> “Enquanto a calculadora do ENIAC possui 18.000 tubos a vácuo e pesa 30 toneladas, no futuro poderemos ter computadores com apenas 1.000 tubos e pesando apenas 1,5 toneladas” (Popular Mechanics, 1949).

Despacho Múltiplo Dinâmico

Paulo Ricardo Lisboa de Almeida

DSBD

Alguns trechos desses slides foram baseados nas aulas de Marco Zanata: web.inf.ufpr.br/mazalves/dis-arquitetura-de-computadores

UFPR
UNIVERSIDADE FEDERAL DO PARANÁ

Especulação

Para aumentar a eficiência da CPU através do paralelismo, é necessária **especulação**.

O processador precisa fazer “adivinhações” sobre as instruções.

Possibilitar a execução de outras instruções que podem depender da instrução especulada.

Já fizemos algo assim no pipeline. Onde?

Especulação

Para aumentar a eficiência da CPU através do paralelismo, é necessária **especulação**.

O processador precisa fazer “adivinhações” sobre as instruções.

Possibilitar a execução de outras instruções que podem depender da instrução especulada.

A unidade de predição de desvios implementada está fazendo **especulação**.

Especulação

Para aumentar a eficiência da CPU através do paralelismo, é necessária **especulação**.

O processador precisa fazer "adivinhações" sobre as instruções.

Possibilitar a execução de outras instruções que podem depender da instrução especulada.

A unidade de predição de desvios implementada está fazendo **especulação**.

Outro exemplo: especular que a sequência *store -> load* não se refere ao mesmo endereço na memória.

Possibilitar que o load seja executado antes de um store, ou em paralelo.

Especulação

Qual o principal problema da especulação?

Especulação

Qual o principal problema da especulação?

Podemos estar errados!

Necessários mecanismos extras para:

- Verificar se a especulação estava correta.
- Mecanismos de rollback (desfazer o que foi feito) caso a especulação esteja incorreta.

Especulação

A especulação pode ser feita via hardware ou software (e.g., compilador).

Por exemplo, o compilador pode mover uma instrução para fora de uma condicional.

Adiciona bits de flag para verificar se a especulação estava correta, caso contrário a operação é descartada.

Exemplo: processadores IA-64 (Itanium) usam uma combinação de especulação por software e hardware.

Despacho múltiplo dinâmico

Processadores com **Despacho Múltiplo Dinâmico**.

Conhecidos também como **superescalares**.

Despacho múltiplo dinâmico

Em processadores simples.

O superescalar escolhe se no próximo ciclo de clock é possível enviar:

0 instruções, ou;

1 instrução, ou;

2 instruções, ...

Qual o papel do compilador?

Despacho múltiplo dinâmico

Em processadores simples.

O superescalar escolhe se no próximo ciclo de clock é possível enviar:

0 instruções, ou;

1 instrução, ou;

2 instruções, ...

Qual o papel do compilador?

O compilador deve ordenar as instruções de forma a reduzir ao máximo as dependências.

Possibilitar que o processador despache múltiplas instruções em um ciclo.

Diferenças para VLIW

1. Em um superescalar, a execução correta das instruções é **garantida pela CPU.**

   Em um VLIW, é papel do compilador montar as instruções de forma a resolver as dependências.

Diferenças para VLIW

1. Em um superescalar, a execução correta das instruções é **garantida pela CPU.**

   Em um VLIW, é papel do compilador montar as instruções de forma a resolver as dependências.

2. Superescalares são menos dependentes do número de unidades funcionais.

   Em alguns VLIW, pode ser necessária uma tradução ou recompilação.

Superescalar

```
        ori $s0,$zero,0
        lui $s0,0x1000
        ori $s1,$zero,396
loop:
        addi $s2,$s0,$s1
        addi $s3,$s2,-4
        lw $t0,0($s2)
        addi $t0,$t0,1
        lw $t1,0($s3)
        addi $t1,$t1,1
        sw $t0,0($s2)
        addi $s1,$s1,-8
        sw $t1,0($s3)
        bne $s1,$zero,loop
```

VLIW


```
        ori $s0,$zero,0              nop
        lui $s0,0x1000               nop
        ori $s1,$zero,396       nop
loop:
        addi $s2,$s0,$s1             nop
        addi $s3,$s2,-4              lw $t0,0($s2)
        addi $t0,$t0,1               lw $t1,0($s3)
        addi $t1,$t1,1               sw $t0,0($s2)
        addi $s1,$s1,-8              sw $t1,0($s3)
        bne $s1,$zero,loop      nop
```

Escalonamento dinâmico em pipeline

Muitos superescalares usam uma combinação de despacho múltiplo dinâmico com **escalonamento dinâmico de pipeline**.

Escalonamento dinâmico em pipeline

Qual o problema com a sequência de instruções a seguir? O que o compilador/programador poderia fazer?

```
lw $s0,0($t0)
add $s1,$s0,$t2
lw $$2,4($t0)
```

Escalonamento dinâmico em pipeline

Qual o problema com a sequência de instruções a seguir? O que o compilador/programador poderia fazer?

```
lw $s0,0($t0)
add $s1,$s0,$t2
lw $$2,4($t0)
```

O add gera um stall, pois precisa esperar o primeiro lw. Um compilador/programador inteligente reordenaria as instruções para evitar o stall.

```
lw $s0,0($t0)
lw $$2,4($t0)
add $s1,$s0,$t2
```

Escalonamento dinâmico em pipeline

Problemas de depender do compilador/programador?

```
lw $s0,0($t0)
lw $$2,4($t0)
add $s1,$s0,$t2
```

Escalonamento dinâmico em pipeline

Problemas de depender do compilador/programador?

- Adicionamos complexidade para o compilador/programador.
    Podem cometer erros ou simplesmente ignorar as otimizações.
- As otimizações feitas para um processador podem não servir para outro.
    Pode exigir recompilação.
    No exemplo, a mudança da ordem das instruções serve para o pipeline de 5 estágios do MIPS32.

```
lw $s0,0($t0)
lw $$2,4($t0)
add $s1,$s0,$t2
```

Escalonamento dinâmico em pipeline

Um processador com escalonamento dinâmico de pipeline **troca a ordem das instruções internamente para tentar evitar stalls**.

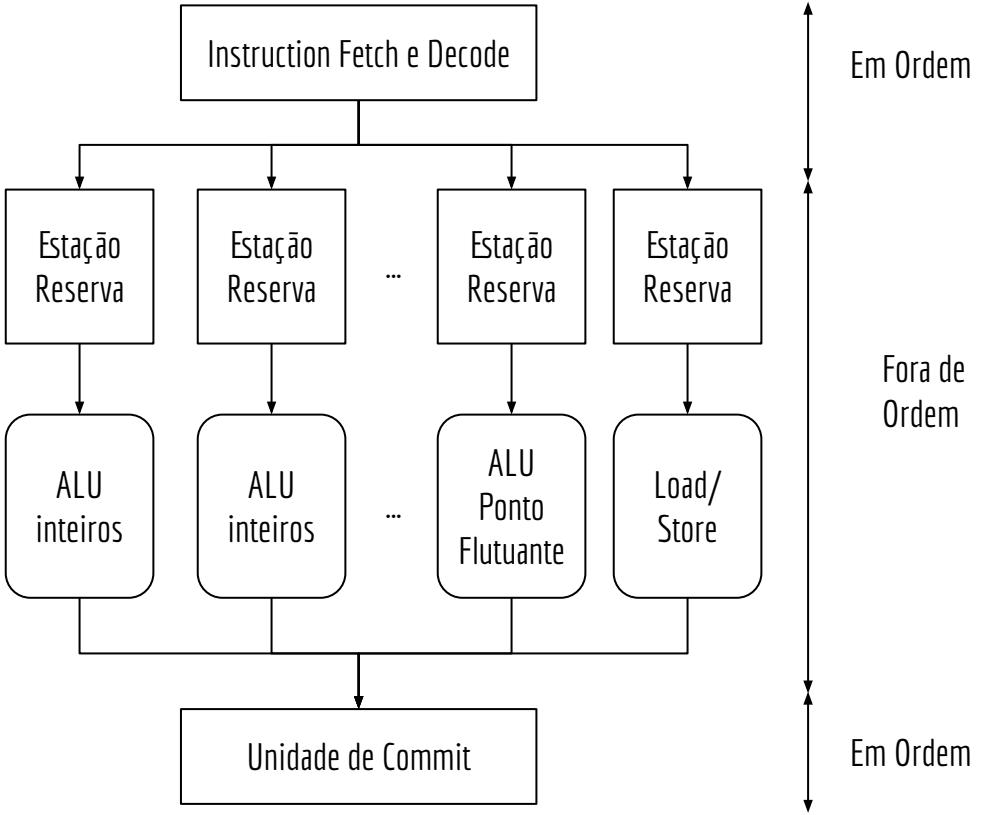
A troca é feita de forma que o programa pareça ser executado na ordem original das instruções.

Execução fora de ordem.

```
lw $s0,0($t0)
add $s1,$s0,$t2
lw $$2,4($t0)
```

```
lw $s0,0($t0)
lw $$2,4($t0)
add $s1,$s0,$t2
```

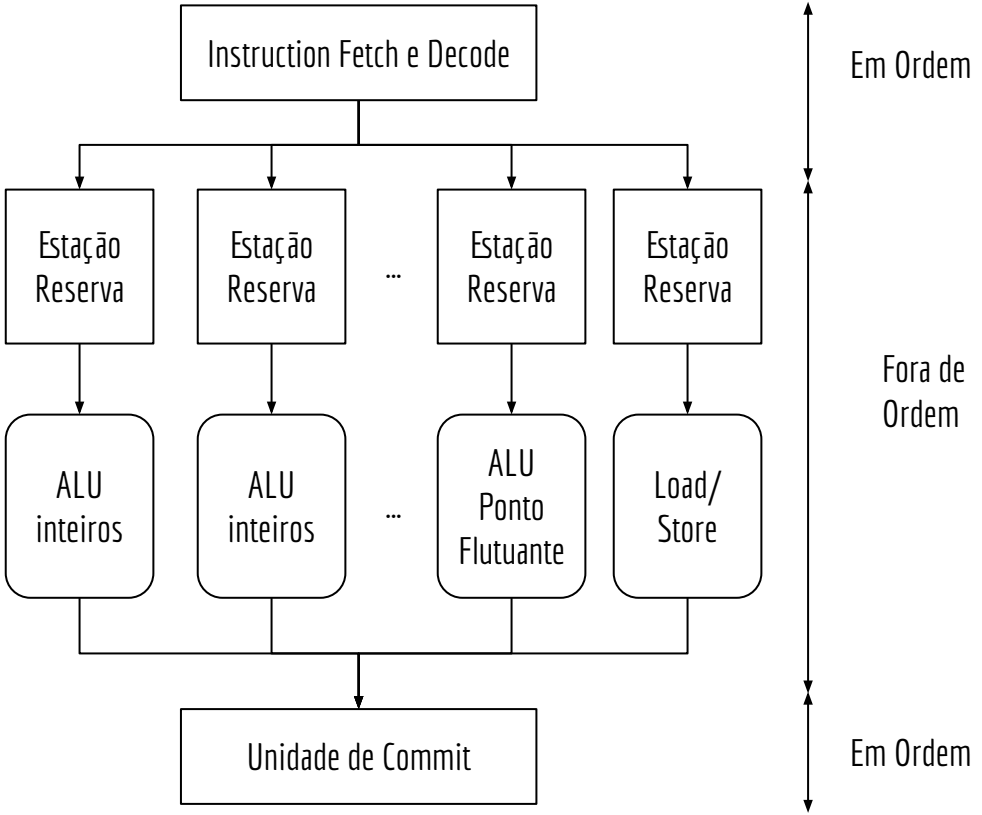
Ideia


Instruction Fetch e Decode
Em Ordem
Estação Reserva
Estação Reserva
...
Estação Reserva
Estação Reserva
Fora de Ordem
ALU inteiros
ALU inteiros
...
ALU Ponto Flutuante
Load/ Store
Unidade de Commit
Em Ordem

Instruction Fetch e Decode

As instruções chegam **em ordem** na unidade de instruction fetch e decode.

```
lw $s0,0($t0)
add $s1,$s0,$t2
add $s2,$s3,$s4
```

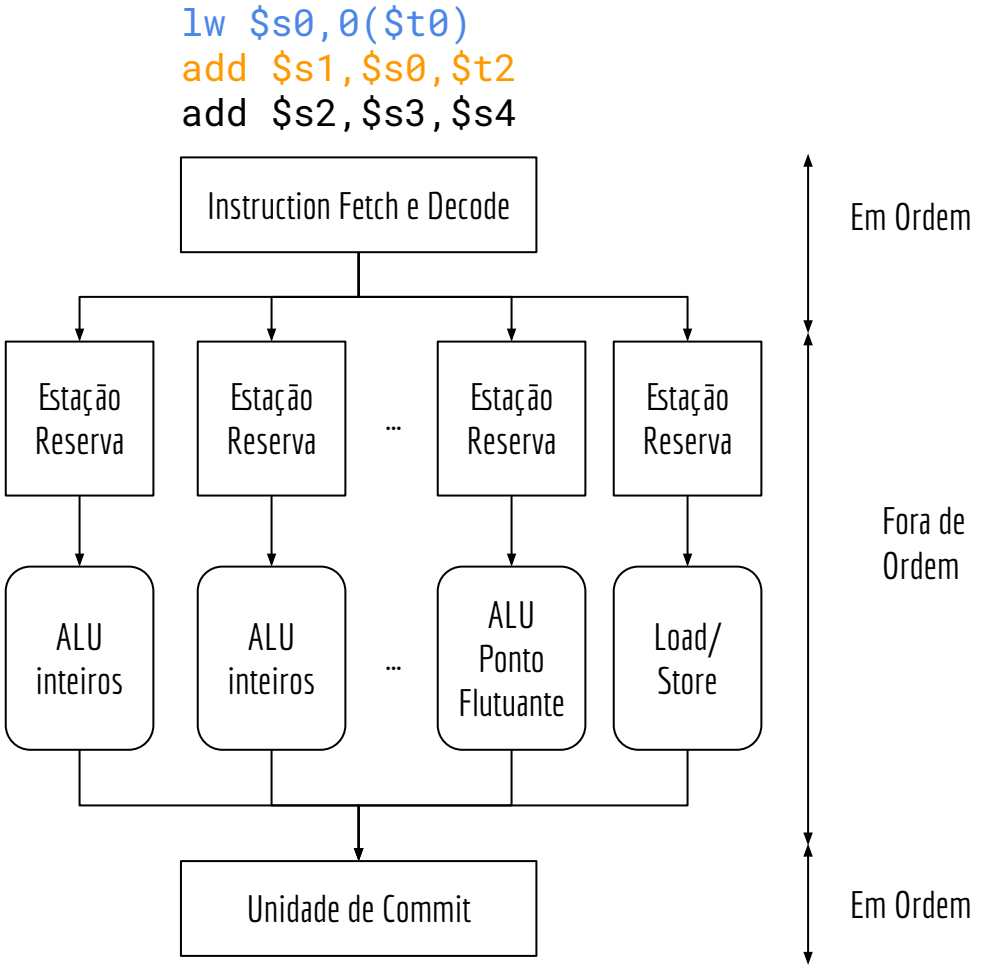


Estações Reserva

A CPU possui múltiplas unidades funcionais (superescalar).
Cada unidade funcional é associada a uma **estação de reserva**.

As estações reserva são buffers que armazenam as instruções enquanto:

- A unidade funcional estiver ocupada;
- A instrução ainda não tem os dados necessários para executar.

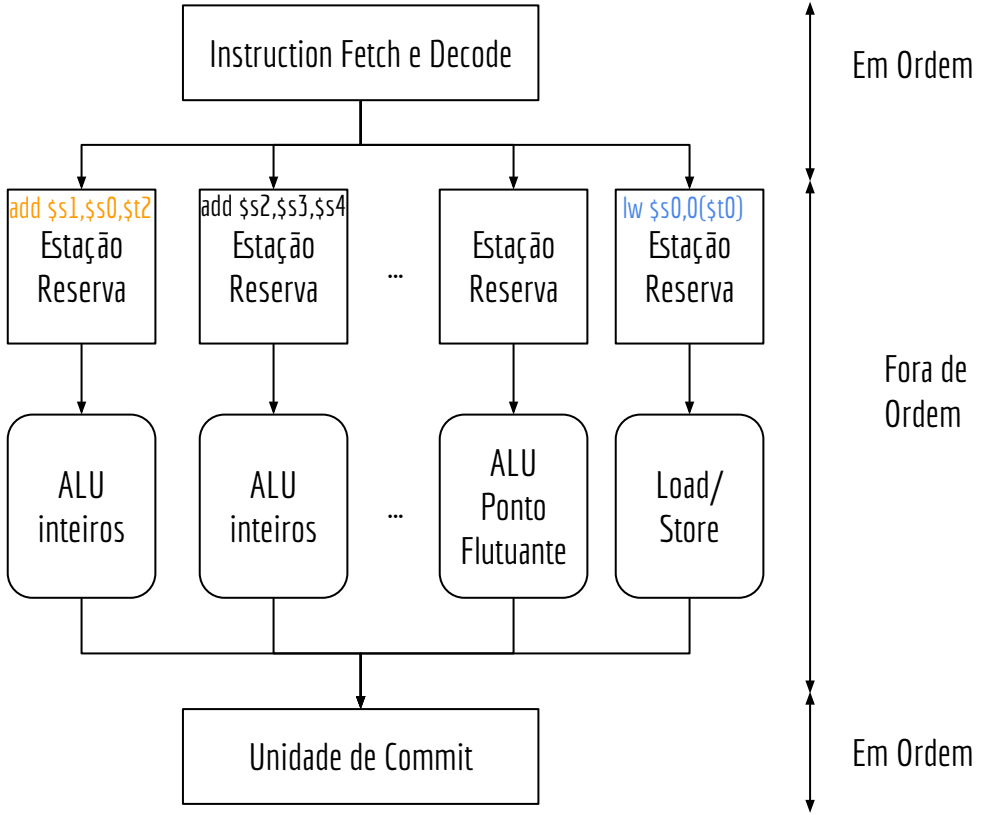


Estações Reserva

A CPU possui múltiplas unidades funcionais.
Cada unidade funcional é associada a uma **estação de reserva**.

As estações reserva são buffers que armazenam as instruções enquanto:

- A unidade funcional estiver ocupada;
- A instrução ainda não tem os dados necessários para executar.




```
lw $s0,0($t0)
add $s1,$s0,$t2
add $s2,$s3,$s4
```

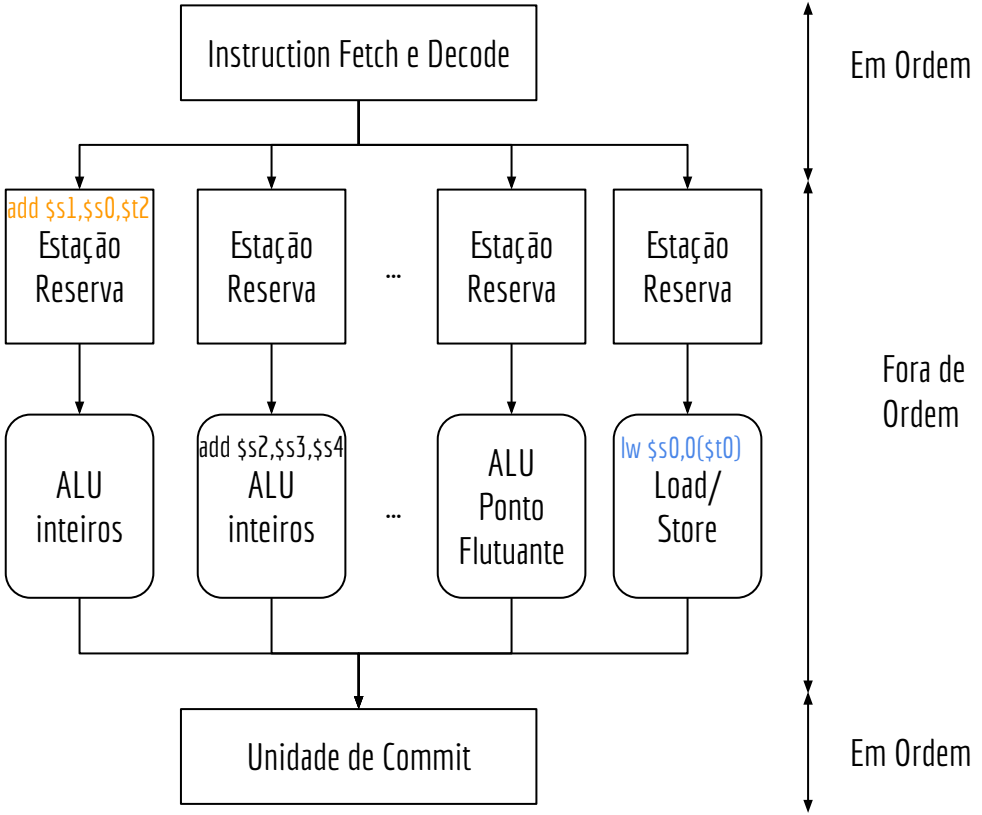
Estações Reserva

A CPU possui múltiplas unidades funcionais.
Cada unidade funcional é associada a uma **estação de reserva**.

As estações reserva são buffers que armazenam as instruções enquanto:

- A unidade funcional estiver ocupada;
- A instrução ainda não tem os dados necessários para executar.

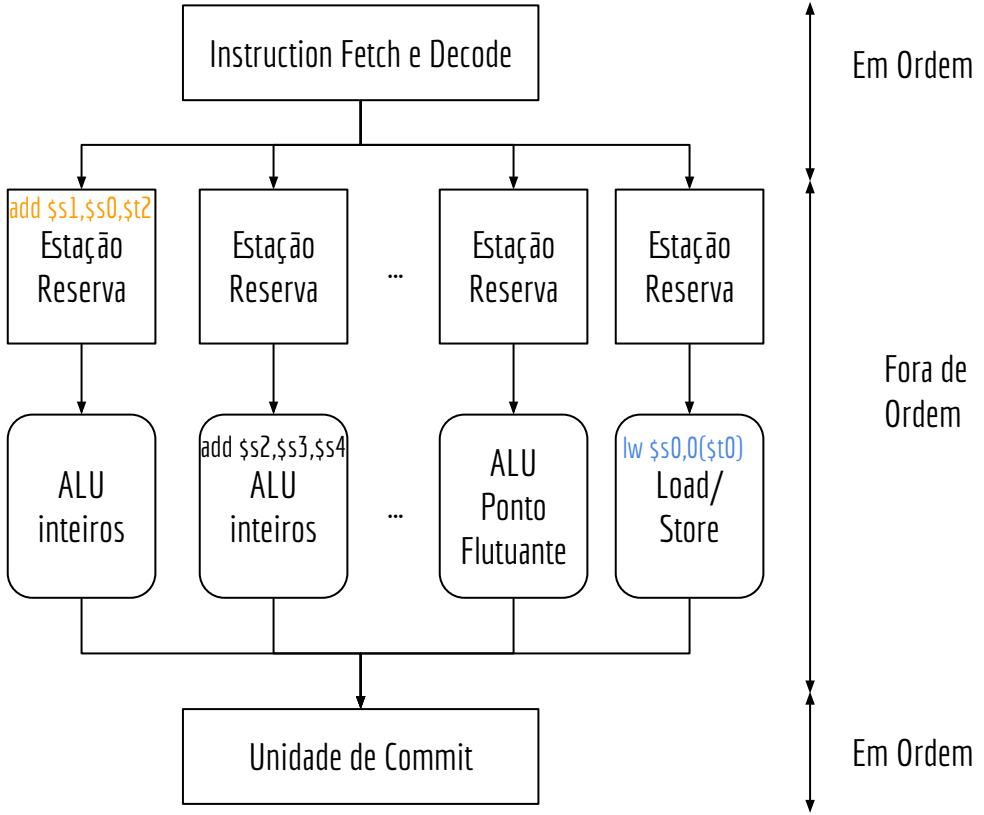
```
lw $s0,0($t0)
add $s1,$s0,$t2
add $s2,$s3,$s4
```



Estações Reserva

A **unidade de commit** – ou **buffer de reordenação**) faz:

- Mantém os resultados das instruções.
- Manda os resultados para o banco de registradores ou memória **quando é seguro**.
  - O envio (commit) é em ordem.

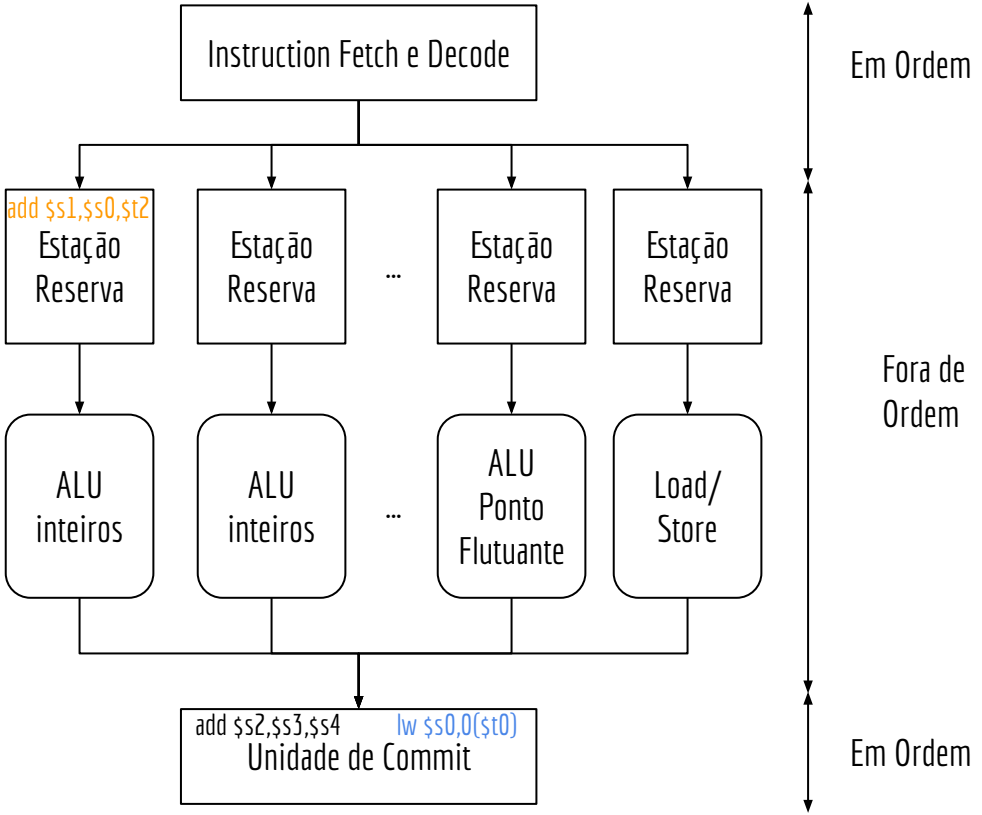



```
lw $s0,0($t0)
add $s1,$s0,$t2
add $s2,$s3,$s4
```

Estações Reserva

A **unidade de commit** – ou **buffer de reordenação**) faz:

- Mantém os resultados das instruções.
- Manda os resultados para o banco de registradores ou memória **quando é seguro**.
  - O envio (commit) é em ordem.



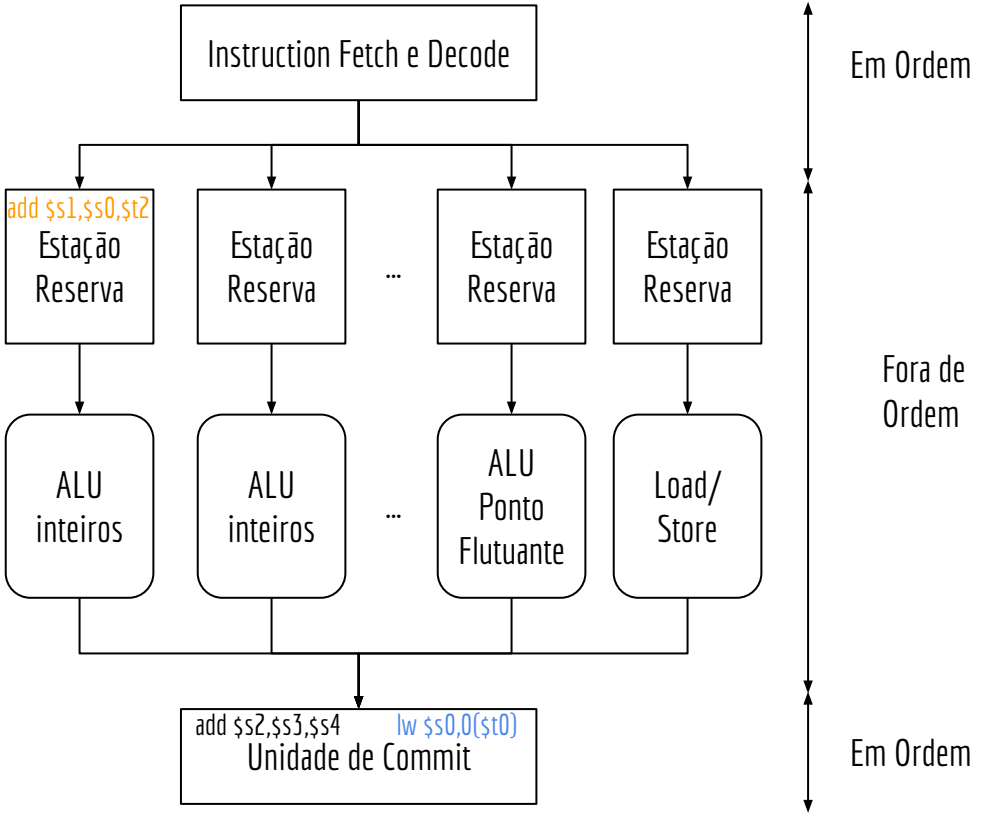

```
lw $s0,0($t0)
add $s1,$s0,$t2
add $s2,$s3,$s4
```

Forwarding

Existem **forwardings** entre todas as unidades de execução, unidade de commit, e as estações reserva.

Os resultados são usados tão logo estão prontos.

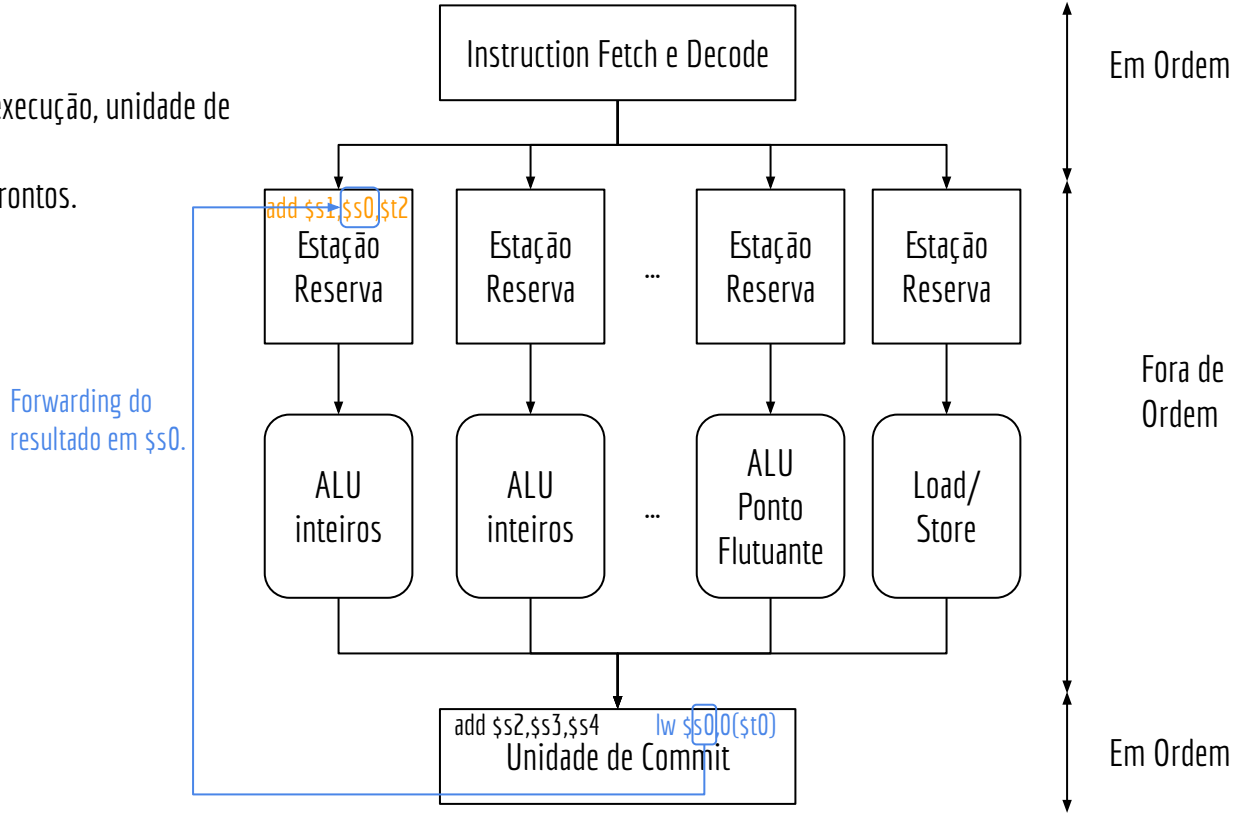
```
lw $s0,0($t0)
add $s1,$s0,$t2
add $s2,$s3,$s4
```



Forwarding

Existem **forwardings** entre todas as unidades de execução, unidade de commit, e as estações reserva.

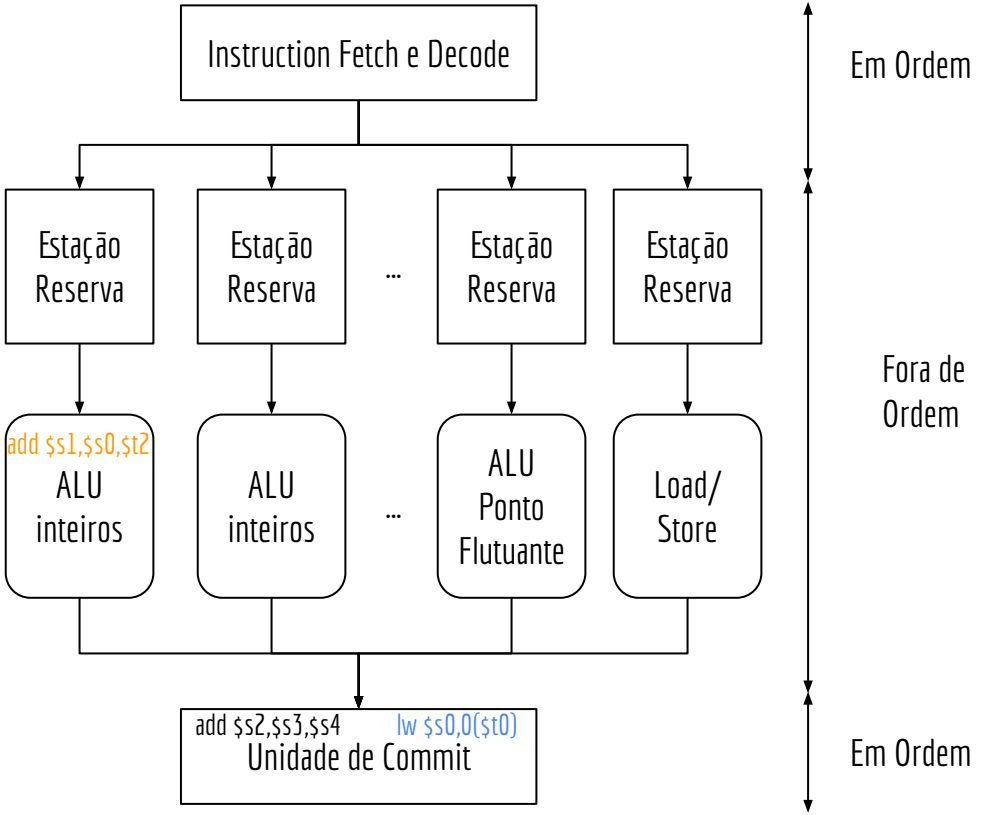
Os resultados são usados tão logo estão prontos.




```
lw $s0,0($t0)
add $s1,$s0,$t2
add $s2,$s3,$s4
```

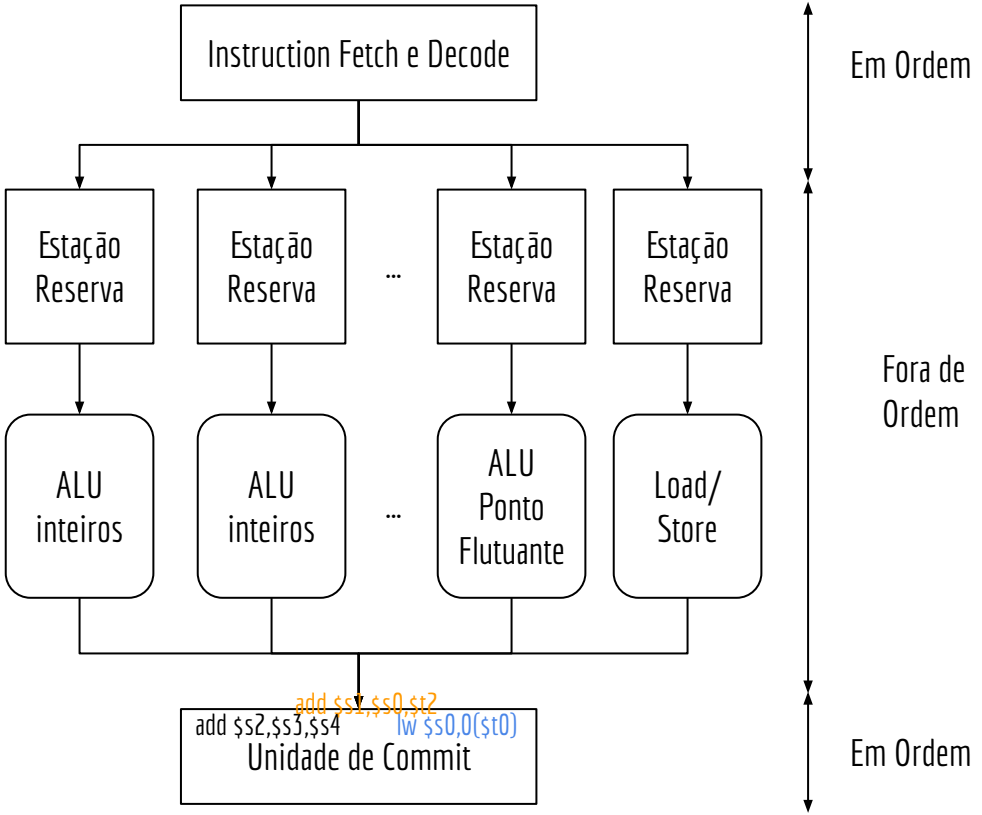
Forwarding

```
lw $s0,0($t0)
add $s1,$s0,$t2
add $s2,$s3,$s4
```



Forwarding

```
lw $s0,0($t0)
add $s1,$s0,$t2
add $s2,$s3,$s4
```



Problemas...

As coisas podem dar (muito) errado.

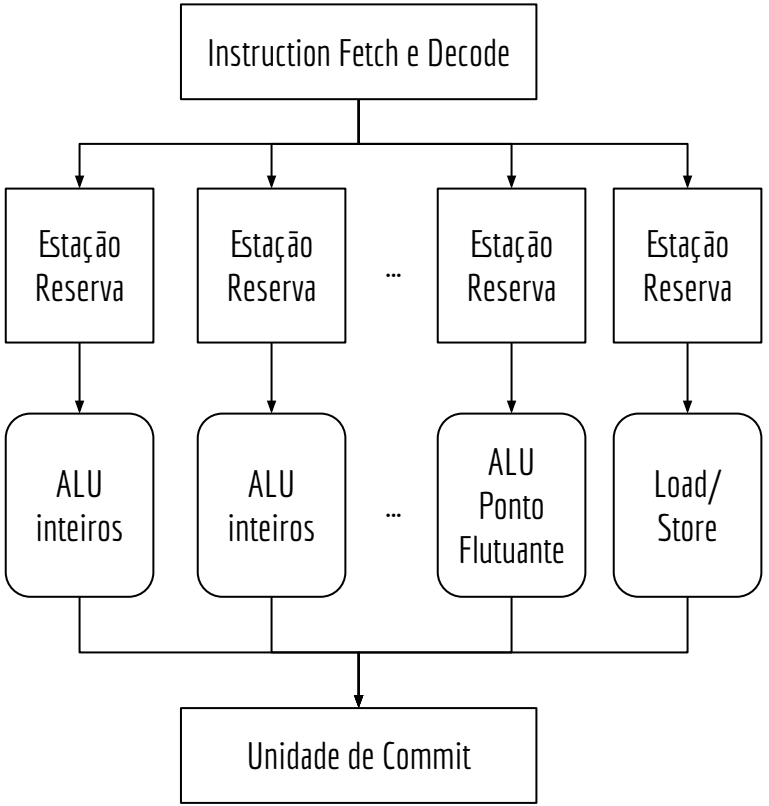
Considere as instruções a seguir, e que:

$s4 possui zero.

Exceções são geradas em divisões por zero.

O que pode dar errado?

```
lw $s0,0($t0)
add $s1,$s0,$t2
div $s2,$s3,$s4
```

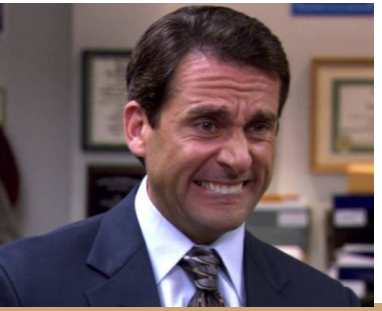


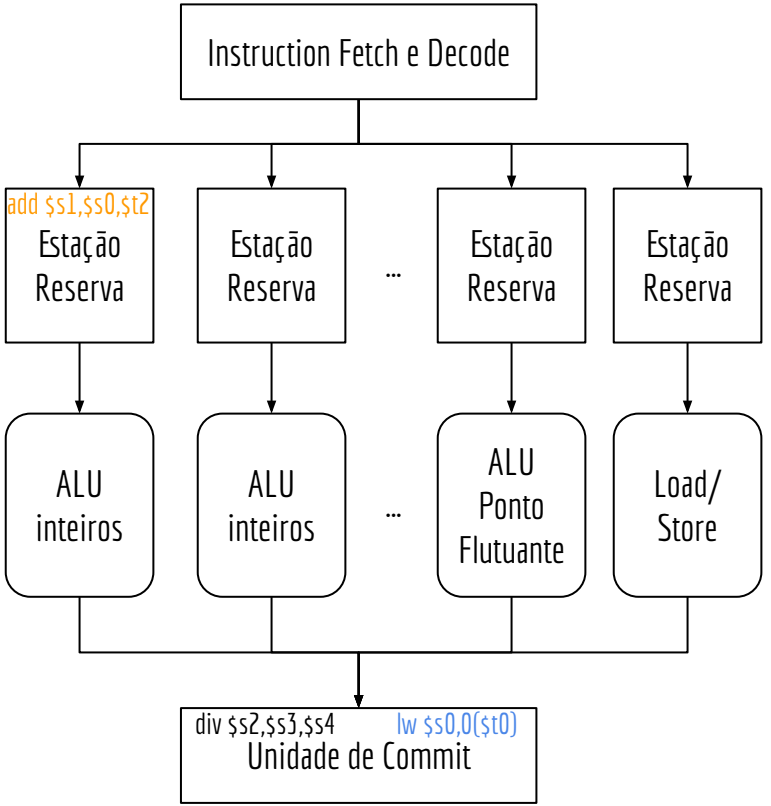
Problemas...

As coisas podem dar (muito) errado.

```
lw $s0,0($t0)
add $s1,$s0,$t2
div $s2,$s3,$s4
```

A terceira instrução vai gerar uma exceção antes da segunda executar. As exceções podem ficar fora de ordem!





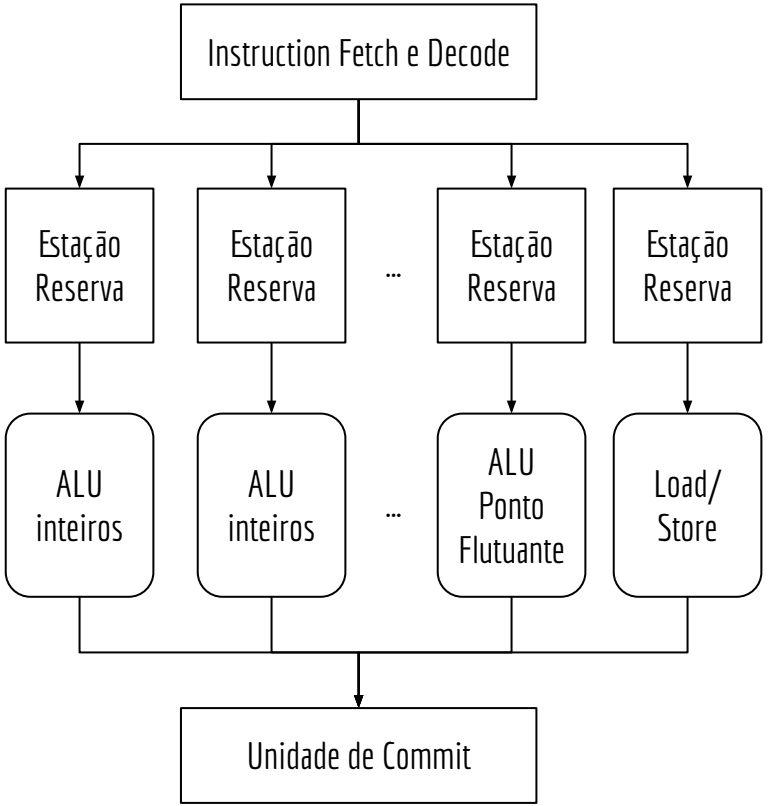
Problemas...

Podemos piorar...

Considere mais uma vez que $s4 possui zero.

E agora, o que pode dar terrivelmente errado?

```
lw $s0,0($t0)
div $s1,$s0,$s4
div $s2,$s3,$s4
```



Problemas...

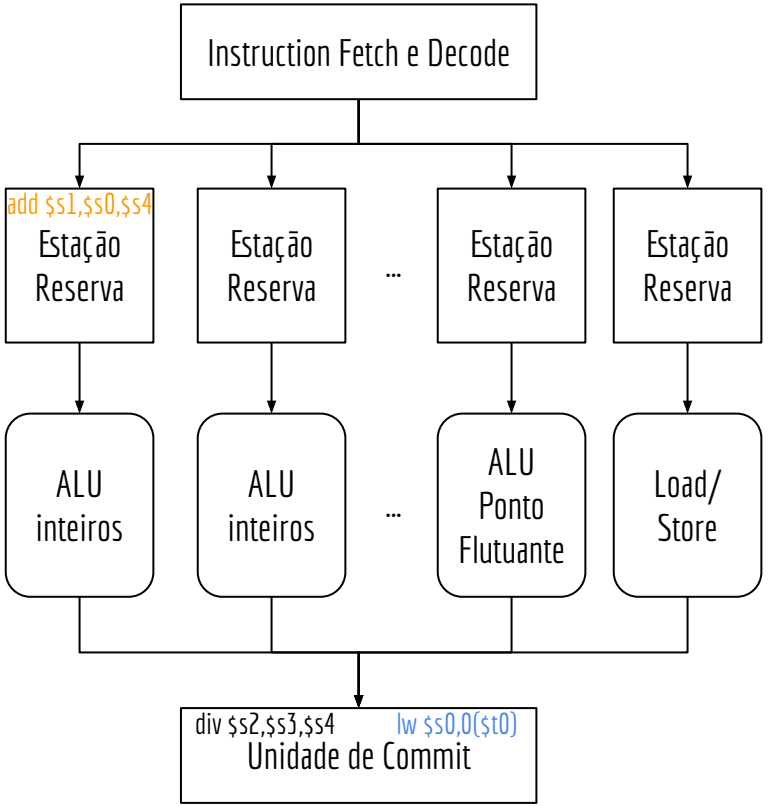
Podemos piorar...

```
lw $s0,0($t0)
div $s1,$s0,$s4
div $s2,$s3,$s4
```

A terceira instrução vai gerar uma exceção.

Mas na ordem original do programa, a segunda instrução geraria uma exceção antes.

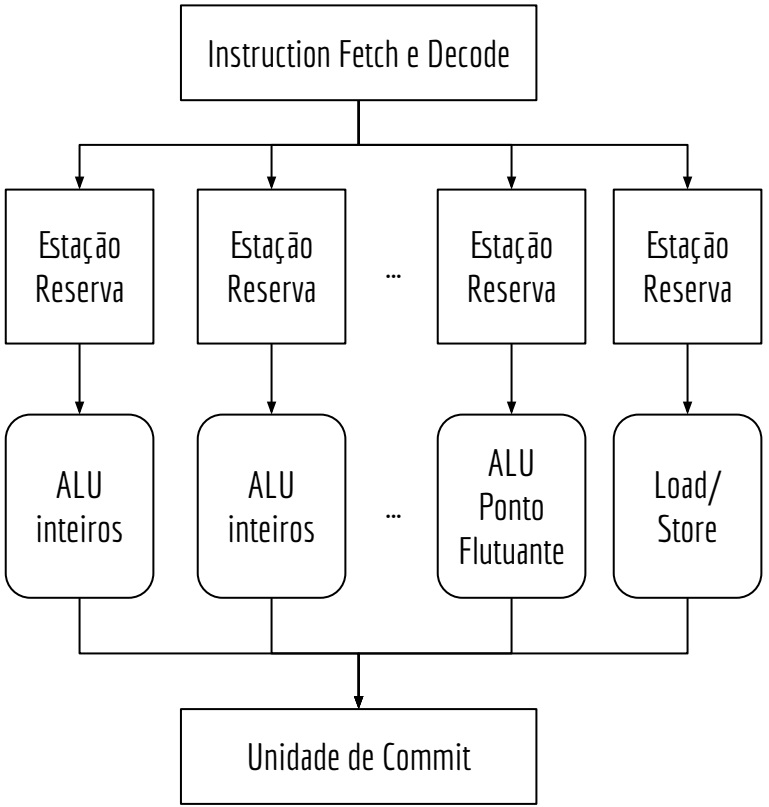
Logo, a terceira instrução não deveria ser executada (nem gerar exceção).



Problemas...

E agora?

```
        lw $s0,0($t0)
        beq $s4,$zero,saida
        div $s2,$s3,$s4
saida:
        ...
```

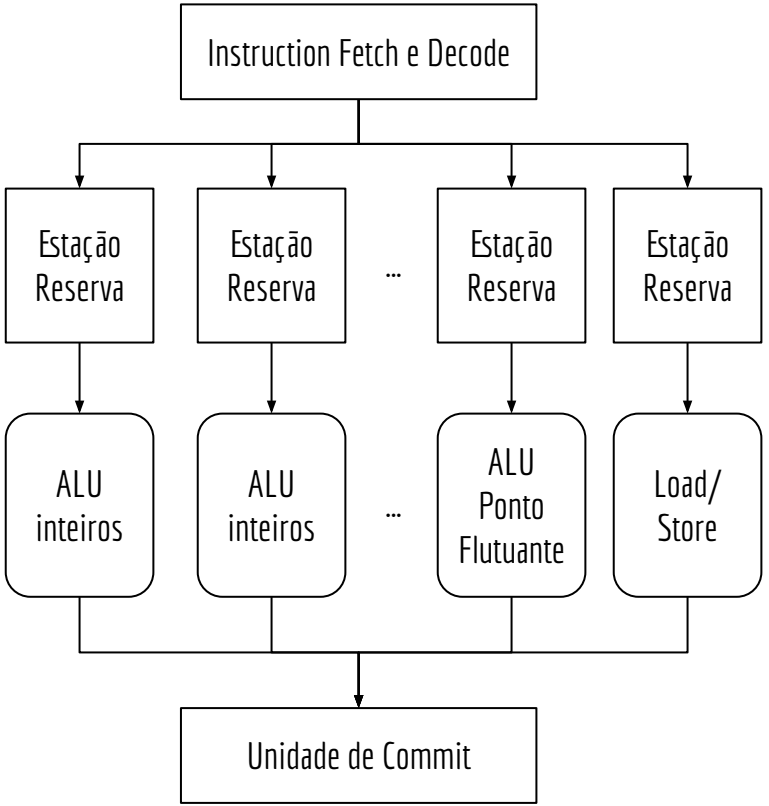


Problemas...

E agora?

```
    lw $s0,0($t0)
    beq $s4,$zero,saida
    div $s2,$s3,$s4
saida:
    …
```

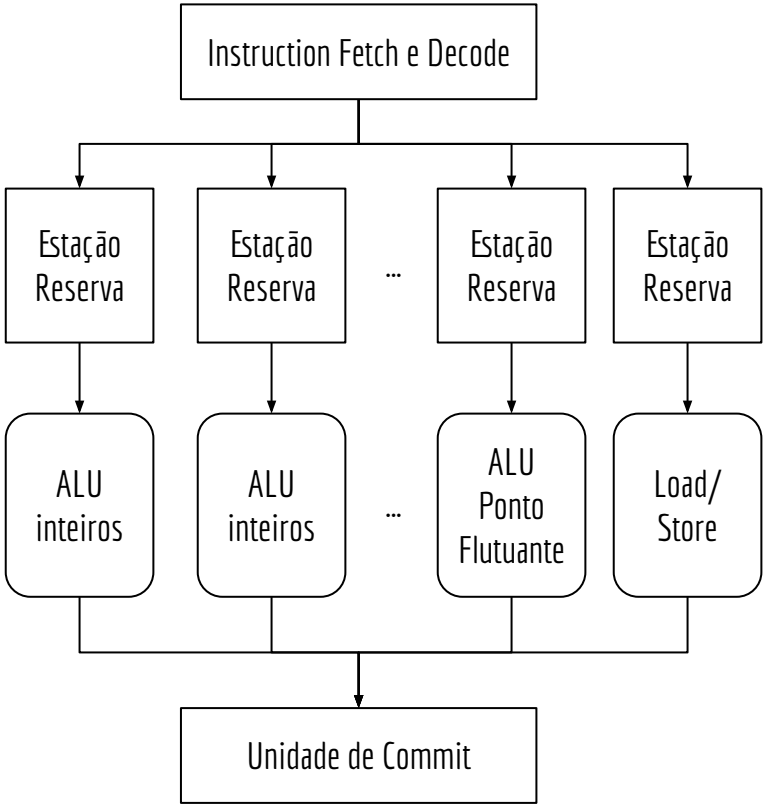
Se a especulação do preditor de desvios estiver incorreto, uma exceção que não deveria existir vai ser gerada.



Unidade de Commit

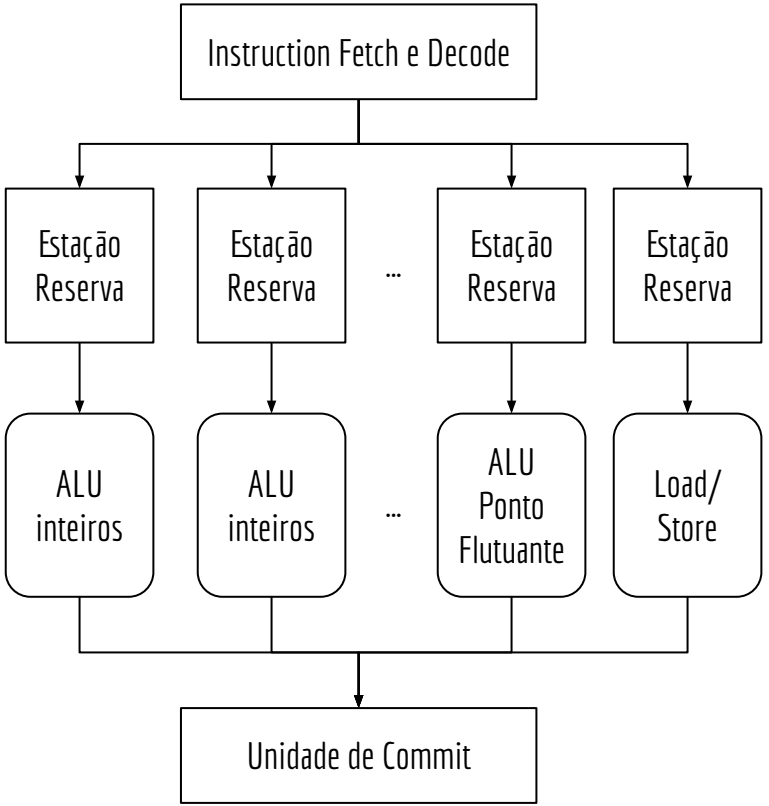
Por conta desses e outros problemas, a unidade de commit precisa estar segura de muitas coisas antes de enviar o resultado.

- O resultado realmente deveria existir, ou é só uma especulação incorreta?
- A exceção realmente deveria existir, ou é só uma especulação incorreta?
- Se ocorrerem múltiplas exceções, qual a ordem?
- Qual a instrução original que gerou a exceção?



Mais problemas

Dá para piorar – é sério isso!?.

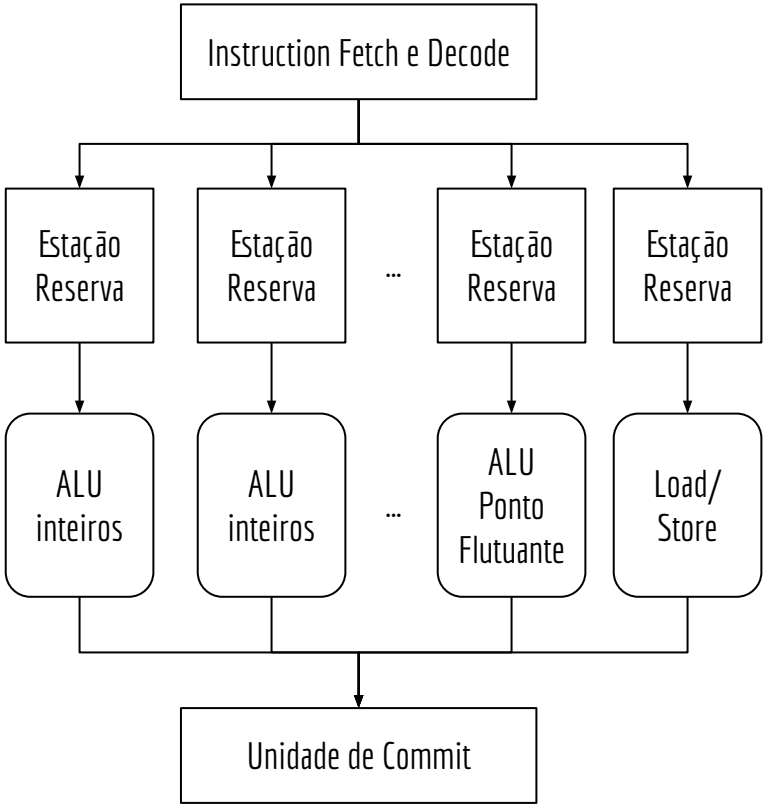


Mais problemas

Considere o trecho.

Quais instruções precisam esperar quais? Por quê?

```
lw $s0,0($t0)
add $s1,$s0,$s4
sw $s1,0($t0)
add $s0,$s3,$s4
sw $s0,4($t0)
```



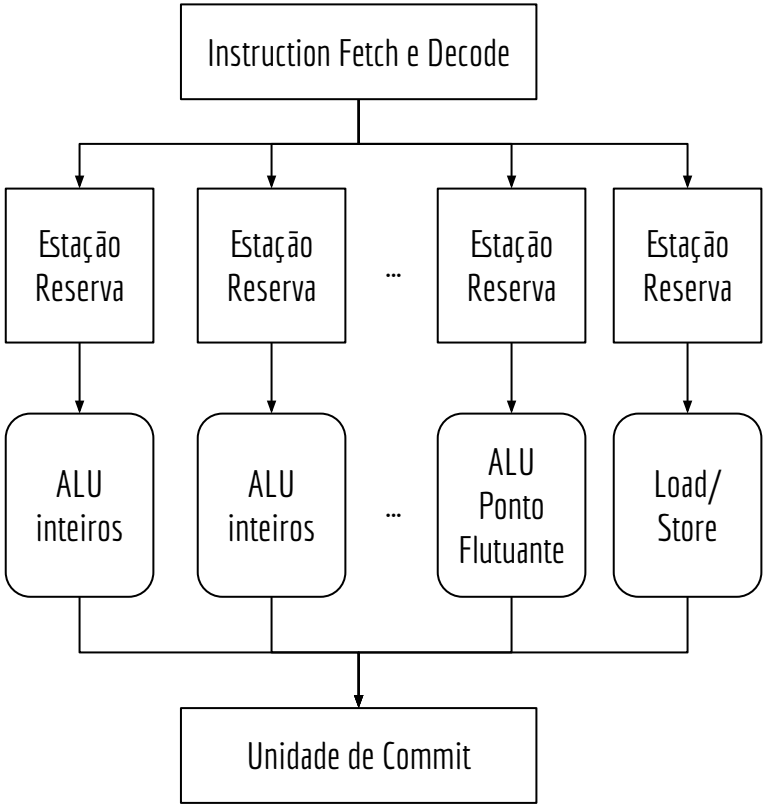
Mais problemas

Considere o trecho.

Quais instruções precisam esperar quais? Por quê?

Esperar o lw.

```
lw $s0,0($t0)
add $s1,$s0,$s4
sw $s1,0($t0)
add $s0,$s3,$s4
sw $s0,4($t0)
```

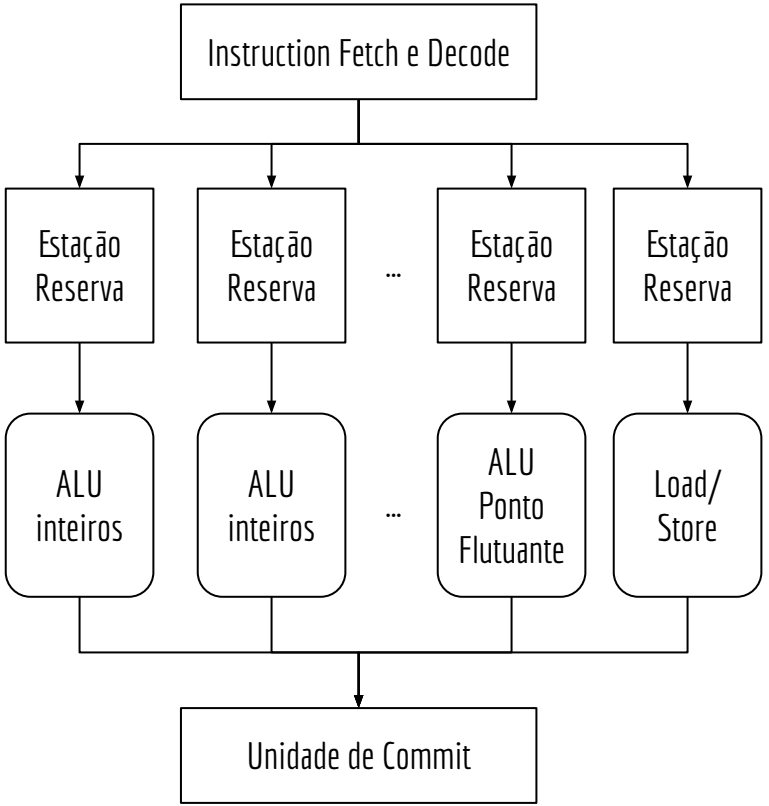


Mais problemas

Considere o trecho.

O que um compilador/programador esperto pode fazer?

```
lw $s0,0($t0)
add $s1,$s0,$s4
sw $s1,0($t0)
add $s0,$s3,$s4
sw $s0,4($t0)
```



Mais problemas

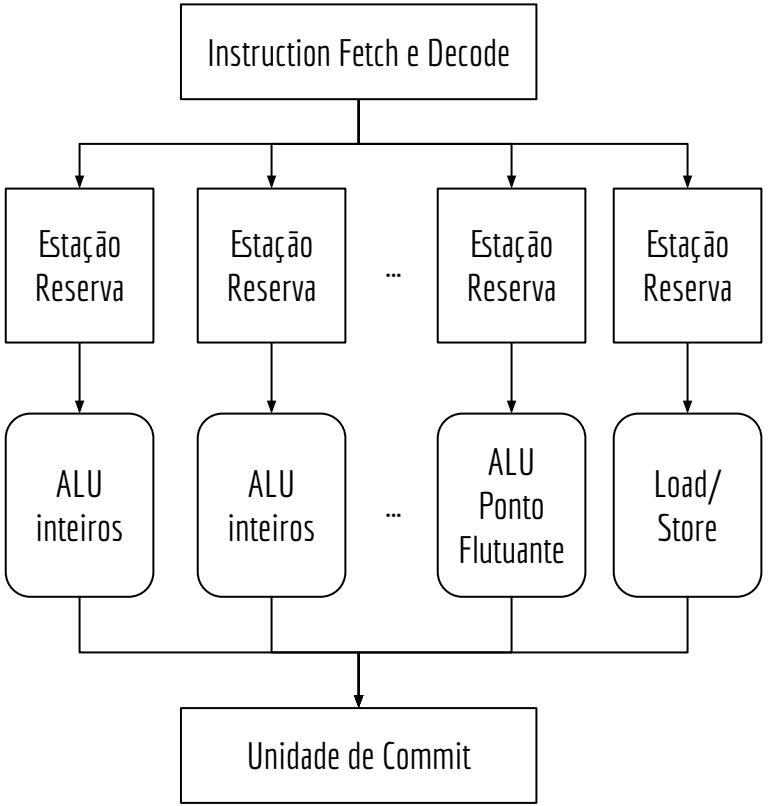
Considere o trecho.

O segundo add pode usar outro registrador.

Só estamos usando o $s0 para reutilizar registradores, não existe dependência de dados real.

Chamamos de **dependência de nomes**.

```
lw $s0,0($t0)
add $s1,$s0,$s4
sw $s1,0($t0)
add $s0,$s3,$s4
sw $s0,4($t0)
```



Mais problemas

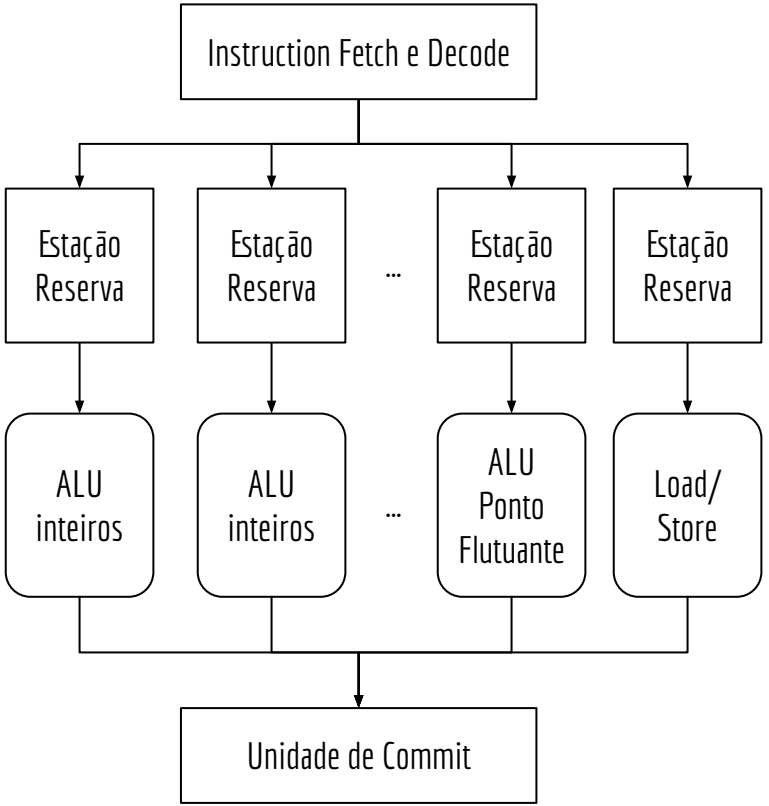
Considere o trecho.

O segundo add pode usar outro registrador.

Só estamos usando o $s0 para reutilizar registradores, não existe dependência de dados real.

Chamamos de **dependência de nomes**.

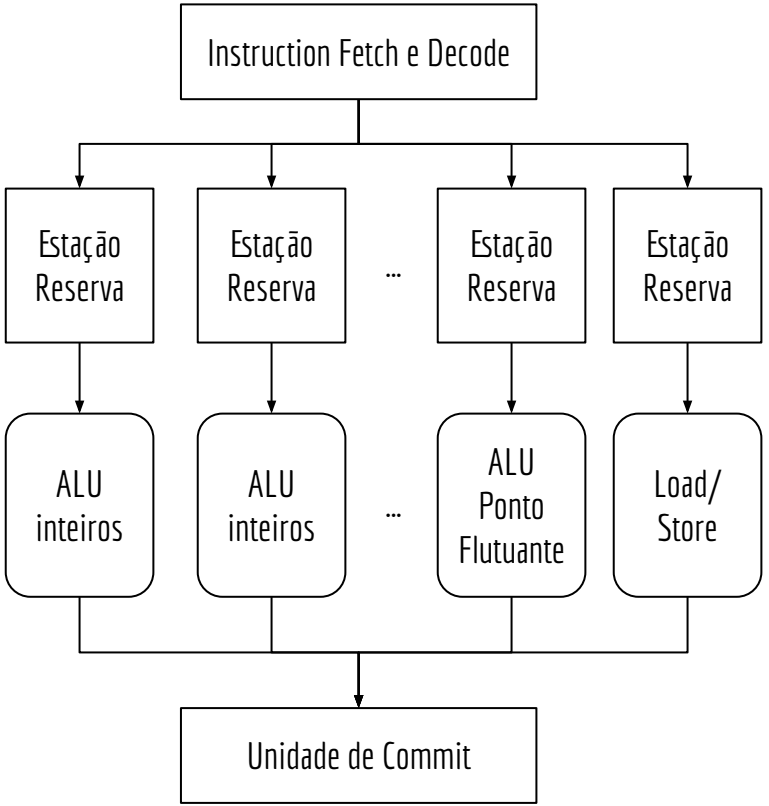
```
lw $s0,0($t0)
add $s1,$s0,$s4
sw $s1,0($t0)
add $s5,$s3,$s4
sw $s5,4($t0)
```



Mais problemas

O lw e o primeiro add possuem uma **dependência de dados real – *true data dependence***, e não podemos resolver usando outros registradores.

```
lw $s0,0($t0)
add $s1,$s0,$s4
sw $s1,0($t0)
add $s5,$s3,$s4
sw $s5,4($t0)
```

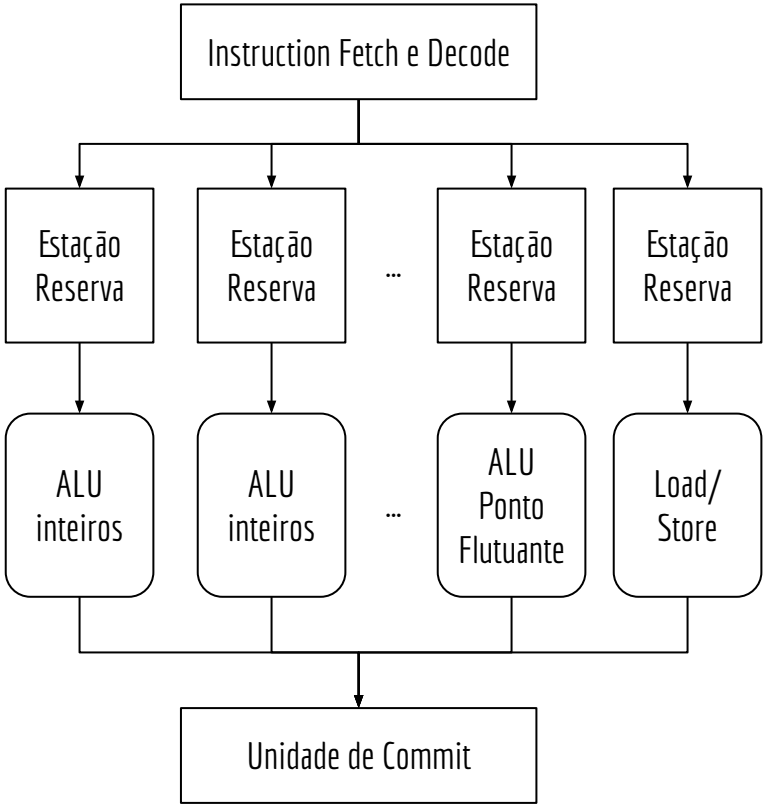


Mais problemas

Deixar a resolução de dependências de nome a cargo do compilador gera os mesmos problemas discutidos anteriormente.

Processadores com **renomeação de registradores** tentam resolver isso autoMAGiCAmente.

```
lw $s0,0($t0)
add $s1,$s0,$s4
sw $s1,0($t0)
add $s5,$s3,$s4
sw $s5,4($t0)
```



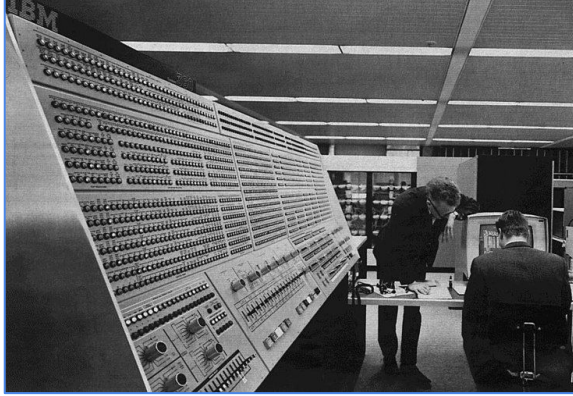
Algoritmo de Tomasulo

Algoritmo criado para o IBM360/91 de 1964.

Permite a execução fora de ordem, e renomeação de registradores.

Versões modernas do algoritmo permitem a geração de exceções precisas através de commit em ordem.

Variantes do algoritmo são usadas para fazer o despacho de múltiplas instruções em um ciclo de clock.



IBM 360/91 - en.wikipedia.org/wiki/IBM_System/360_Model_91

Robert Tomasulo
(31/10/1934 - 03/04/ 2008)
- Algoritmo de Tomasulo

www.computer.org/profiles/robert-tomasulo

Algoritmo de Tomasulo

Veremos um exemplo de implementação do algoritmo.

Serão considerados dois itens que não existiam no método original, mas que são comuns hoje em dia:

- Buffer de reordenação;
- Commit em ordem com exceções precisas.

Vamos considerar duas unidades funcionais, que levam múltiplos ciclos de clock para completar uma operação.

A cada ciclo de clock, apenas uma operação poderá ser enviada para execução.

Processadores superescalares atuais podem despachar múltiplas operações para execução em um único ciclo

Geralmente até 4 instruções podem ser despachadas em um único ciclo.

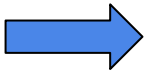
Complica muito o algoritmo – veja na literatura.

Buffer de reordenação

As instruções que partem da unidade de *Instruction Fetch* vão para um *Buffer de Reordenação*.

Se o buffer estiver cheio, *stall*.

Instruction Fetch
`mul $s3,$s0,$s1`
`add $s3,$s3,$s0`
`add $s1,$s6,$s7`
`mul $s3,$s6,$s4`
`mul $s7,$s3,$s5`
…

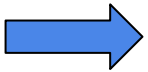


Buffer de Reordenação - Reorder Buffer (ROB)							
#	Ocup.	Opcode	Source	Estado	Destino	Resultado	Exceção
0	0						
1	0						
2	0						
3	0						
4	0						
5	0						
6	0						
7	0						

Instruction Fetch envia para a instrução com suas propriedades para o buffer de reordenação.
Cada entrada do buffer possui um endereço (# na tabela).

Clock 0

Instruction Fetch
`mul $s3,$s0,$s1`
`add $s3,$s3,$s0`
`add $s1,$s6,$s7`
`mul $s3,$s6,$s4`
`mul $s7,$s3,$s5`
…

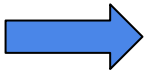


Buffer de Reordenação - Reorder Buffer (ROB)							
#	Ocup.	Opcode	Source	Estado	Destino	Resultado	Exceção
0	1	`mul`	`$s0,$s1`	Pronta	`$s3`	-	-
1	0						
2	0						
3	0						
4	0						
5	0						
6	0						
7	0						

Instruction Fetch envia para a instrução com suas propriedades para o buffer de reordenação.
Cada entrada do buffer possui um endereço (# na tabela).

Clock 1

Instruction Fetch
`mul $s3,$s0,$s1`
`add $s3,$s3,$s0`
`add $s1,$s6,$s7`
`mul $s3,$s6,$s4`
`mul $s7,$s3,$s5`
…



Buffer de Reordenação - Reorder Buffer (ROB)							
#	Ocup.	Opcode	Source	Estado	Destino	Resultado	Exceção
0	1	mul	$s0,$s1	Pronta	$s3	-	-
1	0						
2	0						
3	0						
4	0						
5	0						
6	0						
7	0						

Banco de Registradores		
Reg.	Valor	ROB #
$s0	1	
$s1	2	
$s2	3	
$s3	5	0
$s4	7	
$s5	20	
$s6	11	
$s7	16	
…		

O banco de registradores deve manter informações se determinado registrador é válido atualmente, ou se o seu resultado será gerado por alguma entrada do ROB.

Clock 1

Estações Reserva

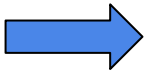
As unidades funcionais são atreladas a **buffers de estações de reserva**.

Quando o buffer da estação de reserva está livre, o ROB coloca a instrução da estação reserva.

A estação reserva recebe uma cópia do valor dos operandos, ou então a Tag de onde o operando vai vir.

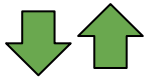
A estação reserva também tem a Tag de destino no ROB.

Instruction Fetch
mul $s3,$s0,$s1
add $s3,$s3,$s0
add $s1,$s6,$s7
mul $s3,$s6,$s4
mul $s7,$s3,$s5
…



Buffer de Reordenação - Reorder Buffer (ROB)

#	Ocup.	Opcode	Source	Estado	Destino	Resultado	Exceção
0	1	mul	$s0,$s1	Pronta	$s3	-	-
1	0						
2	0						
3	0						
4	0						
5	0						
6	0						
7	0						



Estação de Reserva 0 - ADD, SUB, AND, OR, ...

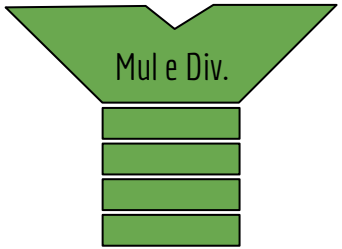
Dest. #	Ocup.	Opcode	Source1	Source2	Resultado
	0				
	0				
	0				
	0				

Estação de Reserva 1 - Mul e Div

Dest. #	Ocup.	Opcode	Source1	Source2	Resultado
0	1	mul	1	2	
	0				
	0				
	0				

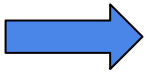
Banco de Registradores

Reg.	Valor	ROB #
$s0	1	
$s1	2	
$s2	3	
$s3	5	0
$s4	7	
$s5	20	
$s6	11	
$s7	16	
…		



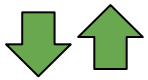
Clock 1

Instruction Fetch
add $s3,$s3,$s0
add $s1,$s6,$s7
mul $s3,$s6,$s4
mul $s7,$s3,$s5
…



Buffer de Reordenação - Reorder Buffer (ROB)

#	Ocup.	Opcode	Source	Estado	Destino	Resultado	Exceção
0	1	mul	$s0,$s1	Executando	$s3	-	-
1	1	add	$s3,$s0	Aguardando	$s3	-	-
2	0						
3	0						
4	0						
5	0						
6	0						
7	0						



Banco de Registradores

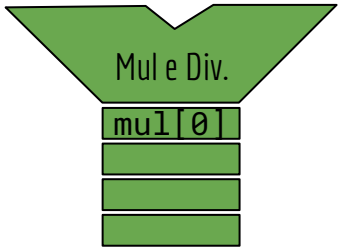
Reg.	Valor	ROB #
$s0	1	
$s1	2	
$s2	3	
$s3	5	1
$s4	7	
$s5	20	
$s6	11	
$s7	16	
…		

Estação de Reserva 0 - ADD, SUB, AND, OR, ...

Dest. #	Ocup.	Opcode	Source1	Source2	Resultado
1	1	add	rob[0]	1	
	0				
	0				
	0				

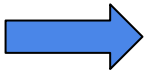
Estação de Reserva 1 - Mul e Div

Dest. #	Ocup.	Opcode	Source1	Source2	Resultado
0	1	mul	1	2	
	0				
	0				
	0				



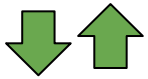
Clock 2

Instruction Fetch
add $s1,$s6,$s7
mul $s3,$s6,$s4
mul $s7,$s3,$s5
…



Buffer de Reordenação - Reorder Buffer (ROB)

#	Ocup.	Opcode	Source	Estado	Destino	Resultado	Exceção
0	1	mul	$s0,$s1	Executando	$s3	-	-
1	1	add	$s3,$s0	Aguardando	$s3	-	-
2	1	add	$s6,$s7	Pronta	$s1	-	-
3	0						
4	0						
5	0						
6	0						
7	0						



Banco de Registradores

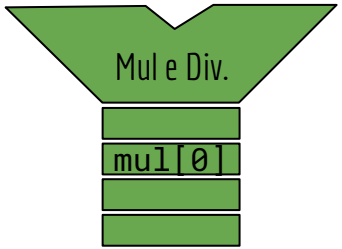
Reg.	Valor	ROB #
$s0	1	
$s1	2	2
$s2	3	
$s3	5	1
$s4	7	
$s5	20	
$s6	11	
$s7	16	
…		

Estação de Reserva 0 - ADD, SUB, AND, OR, ...

Dest. #	Ocup.	Opcode	Source1	Source2	Resultado
1	1	add	rob[0]	1	
2	1	add	11	16	
	0				
	0				

Estação de Reserva 1 - Mul e Div

Dest. #	Ocup.	Opcode	Source1	Source2	Resultado
0	1	mul	1	2	
	0				
	0				
	0				



Clock 3

Instruction Fetch
mul $s3,$s6,$s4
mul $s7,$s3,$s5
…

Buffer de Reordenação - Reorder Buffer (ROB)

#	Ocup.	Opcode	Source	Estado	Destino	Resultado	Exceção
0	1	mul	$s0,$s1	Executando	$s3	-	-
1	1	add	$s3,$s0	Aguardando	$s3	-	-
2	1	add	$s6,$s7	Executando	$s1	-	-
3	1	mul	$s6,$s4	Pronta	$s3	-	-
4	0						
5	0						
6	0						
7	0						

Estação de Reserva 0 - ADD, SUB, AND, OR, ...

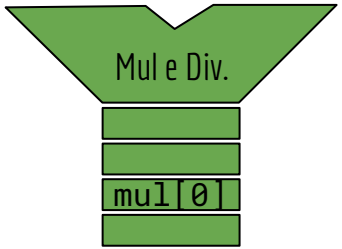
Dest. #	Ocup.	Opcode	Source1	Source2	Resultado
1	1	add	rob[0]	1	
2	1	add	11	16	
	0				
	0				

Estação de Reserva 1 - Mul e Div

Dest. #	Ocup.	Opcode	Source1	Source2	Resultado
0	1	mul	1	2	
3	0	mul	11	16	
	0				
	0				

Banco de Registradores

Reg.	Valor	ROB #
$s0	1	
$s1	2	2
$s2	3	
$s3	5	3
$s4	7	
$s5	20	
$s6	11	
$s7	16	
…		



Clock 4

Instruction Fetch
`mul $s7,$s3,$s5`
…

Buffer de Reordenação - Reorder Buffer (ROB)

#	Ocup.	Opcode	Source	Estado	Destino	Resultado	Exceção
0	1	mul	$s0,$s1	Executando	$s3	-	-
1	1	add	$s3,$s0	Aguardando	$s3	-	-
2	1	add	$s6,$s7	Executando	$s1	-	-
3	1	mul	$s6,$s4	Executando	$s3	-	-
4	1	mul	$s3,$s5	Aguardando	$s7	-	-
5	0						
6	0						
7	0						

Estação de Reserva 0 - ADD, SUB, AND, OR, ...

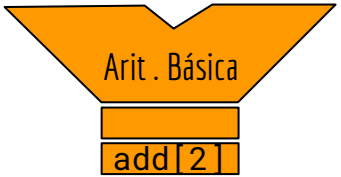
Dest. #	Ocup.	Opcode	Source1	Source2	Resultado
1	1	add	rob[0]	1	
2	1	add	11	16	27
	0				
	0				

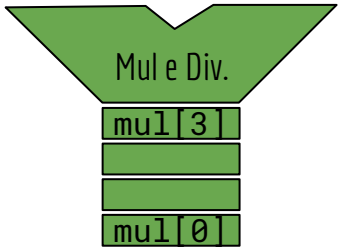
Estação de Reserva 1 - Mul e Div

Dest. #	Ocup.	Opcode	Source1	Source2	Resultado
0	1	mul	1	2	2
3	1	mul	11	16	
4	1	mul	rob[3]	20	
	0				

Banco de Registradores

Reg.	Valor	ROB #
$s0	1	
$s1	2	2
$s2	3	
$s3	5	3
$s4	7	
$s5	20	
$s6	11	
$s7	16	4
…		





Clock 5

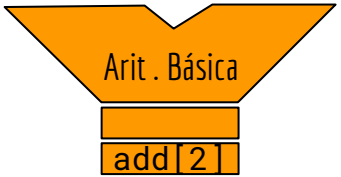
Resultados prontos

Os resultados do mul e do add estão prontos.

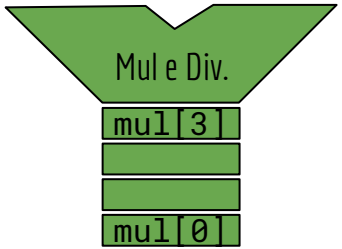
Podem ser enviados para o ROB, e para os demais buffers que estão esperando via forwardings.

Banco de Registradores		
Reg.	Valor	ROB #
$s0	1	
$s1	2	2
$s2	3	
$s3	5	3
$s4	7	
$s5	20	
$s6	11	
$s7	16	4
...		

Estação de Reserva 0 - ADD, SUB, AND, OR, ...					
Dest. #	Ocup.	Opcode	Source1	Source2	Resultado
1	1	add	rob[0]	1	
2	1	add	11	16	27
	0				
	0				



Estação de Reserva 1 - Mul e Div					
Dest. #	Ocup.	Opcode	Source1	Source2	Resultado
0	1	mul	1	2	2
3	1	mul	11	16	
4	1	mul	rob[3]	20	
	0				



Clock 5

Instruction Fetch
...

Buffer de Reordenação - Reorder Buffer (ROB)

#	Ocup.	Opcode	Source	Estado	Destino	Resultado	Exceção
0	1	mul	$s0,$s1	Commit	$s3	2	-
1	1	add	$s3,$s0	Pronta	$s3	-	-
2	1	add	$s6,$s7	Commit	$s1	27	-
3	1	mul	$s6,$s4	Executando	$s3	-	-
4	1	mul	$s3,$s5	Aguardando	$s7	-	-
5	0						
6	0						
7	0						

Banco de Registradores

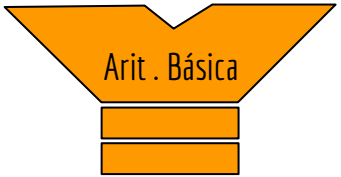
Reg.	Valor	ROB #
$s0	1	
$s1	2	2
$s2	3	
$s3	5	3
$s4	7	
$s5	20	
$s6	11	
$s7	16	4
...		

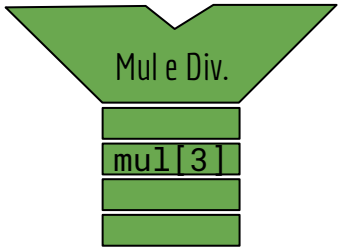
Estação de Reserva 0 - ADD, SUB, AND, OR, ...

Dest. #	Ocup.	Opcode	Source1	Source2	Resultado
1	1	add	2	1	
	0				
	0				
	0				

Estação de Reserva 1 - Mul e Div

Dest. #	Ocup.	Opcode	Source1	Source2	Resultado
	0				
3	1	mul	11	16	
4	1	mul	rob[3]	20	
	0				





Clock 6

Buffer de Reordenação - Reorder Buffer (ROB)							
#	Ocup.	Opcode	Source	Estado	Destino	Resultado	Exceção
0	1	mul	$s0,$s1	Commit	$s3	2	-
1	1	add	$s3,$s0	Pronta	$s3	-	-
2	1	add	$s6,$s7	Commit	$s1	27	-
3	1	mul	$s6,$s4	Executando	$s3	-	-
4	1	mul	$s3,$s5	Aguardando	$s7	-	-
5	0						
6	0						
7	0						

Temos duas instruções em commit (terminaram de executar no ROB).

As instruções efetuam o commit **em ordem**.

Note que a primeira instrução que vai fazer o commit não precisa alterar nada no banco de registradores nem na memória.

Banco de Registradores		
Reg.	Valor	ROB #
$s0	1	
$s1	2	2
$s2	3	
$s3	5	3
$s4	7	
$s5	20	
$s6	11	
$s7	16	4
...		

Clock 6

Instruction Fetch
...

Buffer de Reordenação - Reorder Buffer (ROB)

#	Ocup.	Opcode	Source	Estado	Destino	Resultado	Exceção
0	0						
1	1	add	$s3,$s0	Executando	$s3	-	-
2	1	add	$s6,$s7	Commit	$s1	27	-
3	1	mul	$s6,$s4	Executando	$s3	-	-
4	1	mul	$s3,$s5	Aguardando	$s7	-	-
5	0						
6	0						
7	0						

Estação de Reserva 0 - ADD, SUB, AND, OR, ...

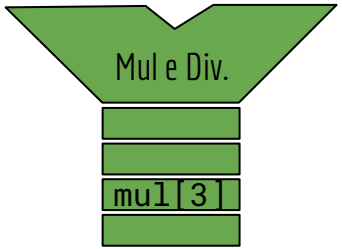
Dest. #	Ocup.	Opcode	Source1	Source2	Resultado
1	1	add	2	1	
	0				
	0				
	0				

Estação de Reserva 1 - Mul e Div

Dest. #	Ocup.	Opcode	Source1	Source2	Resultado
	0				
3	1	mul	11	16	
4	1	mul	rob[3]	20	
	0				

Banco de Registradores

Reg.	Valor	ROB #
$s0	1	
$s1	2	2
$s2	3	
$s3	5	3
$s4	7	
$s5	20	
$s6	11	
$s7	16	4
...		



Clock 7

Buffer de Reordenação - Reorder Buffer (ROB)							
#	Ocup.	Opcode	Source	Estado	Destino	Resultado	Exceção
0	0						
1	1	add	$s3,$s0	Executando	$s3	-	-
2	1	add	$s6,$s7	Commit	$s1	27	-
3	1	mul	$s6,$s4	Executando	$s3	-	-
4	1	mul	$s3,$s5	Aguardando	$s7	-	-
5	0						
6	0						
7	0						

A instrução no ROB2 já está em commit, mas o **commit é em ordem**.

Vai precisar esperar a instrução no ROB1 terminar.

Banco de Registradores		
Reg.	Valor	ROB #
$s0	1	
$s1	2	2
$s2	3	
$s3	5	3
$s4	7	
$s5	20	
$s6	11	
$s7	16	4
...		

Clock 7

Instruction Fetch
...

Buffer de Reordenação - Reorder Buffer (ROB)

#	Ocup.	Opcode	Source	Estado	Destino	Resultado	Exceção
0	0						
1	1	add	$s3,$s0	Executando	$s3	-	-
2	1	add	$s6,$s7	Commit	$s1	27	-
3	1	mul	$s6,$s4	Executando	$s3	-	-
4	1	mul	$s3,$s5	Aguardando	$s7	-	-
5	0						
6	0						
7	0						

Estação de Reserva 0 - ADD, SUB, AND, OR, ...

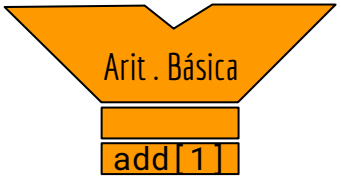
Dest. #	Ocup.	Opcode	Source1	Source2	Resultado
1	1	add	2	1	3
	0				
	0				
	0				

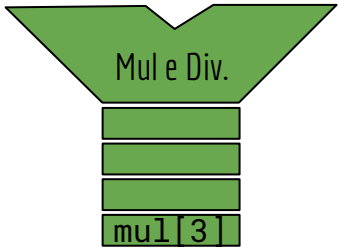
Estação de Reserva 1 - Mul e Div

Dest. #	Ocup.	Opcode	Source1	Source2	Resultado
	0				
3	1	mul	11	16	176
4	1	mul	rob[3]	20	
	0				

Banco de Registradores

Reg.	Valor	ROB #
$s0	1	
$s1	2	2
$s2	3	
$s3	5	3
$s4	7	
$s5	20	
$s6	11	
$s7	16	4
...		





Clock 8

Instruction Fetch
...

Buffer de Reordenação - Reorder Buffer (ROB)

#	Ocup.	Opcode	Source	Estado	Destino	Resultado	Exceção
0	0						
1	1	add	$s3, $s0	Commit	$s3	3	-
2	1	add	$s6, $s7	Commit	$s1	27	-
3	1	mul	$s6, $s4	Commit	$s3	176	-
4	1	mul	$s3, $s5	Executando	$s7	-	-
5	0						
6	0						
7	0						

Estação de Reserva 0 - ADD, SUB, AND, OR, ...

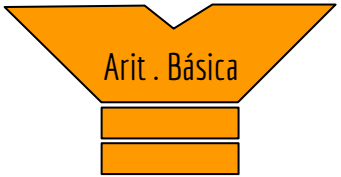
Dest. #	Ocup.	Opcode	Source1	Source2	Resultado
	0				
	0				
	0				
	0				

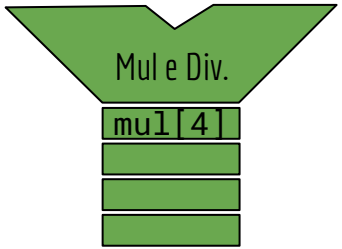
Estação de Reserva 1 - Mul e Div

Dest. #	Ocup.	Opcode	Source1	Source2	Resultado
	0				
	0				
4	1	mul	176	20	
	0				

Banco de Registradores

Reg.	Valor	ROB #
$s0	1	
$s1	2	2
$s2	3	
$s3	5	3
$s4	7	
$s5	20	
$s6	11	
$s7	16	4
...		





Clock 9

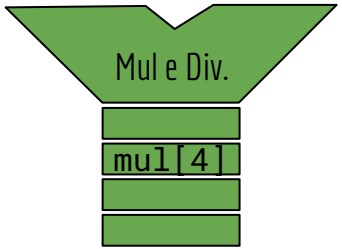
Instruction Fetch
...

Buffer de Reordenação - Reorder Buffer (ROB)							
#	Ocup.	Opcode	Source	Estado	Destino	Resultado	Exceção
0	0						
1	0						
2	1	add	$s6, $s7	Commit	$s1	27	-
3	1	mul	$s6, $s4	Commit	$s3	176	-
4	1	mul	$s3, $s5	Executando	$s7	-	-
5	0						
6	0						
7	0						

Banco de Registradores		
Reg.	Valor	ROB #
$s0	1	
$s1	2	2
$s2	3	
$s3	5	3
$s4	7	
$s5	20	
$s6	11	
$s7	16	4
...		

Estação de Reserva 0 - ADD, SUB, AND, OR, ...					
Dest. #	Ocup.	Opcode	Source1	Source2	Resultado
	0				
	0				
	0				
	0				

Estação de Reserva 1 - Mul e Div					
Dest. #	Ocup.	Opcode	Source1	Source2	Resultado
	0				
	0				
4	1	mul	176	20	
	0				



Clock 10

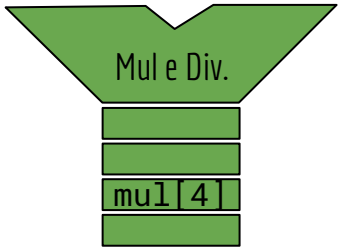
Instruction Fetch
…

Buffer de Reordenação - Reorder Buffer (ROB)							
#	Ocup.	Opcode	Source	Estado	Destino	Resultado	Exceção
0	0						
1	0						
2	0						
3	1	mul	$s6,$s4	Commit	$s3	176	-
4	1	mul	$s3,$s5	Executando	$s7	-	-
5	0						
6	0						
7	0						

Banco de Registradores		
Reg.	Valor	ROB #
$s0	1	
$s1	27	
$s2	3	
$s3	5	3
$s4	7	
$s5	20	
$s6	11	
$s7	16	4
…		

Estação de Reserva 0 - ADD, SUB, AND, OR, …					
Dest. #	Ocup.	Opcode	Source1	Source2	Resultado
	0				
	0				
	0				
	0				

Estação de Reserva 1 - Mul e Div					
Dest. #	Ocup.	Opcode	Source1	Source2	Resultado
	0				
	0				
4	1	mul	176	20	
	0				



Clock 11

Instruction Fetch
…

Buffer de Reordenação - Reorder Buffer (ROB)

#	Ocup.	Opcode	Source	Estado	Destino	Resultado	Exceção
0	0						
1	0						
2	0						
3	0						
4	1	mul	$s3, $s5	Executando	$s7	-	-
5	0						
6	0						
7	0						

Estação de Reserva 0 - ADD, SUB, AND, OR, ...

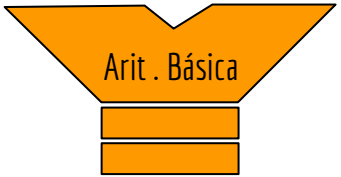
Dest. #	Ocup.	Opcode	Source1	Source2	Resultado
	0				
	0				
	0				
	0				

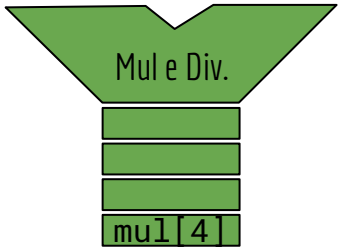
Estação de Reserva 1 - Mul e Div

Dest. #	Ocup.	Opcode	Source1	Source2	Resultado
	0				
	0				
4	1	mul	176	20	3520
	0				

Banco de Registradores

Reg.	Valor	ROB #
$s0	1	
$s1	27	
$s2	3	
$s3	176	
$s4	7	
$s5	20	
$s6	11	
$s7	16	4
…		





Clock 12

Instruction Fetch
…

Buffer de Reordenação - Reorder Buffer (ROB)

#	Ocup.	Opcode	Source	Estado	Destino	Resultado	Exceção
0	0						
1	0						
2	0						
3	0						
4	1	mul	$s3,$s5	Commit	$s7	3520	-
5	0						
6	0						
7	0						

Estação de Reserva 0 - ADD, SUB, AND, OR, ...

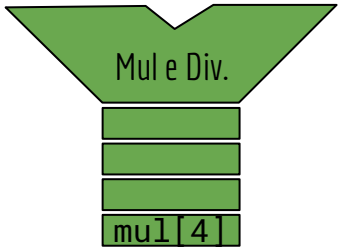
Dest. #	Ocup.	Opcode	Source1	Source2	Resultado
	0				
	0				
	0				
	0				

Estação de Reserva 1 - Mul e Div

Dest. #	Ocup.	Opcode	Source1	Source2	Resultado
	0				
	0				
	0				
	0				

Banco de Registradores

Reg.	Valor	ROB #
$s0	1	
$s1	27	
$s2	3	
$s3	176	
$s4	7	
$s5	20	
$s6	11	
$s7	16	4
…		



Clock 12

Instruction Fetch
...

Buffer de Reordenação - Reorder Buffer (ROB)

#	Ocup.	Opcode	Source	Estado	Destino	Resultado	Exceção
0	0						
1	0						
2	0						
3	0						
4	0						
5	0						
6	0						
7	0						

Estação de Reserva 0 - ADD, SUB, AND, OR, ...

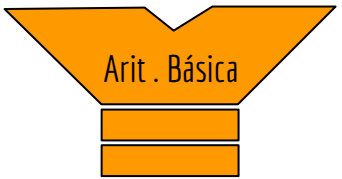
Dest. #	Ocup.	Opcode	Source1	Source2	Resultado
	0				
	0				
	0				
	0				

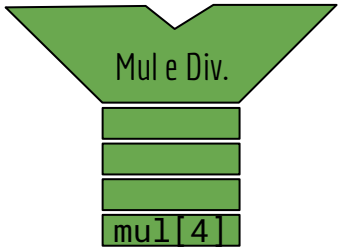
Estação de Reserva 1 - Mul e Div

Dest. #	Ocup.	Opcode	Source1	Source2	Resultado
	0				
	0				
	0				
	0				

Banco de Registradores

Reg.	Valor	ROB #
$s0	1	
$s1	27	
$s2	3	
$s3	176	
$s4	7	
$s5	20	
$s6	11	
$s7	3520	
...		





Clock 12

Elaborando

Note que os buffers das estações de reserva fazem a renomeação de registradores.

- Eliminam a referência aos registradores fonte e destino.
- Armazenam apenas os valores, ou então “ponteiros” para as posições do ROB de onde os valores virão.

Possibilitam uma quantidade enorme de “registradores virtuais”, sem a necessidade de implementar os regitradores diretamente no banco de registradores, o que daria trabalho para o compilador, e forçaria mudanças na ISA.

Elaborando

Com o commit em ordem, mantemos as coisas organizadas.

Se descobrirmos no meio do caminho que alguma especulação estava incorreta, jogamos fora (flush) as instruções do ROB após o erro.

E.g., as instruções após o branch que estão no ROB.

Elaborando

Com o commit em ordem, mantemos as coisas organizadas.

Se descobrirmos no meio do caminho que alguma especulação estava incorreta, jogamos fora (flush) as instruções do ROB após o erro.

E.g., as instruções após o branch que estão no ROB.

Se alguma instrução gerar uma exceção, a exceção é realmente lançada quando o commit é realizado.

Elaborando

Com o commit em ordem, mantemos as coisas organizadas.

Se descobrirmos no meio do caminho que alguma especulação estava incorreta, jogamos fora (flush) as instruções do ROB após o erro.

E.g., as instruções após o branch que estão no ROB.

Se alguma instrução gerar uma exceção, a exceção é realmente lançada quando o commit é realizado.

Antes disso, a instrução pode ser descartada devido a uma especulação incorreta.

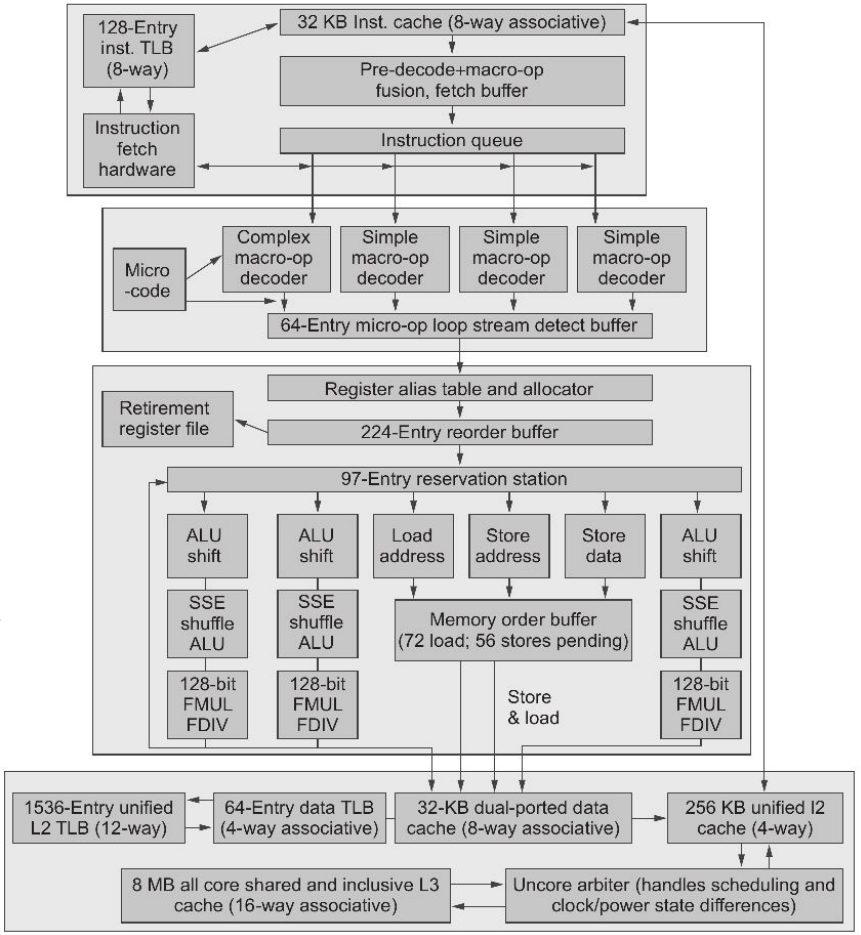
Geramos exceções precisas.

Nota

Vale um comentário, que boa parte das CPUs geram exceções imprecisas nas unidades de ponto flutuante.

Um i7

Unidades Funcionais de um Intel Core i7 6700 - 6a geração.



Unidades de despacho

Microprocessador	Ano	Clock	Estágios Pipeline	Tamanho Despacho	Fora de ordem?	CPUs por Chip	Potência
486	1989	0,025 GHz	5	**1**	Não	1	5 W
Pentium	1993	0,066 GHz	5	**2**	Não	1	10 W
Pentium Pro	1997	0,2 GHz	10	**3**	Sim	1	29 W
Pentium 4 Willamette	2001	2 GHz	22	**3**	Sim	1	75 W
Pentium 4 Prescott	2004	3,6 GHz	31	**3**	Sim	1	103 W
Intel Core	2006	3 GHz	14	**4**	Sim	2	75 W
Core i7 Nehalem	2008	3,6 GHz	14	**4**	Sim	2-4	87 W
Core Westmere	2010	3,73 GHz	14	**4**	Sim	6	130 W
Core i7 Ivy Bridge	2012	3,4 GHz	14	**4**	Sim	6	130 W
Core Broadwell	2014	3,7 GHz	14	**4**	Sim	10	140 W
Core i9 Skylake	2016	3,1 GHz	14	**4**	Sim	14	165 W
Intel Ice Lake	2018	4,2 GHz	14	**4**	Sim	16	185 W

Exercícios

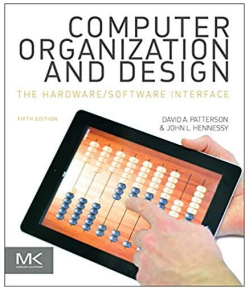
1. Faça um loop para inicializar um vetor com e sem unroll. Compare os assemblys. Como sua CPU x86 lida com isso? Você precisa renomear os registradores ou a CPU lida com isso sozinha? É possível a CPU fazer um unroll automaticamente sem precisarmos fazer isso manualmente?
2. Quais são as dependências reais e de nome no trecho a seguir? Remova as dependências de nome.

```
fdiv.d  f0,f2,f4
fadd.d  f6,f0,f8
fsd     f6,0(x1)
fsub.d  f8,f10,f14
fmul.d  f6,f10,f8
```

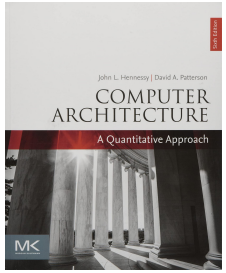
3. Pegue programas seus em MIPS32 criados durante a disciplina de Arquitetura de Computadores, e na disciplina de Projetos, e elimine as dependências de nome de seus programas manualmente.

Referências

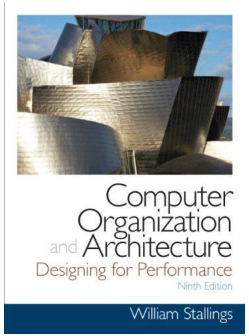
Patterson, Hennessy . Arquitetura e Organização de Computadores: A interface hardware/software. 2014.



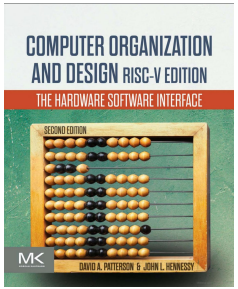
Hennessy, Patterson. Arquitetura de Computadores: uma abordagem quantitativa. 2019.



Stallings, W. Organização de Arquitetura de Computadores. 10a Ed. 2016.



Patterson, Hennessy. Computer Organization and Design RISC-V Edition: The Hardware Software Interface. 2020.



Licença