Arquitecturas de

Paralelismo de Nível de Instrução (Princípios)

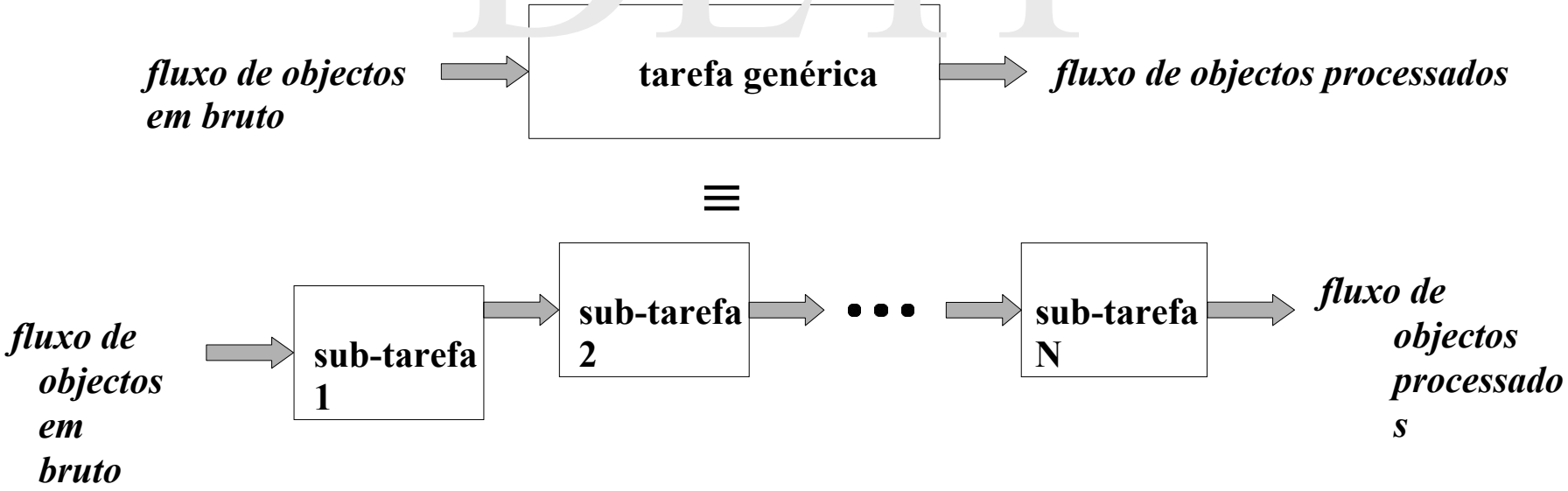
António Rui Borges

Resumo

- *O que é pipelinagem?*
- *Paralelismo ao nível das instruções*
- *Conjunto simplificado de instruções RISC*
- *Implementação não encurtada do conjunto de instruções RISC*
- *Gasoduto clássico de 5 fases para um processador RISC*
- *Principais obstáculos da canalização*
  - *Desempenho de condutas com bancas*
  - *Riscos estruturais*
  - *Perigos dos dados*
  - *Perigos de controlo*
- *Implementação do gasoduto clássico de 5 fases*
- *Excepções*
- *Operações de múltiplos ciclos em gasoduto clássico de 5 fases*
- *Gasoduto MIPS R4000*
- *Leitura sugerida*

O que é pipelinagem? - 3

A *canalização* é uma técnica de implementação onde a execução de uma tarefa genérica sobre os objectos de um riacho é convertida num tandem de subtarefas independentes que operam simultaneamente sobre os sucessivos objectos do riacho. Cada uma das subtarefas individuais, chamadas *fases* ou *segmentos de tubagem*, é executada em sequência e representa uma fracção definida de toda a tarefa. A sua execução ordenada combinada é equivalente à execução da tarefa original em cada objecto do fluxo.



O que é pipelinagem? - 4

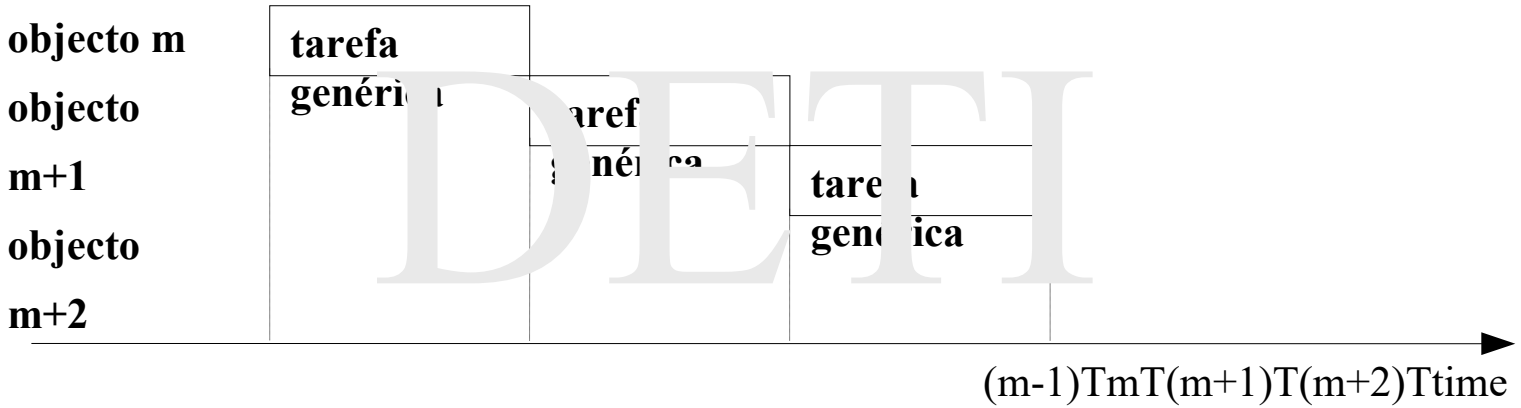
O conceito de canalização foi inventado no contexto da indústria automóvel pela Ford Motor Company no início dos anos 1900, inspirado pelos procedimentos operacionais dos matadouros de gado de Chicago. A primeira fábrica que introduziu uma linha de montagem móvel totalmente desenvolvida, iniciou as operações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

O objectivo de uma linha de montagem móvel era acelerar o processo de construção de um automóvel a partir das suas peças e, consequentemente, baixar o seu preço de mercado e torná-lo disponível a um público de consumidores muito mais vasto. De facto, o tempo de montagem de um carro passou de cerca de 12h e 30m, antes da introdução da linha de montagem em movimento, para apenas 1h e 33m depois.

O impacto das linhas de montagem móveis foi tão grande para a indústria automóvel em geral que a Ford Motor Company produzia 50% de todos os automóveis vendidos nos EUA e cerca de 40% dos automóveis vendidos na Commonwealth Britânica em 1920.

O que é pipelinagem? - 5

versão não encurtada

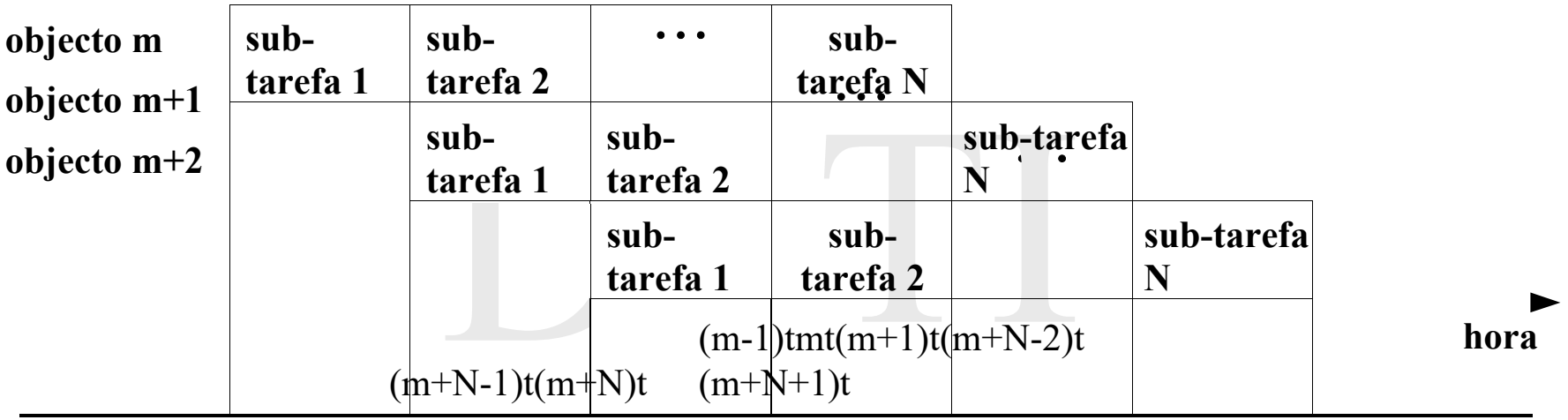


$$\text{produção} = \frac{1}{T}$$

tempo de execução de um fluxo de objectos M= $M \cdot T$

O que é pipelinagem? - 6

Versão encanada em *fase N*



t_n, com $n = 1,2, \cdots,$ tempo de *execução N* para sub-tarefa *n*

produção $= \frac{1}{t}$ onde $t = max\,(t_1, t_2, \cdots, t_N)$

tempo de execução de um fluxo de objectos M $= (N - 1 + M) \cdot t$

O que é pipelinagem? - 7

A *rapidez* obtida com a execução de uma conduta em fase N sobre a execução não compactada da mesma tarefa é expressa por

$$\text{acelerar}_{\substack{\text{Gasoduto em}\\ \text{fase N}}} = \frac{\text{tempo de execução do tempo de execução não encanada}}{\text{la implementação encanada em fase N}} = \frac{M \cdot T}{(N - 1 + M) \cdot t} = \frac{M \cdot N \cdot t^*}{(N - 1 + M) \cdot t} \approx N \cdot \frac{t^*}{t} ,$$

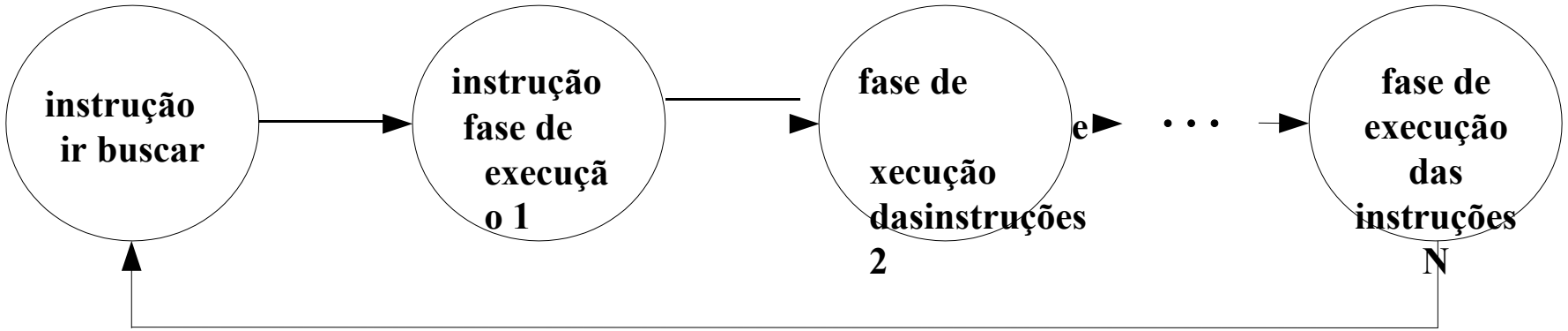
onde $T = M \cdot t^*$ e $N << M$.

Idealmente, se as fases da tubagem estiverem quase perfeitamente equilibradas e as despesas gerais envolvidas na tubagem forem insignificantes, então o rácio t*/t aproxima-se da unidade e a velocidade é aproximadamente igual ao número de fases utilizadas na divisão da tarefa original em subtarefas.

Paralelismo ao nível das instruções

Todos os processadores desde 1985 adoptaram a canalização como meio de sobreposição da execução das instruções e de melhorar o desempenho. Esta potencial sobreposição de instruções durante a sua própria execução é denominada *paralelismo a nível de instruções* (ILP) porque as instruções são, em certo sentido, processadas em paralelo através da decomposição da actividade que tem lugar.

Uma abordagem comum para o fazer é a decomposição da *execução* da *instrução* fase do ciclo do processador em várias fases.



Conjunto simplificado de instruções RISC - 9

Parte da arquitectura central de um típico RISC (Reduced Instruction Set Computer), MIPS64, vai ser utilizada para ilustrar os conceitos básicos de canalização.

As arquitecturas RISC são caracterizadas por algumas propriedades chave, que simplificam dramaticamente a sua implementação

- as únicas operações que afectam a memória são as instruções de *carga* e *armazenamento* que movem dados da memória para um registo do banco de registo, ou de um registo do banco de registo para a memória; as instruções que transferem menos do que o conteúdo de um registo completo também estão frequentemente disponíveis
- as únicas operações sobre dados são as instruções *aritméticas* e *lógicas* que se aplicam aos dados sobre registos do banco de registos; em princípio, alteram todo o registo
- os formatos de instrução são em número reduzido, tendo tipicamente o mesmo tamanho.

Estas propriedades também conduzem a simplificações dramáticas na

Conjunto simplificado de instruções RISC - 10

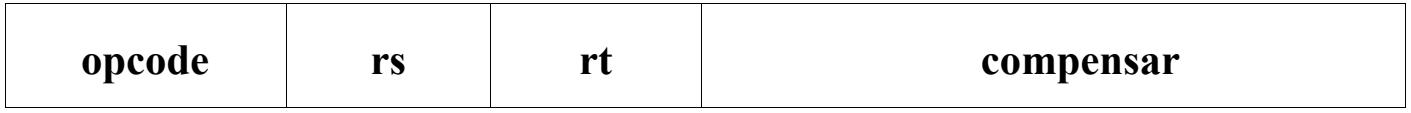
implementação de condutas, que é uma das razões pelas quais os conjuntos de instruções são tão concebidos.

Conjunto simplificado de instruções RISC - 11

Segue-se uma breve descrição da versão de 64 bits do conjunto de instruções MIPS. Todas as instruções são de 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 alargadas de 64 bits são geralmente denotadas por ter um `D` no início, ou no fim da mnemónica. O banco de registo de uso geral contém 32 registos de 64 bits, dos quais o *registo* 0 é apenas de leitura e tem valor zero.

O conjunto de instruções principais tem três classes de instruções

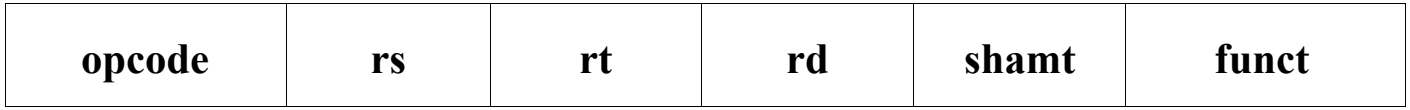
- *instruções de carregamento e armazenamento* - estas instruções tomam como operandos uma fonte de registo, a *base cadastral*, e um campo imediato de 16 bits, o *offset*; a soma do conteúdo da base cadastral e o offset alargado é utilizada como endereço de memória, o *endereço efectivo*; as instruções LD e SD carregam ou armazenam todo o conteúdo do registo de 64 bits



Instrução de tipo I

Conjunto simplificado de instruções RISC - 12

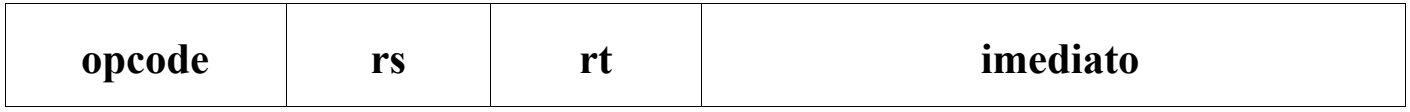
• *instruções aritméticas / lógicas* - estas instruções tomam ou dois registos ou um registo e um valor imediato de sinal estendido, operam sobre eles e armazenam o resultado noutro registo; as instruções *aritméticas* típicas incluem adição (`DADD`) e subtracção (`DSUB`) para as extensões de 64 bits; as instruções *lógicas*, contudo, não diferenciam entre extensões de 32 e 64 bits; as versões *imediatas* utilizam a mesma mnemónica com sufixo `I`; existem formas assinadas e não assinadas das instruções *aritméticas:* os formulários não assinados não geram transbordamento excepto - iões e têm um sufixo `U` no final (`DADDU`, `DSUBU`, `DADDIU`)



1050

31252015

Instrução do tipo R



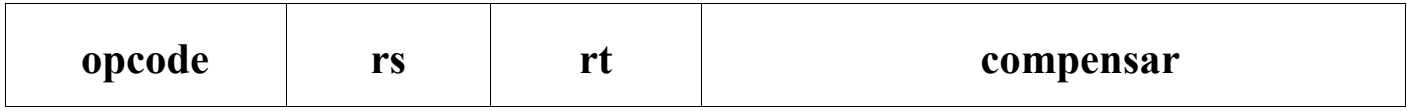
312520150

Instrução de tipo I

Conjunto simplificado de instruções RISC - 13

Conjunto simplificado de instruções RISC - 14

- *instruções de ramificação* - estas instruções são modificações condicionais da sequência estrita da execução da instrução; a *condição de* ramificação é tipicamente especificada ou por um conjunto de *bits de condição*, também chamado *código de condição*, um conjunto limitado de comparações entre um par de registos (*maior*, *maior ou igual*, *menor*, *menor ou igual*, *igual* e *não igual*) ou entre um registo e zero (*igual* e *não igual*); o *destino da* ramificação é obtido pela adição de um offset de sinal estendido, deslocado à esquerda por dois bits para fazer um offset de palavra, ao conteúdo actual do *contador do programa* (PC).



Instrução de tipo I

Implementação não encurtada de um conjunto de instruções RISC - 15

É apresentada uma simples implementação não encenada de parte do conjunto de instruções centrais MIPS64 e será utilizada como um quadro para compreender plenamente o funcionamento interno das condutas. Não é a implementação mais económica ou de maior desempenho, mas foi concebida para conduzir naturalmente à tubagem.

A implementação do conjunto de instruções requer a introdução de vários registos temporários que não fazem parte da arquitectura; eles são introduzidos apenas para simplificar a canalização. Esta implementação centrar-se-á apenas no subconjunto inteiro do conjunto de instruções centrais MIPS64 que consiste na carga/armazém, na ALU inteira e nas operações das sucursais. Serão implementados no máximo em 5 ciclos de relógio.

A operação em cada ciclo do relógio é a seguinte

1. *Ciclo de recolha de instruções* (FI)

   O conteúdo do contador de programas (PC) é utilizado como endereço para ir buscar a próxima instrução à memória, o seu valor é armazenado no registo de instruções (IR). O PC é actualizado a seguir para apontar a instrução seguinte, adicionando 4 ao conteúdo actual.

Implementação não encurtada de um conjunto de instruções RISC - 16

2. *Ciclo de descodificação / registo fetch de instruções* (ID)

   A instrução é descodificada levando às operações

   - o conteúdo dos registos correspondentes aos especificadores das fontes de registo são lidos a partir do banco de registo
   - o teste de igualdade é realizado sobre o conteúdo dos registos à medida que são lidos, para serem posteriormente utilizados num possível ramo
   - o campo de compensação é estendido por sinal no caso de ser necessário
   - o possível endereço alvo do ramo é calculado adicionando o conteúdo incremental do PC ao campo de 2 bits de deslocamento à esquerda do sinal estendido.

   A instrução do *ramo* é completada no final desta fase, armazenando o endereço alvo do ramo no PC se a condição de teste se tornar verdadeira, ou deixando-a inalterada se a condição de teste se tornar falsa.

Implementação não encurtada de um conjunto de instruções RISC - 17

3. *Execução / ciclo de endereço efectivo* (EX)

   A ALU toma os operandos computados no ciclo anterior e, dependendo do tipo de instrução, executa uma das acções abaixo

   - *referência da memória* - a base cadastral e o offset são somados para formar o endereço efectivo
   - *instrução register-to-register* ALU - a operação especificada pelo opcode é realizada sobre os valores lidos no banco de registo
   - *instrução de registo-imediata* ALU - a operação especificada pelo opcode é realizada no primeiro valor lido do banco de registo e o sinal- valor imediato alargado.

Implementação não encurtada de um conjunto de instruções RISC - 18

4. *Acesso à memória* (MEM)

   Quando a instrução é uma *carga*, o endereço efectivo calculado no ciclo anterior é o endereço do local da memória cujos dados devem ser lidos; quando a instrução é um *armazém*, o segundo valor lido do banco de registo deve ser escrito no local da memória cujo endereço é o endereço efectivo.
   A instrução da *loja* é concluída no final desta fase.

5. *Ciclo de retorno de escrita* (WB)

   O resultado de uma instrução de um dos tipos *registo de registo*, *registo imediato*, ou *carga*, é escrito num registo do banco de registo.

Implementação não encurtada de um conjunto de instruções RISC - 19

Nesta implementação, as instruções do *ramo* são executadas em 2 ciclos de relógio, as instruções do *armazém* em 4 ciclos de relógio e todas as outras instruções em 5 ciclos de relógio. Assumindo que um dado referencial tem uma frequência de filial de 12% e uma frequência de filial de 10%, qual é o IPC médio para a execução desta aplicação na implementação não encurtada que acaba de ser descrita?

$$\text{CPI global} = \sum_i \frac{\text{contagem de instruções}_i}{\text{contagem de instruções}} \cdot \text{CPI}_{i=}$$

$$= 0.12 \cdot 2 + 0.10 \cdot 4 + 0.78 \cdot 5 = 4.54.$$

Gasoduto clássico de 5 fases para um processador RISC - 20

A implementação não pipelinada pode ser convertida numa implementação pipelinada - com quase nenhuma mudança estrutural, obtendo uma nova instrução a cada ciclo de relógio.

	Número do relógio								
Número de instrução	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***
m	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 relógio a completar, durante cada ciclo o hardware estará a processar alguma parte de cinco instruções consecutivas.

Gasoduto clássico de 5 fases para um processador RISC - 21

Contudo, para que esta transformação funcione, é preciso determinar o que acontece em cada ciclo de relógio do processador e certificar-se de que não há duas operações a utilizar o mesmo recurso de caminho de dados ao mesmo tempo, ou seja, é preciso verificar cuidadosamente se a sobreposição de instruções na tubagem não está a causar um tal conflito.

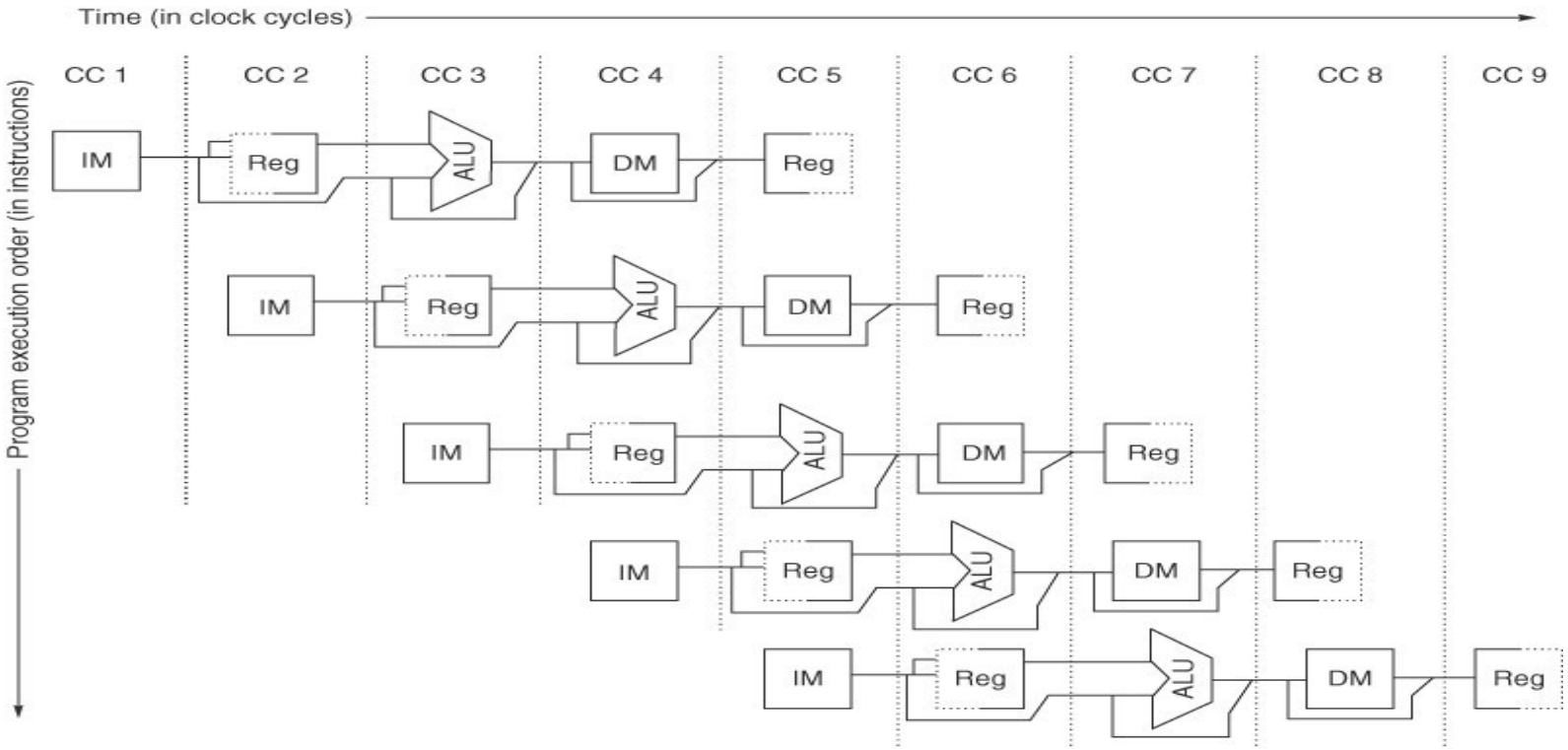
O número de detalhes a considerar é directamente proporcional ao número de fases da tubagem e ao número de instruções que se encontram em execução sobreposta. Por isso, é importante conceber uma representação gráfica da execução que torne visível o que está a acontecer.

Uma forma popular de o fazer é descrever o fluxo de operações como uma série de recursos de percurso de dados em uso nas diferentes fases do gasoduto.

Gasoduto clássico de 5 fases para um processador RISC - 22

Pipeline visto como uma série de recursos de percurso de dados deslocados no tempo

Fonte: Arquitectura de Computadores: Uma Abordagem Quantitativa



CC - ciclo do relógio IM - memória de instruções
DM - memória de

Gasoduto clássico de 5 fases para um processador RISC - 23

dados Reg - banco de registo ALU - unidade aritmética/lógica

Gasoduto clássico de 5 fases para um processador RISC - 24

Algumas observações estão em ordem

- há necessidade de *instruções* e *memórias de dados* separadas, o que significa a implementação de instruções e caches de dados separados para eliminar o conflito de uma única memória que surgiria entre as fases de *instrução fetch* (IF) e de *acesso à memória de dados* (MEM)
- o *banco de registo* é acedido em duas fases: para a leitura, na fase de *descodificação de instruções* / *registo fetch* (ID), e para a escrita, na fase de *write-back* (WB); é necessário efectuar duas leituras e uma escrita em cada ciclo do relógio - para lidar com a leitura e escrita no mesmo registo, a escrita do registo é efectuada na primeira metade do ciclo do relógio e a leitura do registo na segunda metade
- para ir buscar uma instrução a cada ciclo do relógio, o *contador de programa* (PC) deve ser actualizado a cada ciclo do relógio na fase de *busca de instruções* (IF) em preparação para a instrução seguinte; além disso, o endereço alvo potencial do ramo deve ser calculado durante a fase de *descodificação/registo da instrução* (ID); ainda assim, uma vez que o ramo, quando a condição do teste se torna verdadeira, também muda o PC na fase de ID, há um conflito que será tratado mais tarde.

Gasoduto clássico de 5 fases para um processador RISC - 25

Também se deve assegurar que as instruções nas diferentes fases do gasoduto não interferem umas com as outras. A solução é acrescentar registos para armazenamento temporário de dados relevantes entre fases sucessivas, para que no início de um ciclo de relógio todos os resultados de uma determinada fase sejam armazenados nos registos que são utilizados como entradas para a fase seguinte. Os registos são designados para as duas fases que são separadas por esse registo: por exemplo, os registos entre as fases IF e ID são designados por IF/ID.

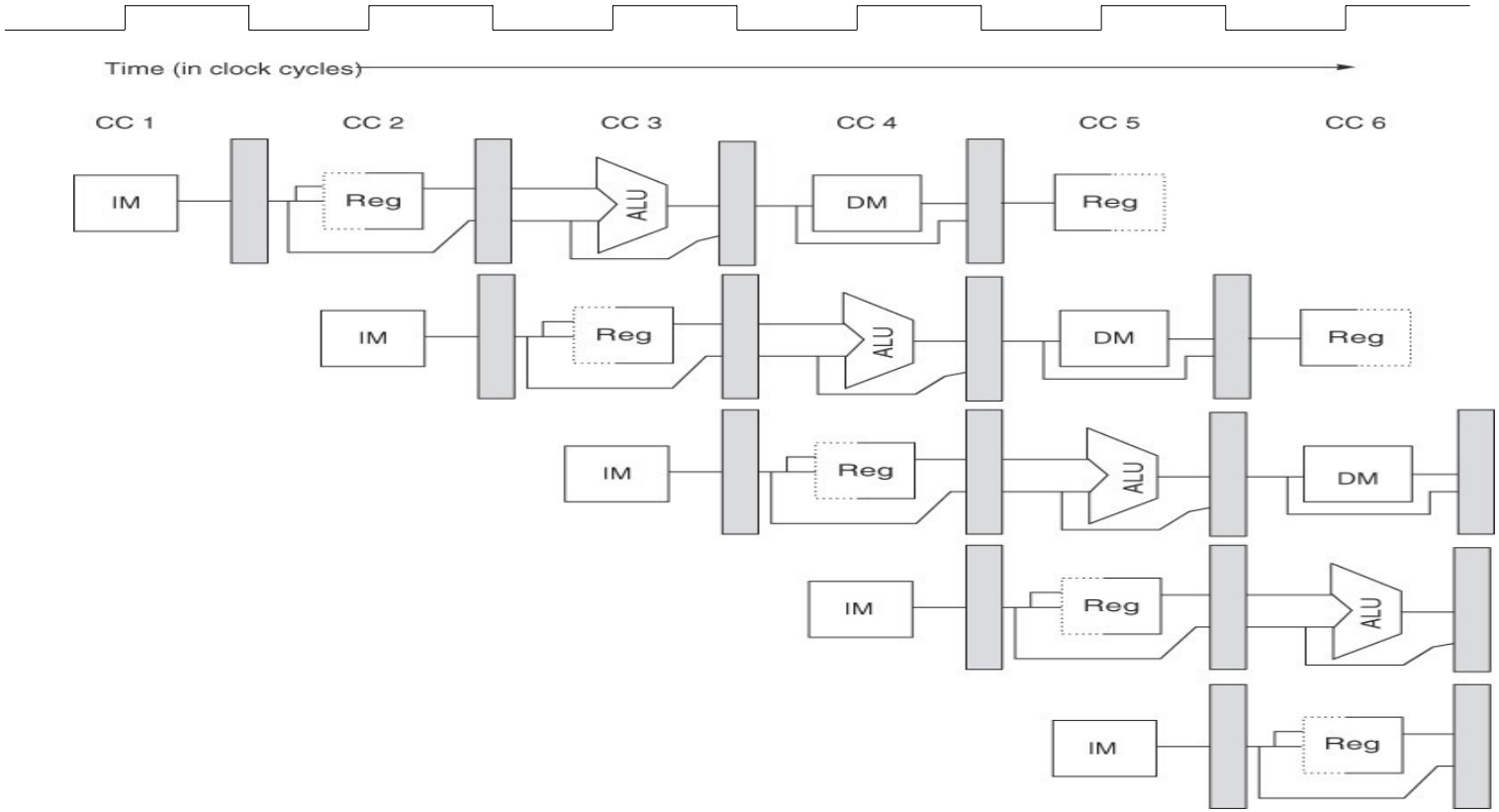
Note-se que não são necessários registos no final da fase WB. Todas as instruções devem actualizar de alguma forma particular o estado do computador (o banco de registo, a memória ou o contador de programas), pelo que um registo separado no final do gasoduto seria redundante para o estado que está a ser actualizado.

O próprio contador do programa pode ser pensado como um registo do gasoduto: um que alimenta a fase IF do gasoduto. Ao contrário dos outros registos, porém, o contador do programa faz parte do estado arquitectónico visível; o seu conteúdo deve ser guardado quando é feita uma excepção, enquanto que o conteúdo dos outros registos do gasoduto é descartado.

Gasoduto clássico de 5 fases para um processador RISC - 26

Pipeline com registos entre etapas sucessivas para armazenamento temporário

Fonte: Adaptado de Computer Architecture: Uma Abordagem Quantitativa



Gasoduto clássico de 5 fases para um processador RISC - 27

A canalização aumenta o rendimento da instrução, mas não reduz o tempo de execução de uma instrução individual. De facto, aumenta ligeiramente devido às despesas gerais necessárias para controlar a tubagem. Note-se que o aumento do rendimento da instrução significa que um programa corre mais rapidamente e tem um tempo de execução mais baixo, mesmo que nenhuma instrução individual corra mais rapidamente.

Considerar o processador não compactado anteriormente descrito. Assumir que tem um ciclo de relógio de 1ns e que corre um ponto de referência com uma frequência de filial de 20% e uma frequência de loja de 10%. Suponhamos que, devido à sobrecarga para controlar a tubagem, o ciclo do relógio da versão pipelinada do processador é de 1,2ns. Qual é a velocidade ideal na taxa de execução de instruções obtida com a pipelinagem?

tempo médio de execução timenonpipelined= global CPInonpipelined ·clock

tempo de ciclo =

= (0,2 ·2 + 0,1· 4 + 0,7 ·5)·1.0 = 4,3 ns

s peed up =

timenonpipelined de execução de instante médio

$$\frac{4.3}{1.2} = 3.6$$

timepelinização de execução de instintos médios

Gasoduto clássico de 5 fases para um processador RISC - 28

Principais obstáculos da canalização - 29

A canalização funcionaria como descrito se as instruções fossem independentes umas das outras. Na realidade, isto não é assim. Existem situações, chamadas *perigos*, que impedem a próxima instrução no fluxo a ser executada durante o seu ciclo de relógio. Assim, reduzindo o desempenho a partir da velocidade ideal ganha por pipelinagem.

Há três classes de perigos

- *riscos estruturais* - surgem de conflitos de recursos quando o hardware não pode 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 na conduta
- *riscos de controlo* - surgem da canalização de instruções de ramo e outras instruções que afectam o PC.

Principais obstáculos da canalização - 30

Os perigos nos oleodutos podem exigir a *paragem da* conduta, ou seja, para evitar o perigo é muitas vezes necessário que algumas instruções na conduta sejam forçadas a permanecer na mesma fase, enquanto outras são autorizadas a prosseguir para a fase seguinte. Quando uma instrução é bloqueada, todas as instruções que foram obtidas mais tarde do que a instrução bloqueada, também são bloqueadas, e todas as instruções obtidas mais cedo devem continuar, pois caso contrário o perigo nunca desapareceria. Como resultado, não são buscadas novas instruções durante uma paragem e ocorre um atraso na execução.

Desempenho de condutas com bancas

$$\text{speed up} = \frac{\text{média de execução timenonpipelined}}{\text{timepelinização de execução de instintos médios}}$$

$$= \frac{\text{CPInonpipelined em geral}}{\text{CPIpipelined global}} \cdot \frac{\text{ciclo de relógio timenonpipelined}}{\text{ciclo de relógio timepelined}} =$$

$$= \frac{\text{CPInonpipelined em geral}}{1 + \text{ciclos de paragem da conduta por instrução}^{\text{pipelinado}}} \cdot \frac{\text{ciclo de relógio timenonpipelined}}{\text{ciclo de relógio timepelined}}.$$

Riscos estruturais - 32

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

O exemplo mais comum de riscos estruturais surge quando alguma unidade funcional não está totalmente canalizada. Depois, uma sequência de instruções em que algumas delas utilizam essa unidade, não pode prosseguir através das fases da tubagem à taxa de uma instrução por ciclo de relógio. Outra causa comum é quando algum recurso não é suficientemente replicado para permitir que todas as combinações de instruções sejam executadas à taxa nominal.

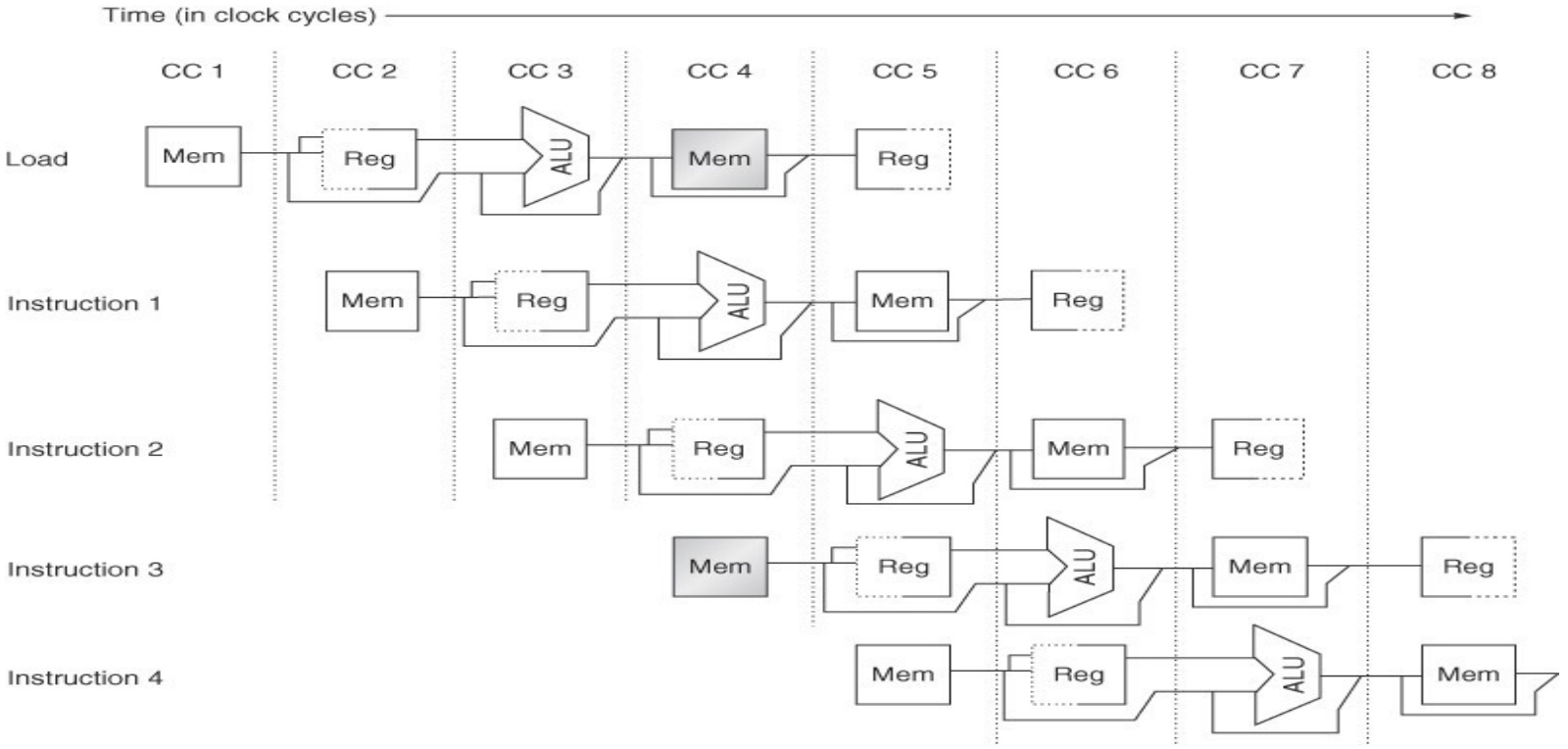
Quando este tipo de perigo for encontrado, o gasoduto irá *empatar* a última instrução a seguir até que a unidade necessária esteja disponível. A barraca gerará aquilo a que normalmente se chama uma *bolha*, uma vez que esta condição pode ser pensada para fluir através das fases da tubagem ocupando espaço, mas não produzindo qualquer trabalho útil.

Riscos estruturais - 33

Alguns processadores têm uma única memória para aceder tanto às instruções como aos dados. Isto gerará um potencial conflito entre as fases de instrução (IF) e de acesso à memória de dados (MEM).

Processador com apenas uma porta de memória

Fonte: Arquitectura de Computadores: Uma Abordagem Quantitativa



Riscos estruturais - 34

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

Fonte: Adaptado de Computer Architecture: Uma Abordagem Quantitativa

	Número do relógio									
Instrução	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***
carga	SE	ID	EX	MEM	WB					
m+1		SE	ID	EX	MEM	WB				
m+2			SE	ID	EX	MEM	WB			
m+3				***estábulo***	SE	ID	EX	MEM	WB	
m+4						SE	ID	EX	MEM	WB
m+5							SE	ID	EX	MEM
m+6								SE	ID	EX

	Número do relógio									
Instrução	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***
carga	SE	ID	EX	MEM	WB					
m+1		SE	ID	EX	MEM	WB				
m+2			SE	ID	EX	MEM	WB			
bolha				*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 - 35

Suponhamos que as instruções de referência de dados (*carga* e *armazenamento*) constituem 40% da mistura de instruções para um dado parâmetro de referência e que o IPC ideal para o processador encanado, ignorando o risco estrutural, é 1. Suponhamos que o processador com o risco estrutural tem uma taxa de relógio 1,05 vezes superior à do processador sem o risco. Desconsiderando quaisquer outras perdas de desempenho, qual o processador que executa o padrão de referência mais rapidamente?

$$\text{tempo médio de execução instante}_{\text{sem nevoeiro}} = \text{IPC global}_{\text{sem nevoeiro}} \cdot \text{clock tempo de ciclo}_{\text{ideal}} =$$

$$= 1 \cdot \text{clock ciclo timeideal}$$

$$\text{tempo médio de execução}_{\text{com haz}} = \text{CPI global com}_{\text{haz}} \cdot \text{clock tempo de ciclo}_{\text{ideal}} =$$

$$= (1 + 0.4 \cdot 1) \cdot \frac{\text{ciclo do relógio timeideal}}{1.05} \approx$$

$$\approx 1.3 \cdot \text{clock ciclo timeideal.}$$

Riscos estruturais - 36

Se todos os factores forem iguais, um processador sem riscos estruturais terá sempre um IPC mais baixo. Por que razão, então, um projectista permitiria riscos estruturais?

A principal razão é reduzir o custo do processador, uma vez que a canalização de todas as unidades funcionais, e/ou a sua reprodução, pode revelar-se demasiado cara. Por exemplo, os processadores que requerem uma instrução e um acesso à cache de dados em cada ciclo do relógio necessitam do dobro da largura de banda total da memória. Da mesma forma, a concepção de um multiplicador de ponto flutuante totalmente encapsulado consome muitos portões e o espaço pode não estar disponível.

Se o risco estrutural for raro, pode não valer a pena o custo de o evitar.

Perigos dos dados - 37

Um efeito importante da canalização é alterar o timing relativo das instruções, sobrepondo a sua execução. Devido a isto, podem surgir riscos de dados. Os *perigos dos dados*, de facto, ocorrem quando a conduta força uma ordem real de operações de leitura/escrita a operandos diferentes dos percebidos através da execução sequencial das instruções num processador não encapsulado.

Considerar a execução do código em pipelina

```
DADDR1,R2,R3
DSUBR4,R1,R5
ANDR6,R1,R7
ORR8,R1,R9
XORR10,R1,R11
```

Perigos dos dados - 38

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

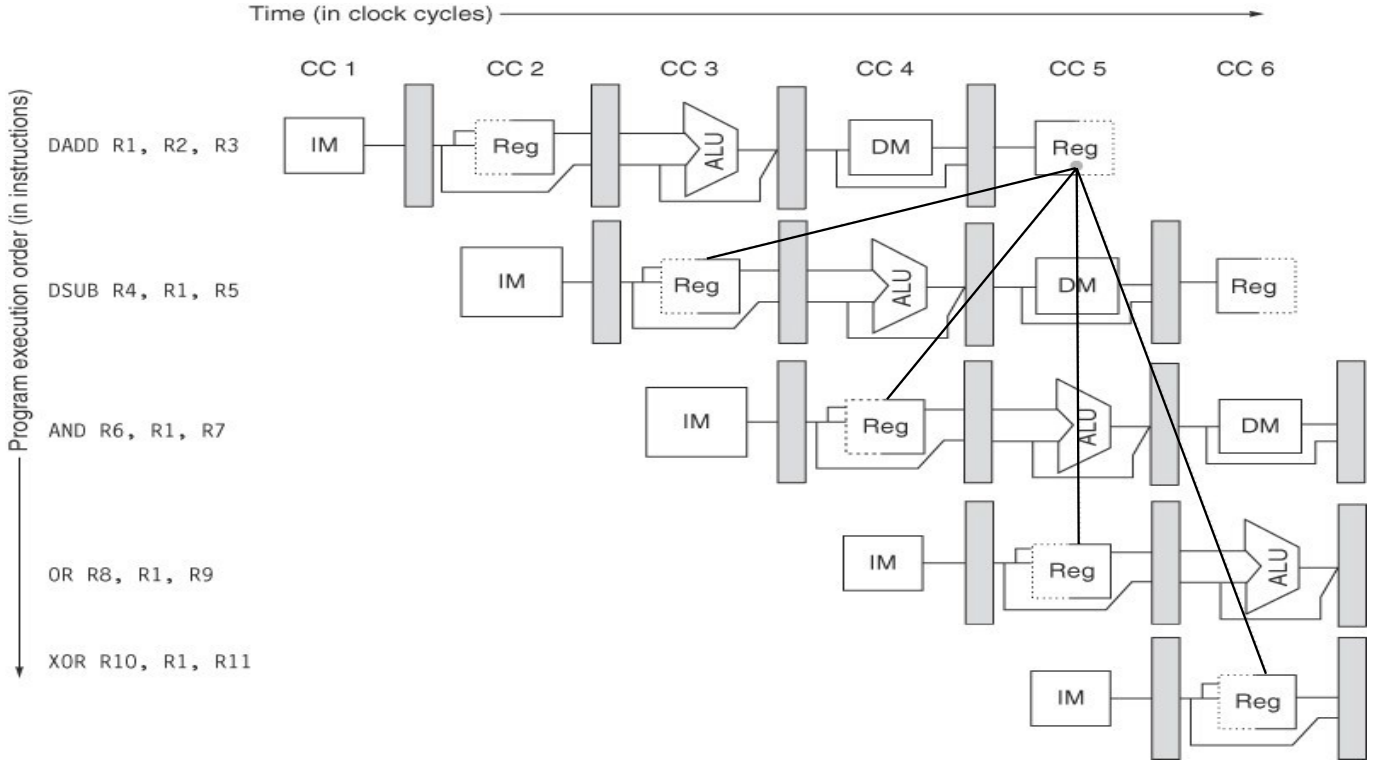
O DADD actualiza o valor de R1 na fase do BM, mas o DSUB lê R1 antes disso na sua fase de identificação. A menos que sejam tomadas precauções para o evitar, o DSUB lerá um valor errado. O valor lido não é sequer determinístico. Parece lógico assumir que o DSUB obterá sempre o valor atribuído a R1 por alguma instrução antes do DADD, mas este pode não ser o caso. Se for efectuada uma interrupção entre as instruções DADD e DSUB, a fase WB do DADD será concluída e o valor de R1 nesse momento será o resultado do DADD.

A instrução AND é também afectada por este perigo. A escrita de R1 não se completa até à primeira metade do ciclo 5 do relógio. Assim, AND que lê o registo na segunda metade do ciclo de relógio 4, receberá um resultado errado. As instruções OR e XOR, contudo, executam correctamente: a primeira lê R1 na segunda metade do ciclo de relógio 5, depois de a escrita ter tido lugar; a segunda lê R1 apenas no ciclo de relógio 6.

Perigos dos dados - 39

O efeito da instrução DADD sobre as instruções que se seguem

Fonte: Arquitectura de Computadores: Uma Abordagem Quantitativa



Perigos dos dados - 40

O problema pode ser resolvido através da utilização de uma técnica de hardware simples chamada *"forwarding"*, *"bypassing"*, ou *curto-circuito*. A principal percepção no *encaminhamento* é que o resultado não é realmente necessário por `DSUB`, ou por `AND`, até depois do `DADD` o produzir realmente. Se o resultado puder ser transferido do registo onde `DADD` o armazena para onde `DSUB` e `AND` precisa dele, então a introdução de bancas pode ser evitada.

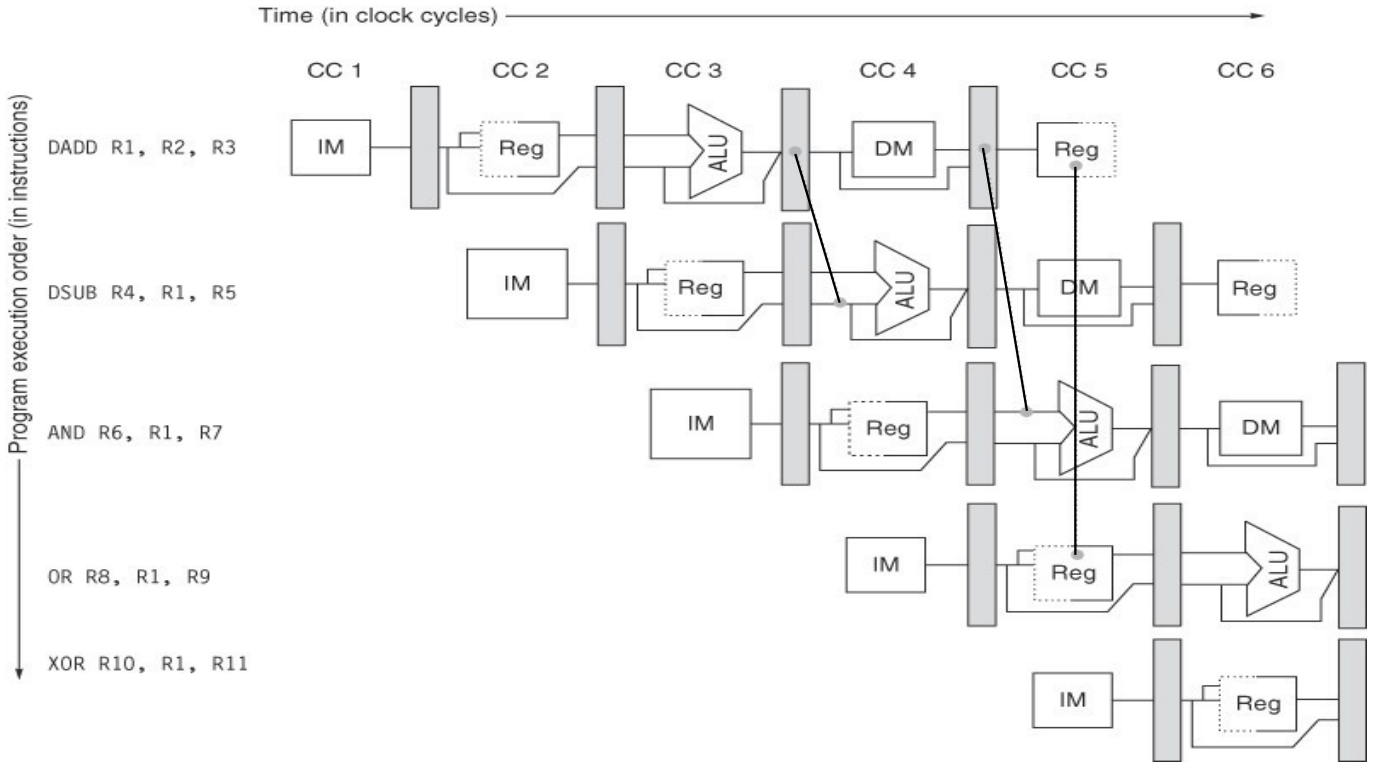
Usando esta observação, o *encaminhamento* funciona da seguinte forma

- o resultado da UAL dos registos dos gasodutos EX/MEM e MEM/WB é sempre alimentado com as entradas da UAL
- se o hardware de envio detectar que uma operação ALU anterior modificou o registo correspondente a uma fonte para a operação ALU actual, a lógica de controlo selecciona o resultado do envio como entrada da ALU em vez do valor lido no banco de registos.

Perigos dos dados - 41

O efeito da instrução DADD sobre as instruções que se seguem é resolvido através do reencaminhamento para evitar um risco de dados

Fonte: Arquitectura de Computadores: Uma Abordagem Quantitativa



Perigos dos dados - 42

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

Considerar a execução do código em pipelina

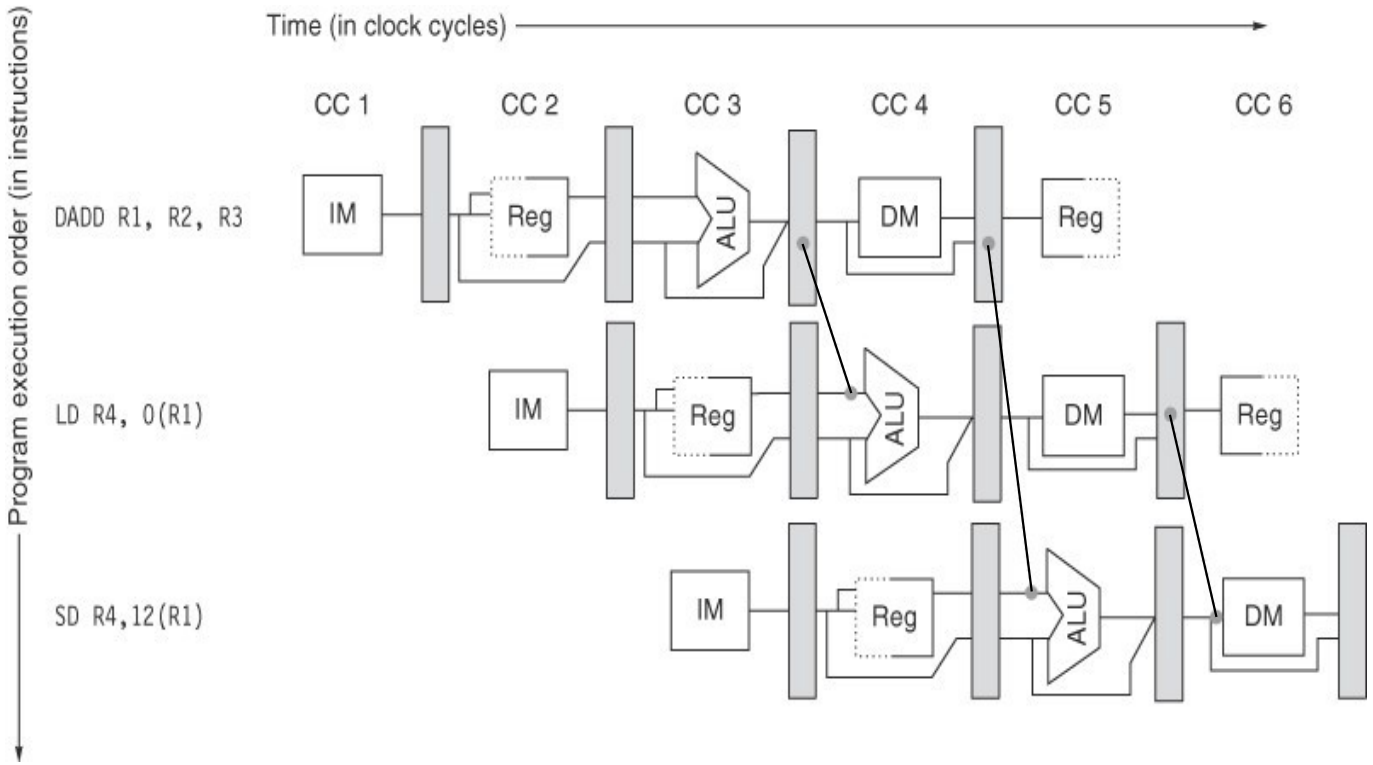
```
DADDR1,R2,R3
LDR4,0(R1)
SDR4,12(R1)
```

Para evitar uma paragem nesta sequência, é necessário reencaminhar os valores da saída da ALU e a saída da memória de dados dos registos da conduta para a ALU e as entradas da memória de dados.

Perigos dos dados - 43

Encaminhamento de operando exigido pela *loja* durante o MEM

Fonte: Arquitectura de Computadores: Uma Abordagem Quantitativa



Perigos dos dados - 44

Infelizmente, nem todos os perigos potenciais dos dados podem ser tratados através do reencaminhamento. Considerar a execução do código em pipelina

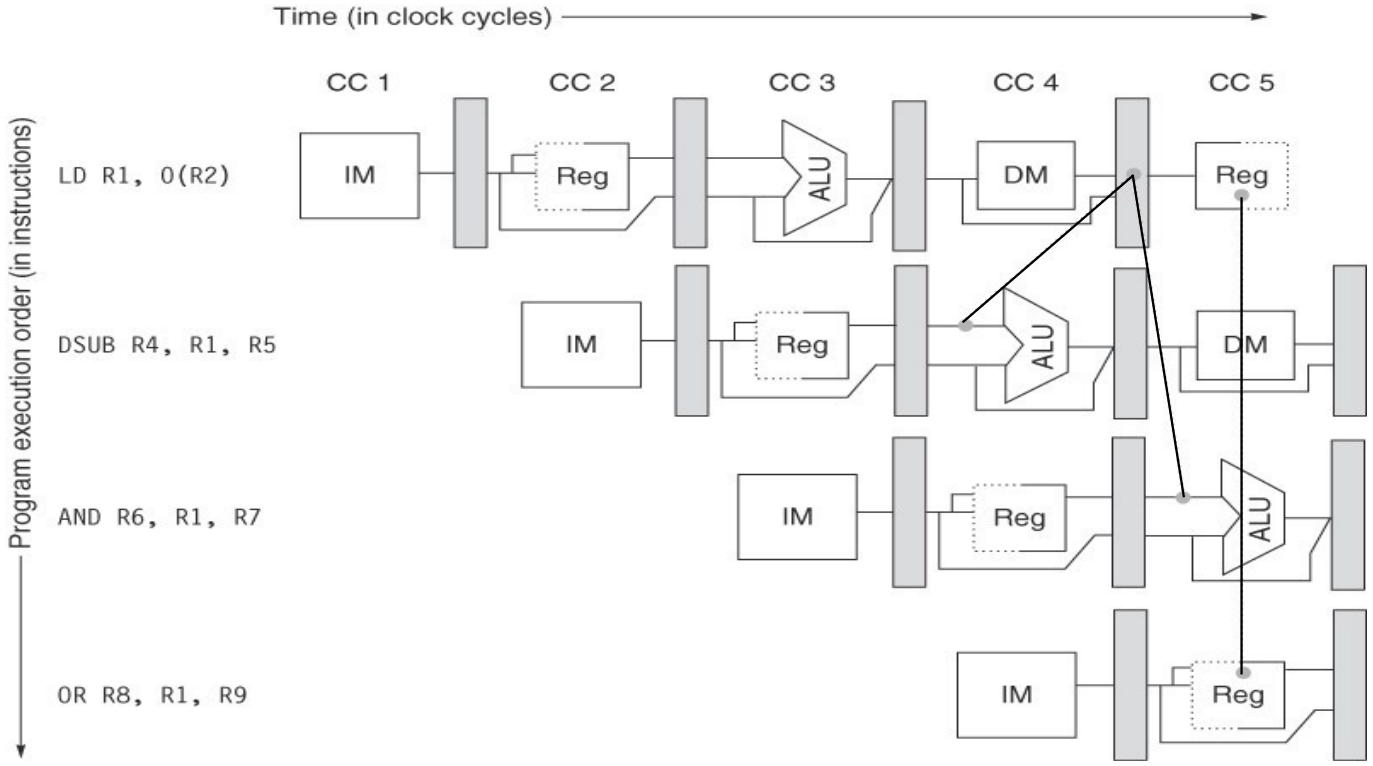
```
LDR1,0(R2)
DSUBR4,R1,R5
ANDR6,R1,R7
ORR8,R1,R9
```

A instrução `LD` não tem os dados até ao fim do ciclo do relógio 4, enquanto a instrução `DSUB` os exige 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 imediatamente encaminhado para o ALU para utilização pela instrução `AND`, que inicia 2 ciclos de relógio após `LD`. Da mesma forma, a instrução `OR` não tem qualquer problema, uma vez que obtém o valor através do ficheiro de registo.

Perigos dos dados - 45

Perigo de dados que não pode ser resolvido através de reencaminhamento

Fonte: Arquitectura de Computadores: Uma Abordagem Quantitativa



Perigos dos dados - 46

Para resolver o problema, é necessário adicionar hardware especial, chamado *encravamento de tubagem*, para preservar o padrão de execução correcto. Em geral, o *encravamento da conduta* detecta um perigo e paralisa a conduta até que o perigo desapareça. Neste caso, o *encravamento da conduta* bloqueia a conduta, começando com a instrução que pretende utilizar os dados até que a instrução da fonte os produza e os disponibilize.

	Número do relógio								
Instrução	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***
`LDR1,0(R2)`	SE	ID	EX	MEM	WB				
`DSUB R4,R1,R5`		SE	ID	***estáb ulo***	EX	MEM	WB		
`E R6,R1,R7`			SE	***estáb ulo***	ID	EX	MEM	WB	
`ORR8,R1,R9`				***estáb ulo***	SE	ID	EX	MEM	WB

Perigos de controlo - 47

Quando uma filial é executada, pode ou não alterar o PC. Diz-se que se uma instrução de sucursal mudar o PC para o seu endereço de destino, é uma sucursal *tomada*; se cair, é uma sucursal *não tomada.* Quando uma instrução é uma sucursal tomada, o PC não é alterado até ao fim da fase de identificação.

O método mais simples para lidar com os ramos é refazer o fetch da instrução que segue o ramo, uma vez detectada a instrução do ramo (durante a fase de identificação). O primeiro ciclo IF é essencialmente um stall, porque nunca realiza trabalho útil. Deve-se notar que se um ramo não for tocado, há uma penalização, a repetição da fase IF é desnecessária, uma vez que a instrução correcta foi de facto obtida.

	Número do relógio								
Instrução	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***
filial	SE	ID	EX	MEM	WB				
sucessor		SE	SE	ID	EX	MEM	WB		
sucessor+1				SE	ID	EX	MEM	WB	
sucessor+2					SE	ID	EX	MEM	WB

Perigos de controlo - 48

Uma alternativa de maior desempenho, designada por esquema de *junção prevista*, é tratar cada ramo como se *não tivesse sido executado*, permitindo simplesmente que o hardware continue como se a instrução do ramo 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. Neste caso, o gasoduto parece não estar a acontecer nada fora do normal. Se o ramal for *tomado*, porém, a instrução de busca é transformada numa instrução *não operacional* e a fase de busca é reiniciada no endereço alvo.

	Número do relógio							
Instrução	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***
ramo não ocupado	SE	ID	EX	MEM	WB			
sucessor		SE	ID	EX	MEM	WB		
sucessor+1			SE	ID	EX	MEM	WB	
sucessor+2				SE	ID	EX	MEM	WB

	Número do relógio							
Instrução	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***
tomada ramo	SE	ID	EX	MEM	WB			
sucessor		SE	***inacti***	***inacti***	***inacti***	***inacti***		

Perigos de controlo - 49

			vo	***vo***	***vo***	***vo***		
alvo			SE	ID	EX	MEM	WB	
alvo+1				SE	ID	EX	MEM	WB

Perigos de controlo - 50

Pode-se pensar o contrário e tratar cada ramo como se fosse *tomado*. Este esquema é obviamente chamado o esquema *previsto.* Só é relevante para processadores que utilizam códigos de condição implícitos ou mais poderosos, e portanto mais lentos, condições do ramo - o endereço alvo do ramo é conhecido antes do resultado do ramo e, devido a isso, o esquema *predito* pode fazer sentido.

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

Perigos de controlo - 51

Um último esquema, chamado esquema de *filial atrasada*, foi muito utilizado nos primeiros processadores RISC. O ciclo de execução com um ramo atrasado de um é definido como

```
Se for tomada uma
instrução de ramo sucessor
sequencial do ramo alvo
```

O sucessor sequencial está na *ranhura de atraso do ramo*. Esta instrução é executada quer o ramo seja tomado ou não e não pode ser ela própria uma instrução do ramo.

	Número do relógio							
Instrução	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***
ramo não ocupado	SE	ID	EX	MEM	WB			
atraso do ramo		SE	ID	EX	MEM	WB		
sucessor			SE	ID	EX	MEM	WB	
sucessor+1				SE	ID	EX	MEM	WB

	Número do relógio							
Instrução	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***
tomada ramo	SE	ID	EX	MEM	WB			

Perigos de controlo

- 52

atraso do ramo		SE	ID	EX	MEM	WB		
alvo			SE	ID	EX	MEM	WB	
alvo+1				SE	ID	EX	MEM	WB

Perigos de controlo - 53

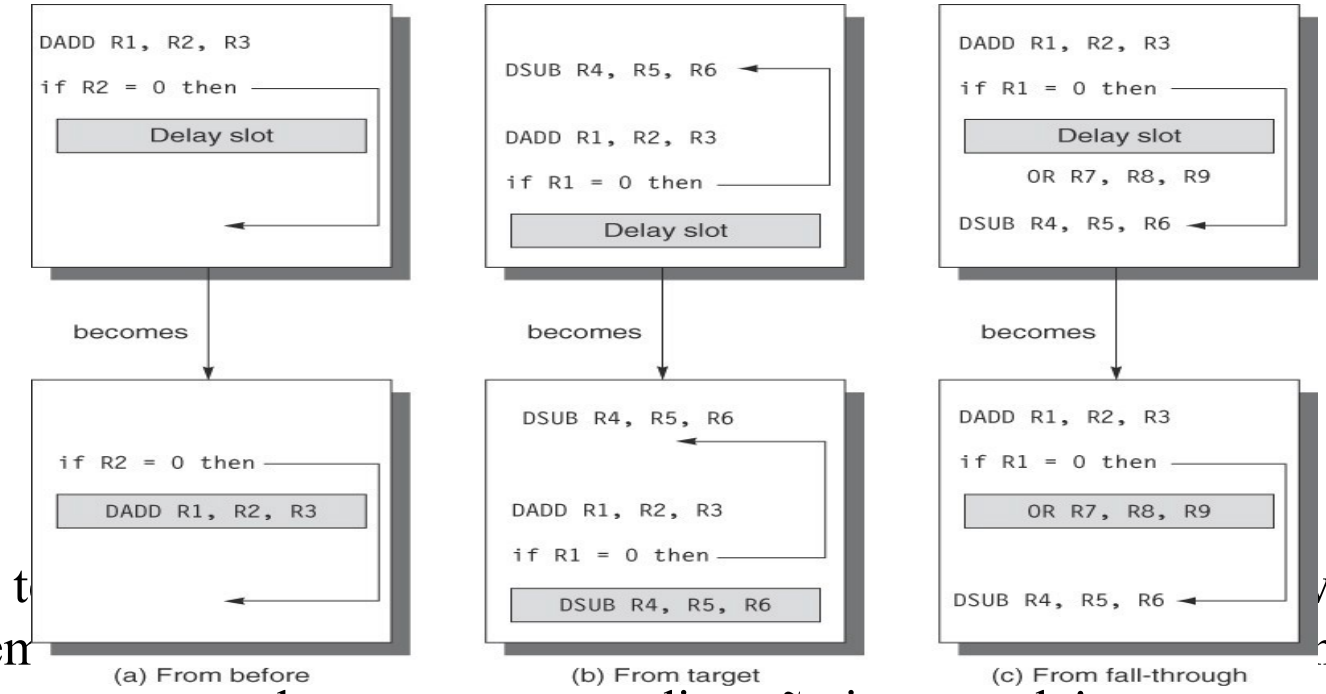
O trabalho do compilador é tornar válida e útil a instrução sequencial do sucessor. Várias alternativas são possíveis

- *de antes* - inserir na *ranhura de atraso do ramo* uma instrução supostamente executada antes da instrução do ramo e que seja independente dela (a situação *ideal* uma vez que não resultam efeitos secundários da sua aplicação)
- *do alvo* - inserir na *ranhura de atraso do ramo* a instrução de que o ramo, se *tomado*, aponta e faz com que o ramo aponte para o seu sucessor (note que quando o ramo *não* é *tomado*, é executada uma instrução extra, não pretendida originalmente)
- *de queda* - inserir na *ranhura de atraso do ramo* a instrução que será executada a seguir se o ramo *não for tomado* (então esta instrução ainda está a ser executada quando o ramo for *tomado*).

Perigos de controlo - 54

Optimizações do compilador de slots de retardamento de ramificação

Fonte: Arquitectura de Computadores: Uma Abordagem Quantitativa



Há que t veis se não implicaren ma correr incorrectamente quando o ramo toma a direcção inesperada!

Perigos de controlo
- 55

Para melhorar a capacidade do compilador de preencher a ranhura de atraso do ramo, a maioria dos processadores com ramos condicionais introduziram uma instrução de *cancelamento* ou *anulação do ramo.* Parece ser a habitual instrução de *ramificação atrasada.* Quando o ramo é *tomado*, a instrução na *ranhura de atraso do ramo* é executada; no entanto, quando o ramo *não* é *tomado*, a instrução na *ranhura de atraso do ramo* é transformada numa instrução de *não abertura* e nada acontece.

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

$$\text{speed up} = \frac{\text{CPInonpipelined}}{1 + \text{ciclos de paragem da conduta por instruções}} \cdot \frac{\text{ciclo de relógio}_{\text{conduta de}}}{\text{tempo de ciclo do relógio}_{\text{pipelined}}}$$

$$= \frac{\text{CPInonpipelined em geral}}{1 + (\text{frequência do ramo} \cdot \text{penalização}_{\text{d}} \text{ o } \text{r}} \cdot \frac{\text{ciclo de relógio}^{\text{o}}}{\text{timenonpipelined}}$$

amo)

Perigos de controlo - 56

pipelinado . tempo de ciclo do relógio pipelinado .

Perigos de controlo - 57

À medida que os oleodutos vão mais fundo e a penalização potencial dos ramos aumenta, os ramos atrasados e esquemas semelhantes são considerados insuficientes. É necessário ser mais agressivo na previsão de ramos.

O problema é abordado de duas maneiras diferentes

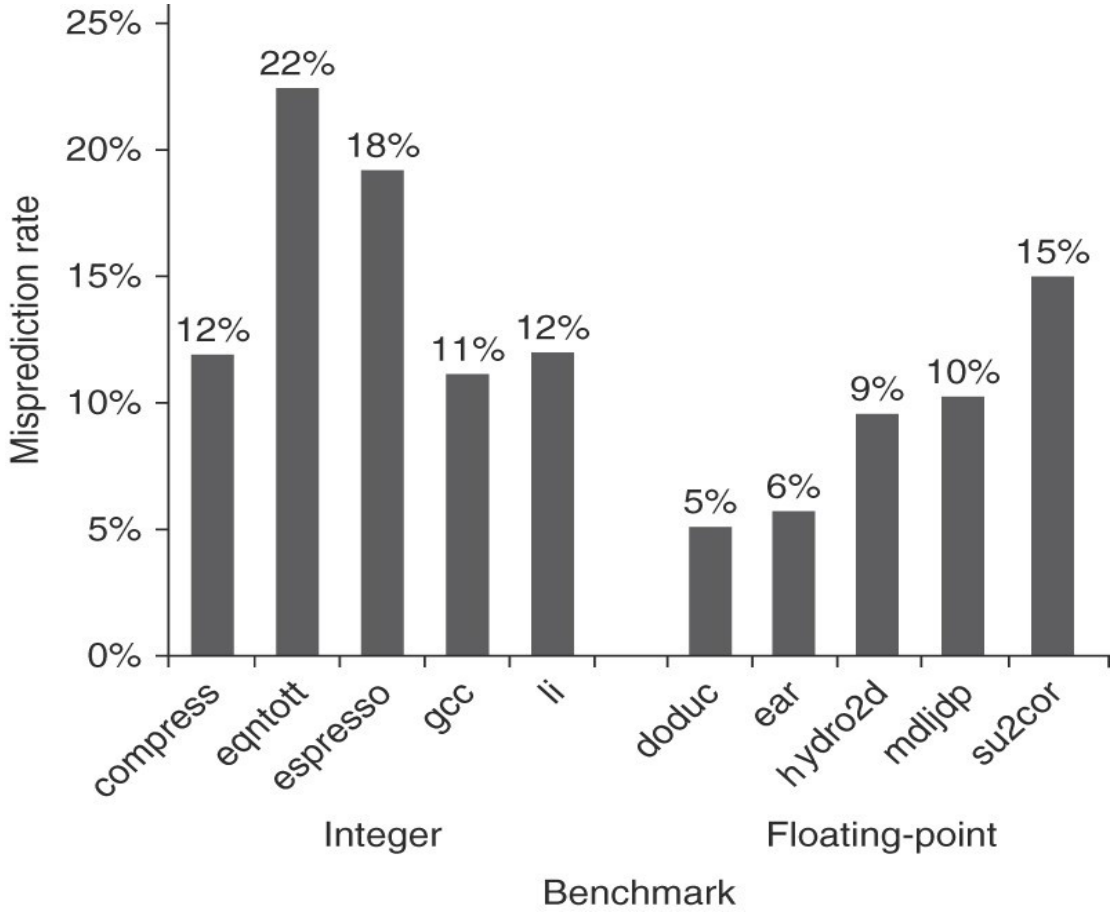
- *esquemas estáticos de baixo custo* - baseiam-se na informação disponível no momento da compilação e na elaboração de perfis de execuções anteriores do mesmo código; a observação chave que faz com que este esforço valha a pena é que o comportamento dos ramos é frequentemente distribuído de forma bimodal, ou seja, cada ramo em particular tende a ser frequentemente muito tendencialmente tendencialmente tendencialmente *tomado* ou *não tomado*
- *esquemas dinâmicos* - consistem em métodos para prever a direcção do ramo 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 só a sua frequência de ramificação é maior, mas também a direcção da ramificação é mais aleatória.

Perigos de controlo
- 58

Taxa de erro de previsão para SPEC92 usando um preditor baseado em perfil

Fonte: Arquitectura de Computadores: Uma Abordagem Quantitativa



Perigos de controlo - 59

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

Com um tal dispositivo, não se sabe realmente se a previsão está correcta - pode ter sido definida ou reiniciada por outra instrução de filial que partilha os mesmos bits de endereço de baixa ordem, ou agora o comportamento da filial pode ser oposto ao que era antes. Se se pensar um pouco sobre isto, não é determinante. Uma previsão é apenas uma dica sobre o futuro que se assume ser verdadeira e, por conseguinte, a obtenção de receitas na direcção prevista. Se mais tarde se provar que é falsa, a dica da previsão é complementada e o fetching é reiniciado.

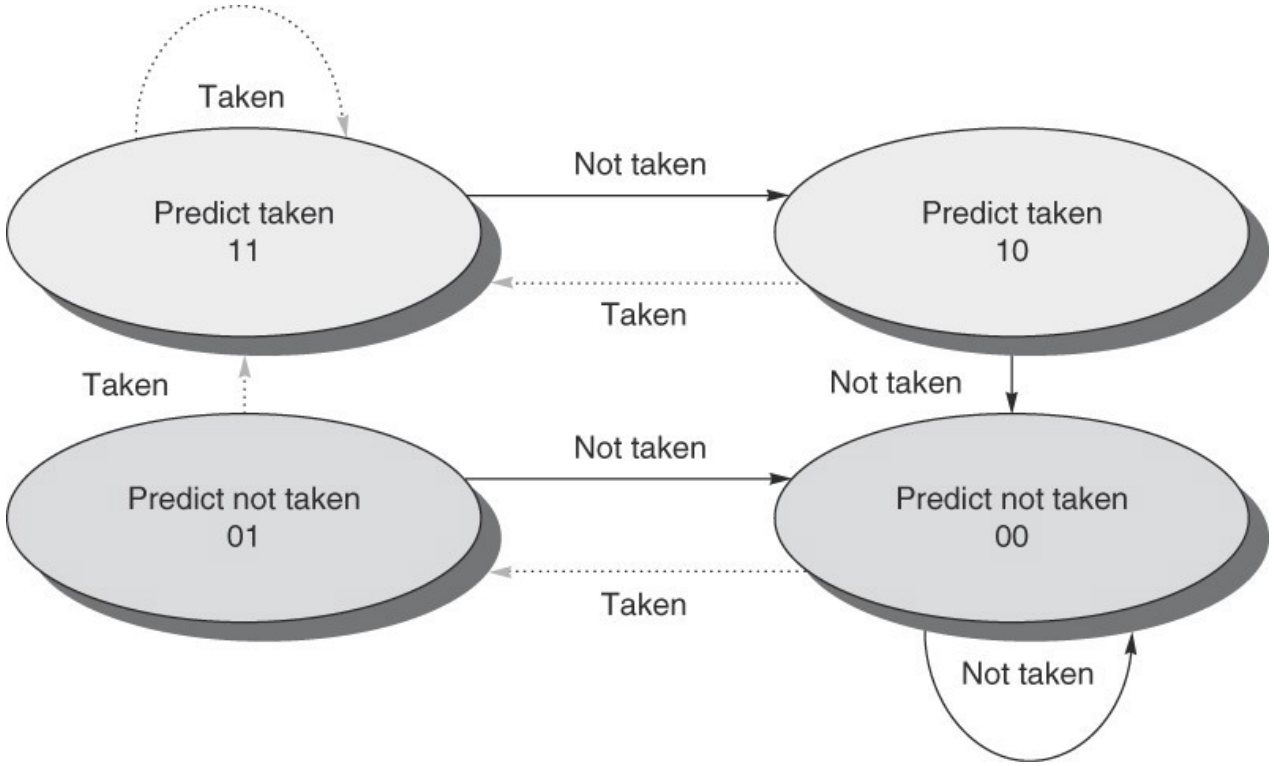
Este simples esquema de previsão de 1 bit tem uma falha de desempenho: mesmo quando o ramo é quase sempre tomado, é provável que preveja incorrectamente duas vezes, em vez de uma, quando não foi tomado. Isto pode ser remediado através da utilização de um esquema de previsão de 2 bits.

Perigos de controlo - 60

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

Esquema de previsão de 2 bits diagrama de estado

Fonte: Arquitectura de Computadores: Uma Abordagem Quantitativa

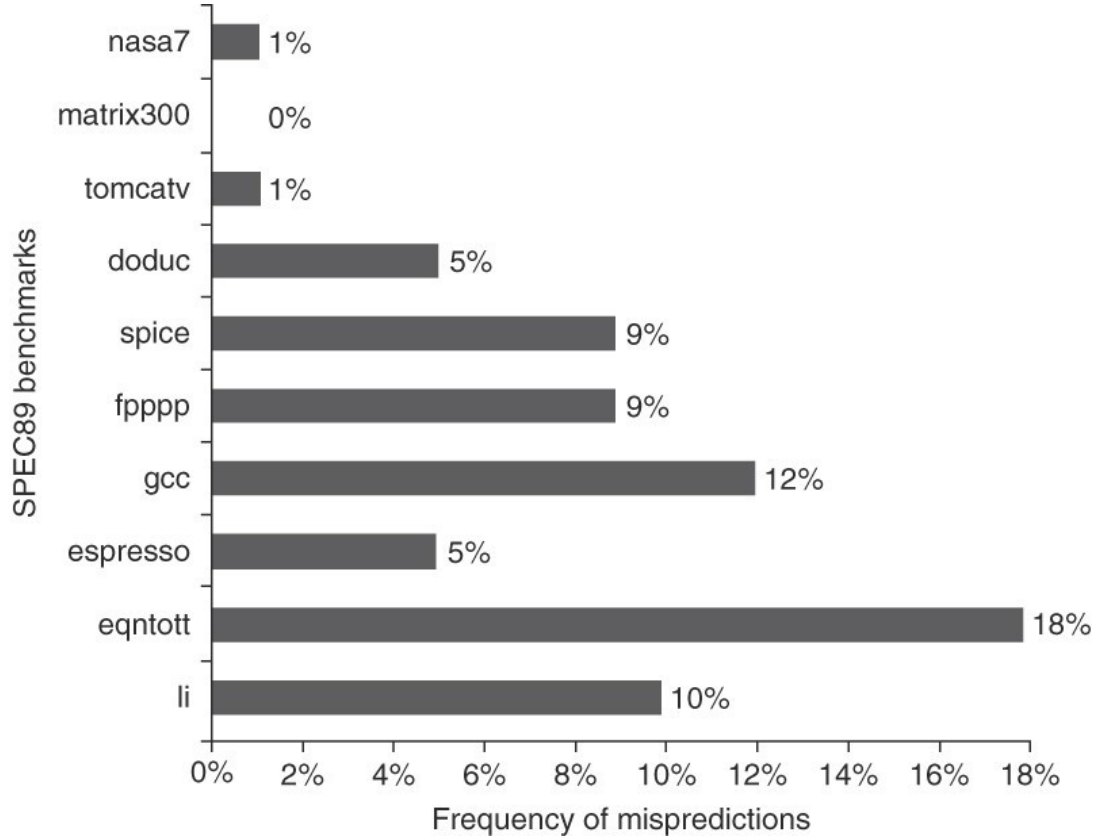


Perigos de controlo - 61

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

Taxa de previsão incorrecta para SPEC89 usando um buffer de previsão de 2 bits de 4096 entradas

Fonte: Arquitectura de Computadores: Uma Abordagem Quantitativa



Perigos de controlo - 62

Estudos mostram que a taxa de sucesso do tampão não é o principal factor limitativo. Na realidade, o comportamento de um tampão de 4096 entradas é muito semelhante ao comportamento de um tampão com número ilimitado de entradas. Aumentar o número de bits do preditor sem alterar a estrutura do preditor também tem pouco impacto.

As abordagens actuais favorecem a utilização de informação tanto local como global para melhorar a precisão da previsão.

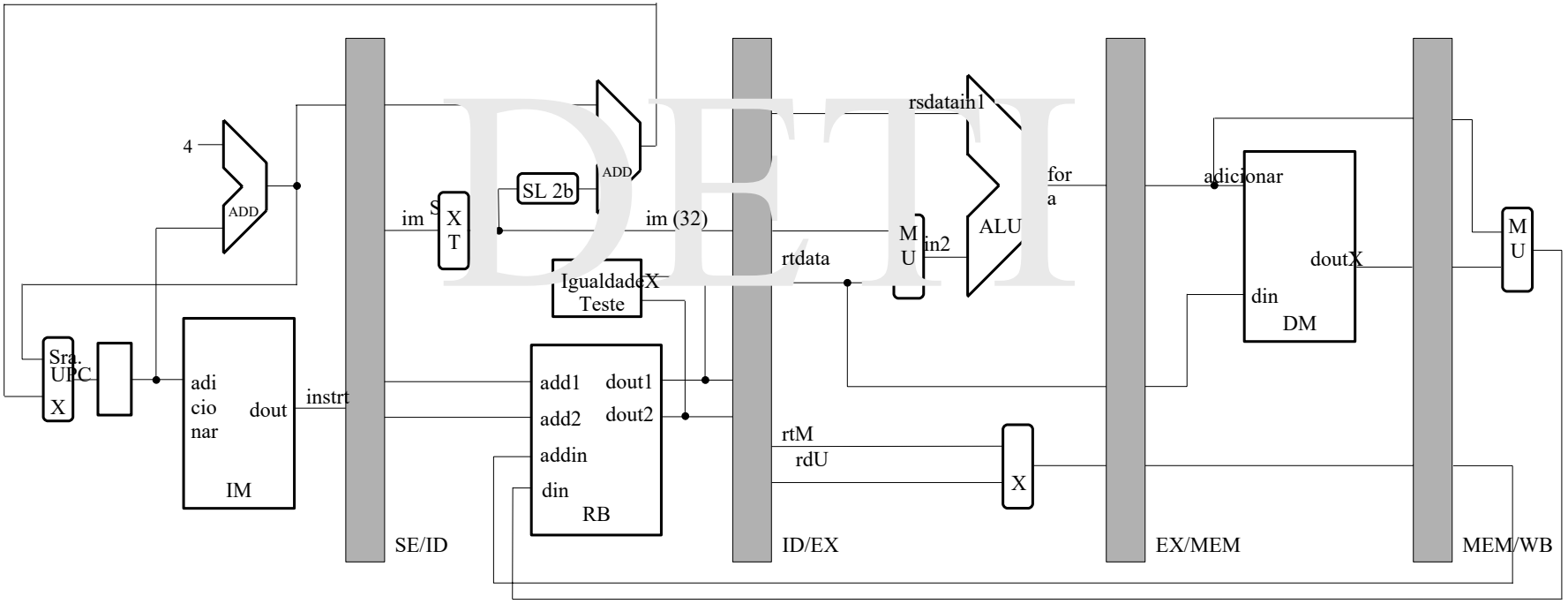
No caso mais simples, um *preditor correlato* pode consistir num preditor de 2 bits para cada ramo, relatando o seu historial em diferentes situações globais.

Mais recentemente, têm sido utilizados *preditores de torneios.* Um *prognosticador de torneios* utiliza múltiplos prognosticadores, rastreando o que produz o melhor resultado para cada ramo e tomando a sugestão vencedora como a próxima previsão.

Implementação do gasoduto clássico de 5 fases - 63

Datapath de 5 fases do gasoduto

Fonte: Adaptado de Computer Architecture: Uma Abordagem Quantitativa



Implementação do gasoduto clássico de 5 fases - 64

Classes de instruções a serem consideradas

instruções aritméticas / lógicas

0	rs	rt	rd	shamt	funct

31 25 20 15 10 5 0

Instrução do tipo R

instruções de carga e armazenamento

opcode	rs	rt	imediato

do ramo

instruções

Implementação do gasoduto clássico de 5 fases - 65

Instrução de tipo I

312520150

23 / 2B	rs	rt	compensação (imediata)

312520150

Instrução de tipo I

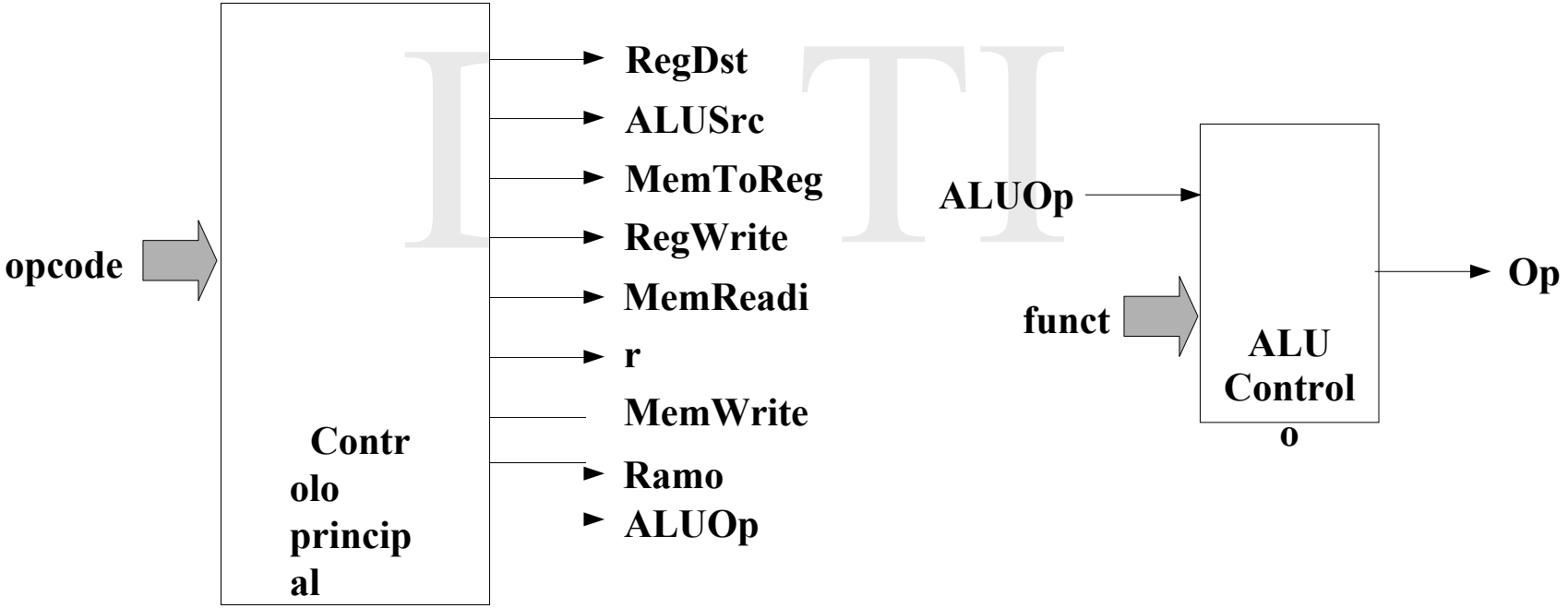
4	rs	rt	compensação (imediata)

312520150

Instrução de tipo I

Implementação do gasoduto clássico de 5 fases - 66

Unidade de controlo de condutas



Descrição funcional do controlo principal

Fonte: Adaptado de Computer Organization and Design: A interface Hardware/Software

		Descodificação da instrução / Fase de Registo	Execução / Etapa de Endereço Eficaz				Estágio de Acesso à Memória		Estágio de Reescrever	
Tipo de Instrução	**OpCode**	**Ramo**	**RegDst**	**ALUSrc**	**ALUOp1**	**ALUOp2**	**MemRead**	**MemWrite**	**RegWrite**	**MemToReg**
Formato R	0x00	0	1	0	1	0	0	0	1	0
lw	0x23	0	0	1	0	0	1	0	1	1
sw	0x2B	0	x	1	0	0	0	1	0	x

Nome do sinal	**Efeito quando desassertado (0)**	**Efeito quando afirmado (1)**
RegDst	o número de destino do registo de escrita vem do campo rt (bits 20:16)	o número de destino do registo de escrita vem do campo rd (bits 15:11)
RegWrite	nenhuma	o registo cujo número está em addin é escrito com o valor em din
ALUSrc	o segundo operando da ALU vem do conteúdo do registo cujo número está no campo rt (bits 20:16)	o segundo operando da ALU é o campo imediato (bits 15:0) após o sinal - aumentando-o para 32 bits
PCSrc	o PC é substituído pela saída do viciador que calcula o valor do PC+4	o PC é substituído pela saída do viciador que calcula o endereço de destino do ramo
MemRead	nenhuma	o conteúdo da localização da memória referenciada por adição é colocado no dout
MemWrite	nenhuma	o conteúdo do local de memória referenciado por adição é substituído pelo valor em din
MemToReg	o valor alimentado para a introdução de dados do registo de escrita provém da UAL	o valor alimentado para a introdução de dados do registo de escrita provém do doutor da memória de dados

Descrição funcional do ALU Control

Fonte: Adaptado de Computer Organization and Design: A interface Hardware/Software

Instrução	ALUOp	Função	Acção ALU pretendida	Operação
lw	00	xxxxxx	adicionar	0010
sw	00	xxxxxx	adicionar	0010
adicionar	10	100000	adicionar	0010
subtrair	10	100010	subtrair	0110
e	10	100100	e	0000
ou	10	100101	ou	0001
slt	10	101010	fixado em menos de	0111

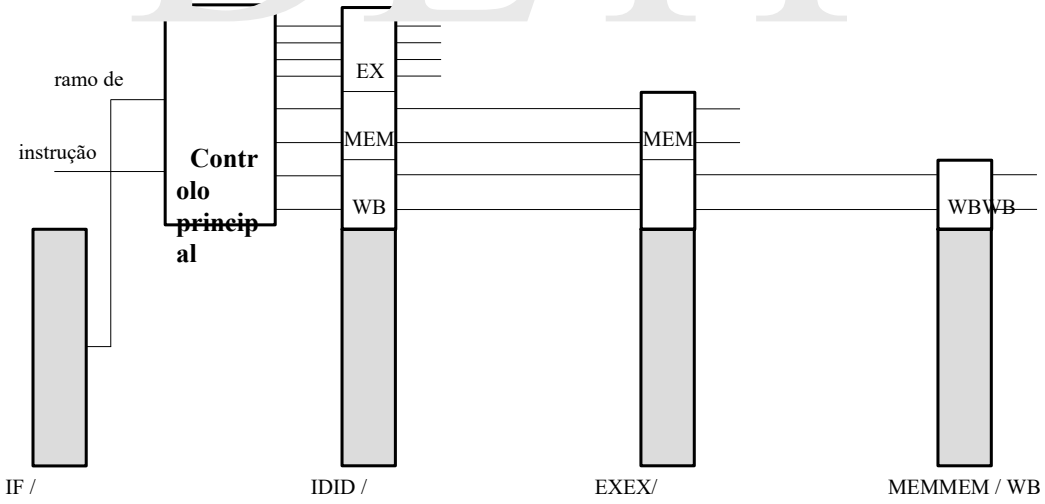
Implementação do gasoduto clássico de 5 fases - 69

Para que os circuitos de controlo funcionem correctamente, os sinais de controlo devem ser ajustados aos valores certos em cada etapa para cada instrução. A forma mais simples de o fazer é alargar os registos dos gasodutos de modo a incluir informação de controlo.

Uma vez que os sinais de controlo começam a ser aplicados na fase EX, as informações de controlo podem ser criadas durante a fase de identificação.

Implantação de sinais de controlo através das diferentes fases da tubagem

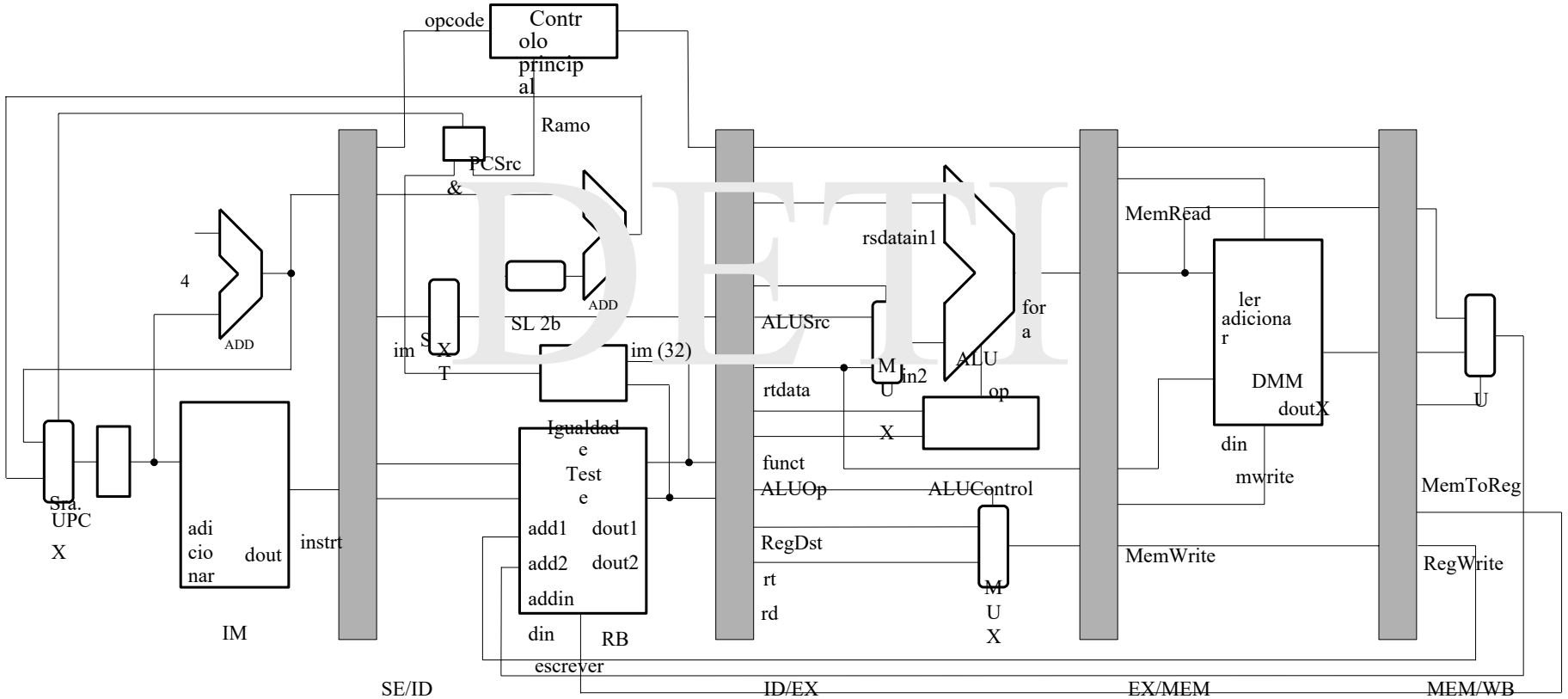
Fonte: Adaptado de Computer Organization and Design: A interface Hardware/Software



Implementação do gasoduto clássico de 5 fases - 70

Integração do controlo no datapath de 5 fases do gasoduto

Fonte: Adaptado de Computer Architecture: Uma Abordagem Quantitativa



Implementação do gasoduto clássico de 5 fases - 71

Os *riscos de dados* que podem ser resolvidos através *do encaminhamento* para esta implementação específica de gasodutos em 5 fases, podem ser classificados nas seguintes classes

- ID *hazard* - o conteúdo de um registo do banco de registo é exigido por uma instrução `beq` para o teste de igualdade e foi modificado por uma instrução prévia e o seu valor actualizado é actualmente armazenado num registo ID/EX, EX/MEM ou MEM/ WB pipeline
- EX *hazard* - o conteúdo de um registo do banco de registo é exigido por uma instrução e foi modificado por uma instrução prévia e o seu valor actualizado é actualmente armazenado num registo de gasodutos EX/MEM ou MEM/WB
- *Perigo* MEM - o conteúdo de um registo do banco de registo é exigido por uma instrução e foi modificado por uma instrução prévia e o seu valor actualizado é presentemente armazenado num registo de gasodutos MEM/WB.

Implementação do gasoduto clássico de 5 fases - 72

Detecção de riscos de dados de identificação

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

Implementação do gasoduto clássico de 5 fases - 73

Detecção de riscos de dados de identificação (cont.)

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

Implementação do gasoduto clássico de 5 fases - 74

Detecção de riscos de dados EX

```
if ( EX/MEM.RegWrite && (EX/MEM.RegDest ≠ 0) &&
      (EX/MEM.RegDest == ID/EX.RegRs) )
     fwdEXA = 10
     senão 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
     senão 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
```

Detecção de riscos de dados MEM

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

Valores de selecção para os multiplexadores de reencaminhamento na fase de identificação

Selecção MUX	Fonte	Explicação
fwdIDA= 00	SE/ID	O primeiro operando da ALU vem do ficheiro de registo
fwdIDA= 01	ID/EX	A primeira operação ALU é enviada do último resultado ALU
fwdIDA= 10	EX/MEM	A primeira operação ALU é encaminhada do primeiro para o último resultado ALU
fwdIDA= 11	MEM/WB	A primeira operação ALU é encaminhada da memória de dados ou o segundo para o último resultado ALU
fwdIDB = 00	SE/ID	O segundo operando da ALU vem do ficheiro de registo
fwdIDB = 01	ID/EX	A segunda operação ALU é enviada a partir do último resultado ALU
fwdIDB = 10	EX/MEM	A segunda operação ALU é encaminhada do primeiro para o último resultado ALU
fwdIDB = 11	MEM/WB	A segunda operação ALU é reencaminhada da memória de dados ou do segundo ao último resultado ALU

Valores de selecção para os multiplexores de expedição na fase EX

Selecção MUX	Fonte	Explicação
fwdEXA= 00	ID/EX	O primeiro operando da ALU vem do ficheiro de registo
fwdEXA= 10	EX/MEM	A primeira operação ALU é enviada do último resultado da ALU
fwdEXA= 11	MEM/WB	A primeira operação ALU é enviada da memória de dados ou do primeiro ao último resultado ALU
fwdEXB = 00	ID/EX	O segundo operando da ALU vem do ficheiro de registo
fwdEXB = 01	ID/EX	O segundo operando da ALU é o sinal de campo imediato - estendido a 32 bits
fwdEXB = 10	EX/MEM	A segunda operação ALU é enviada a partir do último resultado da ALU
fwdEXB = 11	MEM/WB	O segundo operando da ALU é encaminhado da memória de dados ou do primeiro para o último resultado da ALU

Implementação do gasoduto clássico de 5 fases - 78

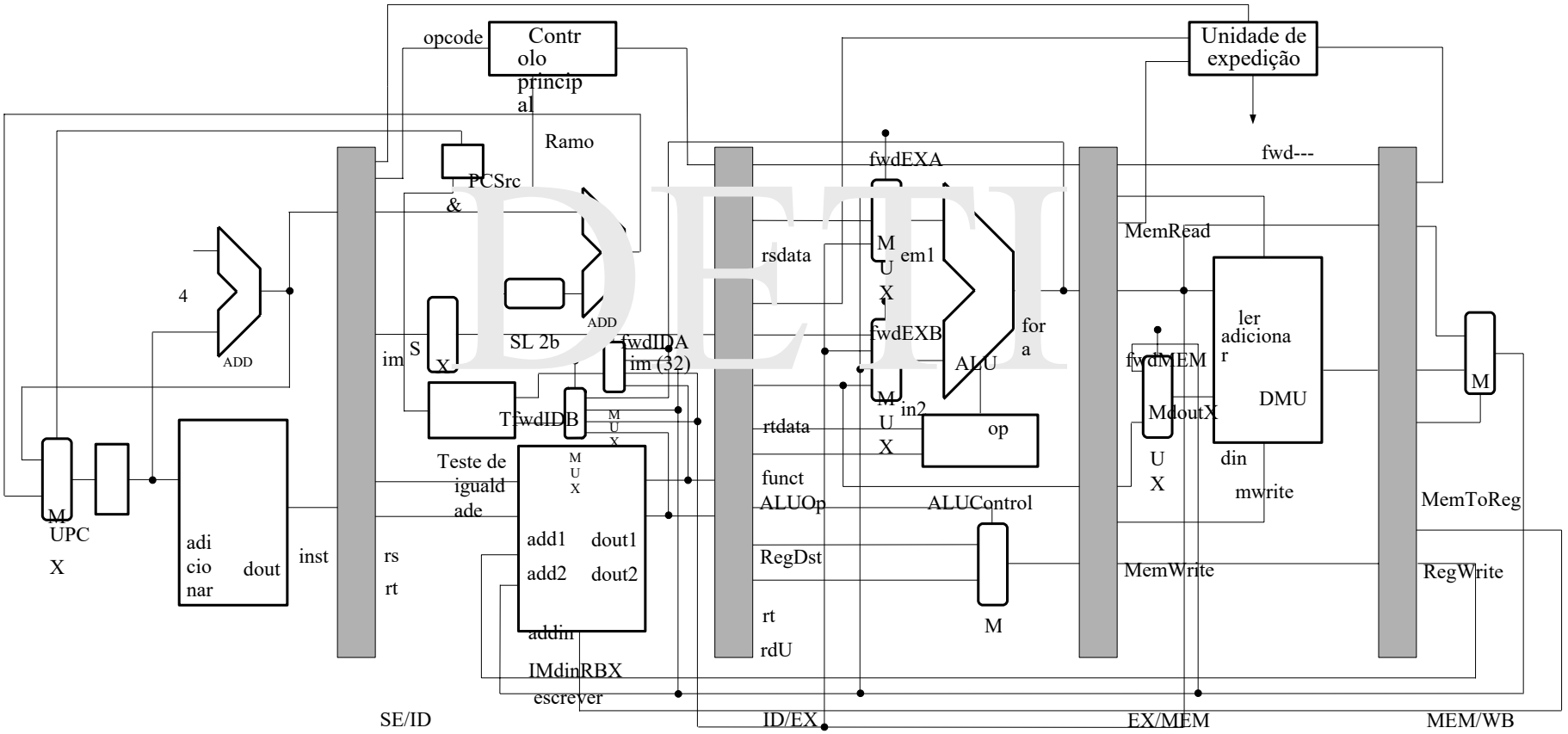
Valores de selecção para os multiplexadores de reencaminhamento na fase MEM

Selecção MUX	Fonte	Explicação
fwdMEM = 0	EX/MEM	O valor a ser possivelmente escrito na memória de dados vem do conteúdo do rt de registo
fwdMEM = 1	MEM/WB	O valor a ser escrito na memória de dados é o valor que tem apenas foi produzido

Implementação do gasoduto clássico de 5 fases - 79

Integrar o encaminhamento nos dados dos oleodutos em 5 fases

Fonte: Adaptado de Computer Architecture: Uma Abordagem Quantitativa



Implementação do gasoduto clássico de 5 fases - 80

Nem todos os *riscos de dados* para esta implementação específica de 5 fases do gasoduto podem ser resolvidos através *do reencaminhamento*. Como se viu, se uma instrução imediatamente a seguir a uma `carga` tentar ler o mesmo registo que é escrito por ela, não há maneira de esta instrução prosseguir até que o novo valor seja efectivamente recuperado da memória. Por isso, o progresso desta instrução no pipeline deve ser *interrompido*.

Além disso, e no que diz respeito aos *riscos de controlo*, se se assumir um esquema de *ramo retardado*, não é necessário considerar novas situações anormais que exigiriam o progresso da instrução.

Quando a instrução que segue a `carga` é uma instrução ALU, é necessário um ciclo de paragem; quando é uma instrução `beq`, são necessários dois ciclos de paragem.

Implementação do gasoduto clássico de 5 fases - 81

Detectar a necessidade de inserção de ciclos de estagnação

```
if ( ID/EX.MemRead && !MemWrite &&
     ((RegDest == IF/ID.RegRs) ||||
      (RegDest == IF/ID.RegRt && !MemRead ))
     empatar o gasoduto
     caso contrário, se ( Ramo && EX/MEM.MemReading &&
                ((EX/MEM.RegDest == IF/ID.RegRs) ||||
                 (EX/MEM.RegDest == IF/ID.RegRt ))
               empatar o gasoduto
```

Implementação do gasoduto clássico de 5 fases - 82

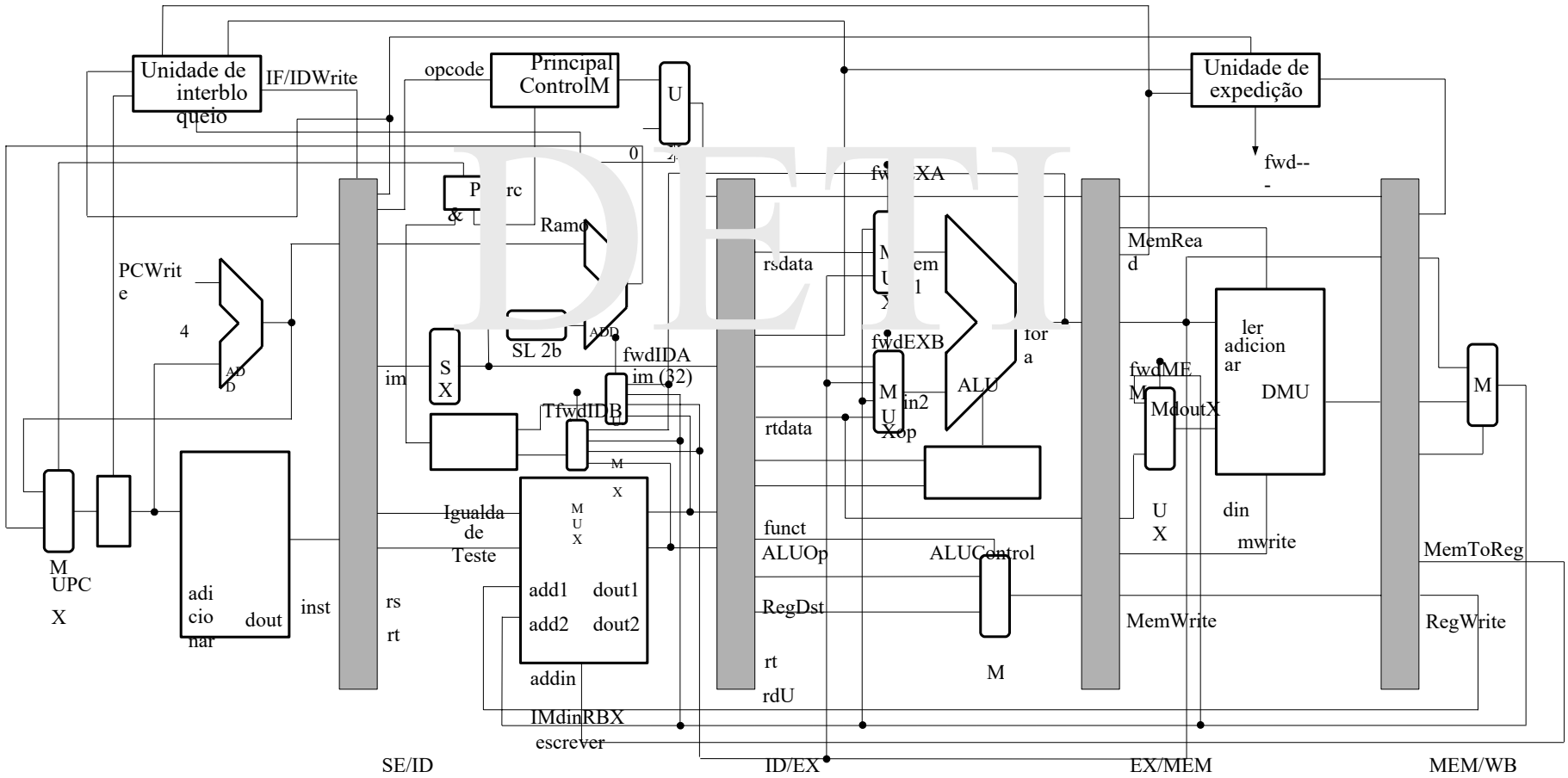
Para que a instrução na fase de identificação seja paralisada, a instrução na fase de IF também deve ser paralisada. Uma forma simples de o conseguir é impedir que tanto o PC como os registos da conduta IF/ID mudem. Desde que o conteúdo destes registos seja preservado, as instruções processadas na fase de FI e ID são mantidas inalteradas.

Por outro lado, as instruções nas outras fases devem proceder como se nada tivesse acontecido. Isto significa que deve ser gerada uma instrução *não operacional* e inserida na fase EX no próximo ciclo do relógio. Isto pode ser conseguido desactivando os sinais de controlo quando estes são escritos nos registos ID/EX dos oleodutos.

Implementação do gasoduto clássico de 5 fases - 83

Integrar o encravamento no caminho de dados de 5 fases do gasoduto

Fonte: Adaptado de Computer Architecture: Uma Abordagem Quantitativa



Excepções - 84

O controlo é o aspecto mais desafiante do design do processador: é tanto a parte mais difícil de acertar como a parte mais difícil de fazer andar depressa. Uma das partes mais difíceis do controlo é permitir *excepções* ou *interrupções* - eventos que provocam a quebra da sequência estrita da execução de instruções, para além de ramos e saltos.

As excepções típicas incluem

- pedidos de atenção de um controlador associado a um dispositivo de E/S
- invocando uma chamada de sistema a partir do nível do utilizador
- execução de instrução de rastreio ou definição de pontos de ruptura
- transbordo de inteiros ou anomalia de ponto flutuante
- falha de página numa organização de memória virtual
- acesso à memória desalinhada
- violação da protecção da memória
- tentando executar uma instrução não definida ou não implementada
- mau funcionamento do hardware
- falha de energia.

Excepções - 85

As excepções individuais têm características importantes que determinam a acção que o hardware é necessário para executar. Estes requisitos podem ser classificados em cinco categorias semi-independentes

- *síncrono* vs. *assíncrono* - ao contrário dos eventos assíncronos, os eventos síncronos ocorrem no mesmo lugar sempre que o programa é executado com os mesmos dados e alocação de memória; os eventos assíncronos, pelo contrário, podem ocorrer em qualquer lugar dentro do programa e podem ser normalmente tratados após a conclusão da instrução actual
- *utilizador solicitado* versus *coagido* - as excepções solicitadas pelo utilizador, sendo previsíveis, não são realmente, em certo sentido, verdadeiras excepções; só são tratadas como tal porque o mesmo mecanismo, que é utilizado para salvar e restaurar o estado do programa, também lhes é aplicado; uma vez que a única função da instrução que desencadeia este tipo de excepção é causar a própria excepção, podem sempre ser tratadas depois de a instrução estar concluída; as excepções coagidas, pelo contrário, são causadas por algum evento de hardware que o programa não controla e são totalmente imprevisíveis

Excepções - 86

- *mascarável* vs. *não mascarável* - algumas excepções permitem ao programa escolher o momento em que o hardware responde a eles
- *dentro* vs. *entre instruções* - um evento pode impedir a conclusão de uma instrução em execução, quando é desencadeado por ela, porque alguma função ou anomalia maléfica ocorreu a nível de software / hardware; as correspondentes excepções são normalmente síncronas e difíceis de implementar porque a instrução deve ser parada e, eventualmente, reiniciada mais tarde; excepções assíncronas que ocorrem dentro das instruções, em contraste, surgem de situações catastróficas e causam sempre o término do programa
- *retomar* versus *terminar* - excepções que não requerem que o programa seja retomado depois de serem tratadas, são mais fáceis de implementar porque não há necessidade de reiniciar o programa.

Excepções - 87

Como são comuns as excepções no esquema de classificação das 5 categorias

Fonte: Adaptado de Computer Architecture: Uma Abordagem Quantitativa

Tipo de Excepção	Síncrono vs. Assíncrono	Utilizador solicitado vs. Utilizador solicitado vs. Utilizador solicitado vs. Utilizador solicitado vs. Coerced	Máscara vs. Não-máscara	Dentro vs. Entre instruções	Retomar vs. Terminar
Pedido de dispositivo de E/S	asynchonous	coagido	mascarável	entre	currículo
invocação do sistema operativo	síncrono	utilizador solicitado	mascarável	entre	currículo
execução de instrução de rastreio ou definição de breahpoints	síncrono	utilizador solicitado	mascarável	entre	currículo
overflow de número inteiro ou anomalia FP	síncrono	coagido	mascarável	em	currículo
falha de página (memória virtual)	síncrono	coagido	não-marcável	em	currículo
acesso à memória desalinhada	síncrono	coagido	não-marcável	em	terminar
violação da protecção da memória	síncrono	coagido	não-marcável	em	terminar
execução de instrução indefinida ou não implementada	síncrono	coagido	não-marcável	em	terminar
mau funcionamento do hardware	asynchonous	coagido	não-marcável	em	terminar
falha de energia	asynchonous	coagido	não-marcável	em	terminar

Excepções - 88

Tal como nas organizações semipipeline, a difícil tarefa é implementar excepções que ocorrem dentro das instruções, normalmente nas fases EX ou MEM, que têm de ser retomadas. A implementação supõe que outro programa, em princípio o sistema operacional, é invocado para salvar o estado do programa de execução, corrigir a causa da excepção e restaurar o estado do programa, antes que a instrução que causou a excepção seja novamente executada - um mecanismo que tem obviamente de ser totalmente transparente para o programa de execução.

Se a conduta tiver a capacidade de lidar com a excepção, salvar o estado e reiniciar sem afectar a execução do programa, diz-se que o processador é *reiniciável*. Embora os primeiros supercomputadores e microprocessadores muitas vezes não tivessem esta propriedade, hoje em dia quase todos os processadores a suportam, pelo menos para o gasoduto inteiro, porque repousa na base de tornar operacional a organização da memória virtual.

Excepções - 89

Para que o sistema operativo possa lidar com uma excepção, deve saber a razão que o desencadeou, além da instrução que o causou ou que seria executada a seguir se a excepção não fosse reparada. Existem dois métodos principais para comunicar o motivo de uma excepção

- *registo de causas* - é um registo de estado que contém um campo que descreve a causa de uma excepção ocorrida, sendo o seu valor definido pelo hardware aquando da sua detecção; quando as excepções são servidas pelo mesmo endereço do ponto de entrada, este é o meio que o sistema operativo tem de determinar a causa da excepção
- *excepções vectorizadas* - existem múltiplos endereços de pontos de entrada para o serviço das excepções; em geral, cada endereço de ponto de entrada está associado a uma excepção específica, pelo que a identificação da causa pelo sistema operativo se torna trivial.

Excepções - 90

Quando uma excepção é servida, o controlo da conduta deve tomar as seguintes medidas para salvar o estado do programa e permitir que o programa seja retomado mais tarde, se for esse o caso.

1. O valor actual do PC é guardado e é substituído pelo endereço do ponto de entrada correspondente à excepção para o próximo ciclo IF. O processador é colocado num modo privilegiado onde está disponível um conjunto de instruções mais amplo e todas as excepções mascaráveis do mesmo nível de prioridade ou de prioridade inferior são desactivadas.
2. Todas as instruções que se seguem actualmente na tubagem, e a própria instrução se a excepção estiver dentro dela, têm de ser transformadas em instruções *não operacionais* para as restantes fases da tubagem. Todas as instruções anteriores, se existirem, por outro lado, devem ser autorizadas a completar para que o estado do programa seja consistente no momento do processamento da excepção.

Excepções - 91

3. Após a rotina de serviço de excepção no sistema operativo começar a ser executada, actualiza imediatamente o valor guardado no PC para o valor correcto, o que depende da causa da excepção (dentro ou entre, e se o primeiro caso for verdadeiro, na fase de tubagem que o desencadeou).
   Também se deve notar que, se for assumido um esquema de *filial atrasada*, já não é possível recriar o estado do processador com o armazenamento de um único valor de PC - as instruções na tubagem podem não estar sequencialmente relacionadas. Para que a rotina de serviço de excepção possa actualizar o PC gravado para o valor correcto, são necessários tantos endereços de PC como o comprimento da *ranhura de retardamento do ramo* mais 1.
4. Finalmente, após o término da rotina do serviço de excepção, instruções especiais do tipo *retorno do* tipo de *excepção* reiniciam o fluxo de instruções anterior, restaurando o valor do PC e o modo de execução anterior.

Excepções - 92

Se a conduta puder ser parada para que as instruções que precedem a instrução de falha sejam completadas e ela própria, juntamente com as que a sucedem, possa ser reiniciada do zero, diz-se que a conduta tem *excepções precisas*.

Idealmente, a instrução de falha não deve alterar o estado de execução e, portanto, para o tratamento correcto de algumas excepções, é necessário que a instrução de falha não produza efeitos. Para outras excepções, contudo, tais como excepções de pontos flutuantes, a instrução de falha em alguns processadores escreve o seu resultado antes de a excepção poder ser tratada. Nesses casos, o hardware deve estar preparado para recuperar os operandos de origem, mesmo que o destino seja o mesmo que um dos operandos de origem.

Para ultrapassar este problema, muitos processadores recentes de alto desempenho introduziram dois modos de funcionamento: um modo tem excepções precisas e o outro, para ser mais performante, não o faz. De facto, o modo de excepção preciso deve ser mais lento porque tem de permitir uma sobreposição muito menor entre as instruções de ponto flutuante.

Excepções - 93

Com a canalização, podem ocorrer múltiplas excepções no mesmo ciclo de relógio porque há múltiplas instruções em execução simultânea.

Típico *dentro das* excepções que podem ocorrer no gasoduto clássico de 5 fases

Fonte: Arquitectura de Computadores: Uma Abordagem Quantitativa

Etapa do oleoduto	Excepções
SE	falha de página na instrução fetch misligned memory access memory memory protection violation
ID	instrução não definida ou não implementada
EX	excepção aritmética

Excepções - 94

MEM	falha de página na busca de dados violação da protecção da memória de acesso à memória
WB	nenhuma

Excepções - 95

Considere a sequência de instruções

LD	SE	ID	EX	M EM	WB	
DADD		SE	ID	EX	M EM	WB

Este par de instruções pode causar uma falha na página de dados e uma excepção aritmética no mesmo ciclo de relógio. A forma de lidar com este problema é responder à excepção da página de dados e depois reiniciar a execução. A excepção aritmética voltará a ocorrer e só então poderá ser tratada de forma independente.

Em geral, as coisas não são tão simples. As excepções podem ocorrer *fora de ordem*, ou seja, uma instrução pode desencadear uma excepção antes de uma anterior, que está em execução simultânea, desencadear também a sua própria excepção. Para visualizar isto, tomar por exemplo a situação em que a instrução de *carga* gera uma falha de página no fetch de dados na fase MEM e a instrução de *adição* gera uma falha de página no fetch de instruções na fase IF.

Para desenvolver uma implementação *precisa* das *excepções*, a excepção desencadeada pela instrução de *carga* deve ser tratada em primeiro lugar. Como é que isso pode ser feito?

Excepções - 96

O gasoduto não pode tratar as excepções tal como elas ocorrem a tempo, uma vez que, ao fazê-lo, as excepções correm o risco de serem processadas a partir da ordem não pipelada.

Em vez disso, o hardware coloca todas as excepções causadas por uma dada instrução num vector de estado associado a essa instrução. O vector de estado continua a mover-se ao longo das fases da tubagem com a instrução. Assim que houver uma indicação de excepção no vector de estado, qualquer sinal de controlo que possa causar a escrita de um valor de dados, é desactivado (isto significa a escrita do registo e os sinais de escrita da memória).

Quando a instrução entra na fase WB, o vector de estado é verificado. Se for afixada qualquer excepção, ela é tratada pela ordem em que ocorreria a tempo numa implementação não pipelada.

Operações de múltiplos ciclos em gasoduto clássico de 5 fases - 97

É impraticável presumir que todas as operações de PF e multiplicações e divisões inteiras se completarão num ciclo de relógio, ou mesmo em dois. Fazê-lo significaria prolongar 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 ambas em conjunto. Em vez disso, a conduta FP contemplará uma latência mais longa para as operações.

A melhor maneira de entender a ideia é imaginar as instruções FP como tendo a mesma conduta que as instruções inteiras do núcleo, com duas diferenças importantes

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

Ocorrerá uma paragem se a instrução a ser emitida causar ou um risco estrutural para a unidade funcional requerida, ou um risco de dados.

Operações de múltiplos ciclos em gasoduto clássico de 5 fases - 98

Para efeitos de discussão, quatro unidades funcionais separadas serão consideradas na implementação do projecto

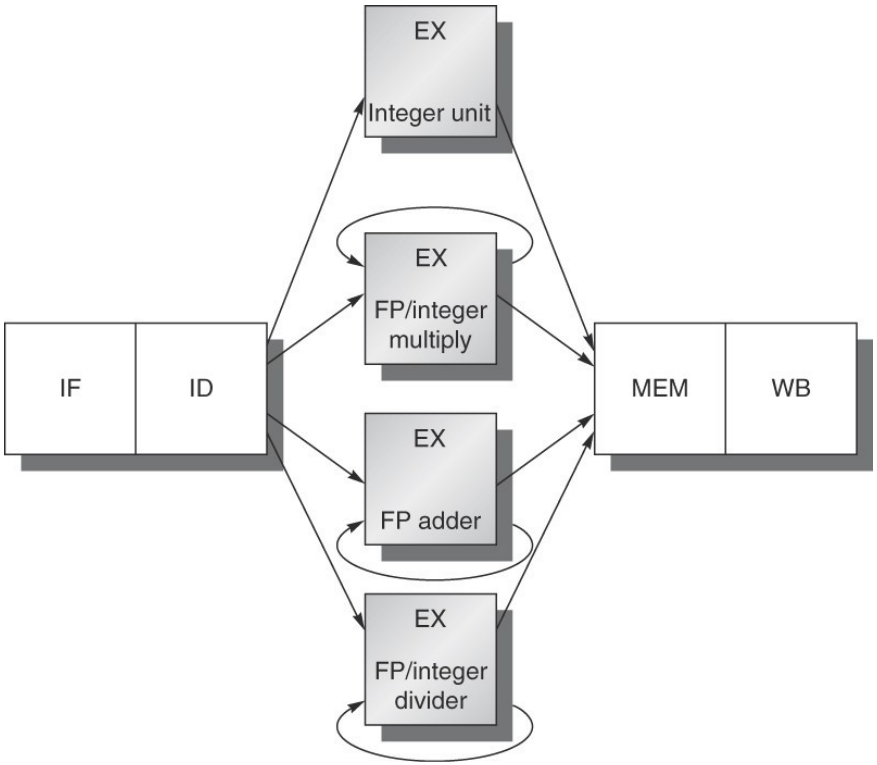
- a unidade inteira principal, que lida com cargas, armazena e a ALU inteira básica operações
- o PF e multiplicador de números inteiros
- a víbora FP, que trata da adição, subtracção e conversão de tipos de dados
- o PF e o divisor de números inteiros.

Assumindo que estas unidades funcionais não estão canalizadas, nenhuma nova instrução usando uma unidade funcional específica pode ser emitida se uma instrução anterior usando a mesma unidade ainda estiver em funcionamento, ou que seja a mesma, ainda não tiver saído da fase EX. Além disso, se uma instrução não puder prosseguir para a fase EX, todo o gasoduto até este ponto será bloqueado.

Operações de múltiplos ciclos em gasoduto clássico de 5 fases - 99

O gasoduto clássico de 5 fases com três unidades funcionais FP adicionais não encapsuladas

Fonte: Arquitectura de Computadores: Uma Abordagem Quantitativa



Operações de múltiplos ciclos em gasoduto clássico de 5 fases - 100

A estrutura de toda a tubagem pode ser generalizada para permitir a canalização de algumas unidades funcionais FP e permitir múltiplas operações em curso. 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 relógio intervenientes entre uma instrução que produz um resultado e uma instrução que utiliza o resultado
- *início* ou *intervalo de repetição* - é o número de ciclos de relógio que devem decorrer entre a emissão de duas operações do mesmo tipo.

As operações de ALU inteira têm uma latência de zero desde que os resultados podem ser utilizados no próximo ciclo do relógio. *As cargas*, por outro lado, têm uma latência de um, uma vez que os resultados podem ser utilizados após um ciclo de relógio interveniente. Além disso, uma vez que a maioria das operações consome os seus operandos no início da fase EX, a latência é normalmente o número de fases após EX que uma instrução necessita para produzir o resultado: zero fases para as operações de ALU inteira e uma fase para as operações de

Operações de múltiplos ciclos em gasoduto clássico de 5 fases - 101

carga. A exceção são os *armazéns* onde a latência é menos um.

Operações de múltiplos ciclos em gasoduto clássico de 5 fases - 102

Latências e intervalos de repetição para as operações nas unidades funcionais

Fonte: Arquitectura de Computadores: Uma Abordagem Quantitativa

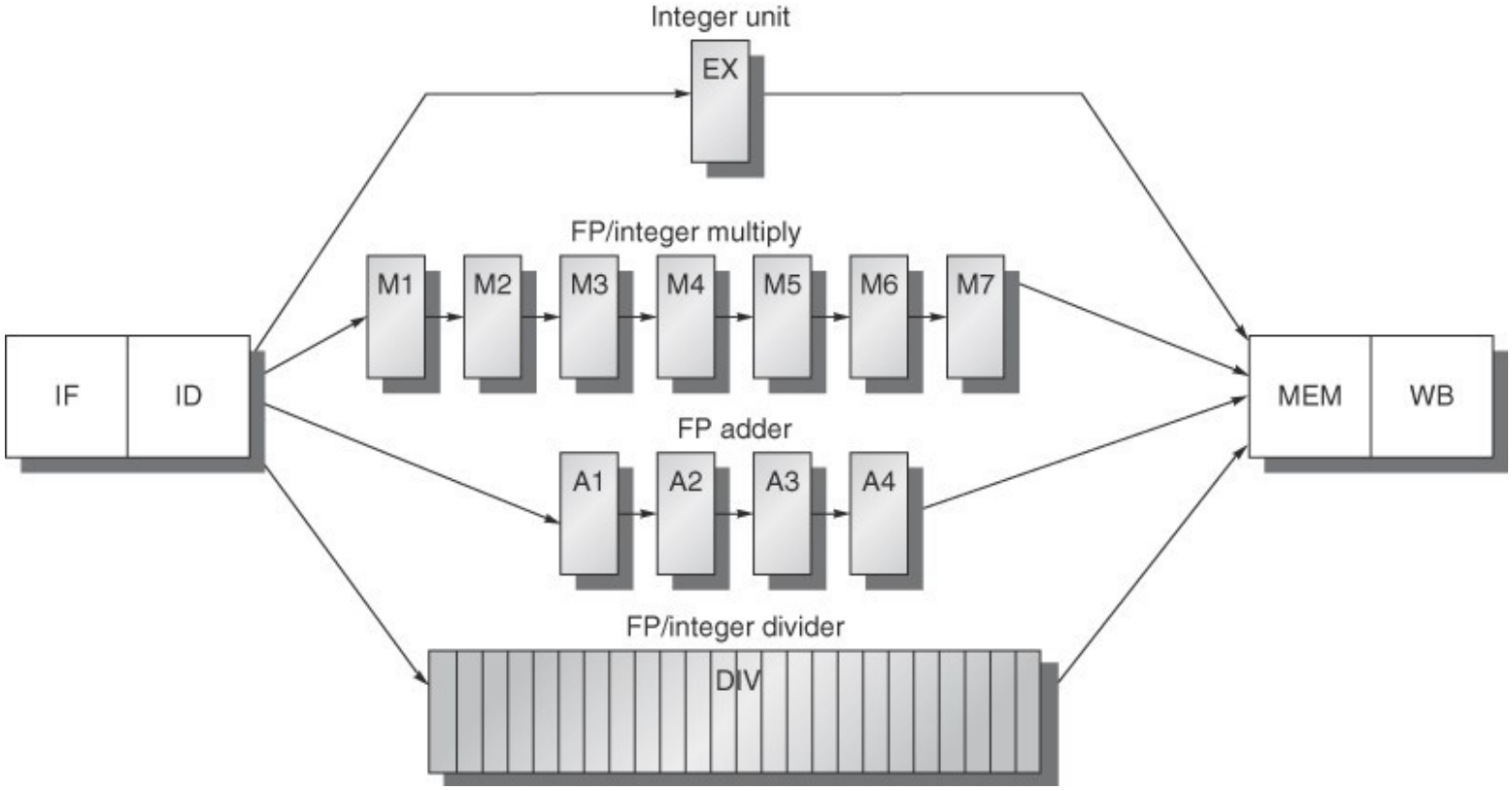
Unidade Funcional	Latência	Intervalo de Repetição
ALU inteira	0	1
Memória de dados (Inteiro e cargas FP)	1	1
Adição de FP	3	1
Multiplicação FP Multiplicação inteira	6	1
Divisão FP Divisão inteira	24	25

Note-se que apenas a adição de FP e a multiplicação de FP / Unidades de multiplicação inteira são pipelinadas, A divisão FP / Unidade de divisão inteira, por outro lado, não é pipelinada e requer 24 ciclos de relógio para produzir um resultado.

Operações de múltiplos ciclos em gasoduto clássico de 5 fases - 103

O gasoduto clássico de 5 fases que suporta múltiplas operações de PF pendentes

Fonte: Arquitectura de Computadores: Uma Abordagem Quantitativa



Operações de múltiplos ciclos em gasoduto clássico de 5 fases - 104

As seguintes observações estão em ordem

- uma vez que a divisão FP / unidade de divisão inteira não está canalizada, podem ocorrer *riscos estruturais*; têm de ser detectados e as instruções de emissão de instruções de utilização da unidade têm de ser interrompidas
- uma vez que agora nem todas as instruções têm o mesmo tempo de execução, o número de registos escritos no mesmo ciclo de relógio pode ser superior a um
- os *perigos dos dados*, em que uma instrução tenta escrever um registo antes de ter sido escrita por outra instrução, são possíveis - o que implica que a ordem de execução não pipelada já não é mantida
- o facto de as instruções poderem ser completadas numa ordem diferente da que foram emitidas, dará origem a problemas quando se tratar de excepções
- os *perigos dos dados*, onde uma instrução tenta ler um registo antes de ter sido escrita por outra instrução, serão mais frequentes.

Operações de múltiplos ciclos em gasoduto clássico de 5 fases - 105

Os *perigos dos dados*, em que uma instrução tenta ler um registo antes de ter sido escrita por outra instrução, seguem um padrão que é fundamentalmente o mesmo encontrado para o gasoduto inteiro.

	Número do relógio																
Instrução	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***	***11***	***12***	***13***	***14***	***15***	***16***	***17***
`L.DF4,0(R2)`	SE	ID	EX	MEM	WB												
`MUL.D F0,F4,F6`		SE	ID	*estábulo*	M1	M2	M3	M4	M5	M6	M7	MEM	WB				
`ADD.D F2,F0,F8`			SE	*estábulo*	ID	*estábulo*	*estábulo*	*estábulo*	*estábulo*	*estábulo*	*estábulo*	A1	A2	A3	A4	MEM	WB
`S.DF2,0(R2)`				SE	*estábulo*	*estábulo*	*estábulo*	*estábulo*	*estábulo*	*estábulo*	*estábulo*	EX	EX	*estábulo*	*estábulo*	*estábulo*	MEM

Note-se que a instrução da *loja* foi paralisada um ciclo extra para evitar o risco estrutural que resultaria do conflito de acesso à fase de MEM. Contudo, uma vez que apenas uma das instruções acede à memória de dados, podem ser feitas para prosseguir para a fase MEM no mesmo ciclo de relógio, se for acrescentado hardware extra.

Operações de múltiplos ciclos em gasoduto clássico de 5 fases - 106

Se se assumir que o banco de registo de FP tem uma única porta de escrita, as sequências de operações de FP, bem como uma instrução de *carga de* FP juntamente com outras operações de FP, podem dar origem a conflitos de acesso à porta de escrita do registo.

	Número do relógio										
Instrução	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***	***11***
MUL.D F0,F4,F6	SE	ID	M1	M2	M3	M4	M5	M6	M7	MEM	WB
. . .		SE	ID	EX	MEM	WB					
. . .			SE	ID	EX	MEM	WB				
ADD.D F2,F4,F6				SE	ID	A1	A2	A3	A4	MEM	WB
. . .					SE	ID	EX	MEM	WB		
. . .						SE	ID	EX	MEM	WB	
L.DF0,0(R2)							SE	ID	EX	MEM	WB

No ciclo de relógio 11, as três instruções chegam à fase de WB e precisam de escrever para o banco de registo FP. Com uma única porta de escrita, o processador tem de completar as instruções em série. O número de portas de escrita poderia ser aumentado para resolver o problema, mas esta abordagem pode não ser muito atractiva uma vez que as portas de escrita extra podem ser raramente

Operações de múltiplos ciclos em gasoduto clássico de 5 fases - 107

utilizadas.

Operações de múltiplos ciclos em gasoduto clássico de 5 fases - 108

Em vez disso, pode-se optar por detectar o risco estrutural e agendar o acesso ao porto de escrita.

Uma solução é acompanhar a utilização do porto de escrita na fase de identificação e, se for necessário, empatar a instrução actual antes de esta ser emitida, tal como é feito para qualquer outro risco estrutural. A implementação desta abordagem necessita de um registo de turno, chamado *registo de reserva*, cujo conteúdo se desloca na direcção oposta à do fluxo de instruções no pipeline em cada ciclo de relógio. O seu objectivo é assinalar os ciclos em que as instruções já emitidas irão escrever no banco de registo FP. Assim, quando a instrução na fase de identificação terá de escrever para um registo do banco de registo de FP, a fase correspondente do registo de turnos é verificada. Se o bit for definido, o que significa que outra instrução já emitida também o fará nesse ciclo do relógio, a instrução é interrompida. Caso contrário, esse bit do registo de turnos é definido e a instrução progride.

Esta abordagem tem a vantagem de manter toda a detecção de interbloqueio e inserção de estagnação restrita à fase de identificação. O custo é o registo de turno extra e toda a lógica para determinar o potencial conflito de escrita.

Operações de múltiplos ciclos em gasoduto clássico de 5 fases - 109

Outra solução é empatar uma instrução conflituosa ao tentar entrar na fase MEM ou na fase WB. Qualquer uma das instruções conflituosas pode ser escolhida para empatar. Uma simples, embora por vezes suboptimizada, heurística é dar prioridade à instrução que sai da unidade com a maior latência, uma vez que é a que mais provavelmente causou a paralisação de outra instrução.

Esta abordagem tem a vantagem de adiar a detecção do conflito até ao momento em que se torna trivial afirmá-lo. A desvantagem é que torna o controlo de condutas mais complexo, uma vez que as bancas podem agora surgir de dois locais.

Operações de múltiplos ciclos em gasoduto clássico de 5 fases - 110

Os *perigos dos dados*, em que uma instrução tenta escrever para um registo antes de ter sido escrita por outra instrução, não são aparentemente dramáticos e podem ser descartados. A lógica é que nunca deveriam ocorrer numa sequência de código bem escrita, porque nenhum compilador geraria duas escritas para o mesmo registo sem uma leitura interveniente. No entanto, se a sequência for inesperada, elas podem de facto ocorrer e produzir resultados errados.


```
                BNZR1,fo
        o
                DIV.DF0,
        F2,F4
        . . .
foo:            L.DF0,0(R3)
```

Se o ramo for tomado (assumindo um esquema de ramo atrasado), a instrução `L.D` chegará à fase WB antes que a instrução `DIV.D` possa ser completada. Assim, originando uma inconsistência no estado do programa a partir deste ponto.

Operações de múltiplos ciclos em gasoduto clássico de 5 fases - 111

Há duas maneiras possíveis de lidar com este perigo. A primeira abordagem é adiar a questão da segunda instrução escrita até a primeira entrar na fase MEM. A segunda é eliminar a primeira instrução, detectando o perigo e desactivando a sua capacidade de alterar o registo - a segunda instrução pode então ser imediatamente emitida...

Devido à sua raridade em aparecer no código, qualquer uma das abordagens é aceitável.

Operações de múltiplos ciclos em gasoduto clássico de 5 fases - 112

Um problema crítico causado por instruções de longa duração é ilustrado pela sequência de códigos abaixo

```
        DIV.DF0,F2,
F4
        ADD.DF10,F1
0,F8
        SUB.DF12,F1
2,F14
```


Embora não haja dependências de dados, a primeira instrução será completada após as duas seguintes. Uma situação que é conhecida como *conclusão fora de ordem*.

Esta condição pode levar a *excepções imprecisas*. Para compreender como, considerar que após a conclusão das instruções `ADD` e `SUB`, a instrução `DIV` irá gerar uma excepção. Uma vez que tanto o `ADD` como o `SUB` alteraram cada um dos seus operandos, o estado do programa foi modificado e não pode ser recuperado,

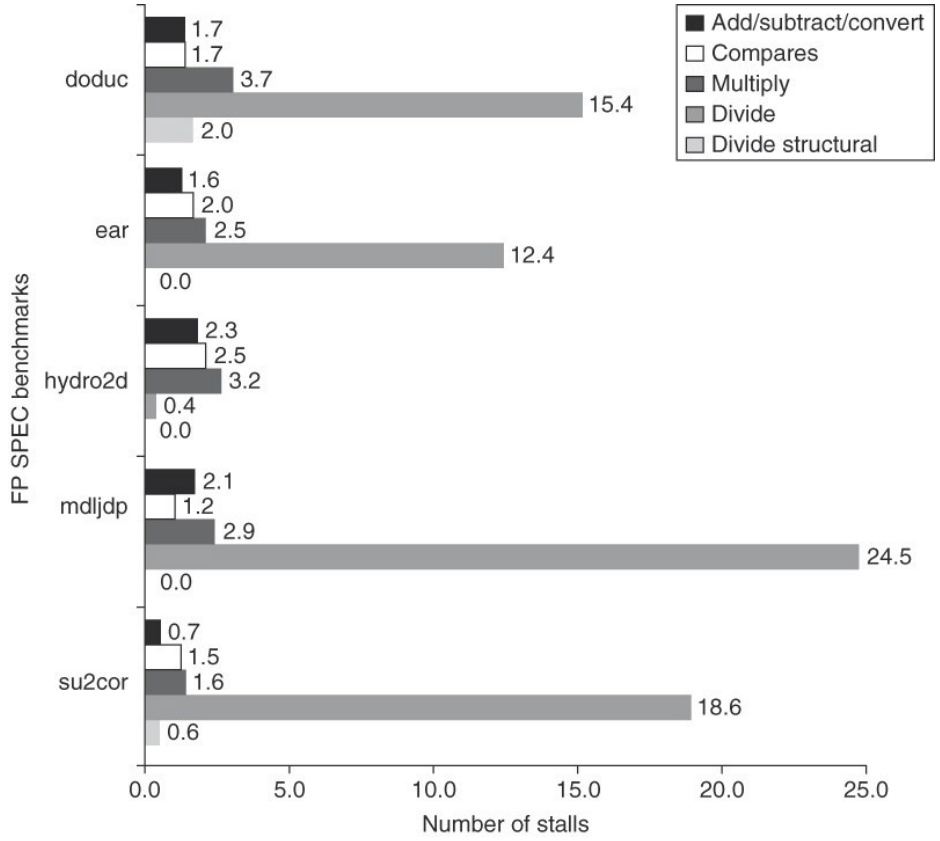
nem mesmo com ajuda de software. Portanto, não é possível reiniciar a instrução `DIV`!

Existem formas de lidar com este problema, mas não serão discutidas aqui.

Operações de múltiplos ciclos em gasoduto clássico de 5 fases - 114

Paradas por operação de PF para cada tipo principal de operação de PF para a referência SPEC89 FP

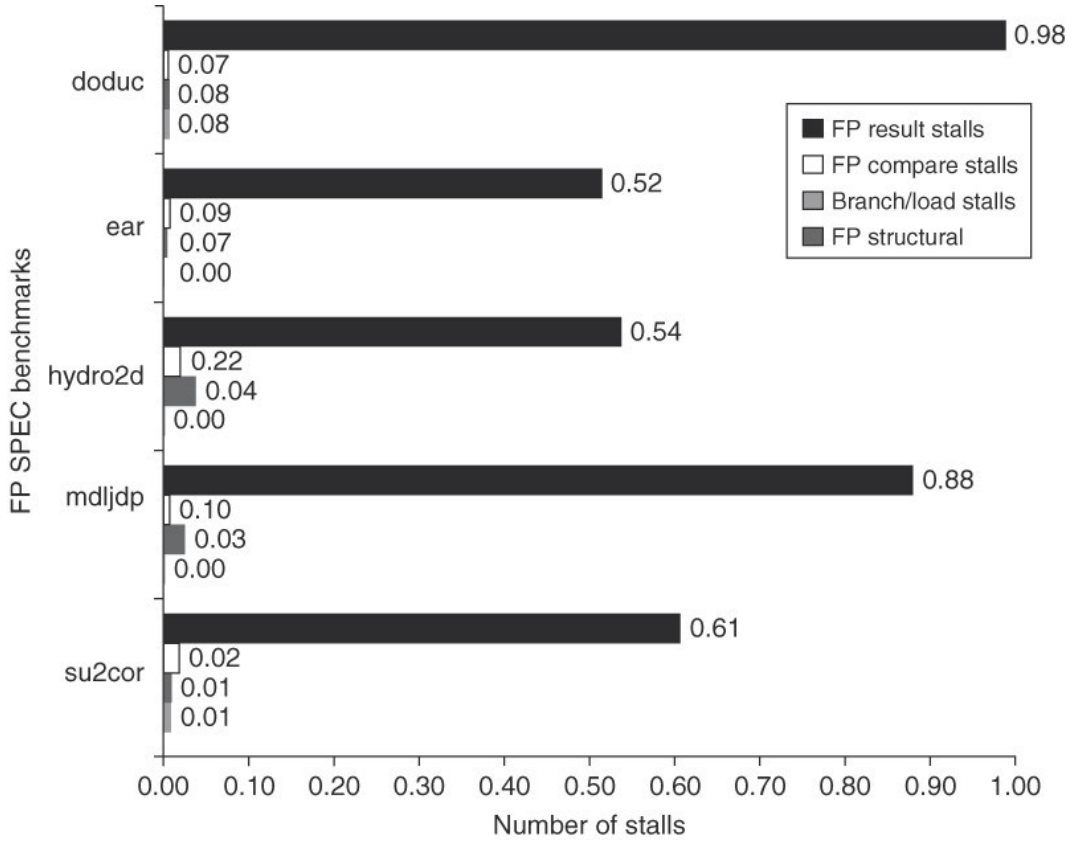
Fonte: Arquitectura de Computadores: Uma Abordagem Quantitativa



Operações de múltiplos ciclos em gasoduto clássico de 5 fases - 115

Paragens por instrução para o gasoduto MIPS FP para a referência SPEC89 FP

Fonte: Arquitectura de Computadores: Uma Abordagem Quantitativa



Gasoduto MIPS R4000 - 116

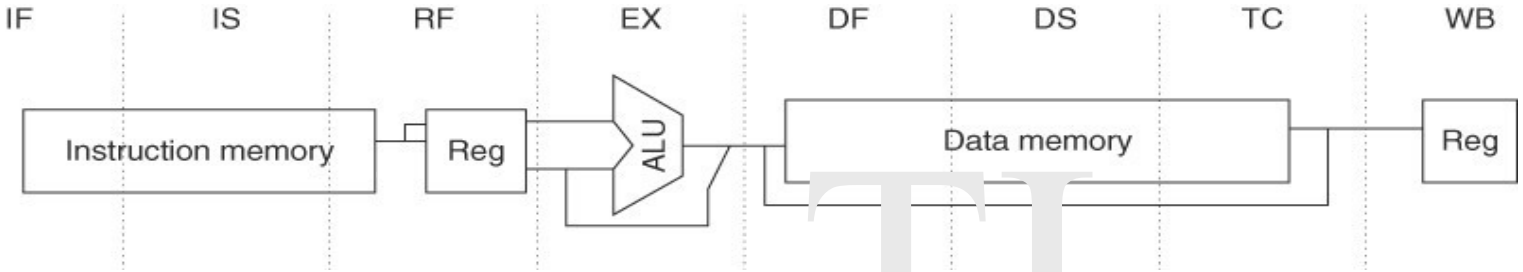
A família de processadores MIPS R4000 implementa o conjunto de instruções MIPS64, mas implementa um gasoduto mais profundo do que o gasoduto clássico de 5 fases que foi descrito, tanto para programas inteiros como de PF.

A conduta mais profunda permite ao processador alcançar uma taxa de relógio mais elevada ao fazer uma decomposição em oito fases, em vez de cinco. Uma vez que o acesso à memória, mesmo através da utilização de caches, é particularmente crítico em termos de tempo, as fases adicionais da tubagem estão relacionadas com a decomposição do acesso à memória. Este tipo de tubagem mais profunda é por vezes chamada *superpipelinagem*.

Gasoduto MIPS R4000 - 117

R4000 organização de oleodutos em oito fases

Fonte: Arquitectura de Computadores: Uma Abordagem Quantitativa



SE - primeira metade do fetch de instrução

RF - segunda metade do fetch de instrução + descodificação de instruções e registo fetch +

verificação dos perigos

EX - execução, incluindo operação de ALU, cálculo eficaz de endereços e alvos de sucursais e avaliação de condições

DF - primeira metade de data fetch DS - segunda metade de data fetch TC - data cache hit detection WB - registo write back

Gasoduto MIPS R4000 - 118

Embora a instrução e os acessos à memória de dados ocupem múltiplos ciclos de relógio, são totalmente canalizados e, portanto, uma nova instrução pode começar em cada ciclo de relógio. De facto, como se pode notar, uma vez que a detecção de acerto de cache tem lugar na última fase de acesso à memória, o pipeline tenta utilizar os dados mesmo antes de a detecção de acerto estar concluída. Se ocorrer um erro, a conduta é apoiada por um ciclo quando os dados correctos estão disponíveis.

Deve-se notar que este gasoduto de latência mais longa não só aumenta a lógica de *encaminhamento* necessária, como também aumenta a *carga* e os atrasos *dos ramos*. O atraso da *carga* é de dois ciclos, uma vez que os dados estão disponíveis no final da fase de DS. O atraso da *ramificação* é de 3 ciclos, uma vez que a condição de ramificação é aqui computada durante a fase EX. A arquitectura R4000, no entanto, utiliza um esquema de *ramificação com* um único ciclo de *atraso*, juntamente com uma estratégia *de junção prevista*, que não produz ciclos ociosos, quando a ramificação não é tomada, e dois ciclos ociosos, quando a ramificação é tomada.

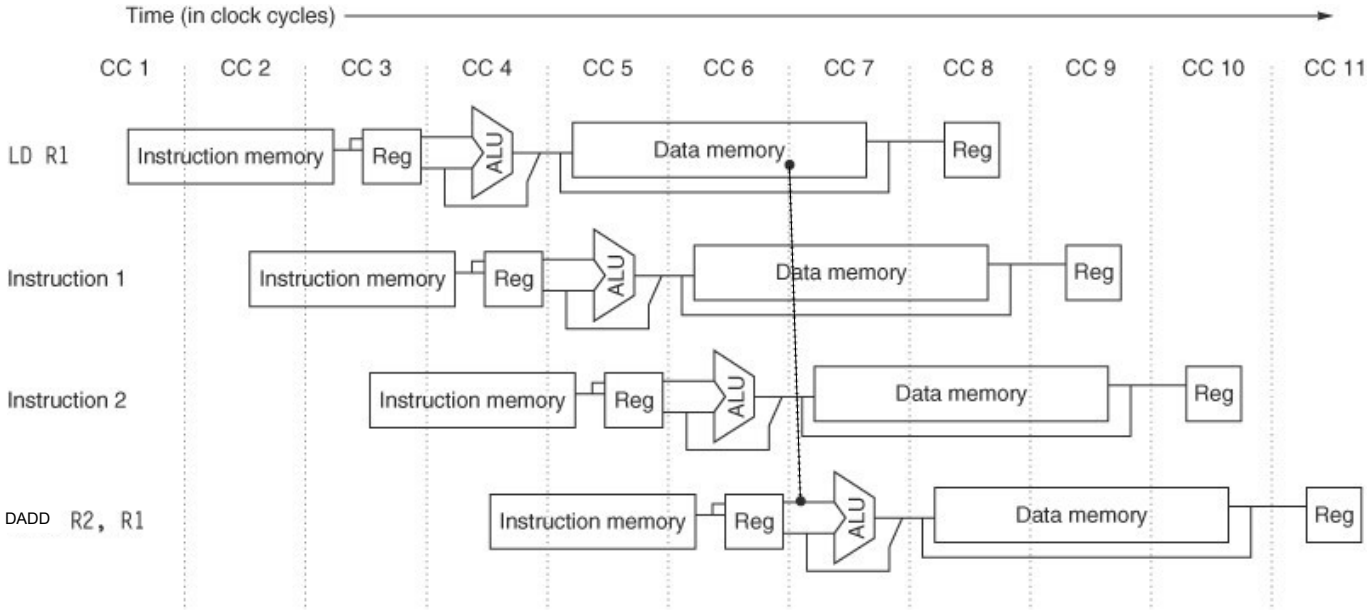
Gasoduto MIPS R4000 - 119

Os encravamentos de tubagem aplicam tanto a penalidade de paragem de 2 ciclos, quando o ramo é tomado, como qualquer risco de dados que surja da ocorrência de uma instrução de carga.

Gasoduto MIPS R4000 - 120

Atraso de dois ciclos de instrução de carga na organização dos oleodutos em oito fases

Fonte: Arquitectura de Computadores: Uma Abordagem Quantitativa



Gasoduto MIPS R4000 - 121

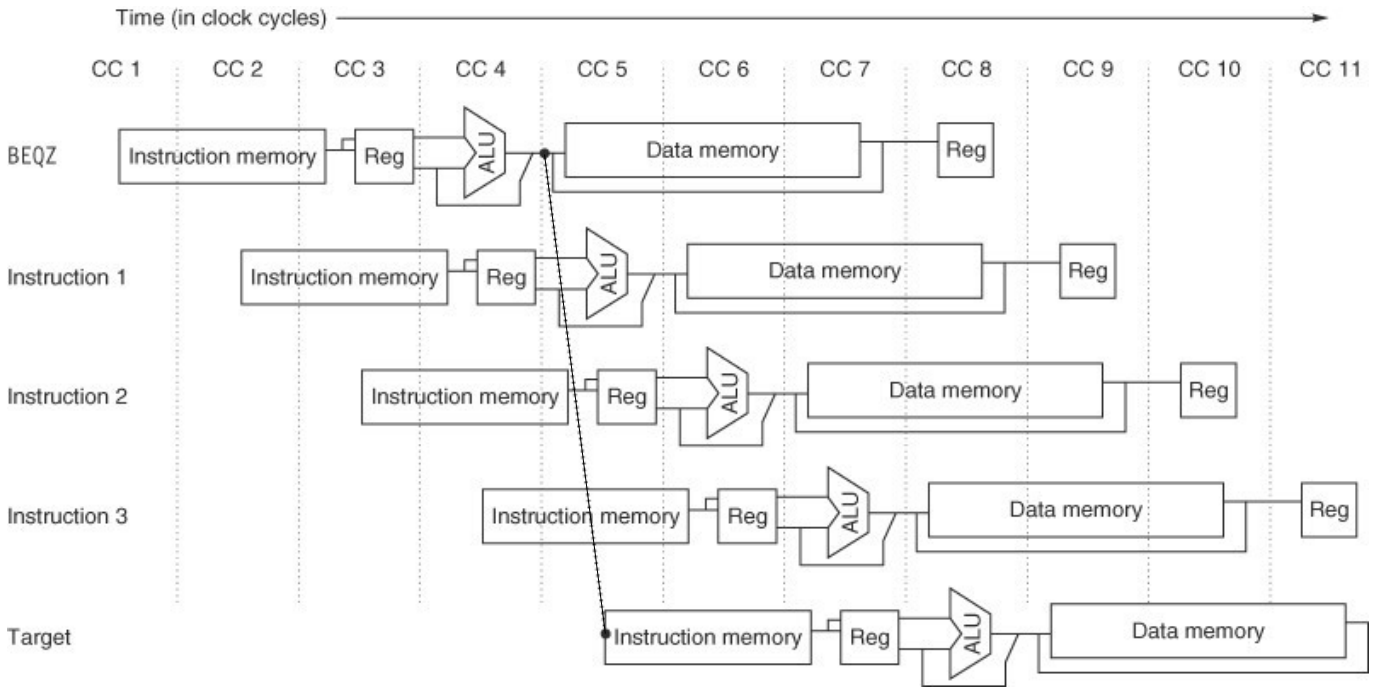
Dois ciclos de paragem produzidos por uma instrução de carga, seguida da utilização imediata do valor carregado, na organização de oito fases do gasoduto

	Número do relógio								
Instrução	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***
`LDR1,0(R10)`	SE	IS	RF	EX	DF	DS	TC	WB	
`DAAD R2,R1,R2`		SE	IS	RF	*está bulo*	*está bulo*	EX	DF	DS
`DSUB R3,R1,R3`			SE	IS	*está bulo*	*está bulo*	RF	EX	DF
`ORR4,R1,R4`				SE	*está bulo*	*está bulo*	IS	RF	EX

Gasoduto MIPS R4000 - 122

Atraso de três ciclos básicos de ramificação na organização dos gasodutos em oito fases

Fonte: Arquitectura de Computadores: Uma Abordagem Quantitativa



Gasoduto MIPS R4000 - 123

Comportamento atrasado do ramo, tanto para os casos *levados* como *não levados,* na organização dos oleodutos em oito fases

	Número do relógio								
Instrução	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***
instrução do ramo	SE	IS	RF	EX	DF	DS	TC	WB	
ranhura de atraso		SE	IS	RF	EX	DF	DS	TC	WB
ciclo ocioso			*está bulo*	*está bulo*	*estábulo*	*está bulo*	*está bulo*	*está bulo*	*está bulo*
ciclo ocioso				*está bulo*	*estábulo*	*está bulo*	*está bulo*	*está bulo*	*está bulo*
alvo do ramo					SE	IS	RF	EX	DF

	Número do relógio								
Instrução	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***
instrução do ramo	SE	IS	RF	EX	DF	DS	TC	WB	
ranhura de atraso		SE	IS	RF	EX	DF	DS	TC	WB
instrução do ramo + 2			SE	IS	RF	EX	DF	DS	TC
instrução do ramo + 3				SE	IS	RF	EX	DF	DS

Gasoduto MIPS R4000 - 124

A unidade de ponto flutuante MIPS R4000 consiste em três unidades funcionais nominais: uma víbora de ponto flutuante, um multiplicador de ponto flutuante e um divisor de ponto flutuante. A lógica da víbora é utilizada nas fases finais de uma operação de multiplicação ou de divisão.

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

Cada unidade funcional pode ser pensada como tendo até oito fases de processamento diferentes. Existe uma única cópia de cada uma destas fases de processamento e instruções diferentes podem utilizar uma determinada fase zero ou mais vezes e combiná-las na sua própria ordem.

Gasoduto MIPS R4000 - 125

As oito fases de processamento nos oleodutos de ponto flutuante R4000

Fonte: Arquitectura de Computadores: Uma Abordagem Quantitativa

Fase de processamento	***Unidade funcional elementar***	***Descrição***
A	Víbora FP	mantissa fase ADD
D	Divisória FP	fase de gasoduto de divisão
E	Multiplicador de FP	fase de teste de excepção
M	Multiplicador de FP	multiplicador primeira fase
N	Multiplicador de FP	multiplicador segunda fase
R	Víbora FP	fase de arredondamento
S	Víbora FP	fase de mudança de operando
U		desempacotar números de ponto flutuante

Gasoduto MIPS R4000 - 126

Latências, intervalos de repetição e composição de fases para operações de ponto flutuante em R4000

Fonte: Arquitectura de Computadores: Uma Abordagem Quantitativa

FP instrução	Latência	Intervalo de repetição	Fases de processamento
adicionar - subtrair	4	3	U - S+A- A+R - R+S
multiplicar	8	4	U - E+M - M - M - M - M - N - N+A- R
dividir	36	35	U - A- R - D28 - D+A - D+R - D+A - D+R - D+R - A- R
raiz quadrada	112	111	U - E - $(A+R)^{108}$ - A- R
negar	2	1	U - S
valor absoluto	2	1	U - S
comparar	3	2	U - A- R

Gasoduto MIPS R4000 - 127

Efeito de uma instrução de multiplicação FP emitida no ciclo 0 sobre uma instrução de adição FP emitida entre os ciclos de relógio 1 a 7

Fonte:Adaptado de Computer Architecture: Uma Abordagem Quantitativa

		Número do relógio										
Instrução	***Decisão***	***0***	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***
multiplicar	edição	U	E+M	M	M	M	N	N+A	R			
adicionar	edição		U	S+A	A+R	R+S						
	edição			U	S+A	A+R	R+S					
	edição				U	S+A	A+R	R+S				
	estábulo					*estábulo*	*estábulo*	U	S+A	A+R	R+S	
	estábulo						*estábulo*	U	S+A	A+R	R+S	
	edição							U	S+A	A+R	R+S	
	edição								U	S+A	A+R	R+S

Gasoduto MIPS R4000 - 128

Efeito de uma instrução de divisão FP emitida no ciclo 0 sobre uma instrução de adição FP emitida entre os ciclos do relógio 26 a 36

Fonte: Adaptado de Computer Architecture: Uma Abordagem Quantitativa

		Número do relógio										
Instrução	***Decisão***	***26***	***27***	***28***	***29***	***30***	***31***	***32***	***33***	***34***	***35***	***36***
dividir	edição em cc 0	D	D	D	D	D	D+A	D+R	D+A	D+R	A	R
adicionar	edição		U	S+A	A+R	R+S						
	edição			U	S+A	A+R	R+S					
	estábulo				*estábulo*	*estábulo*	*estábulo*	*estábulo*	*estábulo*	*estábulo*	U	S+A
	estábulo					*estábulo*	*estábulo*	*estábulo*	*estábulo*	*estábulo*	U	S+A
	estábulo						*estábulo*	*estábulo*	*estábulo*	*estábulo*	U	S+A
	estábulo							*estábulo*	*estábulo*	*estábulo*	U	S+A
	estábulo								*estábulo*	*estábulo*	U	S+A
	estábulo									*estábulo*	U	S+A
	edição										U	S+A
	edição											U

Gasoduto MIPS R4000 - 129

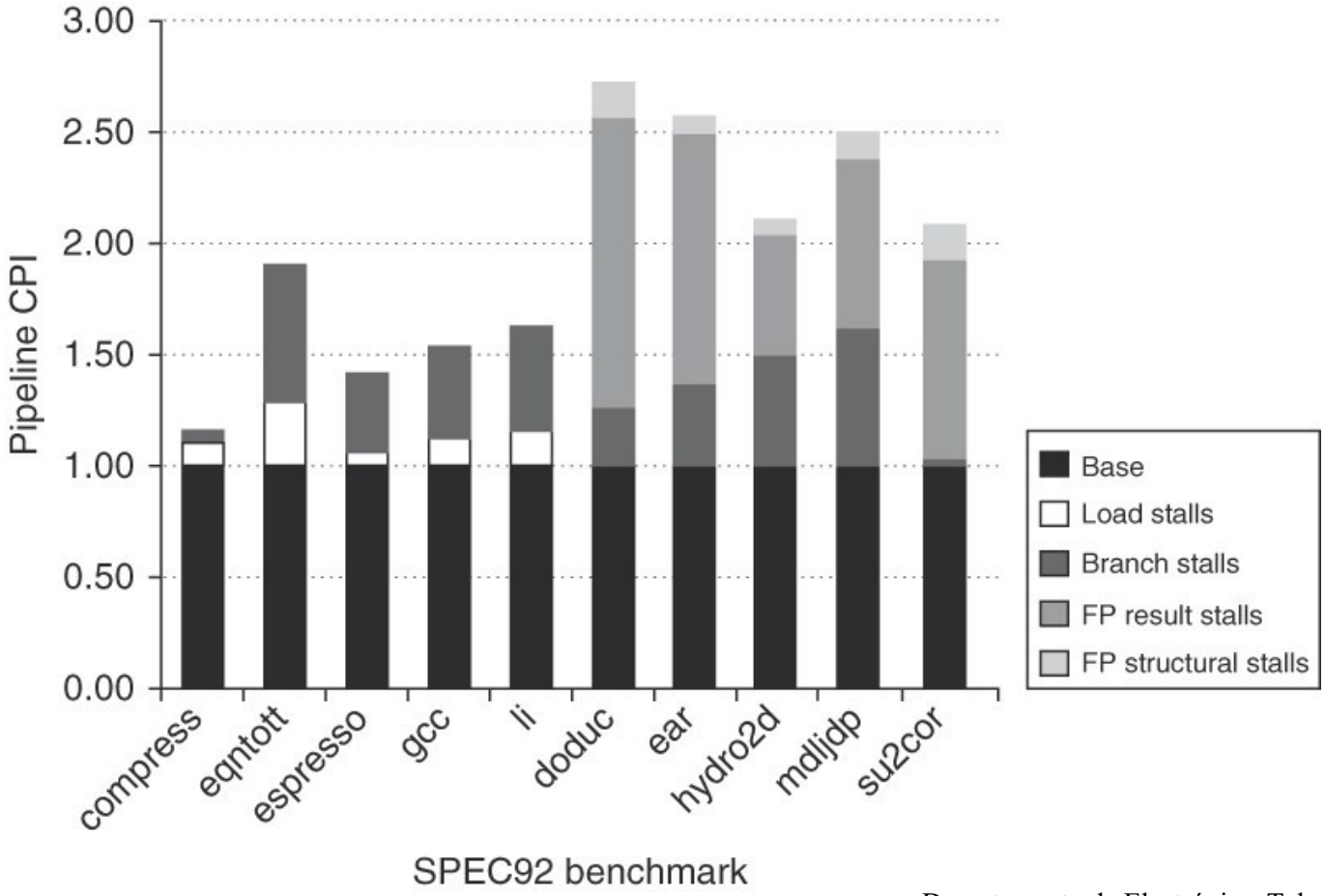
Existem quatro causas principais de barracas de oleodutos ou perdas que impedem que a emissão de instruções à taxa nominal seja atingida

- *paragens de carga* - atrasos decorrentes da utilização do valor da carga um ou dois ciclos de relógio após a execução da carga
- *barracas e perdas de ramos* - dois atrasos de ciclo de relógio após cada ramo *tomado* e um atraso de ciclo de relógio se o atraso do ramo não puder ser preenchido com uma instrução útil
- *Paragens de resultados FP* - atrasos que surgem porque é necessário um operando e ainda não é calculado
- *Paragens estruturais de PF* - atrasos que surgem porque as fases de processamento necessárias na conduta de PF não estão disponíveis quando necessário.

Gasoduto MIPS R4000 - 130

O CPI do gasoduto para 10 das referências da SPEC92, assumindo um cache perfeito

Fonte: Arquitectura de Computadores: Uma Abordagem Quantitativa



Gasoduto MIPS R4000 - 131

O gasoduto R4000 tem atrasos de ramificação muito maiores do que o clássico gasoduto de 5 fases. O atraso mais longo dos ramos aumenta substancialmente os ciclos do relógio gastos nos ramos, especialmente para programas inteiros com uma elevada frequência de ramos.

Um efeito interessante para os programas de PF é que a latência das unidades funcionais do PF conduz a mais barracas de resultados do que as produzidas pelos riscos estruturais, que se devem tanto às limitações de intervalos de repetição como aos conflitos decorrentes da utilização de fases de processamento específicas em diferentes instruções de PF.

Assim, a redução da latência das operações de PF deve ser a primeira tarefa a ser cuidada num esforço de optimização, em vez de aumentar as condutas ou replicar as fases de processamento das unidades funcionais.

Leitura sugerida

- *Arquitectura informática: A Quantitative Approach*, Hennessy J.L., Patterson D.A., 6ª Edição, Morgan Kaufmann, 2017
  - Apêndice A: *Princípios do Conjunto de Instrução* (Secções 1 a 8)
  - Apêndice C: *Pipelinagem: Conceitos Básicos e Intermediários* (Secções 1 a 7)
- *Organização e Arquitectura de Computadores: Designing for Performance*, Stallings W., 10ª Edição, Pearson Education, 2016
  - 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 de Conjunto de Instrução Reduzida*