 3,6.$$

## ***Principais obstáculos do pipeline - 1***

O pipeline funcionaria conforme descrito se as instruções fossem independentes umas das outras. Na realidade, não é assim. Existem situações, chamadas de *perigos*, que impedem que a próxima instrução no fluxo seja executada durante o ciclo de clock designado. Assim, reduzindo o desempenho da velocidade ideal obtida pelo pipelining.

Existem três classes de perigos

- *riscos estruturais* – surgem de conflitos de recursos quando o hardware não consegue suportar todas as combinações possíveis de instruções simultaneamente em execução sobreposta
- *riscos de dados* – surgem quando uma instrução depende dos resultados de uma instrução anterior de uma forma que é exposta pela sobreposição das instruções no pipeline
- *riscos de controle* – eles surgem do pipeline de instruções da filial e outras instruções que afetam o PC.

## ***Principais obstáculos do pipeline - 2***

Os perigos nos oleodutos podem exigir a *paralisação* do oleoduto, ou seja, para evitar o perigo, muitas vezes é necessário que algumas instruções no oleoduto sejam forçadas a permanecer no mesmo estágio, enquanto outras podem prosseguir para o próximo estágio. Quando uma instrução é paralisada, todas as instruções que foram buscadas depois da instrução paralisada também são paralisadas, e todas as instruções buscadas anteriormente devem continuar, caso contrário o perigo nunca desapareceria. Como resultado, nenhuma instrução nova é buscada durante uma parada e ocorre um atraso na execução.

## ***Desempenho de dutos com paradas***

$$\begin{aligned}
 \text{acelerar} &= \frac{\text{tempo médio de inst exec sem pipeline}}{\text{tempo médio de inst exec com pipeline}} = \\
 &= \frac{\text{CPI geral não pipelined}}{\text{CPI geral pipelined}} \cdot \frac{\text{tempo de ciclo de relógio sem pipeline}}{\text{tempo de ciclo de relógio com pipeline}} = \\
 &= \frac{\text{CPInonpipelined geral 1}}{\text{+ ciclos de paralisação do pipeline por instrução pipelined}} \cdot \frac{\text{tempo de ciclo de relógio sem pipeline}}{\text{tempo de ciclo de relógio com pipeline}}
 \end{aligned}$$

## Riscos estruturais - 1

Quando um processador é pipeline, a execução sobreposta de instruções requer o mesmo nível de pipeline de todas as unidades funcionais e a duplicação de recursos, para permitir todas as combinações possíveis de instruções no pipeline. Se alguma combinação de instruções não puder ser acomodada devido a conflitos de recursos, diz-se que o processador incorre em um *risco estrutural*.

O caso mais comum de riscos estruturais surge quando alguma unidade funcional não está totalmente encadeada. Então, uma sequência de instruções em que algumas delas usam essa unidade não pode prosseguir através dos estágios do tubo à taxa de uma instrução por ciclo de clock. Outra causa comum é quando algum recurso não é replicado o suficiente para permitir que todas as combinações de instruções sejam executadas na taxa nominal.

Quando esse tipo de perigo é encontrado, o pipeline irá *paralisar* a instrução mais recente até que a unidade necessária esteja disponível. A parada irá gerar o que normalmente é chamado de *bolha*, já que esta condição pode ser considerada como fluindo através dos estágios do tubo ocupando espaço, mas não produzindo nenhum trabalho útil.

## ***Riscos estruturais - 2***

Alguns processadores possuem uma única memória para acessar instruções e dados. Isso gerará um conflito potencial entre os estágios de busca de instrução (IF) e acesso à memória de dados (MEM).

### **Processador com apenas uma porta de memória**

Fonte: Arquitetura de Computadores: Uma Abordagem Quantitativa



# Riscos estruturais - 3

**Processador com apenas uma porta de memória: o efeito de uma instrução de carregamento**

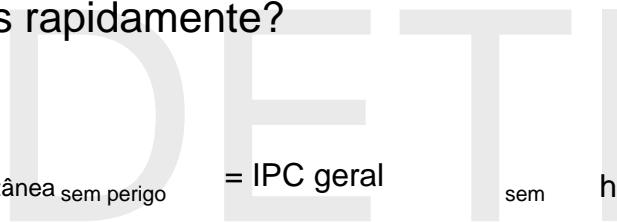
Fonte: Adaptado de Arquitetura de Computadores: Uma Abordagem Quantitativa

<i>Instrução</i>	<i>Número do relógio</i>									
	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>6</b>	<b>7</b>	<b>8</b>	<b>9</b>	<b>10</b>
carregar	SE	ID EX MEM WB								
m+1		SE		WB						
m+2				ID EX MEM W						
m+3				SE		D EX ME	WB			
m+4					SE	ID	X MEM WB			
m+5						SE	ID EX-MEM			
m+6							SE	ID EX		

<i>Instrução</i>	<i>Número do relógio</i>									
	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>6</b>	<b>7</b>	<b>8</b>	<b>9</b>	<b>10</b>
carregar	SE	ID EX MEM WB								
m+1		SE	ID EX MEM WB							
m+2			SE	ID EX MEM WB						
<b>bolha</b>				SE	ID EX MEM WB					
m+3					SE	ID EX MEM WB				
m+4						SE	ID EX MEM WB			
m+5							SE	ID EX-MEM		
m+6								SE	ID EX	

## Riscos estruturais - 4

Suponha que as instruções de referência de dados (*carregar e armazenar*) constituam 40% do mix de instruções para um determinado benchmark e que o CPI ideal para o processador em pipeline, ignorando o risco estrutural, seja 1. Suponha que o processador com o risco estrutural tenha um clock taxa que é 1,05 vezes maior do que o processador sem o perigo. Desconsiderando quaisquer outras perdas de desempenho, qual processador executa o benchmark mais rapidamente?



$$\begin{aligned}
 & \text{tempo médio de execução instantânea sem perigo} = \text{IPC geral} \cdot \text{tempo de ciclo de tempo ideal} = \\
 & = 1 \cdot \text{tempo de ciclo de relógio ideal} \\
 & \text{tempo médio de execução instantânea com perigo} = \text{CPI geral com tempo de ciclo de hazyclock ideal} = \\
 & = (1 + 0,4) \cdot \frac{\text{tempo de ciclo do relógioideal}}{1.05} \\
 & = 1,3 \cdot \text{tempo ideal do ciclo de relógio} .
 \end{aligned}$$

## ***Riscos estruturais - 5***

Se todos os fatores forem iguais, um processador sem riscos estruturais sempre terá um menor IPC. Por que, então, um projetista permitiria riscos estruturais?

A principal razão é reduzir o custo do processador, uma vez que canalizar todas as unidades funcionais e/ou replicá-las pode ser muito caro. Por exemplo, processadores que exigem acesso a uma instrução e ao cache de dados a cada ciclo de clock precisam do dobro da largura de banda total da memória. Da mesma forma, projetar um multiplicador de ponto flutuante totalmente pipeline consome muitas portas e o espaço pode não estar disponível.

Se o risco estrutural for raro, pode não valer a pena o custo para evitá-lo.

## Perigos de dados - 1

Um efeito importante do pipelining é alterar o tempo relativo das instruções, sobrepondo sua execução. Devido a isso, podem surgir riscos de dados. Os *perigos de dados*, na verdade, ocorrem quando o pipeline força uma ordem real para operações de leitura/gravação para operandos diferente daquela percebida pela execução sequencial das instruções em um processador sem pipeline.

Considere a execução em pipeline do código

PAI	R1, R2, R3
DSUB	R4,R1,R5
E	R6,R1,R7
OU	R8,R1,R9
XOR	R10,R1,R11

## Perigos de dados - 2

Todas as instruções após DADD usam o resultado da instrução DADD.

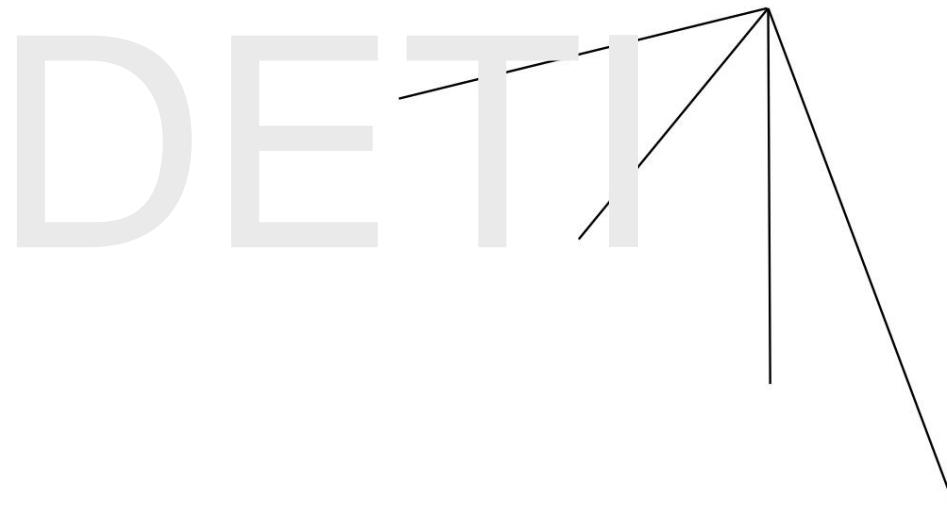
O DADD atualiza o valor de R1 no estágio WB, mas o DSUB lê R1 antes disso em seu estágio ID. A menos que sejam tomadas precauções para evitá-lo, o DSUB lerá um valor errado. O valor lido nem é determinístico. Parece lógico supor que DSUB sempre obterá o valor atribuído a R1 por alguma instrução anterior a DADD, mas pode não ser o caso. Se uma interrupção for atendida entre as instruções DADD e DSUB, o estágio WB de DADD será concluído e o valor de R1 nesse ponto será o resultado de DADD.

A instrução AND também é afetada por este perigo. A escrita de R1 não é completada até a primeira metade do ciclo de clock 5. Assim, AND que lê o registrador na segunda metade do ciclo de clock 4, receberá um resultado errado. As instruções OR e XOR, entretanto, são executadas corretamente: a primeira lerá R1 na segunda metade do ciclo de clock 5, após a escrita ter ocorrido; o último lê R1 apenas no ciclo de clock 6.

## ***Perigos de dados - 3***

### **O efeito da instrução DADD nas instruções a seguir**

Fonte: Arquitetura de Computadores: Uma Abordagem Quantitativa



## Perigos de dados - 4

O problema pode ser resolvido através do uso de uma técnica simples de hardware chamada *encaminhamento, desvio* ou *curto-circuito*. O principal insight no *encaminhamento* é que o resultado não é realmente necessário para o DSUB, ou para o AND, até que o DADD realmente o produza.

Se o resultado puder ser movido do registrador onde DADD o armazena para onde DSUB e AND precisam dele, então a introdução de barracas poderá ser evitada.

Usando esta observação, o *encaminhamento* funciona da seguinte maneira:

- o resultado da ALU dos registradores de pipeline EX/MEM e MEM/W é sempre realimentado para as entradas da ALU
- se o hardware de encaminhamento detectar que uma operação anterior da ALU modificou o registro correspondente a uma fonte para a operação atual da ALU, a lógica de controle seleciona o resultado de encaminhamento como a entrada da ALU em vez do valor lido do banco de registros.

## *Perigos de dados - 5*

**O efeito da instrução DADD nas instruções a seguir é resolvido pelo encaminhamento para evitar risco de dados**

Fonte: Arquitetura de Computadores: Uma Abordagem Quantitativa



## Perigos de dados - 6

O encaminhamento pode ser generalizado para incluir a passagem de um resultado para a unidade funcional que o requer. Um resultado é encaminhado do registrador do pipeline correspondente à saída de uma unidade para a entrada de outra, em vez de apenas do resultado de uma unidade para a entrada da mesma unidade.

Considere a execução em pipeline do código

PAI	R1, R2, R3
LD	R4,0(R1)
SD	R4,12(R1)

Para evitar uma parada nesta sequência, é necessário encaminhar os valores da saída da ALU e da saída da memória de dados dos registradores do pipeline para a ALU e as entradas da memória de dados.

## ***Perigos de dados - 7***

### **Encaminhamento de operando requerido pela *loja* durante MEM**

Fonte: Arquitetura de Computadores: Uma Abordagem Quantitativa



## Perigos de dados - 8

Infelizmente, nem todos os riscos potenciais aos dados podem ser resolvidos por meio do encaminhamento.

Considere a execução em pipeline do código

LD	R1,0(R2)
DS	R4 R1,R3
E	R6 R1 R7
OU	R8 R1, R9

A instrução LD não possui dados válidos no final do ciclo de clock 4, enquanto a instrução DSUB os requer no início deste ciclo. Assim, o perigo de utilizar o resultado de uma instrução de carga não pode ser completamente eliminado. O resultado pode ser encaminhado imediatamente para a ALU para uso pela instrução AND , que inicia 2 ciclos de clock após LD. Da mesma forma, a instrução OR não tem problema, pois obtém o valor por meio do arquivo de registradores.

## ***Perigos de dados - 9***

### **Perigo de dados que não pode ser resolvido encaminhando**

Fonte: Arquitetura de Computadores: Uma Abordagem Quantitativa



## Perigos de dados - 10

Para resolver o problema, é necessário adicionar hardware especial, chamado *intertravamento de pipeline*, para preservar o padrão de execução correto. Em geral, o *intertravamento da tubulação* detecta um perigo e paralisa a tubulação até que o perigo desapareça. Nesse caso, o *intertravamento do pipeline* paralisa o pipeline, começando com a instrução que deseja usar os dados até que a instrução de origem os produza e os disponibilize.

<i>Instrução</i>	<i>Número do relógio</i>								
	1	2	3	4	5	6	7	8	9
LD R1,0(R2)	SE	ID EX	MEM WB						
DSUB R4,R1,R5		SE	EUIA	<i>parar</i>	EX MEM WB				
E R6,R1,R7			SE	<i>parar</i>	ID EX	MEM WB			
OU R8,R1,R9				<i>parar</i>	SE	ID EX	MEM WB		

## Riscos de controle - 1

Quando uma ramificação é executada, ela pode ou não alterar o PC. Diz-se que se uma instrução de desvio altera o PC para seu endereço de destino, é um desvio *assumido*; se cair, é um galho *não arrancado*. Quando uma instrução é um desvio, o PC não é alterado até o final do estágio de ID.

O método mais simples para lidar com desvios é refazer a busca da instrução após o desvio, uma vez que a instrução de desvio é detectada (durante o estágio de ID). O primeiro ciclo IF é essencialmente uma parada, porque nunca realiza trabalho útil. Deve-se notar que se um desvio for desfeito, há uma penalidade, a repetição do estágio IF é desnecessária, pois a instrução correta foi de fato buscada.

<i>Instrução</i>	<i>Número do relógio</i>								
	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>6</b>	<b>7</b>	<b>8</b>	<b>9</b>
<i>filial</i>	SE		ID EX	MEM WB					
<i>sucessor</i>		SE	SE	ID EX	MEM WB				
<i>sucessor+1</i>				SE	ID EX	MEM WB			
<i>sucessor+2</i>					SE	ID EX	MEM WB		

## Riscos de controle - 2

Uma alternativa de maior desempenho, chamada de esquema *previsto não realizado*, é tratar cada desvio como *não realizado*, simplesmente permitindo que o hardware continue como se a instrução de desvio não tivesse sido executada. A complexidade deste esquema surge da necessidade de verificar quando o estado foi alterado por uma instrução e de reverter tal alteração. Parece que nada fora do comum está acontecendo no pipeline neste caso. Se a ramificação for *tomada*, no entanto, a instrução buscada será transformada em um *no-op*. instrução e o estágio de busca é reiniciado no endereço de destino.

<i>Instrução</i>	<i>Número do relógio</i>							
	1	2	3	4	5	6	7	8
<i>ramo não tomado</i>	SE	ID EX	MEM WB					
<i>sucessor</i>		SE	ID EX	MEM WB				
<i>sucessor+1</i>			SE	ID EX	MEM WB			
<i>sucessor+2</i>				SE	ID EX	MEM WB		

<i>Instrução</i>	<i>Número do relógio</i>							
	1	2	3	4	5	6	7	8
<i>ramo tomado</i>	SE	ID EX	MEM WB					
<i>sucessor</i>		SE	<i>parado</i>	<i>parado</i>	<i>parado</i>	<i>parado</i>		
<i>alvo</i>			SE	ID EX	MEM WB			
<i>alvo+1</i>				SE	ID EX	MEM WB		

## Riscos de controle - 3

Pode-se pensar o contrário e tratar cada ramo como *tomado*. Este esquema é obviamente chamado de esquema *previsto*. É relevante apenas para processadores que usam códigos de condição de conjunto implícitos ou condições de ramificação mais poderosas e, portanto, mais lentas - o endereço de destino da ramificação é conhecido antes do resultado da ramificação e, por causa disso, o esquema previsto pode fazer sentido.

Em qualquer um dos esquemas previstos, o compilador desempenha um papel importante, pois pode melhorar o desempenho organizando o código de forma que o caminho mais frequente corresponda à escolha do hardware.

## Riscos de controle - 4

Um último esquema, denominado esquema de *ramificação atrasada*, foi muito utilizado no início do RISC. processadores. O ciclo de execução com atraso de ramificação de um é definido como

- instrução de ramo
- sucessor sequencial
- alvo do ramo se for tomado

O sucessor sequencial está no *slot de atraso de ramificação*. Esta instrução é executada se a ramificação é ou não tomada e não pode ser ela própria uma instrução de ramificação.

<i>Instrução</i>	<i>Número do relógio</i>							
	1	2	3	4	5	6	7	8
<i>ramo não tomado</i>	SE	ID EX	MEM WB					
<i>atraso na filial</i>		SE	ID EX	MEM WB				
<i>sucessor</i>			SE	ID EX	MEM WB			
<i>sucessor+1</i>				SE	ID EX	MEM WB		

<i>Instrução</i>	<i>Número do relógio</i>							
	1	2	3	4	5	6	7	8
<i>ramo tomado</i>	SE	ID EX	MEM WB					
<i>atraso na filial</i>		SE	ID EX	MEM WB				
<i>alvo</i>			SE	ID EX	MEM WB			
<i>alvo+1</i>				SE	ID EX	MEM WB		

## Riscos de controle - 5

A tarefa do compilador é tornar a instrução sucessora sequencial válida e útil. Várias alternativas são possíveis

- *from before* – insere no *slot de atraso de desvio* uma instrução que deveria ser executada antes da instrução de desvio e que é independente dela (a situação *ideal* já que não resultam efeitos colaterais de sua aplicação)
- *from target* – insere no *slot de atraso do desvio* a instrução para a qual o desvio, se *tomado*, aponta e faz com que o desvio aponte para seu sucessor (observe que quando o desvio *não é tomado*, uma instrução extra, não pretendida originalmente, é executada)
- *from fall-through* – insere no *slot de atraso do desvio* a instrução que será executada a seguir se o desvio *não for executado* (então esta instrução ainda estará sendo executada quando o desvio *for executado*).

## ***Riscos de controle - 6***

**Otimizações do compilador de slot de atraso de ramificação**

Fonte: Arquitetura de Computadores: Uma Abordagem Quantitativa



É preciso ter em mente que as duas últimas otimizações só são possíveis se não implicarem como efeito colateral algo que faça o programa rodar incorretamente quando o desvio toma a direção inesperada!

## Riscos de controlo - 7

Para melhorar a capacidade do compilador de preencher o slot de atraso de ramificação, a maioria dos processadores com ramificações condicionais introduziu um *cancelamento* ou *anulação*. instrução de ramo . Parece a instrução usual de ramificação atrasada . Quando a ramificação é executada, a instrução no slot de atraso da ramificação é executada; entretanto, quando a ramificação não é executada, a instrução no slot de atraso da ramificação é transformada em uma instrução não operacional e nada acontece.

A aceleração efectiva destes esquemas para lidar com os riscos de controlo quando o código é executado, pode ser expresso por

$$\begin{aligned}
 \text{acelerar} &= \frac{\text{CPI geral não pipeline}}{1 + \text{ciclos de parada de pipeline por instrução pipelined}} \cdot \frac{\text{tempo de ciclo do relógio sem pipeline}}{\text{ciclo de clock com pipeline}} \\
 &= \frac{\text{CPI geral não pipeline}}{1 + (\text{frequência de ramificação} \cdot \text{penalidade de ramificação})} \cdot \frac{\text{tempo de ciclo do relógio sem pipeline}}{\text{ciclo de clock com pipeline}}
 \end{aligned}$$

## Riscos de controle - 8

À medida que os gasodutos se aprofundam e a penalização potencial das ramificações aumenta, as ramificações atrasadas e esquemas semelhantes são considerados insuficientes. É preciso ser mais agressivo na previsão de ramificações.

O problema é abordado de duas maneiras diferentes

- *esquemas estáticos de baixo custo* – baseiam-se em informações disponíveis no momento do compilador e na criação de perfis de execuções anteriores do mesmo código; a principal observação que faz esse esforço valer a pena é que o comportamento do ramo é frequentemente distribuído bimodalmente, ou seja, cada ramo em particular tende a ser frequentemente altamente tendencioso para ser *tomado* ou *não*.
- *esquemas dinâmicos* – consistem em métodos para prever a direção do ramal durante o tempo de execução.

Em geral, a taxa de erro de previsão para programas inteiros é maior do que para programas de ponto flutuante, não apenas a frequência de ramificação é maior, mas também a direção da ramificação é mais aleatória.

## ***Riscos de controle - 9***

**Taxa de previsão incorreta para SPEC92 usando um preditor baseado em perfil**

Fonte: Arquitetura de Computadores: Uma Abordagem Quantitativa

DETI

## Riscos de controle - 10

O esquema de previsão dinâmica de ramificação mais simples usa um *buffer de previsão de ramificação* (BPB) ou uma *tabela de previsão de ramificação* (BPT). Um *buffer de previsão de desvio* é uma pequena memória indexada pela parte inferior do endereço da instrução de desvio. A memória possui um bit por posição que indica se o ramo foi *tomado* recentemente ou *não*.

Com tal dispositivo, não se sabe realmente se a previsão está correta – ela pode ter sido definida ou redefinida por outra instrução de desvio que compartilha os mesmos bits de endereço de ordem inferior, ou agora o comportamento do desvio pode ser oposto ao que era antes. Se pensarmos um pouco sobre isso, não é determinante. Uma previsão é apenas uma dica sobre o futuro que se presume ser verdadeiro e, portanto, a busca prossegue na direção prevista. Se posteriormente for provado que é falso, o bit de predição é complementado e a busca é reiniciada.

Este esquema simples de previsão de 1 bit tem uma falha de desempenho: mesmo quando o desvio é quase sempre tomado, será provável que ele preveja incorretamente duas vezes, em vez de uma, quando não foi tomado. Isso pode ser remediado usando um esquema de previsão de 2 bits.

## ***Riscos de controle - 11***

O esquema de previsão de 2 bits introduz histerese no processo de previsão.

### **Diagrama de estado do esquema de previsão de 2 bits**

Fonte: Arquitetura de Computadores: Uma Abordagem Quantitativa



## ***Riscos de controle - 12***

A melhoria em relação a um preditor baseado em perfil é real, mas não espetacular.

**Taxa de previsão incorreta para SPEC89 usando um buffer de previsão de 2 bits com 4.096 entradas**

Fonte: Arquitetura de Computadores: Uma Abordagem Quantitativa

DETI

## Riscos de controle - 13

Estudos mostram que a taxa de acerto do buffer não é o principal fator limitante. Na verdade, o comportamento de um buffer com 4.096 entradas é muito semelhante ao comportamento de um buffer com número ilimitado de entradas. Aumentar o número de bits do preditor sem alterar a estrutura do preditor também tem pouco impacto.

As abordagens atuais favorecem o uso de informações locais e globais para melhorar precisão da previsão.

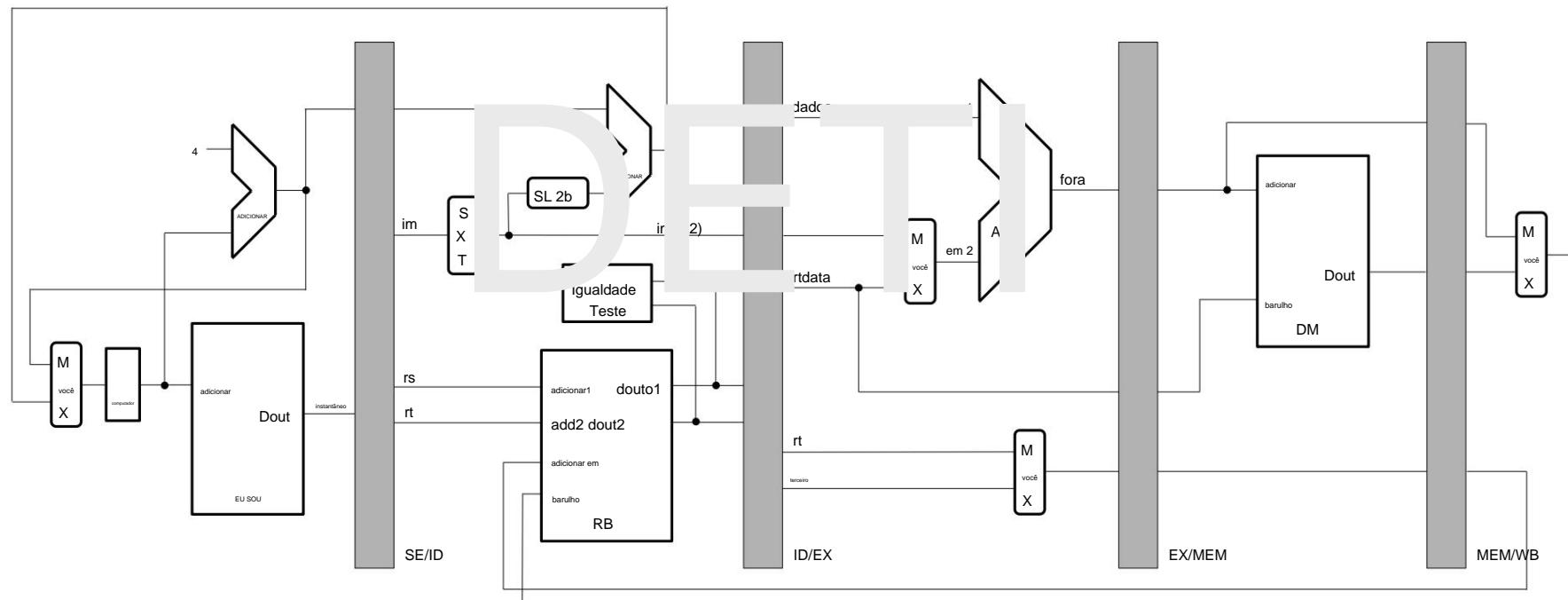
No caso mais simples, um *preditor correlacionado* pode consistir em um preditor de 2 bits para cada filial, relatando sua história passada em diferentes situações globais.

Mais recentemente, foram utilizados *preditores de torneios*. Um *preditor de torneio* usa vários preditores, rastreando qual produz o melhor resultado para cada ramo e considerando a sugestão vencedora como a próxima previsão.

# Implementação do pipeline clássico de 5 estágios - 1

## Caminho de dados do pipeline de 5 estágios

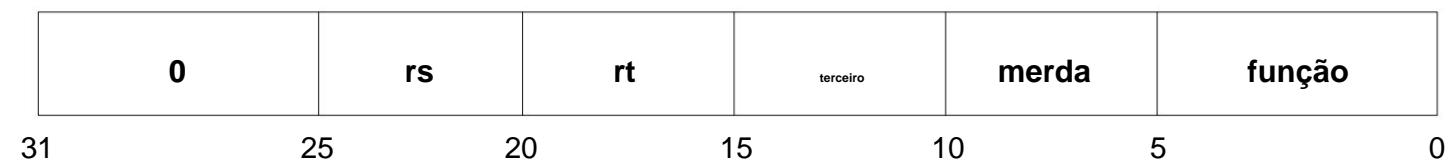
Fonte: Adaptado de Arquitetura de Computadores: Uma Abordagem Quantitativa



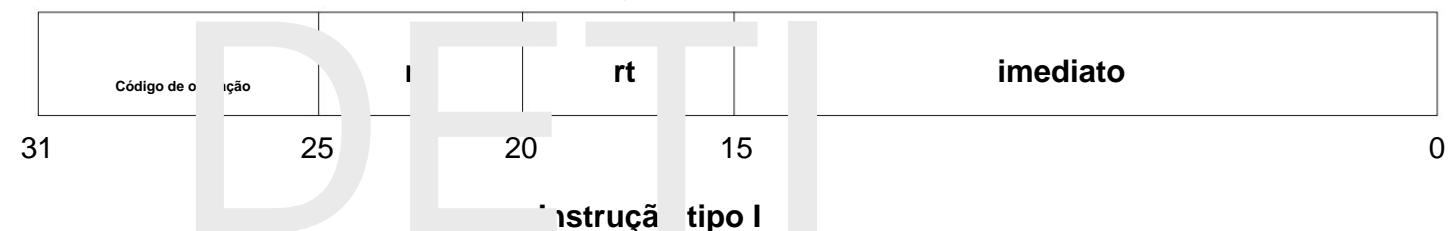
## Implementação do pipeline clássico de 5 estágios - 2

### Classes de instruções a serem consideradas

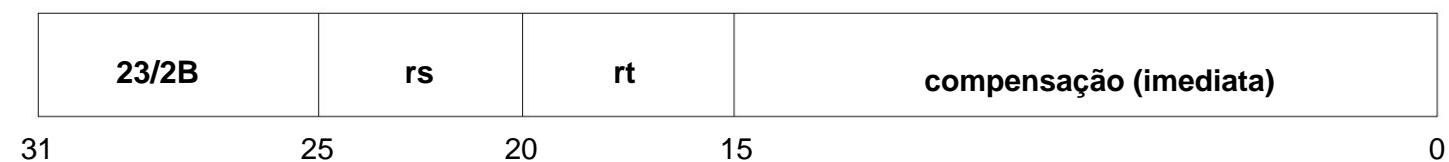
aritmética / lógica  
instruções



Instrução tipo R

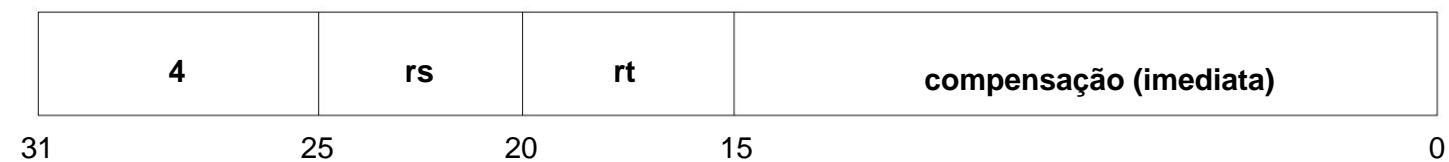


carregar e armazenar  
instruções



Instrução tipo I

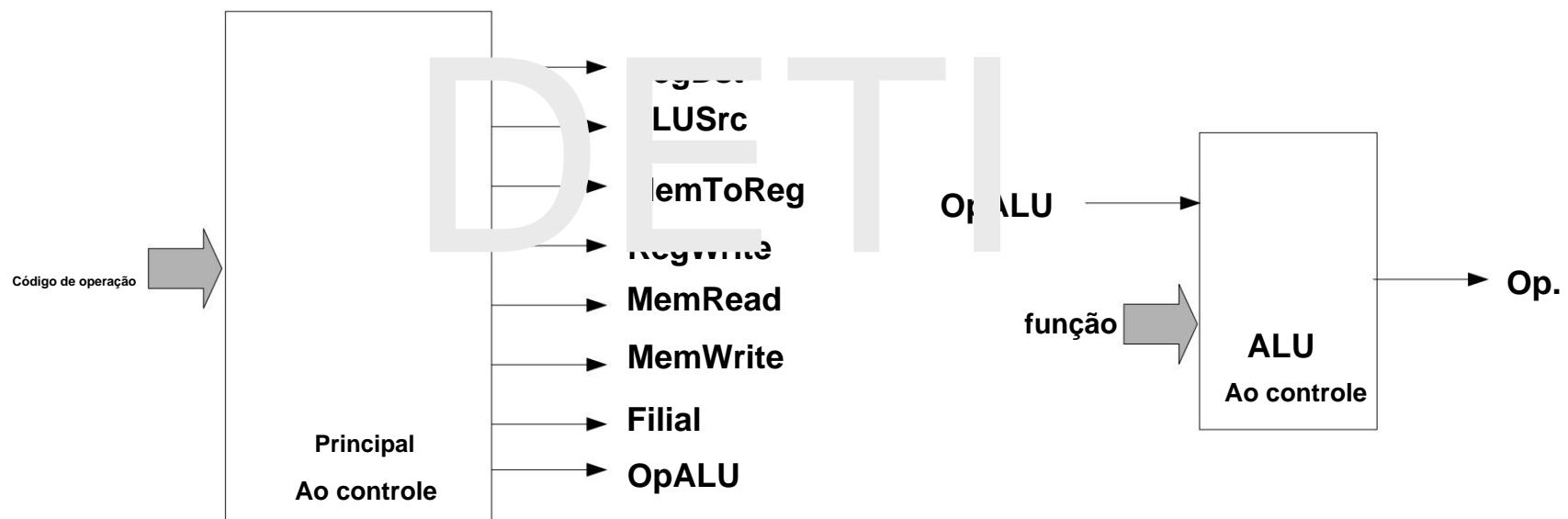
filial  
instruções



Instrução tipo I

## Implementação do pipeline clássico de 5 estágios - 3

Unidade de controle de pipeline



# Implementação do pipeline clássico de 5 estágios - 4

## Descrição Funcional do Controle Principal

Fonte: Adaptado de Organização e Design de Computadores: A Interface Hardware/Software

Instrução Tipo	Código de operação	Decodificação de instrução / Registrar estágio de busca	Estágio de Execução/Endereço Efetivo						Estágio de acesso à memória		Escreva nos bastidores	
			Filial	RegDst	ALUSrc	ALUOp1	ALUOp2	MemRead	MemWrite	RegWrite	MemToReg	
Formato	0x00		0	1	0		1	0	0	0	1	0
R lw	0x23					1		0	1	0	1	1
sw	0x2B		0	x	1		0	0	0	1	0	x
beq	0x04		1	x	0		0	0	0	0	0	x

Efeito do nome do sinal quando desativado (0), o		Efeito quando ativado (1)
RegDst	número de destino do registro para o registrador de gravação vem de o campo rt (bits 20:16)	o número de destino do registro para o registro de gravação vem de o terceiro campo (bits 15:11)
RegWrite	nenhum	o registro cujo número está em addin é escrito com o valor em din
ALUSrc	o segundo operando da ALU vem do conteúdo do registrador cujo número está no campo rt (bits 20:16)	o segundo operando da ALU é o campo imediato (bits 15:0) após estendendo o sinal para 32 bits
PCSrc	o PC é substituído pela saída do somador que calcula o valor de PC+4	o PC é substituído pela saída do somador que calcula o endereço de destino da filial
MemLeia	nenhum	o conteúdo do local de memória referenciado por add é colocado em ponto
MemWrite	nenhum	o conteúdo do local de memória referenciado por add é substituído pelo valor em din
MemToReg	o valor alimentado para a entrada de dados do registrador de gravação vem de a ULA	o valor alimentado para a entrada de dados do registrador de gravação vem de a falta de memória de dados

## **Implementação do pipeline clássico de 5 estágios - 5**

### **Descrição funcional do controle ALU**

Fonte: Adaptado de Organização e Design de Computadores: A Interface Hardware/Software

Instrução	Função ALU	Op	Ação ALU desejada	Operação
eu	00	xxxxxx		0010
sw	00	xxxxxx		0010
	10	100.000		0010
adicionar	10	100010	adicionar	0110
subtrair e	10	100100	adicionar subtrair e	0000
ou	10	100101	ou	0001
slt	10	101010	definido em menos de	0111

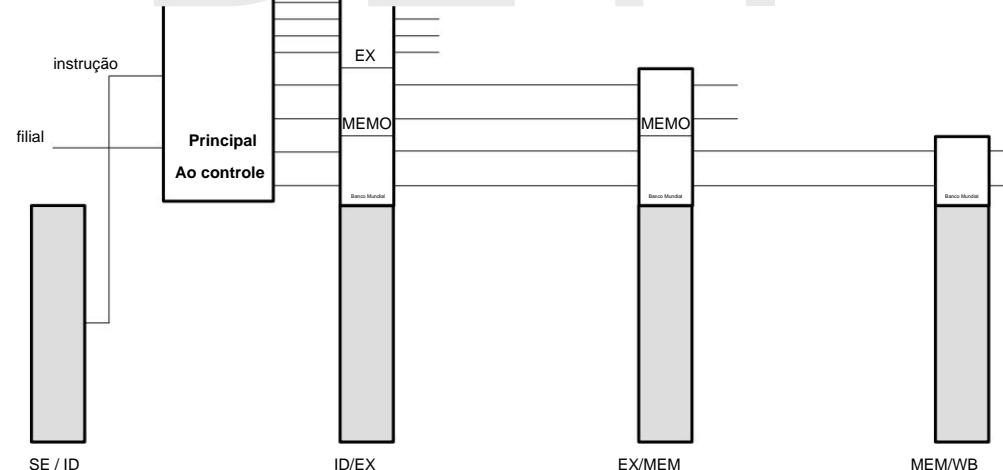
## Implementação do pipeline clássico de 5 estágios - 6

Para que o circuito de controle funcione corretamente, os sinais de controle devem ser ajustados com os valores corretos em cada estágio de cada instrução. A maneira mais simples de fazer isso é estender os registradores do pipeline para incluir informações de controle.

Como os sinais de controle começam a ser aplicados no estágio EX, as informações de controle pode ser criado durante o estágio de ID.

### Implantação de sinais de controle através dos diferentes estágios do tubo

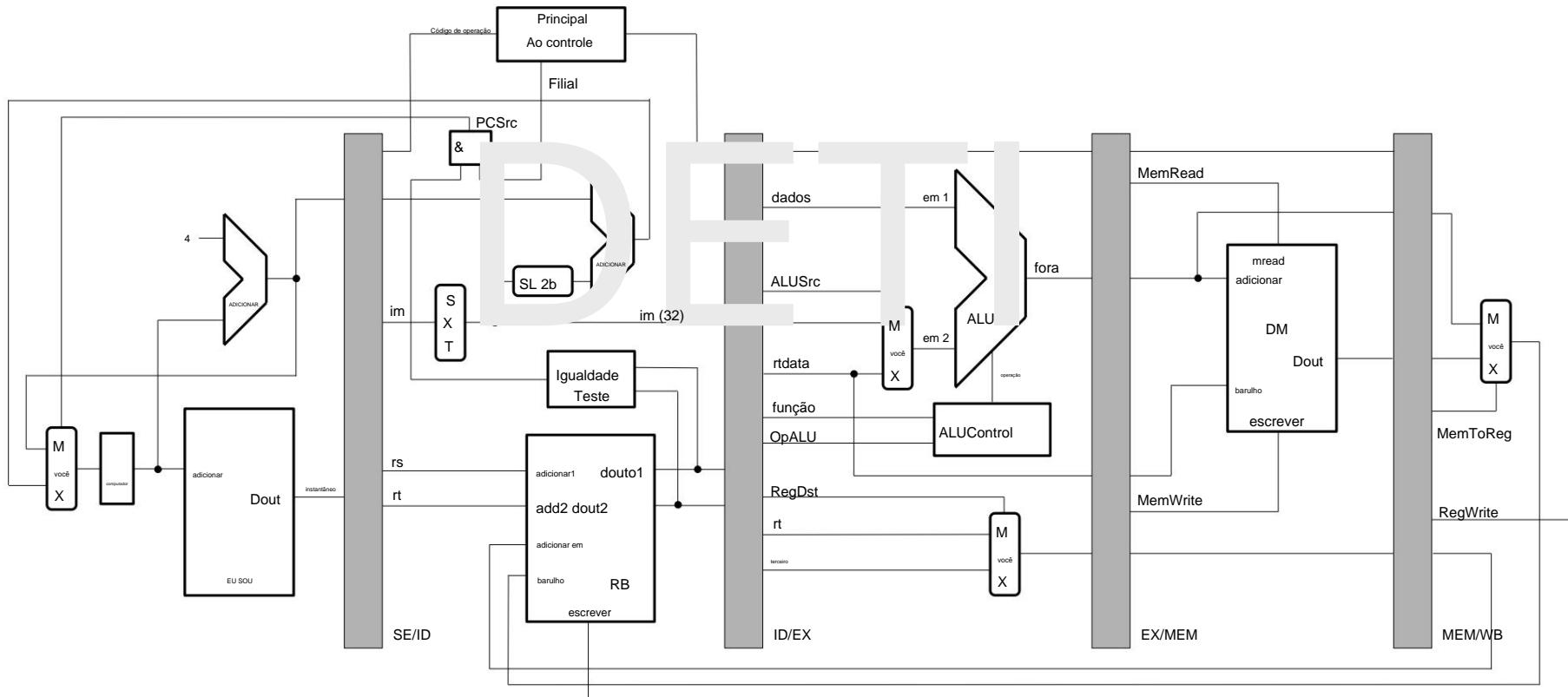
Fonte: Adaptado de Organização e Design de Computadores: A Interface Hardware/Software



# Implementação do pipeline clássico de 5 estágios - 7

## Integrando o controle no caminho de dados do pipeline de 5 estágios

Fonte: Adaptado de Arquitetura de Computadores: Uma Abordagem Quantitativa



## ***Implementação do pipeline clássico de 5 estágios - 8***

Os *perigos de dados* que podem ser resolvidos pelo *encaminhamento* para esta implementação específica de pipeline de 5 estágios podem ser classificados nas seguintes classes

- *Perigo de identificação* – o conteúdo de um registro do banco de registros é exigido por um beq instrução para teste de igualdade e foi modificado por uma instrução anterior e seu valor atualizado está atualmente armazenado em um ID/EX, EX/MEM ou MEM/ Registro de pipeline WB
- *Perigo EX* – o conteúdo de um registro do banco de registros é exigido por uma instrução e foi modificado por uma instrução anterior e seu valor atualizado é atualmente armazenado em um registro de pipeline EX/MEM ou MEM/WB
- *Risco MEM* – o conteúdo de um registro do banco de registros é exigido por uma instrução e foi modificado por uma instrução anterior e seu valor atualizado é atualmente armazenado em um registro pipeline MEM/WB.

## **Implementação do pipeline clássico de 5 estágios - 9**

### **Detectando riscos de dados de ID**

```
if (Branch && ID/EXE.RegWrite && (ID/EXE.RegDest == 0) && (ID/EXE.RegDest == IF/
    ID.RegRs) )
    fwdIDA = 01
caso contrário, se (Branch && EX/MEM.RegWrite && (EX/MEM.RegDest == 0) &&
    (EX/MEM.RegDest == SE/ID.RegRs) )
    fwdIDA = 10
senão if (Branch && MEM/WB.RegWrite) && (MEM/
    WB.RegDest == 0) && (MEM/
    WB.RegDest == IF/ID.RegRs) )
    fwdIDA = 11
senão fwdIDA = 00
```

## Implementação do pipeline clássico de 5 estágios - 10

### Detectando riscos de dados de ID (cont.)

```
if (Branch && ID/EXE.RegWrite && (ID/EXE.RegDest == 0) && (ID/EXE.RegDest == IF/
    ID.RegRt) )
    fwdIDB = 01
caso contrário, se (Branch && EX/MEM.RegWrite && (EX/MEM.RegDest == 0) &&
    (EX/MEM.RegDest == SE/ID.RegRt) )
    fwdIDB = 10
senão if (Branch && MEM/WB.RegWrite) && (MEM/
    WB.RegDest == 0) && (MEM/
    WB.RegDest == IF/ID.RegRt) )
    fwdIDB = 11
senão fwdIDB = 00
```

# Implementação do pipeline clássico de 5 estágios - 11

## Detectando perigos de dados EX

```
if (EX/MEM.RegWrite && (EX/MEM.RegDest != 0) &&
    (EX/MEM.RegDest == ID/EX.RegRs))
    fwdEXA = 10
  caso contrário, se (MEM/WB.RegWrite && (MEM/WB.RegDest != 0) && (MEM/
    WB.RegDest == ID/EX.RegRs),
    fwdEXA = 11
  senão fwdEXA = 00

if (EX/MEM.RegWrite && (EX/MEM.RegDest != 0) &&
    (EX/MEM.RegDest == ID/EX.RegRt) && !ID/EX.ALUSrc )
    fwdEXB = 10
  caso contrário, se (MEM/WB.RegWrite && (MEM/WB.RegDest != 0) &&
    (MEM/WB.RegDest == ID/EX.RegRt) && !ID/EX.ALUSrc )
    fwdEXB = 11
  senão se (ID/EX.ALUSrc)
    fwdEXB = 01
  senão fwdEXB = 00
```

## ***Implementação do pipeline clássico de 5 estágios - 12***

### **Detectando riscos de dados MEM**

```
if (EX/MEM.MemWrite && (EX/MEM.RegDest != 0)&& MEM/  
WB.RegWrite && (EX/  
MEM.RegDest == MEM/WR.RegDest )) fwdMEM = 1  
  
else fwdMEM = 0
```

# Implementação do pipeline clássico de 5 estágios - 13

## Valores de seleção para os multiplexadores de encaminhamento no estágio ID

Seleção MUX	Explicação da fonte
fwdIDA = 00	SE/ID O primeiro operando ALU vem do arquivo de registro
fwdIDA = 01	ID/EX O primeiro operando da ALU é encaminhado do resultado da ALU
fwdIDA = 10 EX/MEM O primeiro operando da ALU é encaminhado do resultado da ALU para o último resultado da ALU	O resultado da ALU é encaminhado para o último resultado da ALU
fwdIDA = 11 MEM/WB	O resultado da ALU é encaminhado da memória de dados ou do segundo para o último resultado da ALU
fwdIDB = 00	SE/ID O segundo operando ALU vem do arquivo de registro
fwdIDB = 01	ID/EX O segundo operando da ALU é encaminhado do resultado da ALU
fwdIDB = 10 EX/MEM O segundo operando da ALU é encaminhado do resultado da ALU para o último resultado da ALU	O segundo operando da ALU é encaminhado da memória de dados ou do segundo para o último resultado da ALU
fwdBID = 11 MEM/WB	O segundo operando da ALU é encaminhado da memória de dados ou do segundo para o último resultado da ALU

# **Implementação do pipeline clássico de 5 estágios - 14**

## **Valores de seleção para os multiplexadores de encaminhamento no estágio EX**

Seleção MUX	Explicação	Fonte	Operando da ALU	Operando da ALU	Resultado da ALU
fwdEXA = 00	ID/EX O primeiro operando da ALU vem do arquivo de registro				
fwdEXA = 10 EX/MEM	O primeiro operando da ALU é o resultado da ALU				
fwdEXA = 11 MEM/WB	O primeiro operando da ALU é encaminhado da memória de dados ou do resultado da ALU para o último resultado da ALU				
fwdEXB = 00	ID/EX O segundo operando da ALU vem do arquivo de registro				
fwdEXB = 01	ID/EX O segundo operando da ALU é o campo imediato estendido com sinal para 32 bits				
fwdEXB = 10 EX/MEM	O segundo operando da ALU é encaminhado do último resultado da ALU				
fwdEXB = 11 MEM/WB	O segundo operando da ALU é encaminhado da memória de dados ou do resultado da ALU para o último resultado da ALU				

## **Implementação do pipeline clássico de 5 estágios - 15**

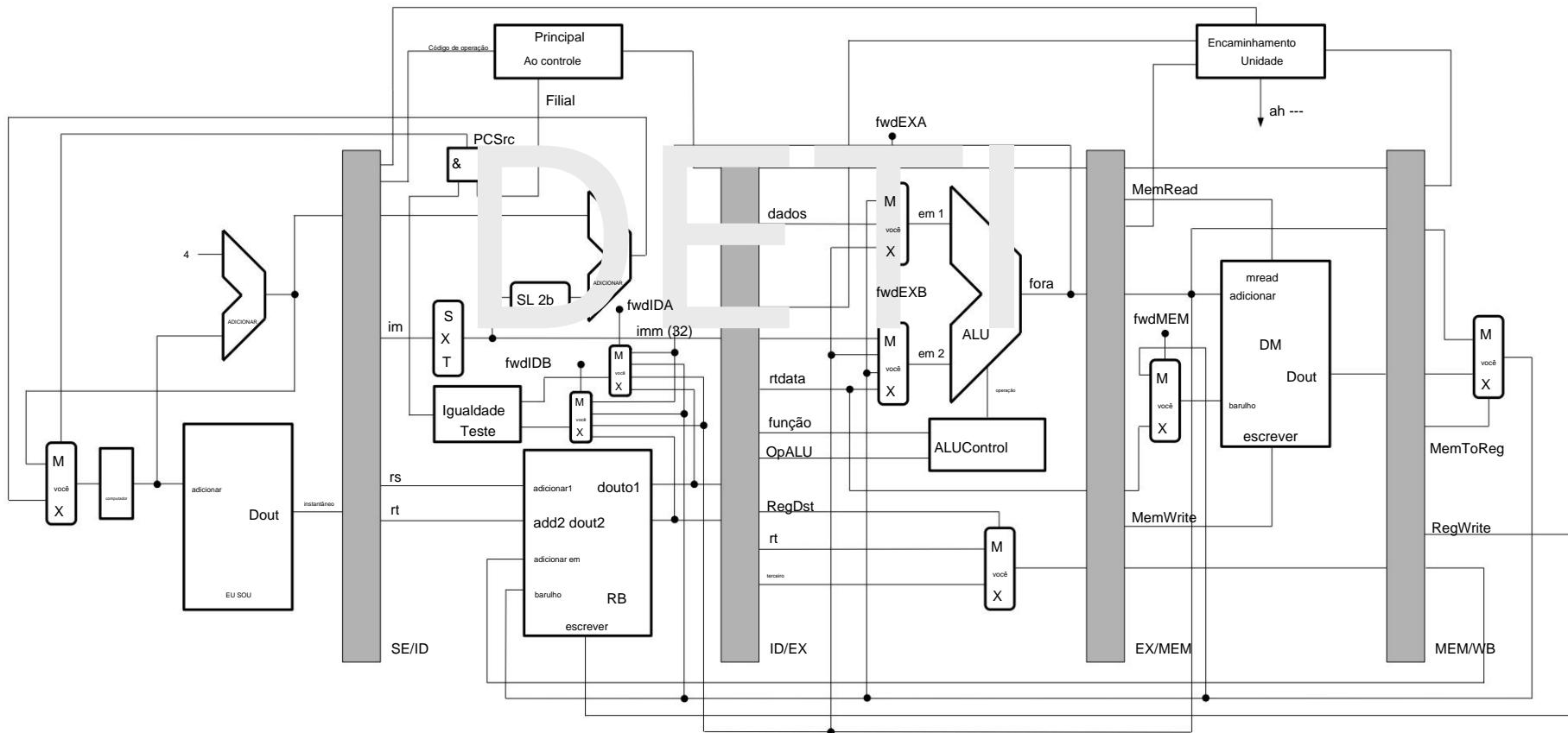
### **Valores de seleção para os multiplexadores de encaminhamento no estágio MEM**

Seleção MUX	Explicação da fonte
fwdMEM = 0 EX/MEM	O valor a ser possivelmente escrito na memória de dados vem do conteúdo do registrador rt
fwdMEM = 1 MEM/WB	O valor a ser escrito na memória de dados é o valor que acabou de ser produzido

# Implementação do pipeline clássico de 5 estágios - 16

## Integrando o encaminhamento no caminho de dados do pipeline de 5 estágios

Fonte: Adaptado de Arquitetura de Computadores: Uma Abordagem Quantitativa



## ***Implementação do pipeline clássico de 5 estágios - 17***

Nem todos os *perigos de dados* para esta implementação específica do pipeline de 5 estágios podem ser resolvidos pelo *encaminhamento*. Como foi visto, se uma instrução imediatamente após uma carga tentar ler o mesmo registrador por ela escrito, não há como esta instrução prosseguir até que o novo valor seja efetivamente recuperado da memória. Portanto, o progresso desta instrução no pipeline deve ser *paralisado*.

Além disso, e no que diz respeito aos *riscos de controlo*, se uma *ramificação atrasada* Se o esquema for assumido, nenhuma nova situação anormal que exigiria a paralisação do progresso da instrução precisa ser considerada.

Quando a instrução que segue a carga é uma instrução ALU, um ciclo de estol é obrigatório; quando se trata de uma instrução beq , são necessários dois ciclos de estol.

## Implementação do pipeline clássico de 5 estágios - 18

### Detectando a necessidade de inserção de ciclos de estol

```
if (ID/EX.MemRead && !MemWrite && ((ReaDest  
== IF/ID.Re Rs) ||  
(RegDest == IF/ID.RegRt && !Mem Read ))  
interromper pipeline,  
caso contrário, se (Branch && EX/ID.MEM.MemRead  
&& ((EX/MEM.RegDest == IF/ID.RegRs) || (EX/  
MEM.RegDest == IF/ID.RegRt)) interromper o  
pipeline
```

## Implementação do pipeline clássico de 5 estágios - 19

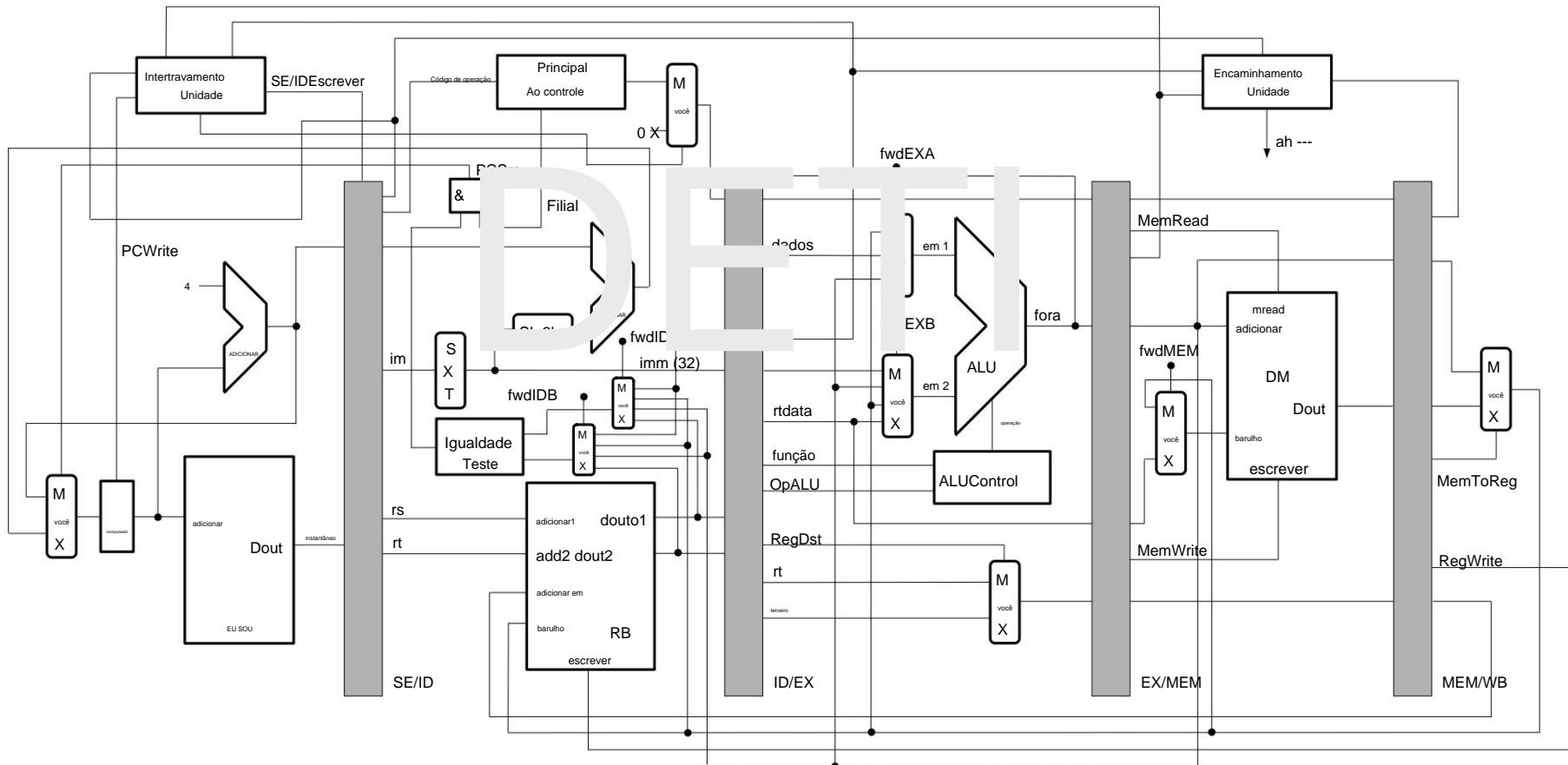
Para que a instrução no estágio ID seja paralisada, a instrução no estágio IF também deve ser paralisada. Uma maneira simples de conseguir isso é evitar que os registros do pipeline IF/ID sejam alterados. Desde que o conteúdo desses registradores seja preservado, as instruções processadas nos estágios IF e ID são mantidas iguais.

Por outro lado, as instruções nas demais etapas devem prosseguir como se nada tivesse acontecido. Isso significa que uma instrução *no-op* deve ser gerada e inserida no estágio EX no próximo ciclo de clock. Isso pode ser conseguido desabilitando os sinais de controle quando eles são gravados nos registradores do pipeline ID/EX.

# Implementação do pipeline clássico de 5 estágios - 20

## Integrando intertravamento no caminho de dados do pipeline de 5 estágios

Fonte: Adaptado de Arquitetura de Computadores: Uma Abordagem Quantitativa



# Exceções - 1

O controle é o aspecto mais desafiador do design do processador: é ao mesmo tempo a parte mais difícil de acertar e a parte mais difícil de acelerar. Uma das partes mais difíceis do controle é permitir exceções ou *interrupções* – eventos que causam a quebra da sequencialidade estrita da execução das instruções, além de desvios e saltos.

Exceções típicas incluem

- solicitações de atenção por um controlador associado a um dispositivo de E/S
- invocar uma chamada de sistema no nível do usuário
- rastrear a execução de instruções ou definir pontos de interrupção
- estouro de número inteiro ou anomalia de ponto flutuante
- falha de página em uma organização de memória virtual
- acesso desalinhado à memória
- violação de proteção de memória
- tentar executar uma instrução indefinida ou não implementada
- mau funcionamento de hardware
- falha de energia.

## Exceções - 2

As exceções individuais possuem características importantes que determinam qual ação o hardware deve executar. Esses requisitos podem ser classificados em cinco categorias semi-independentes

- *síncrono versus assíncrono* – diferentemente dos eventos assíncronos, os eventos síncronos ocorrem no mesmo local toda vez que o programa é executado com os mesmos dados e alocação de memória; eventos assíncronos, por outro lado, podem ocorrer em qualquer lugar do programa e geralmente podem ser tratados após a conclusão da instrução atual.
- *solicitadas pelo usuário versus coagidas* – as exceções solicitadas pelo usuário, sendo previsíveis, não são, em certo sentido, exceções verdadeiras; eles só são tratados como tal porque o mesmo mecanismo, que é usado para salvar e restaurar o estado do programa, também lhes é aplicado; como a única função da instrução que dispara esse tipo de exceção é causar a própria exceção, elas sempre podem ser tratadas após a conclusão da instrução; exceções coagidas, por outro lado, são causadas por algum evento de hardware que o programa não controla e são totalmente imprevisíveis

## Exceções - 3

- *mascarável* versus *não mascarável* – algumas exceções permitem que o programa escolha o momento em que o hardware responde a elas
- *dentro* vs. *entre instruções* – um evento pode impedir a conclusão de uma instrução em execução, quando é acionado por ela, porque ocorreu algum mau funcionamento ou anomalia ao nível de software/hardware; as exceções correspondentes são geralmente síncronas e difíceis de implementar porque a instrução deve ser interrompida e, eventualmente, reiniciada posteriormente; exceções assíncronas que ocorrem dentro de instruções, por outro lado, surgem de situações catastróficas e sempre causam o encerramento do programa
- *retomar* versus *encerrar* – exceções que não exigem que o programa seja retomado após serem tratadas são mais fáceis de implementar porque não há necessidade de reiniciar o programa.

# Exceções - 4

## Como estão as exceções comuns no esquema de classificação de 5 categorias

Fonte: Adaptado de Arquitetura de Computadores: Uma Abordagem Quantitativa

Exceção Tipo	Síncrono vs. Assíncrono	Solicitado pelo usuário vs. Coagido	Mascarável vs. Não mascarável	Dentro vs. Entre instruções	Curriculum vs. Encerrar
Solicitação de dispositivo	assíncrono	coagido	não mascarável	entre	retomar
de E/S, invocação do sistema	síncrono	usuário solicitou	não mascarável	entre	retomar
operacional, rastreamento de execução ou definição de instruções	síncrono	usuário solicitou	não mascarável	entre	retomar
pontos de interrupção					
estouro de inteiro ou anomalia de FP	síncrono	coagido	não mascarável	dentro	retomar
falha de página (memória virtual) memória desalinhada	síncrono	coagido	não mascarável	dentro	retomar
acesso à memória violação de proteção instrução indefinida ou não implementada	síncrono	coagido	não mascarável	dentro dentro	encerrar
execução					
falha de hardware falha de energia	assíncrono	coagido	não mascarável	dentro	encerrar
	assíncrono	coagido	não mascarável	de dentro	encerrar

## Exceções - 5

Tal como acontece nas organizações que não são de pipeline, a difícil tarefa é implementar exceções que ocorrem nas instruções, normalmente nos estágios EX ou MEM, que devem ser retomadas. A implementação supõe que outro programa, em princípio o sistema operacional, seja invocado para salvar o estado do programa em execução, corrigir a causa da exceção e restaurar o estado do programa, antes que a instrução que causou a exceção seja executada novamente – uma mecanismo que obviamente deve ser totalmente transparente para o programa em execução.

Se o pipeline tiver a capacidade de tratar a exceção, salvar o estado e reiniciar sem afetar a execução do programa, o processador será reinicializado . Embora os primeiros supercomputadores e microprocessadores muitas vezes não possuíssem essa propriedade, quase todos os processadores hoje a suportam, pelo menos para o pipeline de números inteiros, porque ela está na base da organização operacional da memória virtual.

## Exceções - 6

Para que o sistema operacional seja capaz de tratar uma exceção, ele deve saber o motivo que a desencadeou, além da instrução que a causou ou que seria executada em seguida se a exceção não fosse atendida. Existem dois métodos principais para comunicar o motivo de uma exceção

- *registrator de causa* – é um registrador de status que contém um campo que descreve a causa de uma exceção ocorrida, sendo seu valor definido pelo hardware no momento de sua detecção; quando as exceções são atendidas pelo mesmo endereço de ponto de entrada, este é o meio que o sistema operacional tem para determinar a causa da exceção
- *exceções vetorizadas* – existem múltiplos endereços de pontos de entrada para atendimento das exceções; em geral, cada endereço de ponto de entrada está associado a uma exceção específica, de modo que a identificação da causa pelo sistema operacional torna-se trivial.

## Exceções - 7

Quando uma exceção é atendida, o controle do pipeline deve executar as etapas a seguir para salvar o estado do programa e permitir que o programa seja retomado mais tarde, se for esse o caso.

1. O valor atual do PC é salvo e substituído pelo endereço do ponto de entrada correspondente à exceção para o próximo ciclo IF. O processador é colocado em um modo privilegiado onde um conjunto de instruções mais amplo está disponível e todas as exceções mascaráveis do mesmo nível de prioridade ou inferior são desabilitadas.
2. Todas as instruções subsequentes atualmente no pipeline, e a própria instrução, se a exceção estiver dentro, devem ser transformadas em instruções *não operacionais* para os estágios restantes do pipeline. Por outro lado, todas as instruções anteriores, se houver, devem ser concluídas para que o estado do programa seja consistente no momento do processamento da exceção.

## Exceções - 8

3. Depois que a rotina de serviço de exceção no sistema operacional começa a ser executada, ela atualiza imediatamente o valor salvo do PC para o valor correto, que depende da causa da exceção (dentro ou entre, e se o primeiro caso for verdadeiro, no estágio de tubo que o acionou).

Deve-se notar também que, se for assumido um esquema de *ramificação atrasada*, não é mais possível recriar o estado do processador com o armazenamento de um único valor de PC – as instruções no pipeline podem não estar relacionadas sequencialmente. Para que a rotina de serviço de exceção seja capaz de atualizar o PC salvo para o valor correto, são necessários tantos endereços de PC quanto o comprimento do *slot de atraso da ramificação* mais 1.

4. Finalmente, após o término da rotina de serviço de exceção, instruções especiais do tipo *retorno da exceção* reiniciam o fluxo de instruções anterior, restaurando o valor do PC e o modo de execução anterior.

## Exceções - 9

Se o pipeline puder ser interrompido de modo que as instruções que precedem a instrução com falha sejam concluídas e ela própria, juntamente com as que a sucedem, possam ser reiniciadas do zero, diz-se que o pipeline tem exceções precisas .

Idealmente, a instrução faltosa não deve alterar o estado de execução e, portanto, para tratar corretamente algumas exceções, é necessário que a instrução falsificada não produza efeitos. Para outras exceções, entretanto, como exceções de ponto flutuante, a instrução faltosa em alguns processadores escreve seu resultado antes que a exceção possa ser tratada. Nesses casos, o hardware deve estar preparado para recuperar os operandos de origem, mesmo que o destino seja igual a um dos operandos de origem.

Para superar esse problema, muitos processadores recentes de alto desempenho introduziram dois modos de operação: um modo tem exceções precisas e o outro, para ter melhor desempenho, não. Na verdade, o modo de exceção preciso deve ser mais lento porque permite muito menos sobreposição entre instruções de ponto flutuante.

# Exceções - 10

Com o pipelining, múltiplas exceções podem ocorrer no mesmo ciclo de clock porque há múltiplas instruções em execução simultânea.

## Típico dentro das exceções que podem ocorrer no pipeline clássico de 5 estágios

Fonte: Arquitetura de Computadores: Uma Abordagem Quantitativa

Gasoduto Estágio	Exceções
SE	falha de página na busca de instrução acesso incorreto à memória violação de proteção de memória
EU IA	instrução indefinida ou não implementada
EX	exceção aritmética
MEMO	falha de página na busca de dados acesso incorreto à memória violação de proteção de memória
WB nenhum	

# Exceções - 11

Considere a sequência de instruções

LD	SE	EU IA	EX M	EM WB		
PAI		SE	EU IA	EX M	EM WB	

Este par de instruções pode causar uma falha na página de dados e uma exceção aritmética no mesmo ciclo de clock. A maneira de lidar com esse problema é responder à exceção da página de dados e então reiniciar a execução. A exceção aritmética ocorrerá novamente e só então poderá ser tratada de forma independente.

Em geral, as coisas não são tão simples. Exceções podem ocorrer *fora de ordem*, ou seja, uma instrução pode disparar uma exceção antes que uma anterior, que está em execução simultânea, também desencadeie a sua própria. Para visualizar isso, considere, por exemplo, a situação em que a instrução de carregamento gera uma falha de página na busca de dados no estágio MEM e a instrução add gera uma falha de página na busca de instrução no estágio IF.

Para desenvolver uma implementação *precisa* do pipeline de exceções, a exceção acionada pela instrução de carga deve ser tratada primeiro. Como isso pode ser feito?

## Exceções - 12

O pipeline não pode lidar com exceções conforme elas ocorrem no tempo, pois ao fazer isso as exceções correm o risco de serem processadas fora do pedido sem pipeline.

Em vez disso, o hardware publica todas as exceções causadas por uma determinada instrução em um vetor de status associado a essa instrução. O vetor de status continua se movendo ao longo dos estágios do tubo com a instrução. Assim que houver uma indicação de exceção no vetor de status, qualquer sinal de controle que possa causar a escrita de um valor de dado é desativado (isto significa a escrita no registro e os sinais de escrita na memória).

Quando a instrução entra no estágio WB, o vetor de status é verificado. Se alguma exceção for lançada, ela será tratada na ordem em que ocorreria no tempo em uma implementação sem pipeline.

## ***Operações multiciclo em pipeline clássico de 5 estágios - 1***

É impraticável assumir que todas as operações FP e multiplicações e divisões de inteiros serão concluídas em um ciclo de clock, ou mesmo em dois. Fazer isso significaria estender o período do relógio para permitir a execução das operações dentro dele, ou usar uma enorme quantidade de lógica na implementação das unidades funcionais, ou ambos juntos. Em vez disso, o pipeline de FP contemplará uma latência maior para as operações.

A melhor maneira de compreender a ideia é imaginar as instruções FP como tendo o mesmo pipeline que as instruções inteiras principais, com duas diferenças importantes

- o ciclo EX pode ser repetido quantas vezes forem necessárias para completar a operação – o número de repetições pode variar de acordo com a operação
- pode haver múltiplas unidades funcionais.

Uma parada ocorrerá se a instrução a ser emitida causar um risco estrutural para a unidade funcional necessária ou um risco para os dados.

## ***Operações multiciclo em pipeline clássico de 5 estágios - 2***

Para fins de discussão, quatro unidades funcionais separadas serão consideradas na implementação do pipeline

- a unidade inteira principal, que lida com cargas, armazenamentos e operações básicas de inteiros da ALU
- o FP e o multiplicador inteiro
- o somador FP, que lida com adição, subtração e conversões de tipo de dados
- o FP e o divisor inteiro.

Supondo que essas unidades funcionais não sejam pipeline, nenhuma nova instrução utilizando uma unidade funcional específica poderá ser emitida se uma instrução anterior utilizando a mesma unidade ainda estiver em operação, ou que seja a mesma, ainda não tenha saído do estágio EX. Além disso, se uma instrução não puder prosseguir para o estágio EX, todo o pipeline até este ponto ficará paralisado.

## ***Operações multiciclo em pipeline clássico de 5 estágios - 3***

**O pipeline clássico de 5 estágios com três não pipeline adicionais  
Unidades funcionais FP**

Fonte: Arquitetura de Computadores: Uma Abordagem Quantitativa

DETI

## Operações multiciclo em pipeline clássico de 5 estágios - 4

A estrutura de todo o pipeline pode ser generalizada para permitir o pipeline de algumas unidades funcionais de FP e permitir múltiplas operações contínuas. Para facilitar a forma como a descrição é feita, são introduzidas duas definições para as unidades funcionais EX

- *latência* – é o número de ciclos de clock intermediários entre uma instrução que produz um resultado e uma instrução que usa o resultado
- *intervalo de iniciação ou repetição* – é o número de ciclos de clock que devem decorrer entre a emissão de duas operações do mesmo tipo.

As operações inteiras da ALU têm latência zero, pois os resultados podem ser usados no próximo ciclo de clock. As cargas, por outro lado, têm latência igual a um, pois os resultados podem ser usados após um ciclo de clock intermediário. Além disso, como a maioria das operações consome seus operandos no início do estágio EX, a latência é geralmente o número de estágios após EX que uma instrução precisa para produzir o resultado: zero estágios para as operações inteiras da ALU e um estágio para as operações de carga. A exceção são as *lojas* onde a latência é menos um.

# Operações multiciclo em pipeline clássico de 5 estágios - 5

## Latências e intervalos de repetição para as operações nas unidades funcionais

Fonte: Arquitetura de Computadores: Uma Abordagem Quantitativa

Funcional Unidade	Latência	Repetição Intervalo
ALU inteiro	0	1
Memória de dados (Cargas de números inteiros e FP)		1
Adição de FP		1
Multiplicação de FP	6	1
Multiplicação inteira		
Divisão FP	24	25
Divisão inteira		

Observe que apenas a adição de FP e as unidades de multiplicação de FP/multiplicação de números inteiros são pipeline. A unidade de divisão de FP/divisão de números inteiros, por outro lado, não é pipeline e requer 24 ciclos de clock para produzir um resultado.

## ***Operações multiciclo em pipeline clássico de 5 estágios - 6***

**O pipeline clássico de 5 estágios que suporta múltiplas operações de FP pendentes**

Fonte: Arquitetura de Computadores: Uma Abordagem Quantitativa

DETI

## Operações multiciclo em pipeline clássico de 5 estágios - 7

As seguintes observações estão em ordem

- como a divisão FP/unidade de divisão inteira não possui pipeline, podem ocorrer *riscos estruturais*; eles devem ser detectados e as instruções de emissão que precisam usar a unidade devem ser interrompidas
- como agora nem todas as instruções têm o mesmo tempo de execução, o número de escritas em registradores no mesmo ciclo de clock pode ser maior que um
- *riscos de dados*, onde uma instrução tenta escrever um registro antes de ter sido escrito por outra instrução, são possíveis – o que implica que a ordem de execução sem pipeline não é mais mantida
- o facto de as instruções poderem ser concluídas numa ordem diferente daquela em que foram emitidas, dará origem a problemas no tratamento de excepções
- os *riscos de dados*, onde uma instrução tenta ler um registrador antes de ter sido escrito por outra instrução, serão mais frequentes.

## Operações multiciclo em pipeline clássico de 5 estágios - 8

Os riscos de dados, onde uma instrução tenta ler um registrador antes de ter sido escrito por outra instrução, seguem um padrão que é fundamentalmente o mesmo encontrado para o pipeline de inteiros.

Instrução	Número do relógio																
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
LD F4,0(R2)	SE	ID EX	MEM WB														
MUL.D F0,F4,F6		SE	Tenda c	Identificação N	M2 N	M4 M5 M6 M7 MEM WB											
ADICIONAR.D F2,F0,F8			SE	par	parada	parada parada parada	parada	parada	parada	parada	A2	A3	A4 MEM WB				
DP F2,0(R2)				SE	parada	parada parada parada parada	parada	parada	parada	ID EX parada				parar	parar MEM		

Observe que a instrução *store* foi paralisada por um ciclo extra para evitar o risco estrutural que surgiria do conflito de acesso ao estágio MEM.

Entretanto, como apenas uma das instruções acessa a memória de dados, elas podem prosseguir para o estágio MEM no mesmo ciclo de clock, se hardware extra for adicionado.

## Operações multiciclo em pipeline clássico de 5 estágios - 9

Se assumirmos que o banco de registradores FP possui uma única porta de gravação, sequências de operações FP, bem como uma instrução *de carregamento* de FP junto com outras operações FP, podem dar origem a conflitos de acesso à porta de gravação de registradores.

Instrução	Número do relógio										
	1	2	3	4	5	6	7	8	9	10	11
MUL.D F0,F4,F6	SE	M1	M3	M4	M5	M6	M7	M WB			
.			ID EX	MEM							
.			SE	ID EX	MEM	/B					
ADICIONAR.D F2,F4,F6			SE		EU	A1	A2	A3	A4	MEM	WB
.					SE	ID EX	MEM	WB			
.						SE	ID EX	MEM	WB		
LDF0,0(R2)							SE	ID EX	MEM	WB	

No ciclo de clock 11, todas as três instruções atingem o estágio WB e precisam ser escritas no banco de registradores FP. Com uma única porta de gravação, o processador deve serializar a conclusão da instrução. O número de portas de gravação poderia ser aumentado para resolver o problema, mas esta abordagem pode não ser muito atraente, uma vez que as portas de gravação extras raramente serão utilizadas.

## **Operações multiciclo em pipeline clássico de 5 estágios - 10**

Em vez disso, pode-se optar por detectar o risco estrutural e programar o acesso para a porta de gravação.

Uma solução é rastrear o uso da porta de gravação no estágio de ID e paralisar, se necessário, a instrução atual antes de ser emitida, assim como é feito para qualquer outro risco estrutural. A implementação desta abordagem necessita de um registrador de deslocamento, denominado *registrador de reserva*, cujo conteúdo se move na direção oposta ao fluxo de instruções no pipeline a cada ciclo de clock. Sua finalidade é sinalizar os ciclos onde as instruções já emitidas serão gravadas no banco de registros do FP. Assim, quando a instrução no estágio ID tiver que escrever em um registro do banco de registros FP, o estágio correspondente do registro de deslocamento é verificado. Se o bit estiver energizado, o que significa que outra instrução já emitida também fará isso naquele ciclo de clock, a instrução será interrompida. Caso contrário, esse bit do registrador de deslocamento é setado e a instrução progride.

Esta abordagem tem a vantagem de manter toda a detecção de intertravamento e inserção de bloqueio restrita ao estágio ID. O custo é o registrador de deslocamento extra e toda a lógica para determinar o potencial conflito de gravação.

## ***Operações multiciclo em pipeline clássico de 5 estágios - 11***

Outra solução é paralisar uma instrução conflitante enquanto ela tenta entrar no estágio MEM ou WB. Qualquer uma das instruções conflitantes pode ser escolhida para parar. Uma heurística simples, embora às vezes abaixo do ideal, é dar prioridade à instrução que sai da unidade com a latência mais alta, uma vez que é a que tem maior probabilidade de ter causado o bloqueio de outra instrução.

Esta abordagem tem a vantagem de adiar a detecção do conflito até o momento em que se torna trivial afirmá-lo. A desvantagem é que isso torna o controle da tubulação mais complexo, já que os bloqueios agora podem surgir de dois locais.

## Operações multiciclo em pipeline clássico de 5 estágios - 12

Os *riscos de dados*, onde uma instrução tenta escrever em um registrador antes de ter sido escrita por outra instrução, aparentemente não são dramáticos e podem ser descartados. A justificativa é que eles nunca deveriam ocorrer em uma sequência de código bem escrita porque nenhum compilador geraria duas escritas no mesmo registrador sem uma leitura intermediária. No entanto, se a sequência for inesperada, eles podem ocorrer e produzir resultados errados.

BNZ	R1,foo
DIV.D	F0,F2,F4
.	.
foo: LD	F0,0(R3)

Se o desvio for realizado (assumindo um esquema de desvio atrasado), a instrução LD alcançará o estágio WB antes que a instrução DIV.D possa ser concluída. Originando assim uma inconsistência no estado do programa a partir deste ponto.

## ***Operações multiciclo em pipeline clássico de 5 estágios - 13***

Existem duas maneiras possíveis de lidar com esse perigo. A primeira abordagem é atrasar a emissão da segunda instrução de escrita até que a primeira entre no estágio MEM. A segunda é eliminar a primeira instrução, detectando o perigo e desabilitando sua capacidade de alterar o registro – a segunda instrução pode então ser emitida imediatamente.

Devido à raridade de aparecer no código, qualquer abordagem é aceitável.

## Operações multiciclo em pipeline clássico de 5 estágios - 14

Um problema crítico causado por instruções de longa execução é ilustrado pela sequência de código abaixo

```
DIV.D F0,F2,F4  
ADICIONAR.D F10,F10,F8  
SUB D F12,F12 F14
```

Embora não haja dependências de dados entre as primeiras instruções, a instrução DIV será concluída após os próximos dois. Uma situação conhecida como *conclusão errada de ordem*.

Esta condição pode levar a exceções *imprecisas*. Para entender como, considere que após a conclusão das instruções ADD e SUB, a instrução DIV irá gerar uma exceção. Como tanto ADD quanto SUB alteraram cada um de seus operandos, o estado do programa foi modificado e não pode ser recuperado, nem mesmo com ajuda de software. Portanto, não é possível reiniciar a instrução DIV !

Existem maneiras de lidar com esse problema, mas elas não serão discutidas aqui.

## ***Operações multiciclo em pipeline clássico de 5 estágios - 15***

**Paradas por operação FP para cada tipo principal de operação FP para  
Referência SPEC89 FP**

Fonte: Arquitetura de Computadores: Uma Abordagem Quantitativa

DETI

## ***Operações multiciclo em pipeline clássico de 5 estágios - 16***

**Paradas por instrução para o pipeline MIPS FP para  
Referência SPEC89 FP**

Fonte: Arquitetura de Computadores: Uma Abordagem Quantitativa

DETI

## **Gasoduto MIPS R4000 - 1**

A família de processadores MIPS R4000 implementa o conjunto de instruções MIPS64, mas implementa um pipeline mais profundo do que o pipeline clássico de 5 estágios que foi descrito, tanto para programas inteiros quanto para programas FP.

O pipeline mais profundo permite que o processador atinja uma taxa de clock mais alta fazendo uma decomposição em oito estágios, em vez de cinco. Como o acesso à memória, mesmo pelo uso de caches, é particularmente crítico em termos de tempo, os estágios extras do pipe estão preocupados com a decomposição do acesso à memória. Esse tipo de pipeline mais profundo às vezes é chamado de *superpipelining*.

# **Gasoduto MIPS R4000 - 2**

## **Organização de pipeline de oito estágios R4000**

Fonte: Arquitetura de Computadores: Uma Abordagem Quantitativa

SE – primeira metade da busca de instrução  
IS – segunda metade da busca de instrução

RF – detecção de acertos no cache de instruções + decodificação de instruções e busca de registro + verificação de perigos

EX – execução, incluindo operação ALU, endereço efetivo e alvo de ramificação  
cálculo e avaliação de condição

DF – primeira metade da busca de dados

DS – segunda metade da busca de dados

TC – detecção de acertos no cache de dados

WB – registro de write-back



## Gasoduto MIPS R4000 - 3

Embora os acessos à memória de instruções e dados ocupem vários ciclos de clock, eles são totalmente pipeline e, portanto, uma nova instrução pode iniciar a cada ciclo de clock. Na verdade, como pode ser notado, como a detecção de acertos no cache ocorre no último estágio de acesso à memória, o pipeline tenta usar os dados antes mesmo de a detecção de acertos ser concluída. Se ocorrer uma falha, será feito backup do pipeline em um ciclo quando os dados corretos estiverem disponíveis.

Deve-se notar que esse pipeline de latência mais longa não apenas aumenta a lógica *de encaminhamento* necessária , mas também aumenta os atrasos *de carga e ramificação* . O atraso *de carregamento* tem dois ciclos, pois os dados estão disponíveis no final do estágio DS. A *filial* o atraso tem 3 ciclos de duração, uma vez que a condição de ramificação é calculada aqui durante o estágio EX. A arquitetura R4000, no entanto, usa um esquema *de ramal com atraso de ciclo único* , juntamente com uma estratégia *de não tomada prevista* , que não produz ciclos ociosos, quando o ramal não é tomado, e dois ciclos ociosos, quando o ramal é tomado.

Os *intertravamentos* de pipeline impõem a penalidade de bloqueio de ramificação de 2 ciclos, quando a ramificação é tomada e qualquer risco de dados que surja da ocorrência de uma instrução de carga.

## ***Pipeline MIPS R4000 - 4***

**Atraso de instrução de carga de dois ciclos na organização do pipeline de oito estágios**

Fonte: Arquitetura de Computadores: Uma Abordagem Quantitativa



PAI

# Pipeline MIPS R4000 - 5

**Parada de dois ciclos produzida por uma instrução de carregamento, seguida pelo uso imediato do valor carregado, na organização do pipeline de oito estágios**

<i>Instrução</i>	<i>Número do relógio</i>								
	1	2	3	4	5	6	7	8	9
LD R1,0(R10)	SE	É	RF EX	DF DS TC WB					
DAAD R2,R1,R2		SE	É	<i>Bloqueio de RF bloqueio EX DF DS</i>					
DSUB R3,R1,R3			SE	É	<i>parada parada RF EX DF</i>				
OU R4,R1,R4				SE	<i>barraca</i>		É RF EX		

## ***Gasoduto MIPS R4000 - 6***

**Atraso básico de ramal de três ciclos na organização de pipeline de oito estágios**

Fonte: Arquitetura de Computadores: Uma Abordagem Quantitativa

DETI

# Gasoduto MIPS R4000 - 7

**Comportamento atrasado da ramificação tanto para os casos aceitos quanto para os não atendidos na organização do pipeline de oito estágios**

<i>Instrução</i>	<i>Número do relógio</i>								
	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>6</b>	<b>7</b>	<b>8</b>	<b>9</b>
instrução de ramo	SE	É	RF EX	DF DS TC	WB				
atraso slot	SE	É	RF EX	F DS TC	WB				
ciclo ocioso		parar	parar	parar	parar	parar	parar	barraca	
ciclo ocioso			parar	parar	parar	parar	parar	barraca	
alvo de ramificação				SE	É RF EX DF				

<i>Instrução</i>	<i>Número do relógio</i>								
	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>6</b>	<b>7</b>	<b>8</b>	<b>9</b>
instrução de ramo	SE	É	RF EX	DF DS TC	WB				
slot de atraso		SE	É	RF EX	DF DS TC	WB			
instrução de ramo + 2			SE	É	RF EX	DF DS TC			
instrução de ramo + 3				SE	É	RF EX	DF DS		

## ***Gasoduto MIPS R4000 - 8***

A unidade de ponto flutuante MIPS R4000 consiste em três unidades funcionais nominais: um somador de ponto flutuante, um multiplicador de ponto flutuante e um divisor de ponto flutuante. A lógica do somador é usada nos estágios finais de uma operação de multiplicação ou divisão.

As operações de precisão dupla podem levar de 2 ciclos (para negação) a 112 ciclos (para uma raiz quadrada).

Cada unidade funcional pode ser considerada como tendo até oito estágios de processamento diferentes. Há uma única cópia de cada uma dessas etapas de processamento e diferentes instruções podem utilizar uma determinada etapa zero ou mais vezes e combiná-las em sua própria ordem.

# Gasoduto MIPS R4000 - 9

## Os oito estágios de processamento nos pipelines de ponto flutuante R4000

Fonte: Arquitetura de Computadores: Uma Abordagem Quantitativa

<i>Em processamento</i> <i>estágio</i>	<i>Elementar</i> <i>unidade funcional</i>	<i>Descrição</i>
A	Somador FP	estágio ADD mantissa
D	Divisor FP	dividir o estágio do pipeline
E	Multiplicador de FP	estágio de teste de exceção
M	Multiplicador de PF	primeiro estágio do multiplicador
N	Multiplicador de FP	segundo estágio do multiplicador
R	Somador FP	estágio de arredondamento
S	Somador FP	estágio de mudança de operando
você		descompactar números de ponto flutuante

# Gasoduto MIPS R4000 - 10

## Latências, intervalos de repetição e composição de estágios para operações de ponto flutuante no R4000

Fonte: Arquitetura de Computadores: Uma Abordagem Quantitativa

PF instrução	Latência	Repetição intervalo	Estágios de processamento
adicionar – subtrair	4	3	você – S+A – A+R – R+S
multiplicar	8	4	você – E+M – M – M – M – N – N+A – R
dividir	36	35	U – A – R – D28 – D+A – D+R – D+A – D+R – A – R
raiz quadrada	112	111	U – E – (A+R)108 – A – R
negar	2	1	NÓS
valor absoluto	2	1	NÓS
comparar	3	2	U – A – R

# Gasoduto MIPS R4000 - 11

**Efeito de uma instrução de multiplicação FP emitida no ciclo de clock 0 em uma instrução de adição de FP emitida entre os ciclos de clock 1 a 7**

Fonte: Adaptado de Arquitetura de Computadores: Uma Abordagem Quantitativa

<i>Instrução</i>	<i>Decisão</i>	Número do relógio									
		0	1	2	3	4	5	6	7	8	9
multiplicar	emitir		U E+MMMMN N+AR								
adicionar	emitir			U S+A A+R R+S							
	emitir				U S+A A+R R+S						
	emitir					U S+A A+R R+S					
	parar					parar	parar U	S+A A+R R+S			
	parar						parar U	S+A A+R R+S			
	emitir							U S+A A+R R+S			
	emitir								U S+A A+R R+S		

# Gasoduto MIPS R4000 - 12

**Efeito de uma instrução de divisão FP emitida no ciclo de clock 0 em uma instrução de adição de FP emitida entre os ciclos de clock 26 a 36**

Fonte: Adaptado de Arquitetura de Computadores: Uma Abordagem Quantitativa

<i>Instrução</i>	<i>Número do relógio</i>											
	<i>Decisão</i>	26	27	28	29	30	31	32	33	34	35	36
dividir	problema em cc 0	D	D	DDD	D+A	D+R	D+A	D+RAR				
adicionar	emitir			U	S+A	A+R	R+\$					
	emitir			U	S+A	A+R	R+\$					
	parar				parar	parar	parar	parar	parar	parar EUA + A		
	parar					parar	parar	parar	parar	parar EUA + A		
	parar						parar	parar	parar	parar EUA + A		
	parar							parar	parar	parar EUA + A		
	parar								parar	parar EUA + A		
	emitir									EUA + A		
	emitir										você	

## **Gasoduto MIPS R4000 - 13**

Existem quatro causas principais para paralisações ou perdas em oleodutos que impedem isso.  
a emissão de instruções à taxa nominal seja alcançada

- *travamentos de carga* – atrasos decorrentes do uso do valor de carga um ou dois ciclos de clock após a execução da carga
- *travamentos e perdas de ramificação* – dois atrasos de ciclo de clock após cada ramificação *tomada* e um atraso de ciclo de clock se o slot de atraso de ramificação não puder ser preenchido com uma instrução útil
- *Paradas no resultado do FP* – atrasos que surgem porque um operando é necessário e ainda não foi computado
- *Paralisações estruturais do FP* – atrasos que surgem porque os estágios de processamento necessários no pipeline do FP não estão disponíveis quando necessários.

## ***Gasoduto MIPS R4000 - 14***

**O CPI do pipeline para 10 dos benchmarks SPEC92 assumindo um cache perfeito**

Fonte: Arquitetura de Computadores: Uma Abordagem Quantitativa

DETI

## **Gasoduto MIPS R4000 - 15**

O pipeline R4000 tem atrasos de ramificação muito mais longos do que o pipeline clássico de 5 estágios. O atraso de ramificação mais longo aumenta substancialmente os ciclos de clock gastos nas ramificações, especialmente para programas inteiros com alta frequência de ramificação.

Um efeito interessante para programas de FP é que a latência das unidades funcionais de FP leva a mais paralisações de resultados do que aquelas produzidas pelos riscos estruturais, que são devidas tanto às limitações do intervalo de repetição quanto a conflitos decorrentes do uso de estágios de processamento específicos em diferentes FP. instruções.

Assim, reduzir a latência das operações FP deve ser a primeira tarefa a ser realizada em um esforço de otimização, ao invés de aumentar o pipeline ou replicar os estágios de processamento das unidades funcionais.

## ***Leitura sugerida***

- *Arquitetura de Computadores: Uma Abordagem Quantitativa*, Hennessy JL, Patterson DA,  
6<sup>a</sup> Edição, Morgan Kaufmann, 2017 – Apêndice A:  
*Princípios do Conjunto de Instruções (Seções 1 a 8)*  
– Apêndice C: *Pipelining, Conceitos Básicos e Terminologia (Seções 1 a 8)*
- 7) • *Organização e Arquitetura de Computadores: Projetar para Desempenho*, Stallings W.,  
10<sup>a</sup> Edição, Pearson Education, 2010  
– Capítulo 12: *Conjuntos de instruções: características e funções*  
– Capítulo 13: *Conjuntos de instruções: modos e formatos de endereçamento*  
– Capítulo 14: *Estrutura e Função do Processador*  
– Capítulo 15: *Computadores com conjunto de instruções reduzido*