

Infraestrutura de Hardware

Melhorando Desempenho de Pipeline
Processadores Superpipeline,
Superescalares, VLIW



UNIVERSIDADE
FEDERAL
DE PERNAMBUCO

Perguntas que Devem ser Respondidas ao Final do Curso

- Como um programa escrito em uma linguagem de alto nível é entendido e executado pelo HW?
- Qual é a interface entre SW e HW e como o SW instrui o HW a executar o que foi planejado?
- O que determina o desempenho de um programa e como ele pode ser melhorado?
- **Que técnicas um projetista de HW pode utilizar para melhorar o desempenho?**

Pipeline

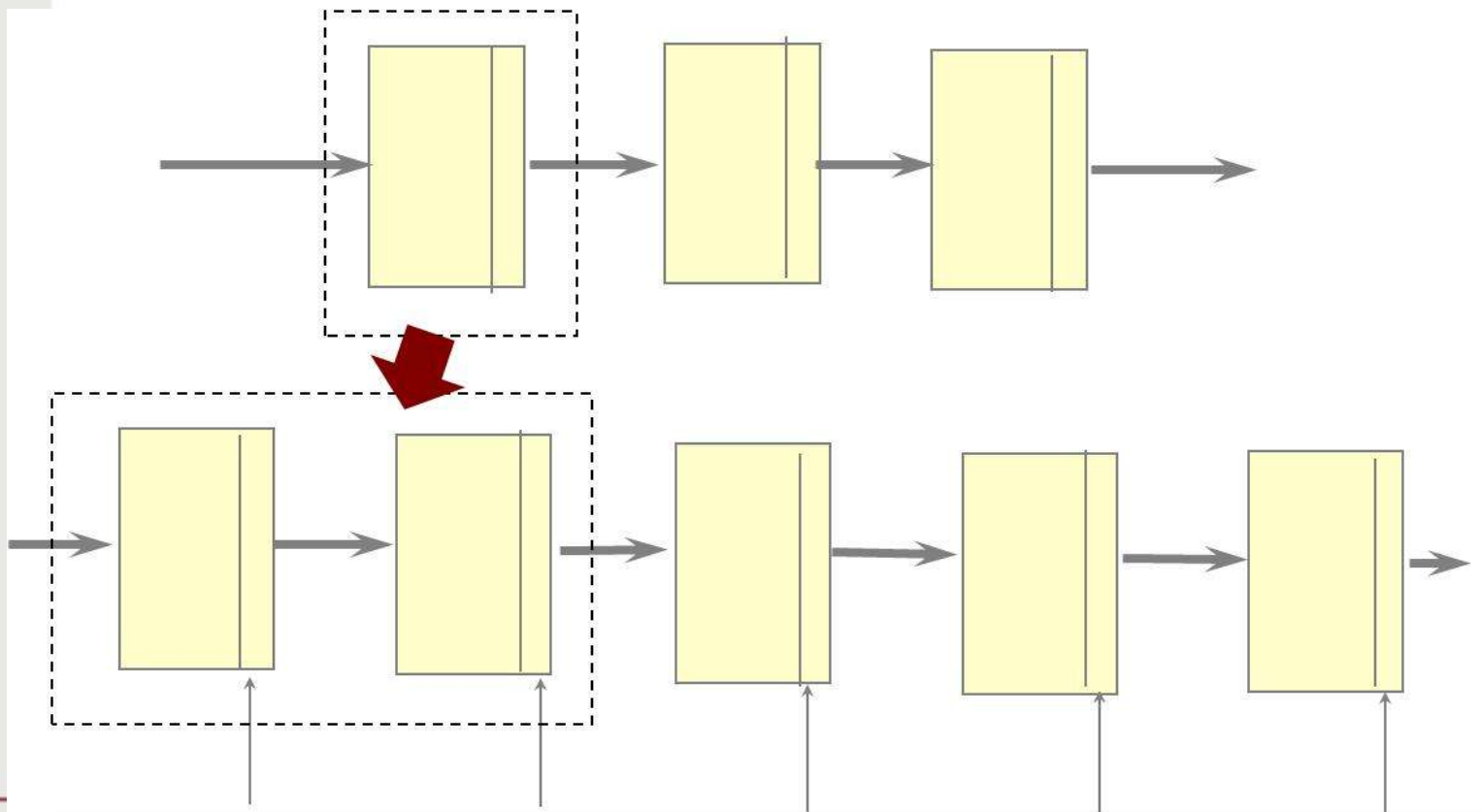
- Pipeline é uma técnica que visa aumentar o nível de paralelismo de execução de instruções
 - ILP (Instruction-**L**evel **P**aralellism)
- Permite que várias instruções sejam processadas simultaneamente com cada parte do HW atuando numa instrução distinta
 - Instruções quebradas em estágios
 - Sobreposição temporal
- Visa aumentar desempenho
 - Latência de instruções é a mesma ou maior
 - Throughput aumenta
- Tempo de execução de instrução é o mesmo ou maior, **MAS** tempo de execução de programa é menor

Como Melhorar Desempenho de Pipeline?

- Aumentando o número de estágios
 - Estágios de menor duração
 - Frequência do clock maior
 - Superpipeline**
- Aumentando a quantidade de instruções que executam em paralelo
 - Paralelismo real
 - Replicação de recursos de HW
 - Superescalar e VLIW**

Superpipeline

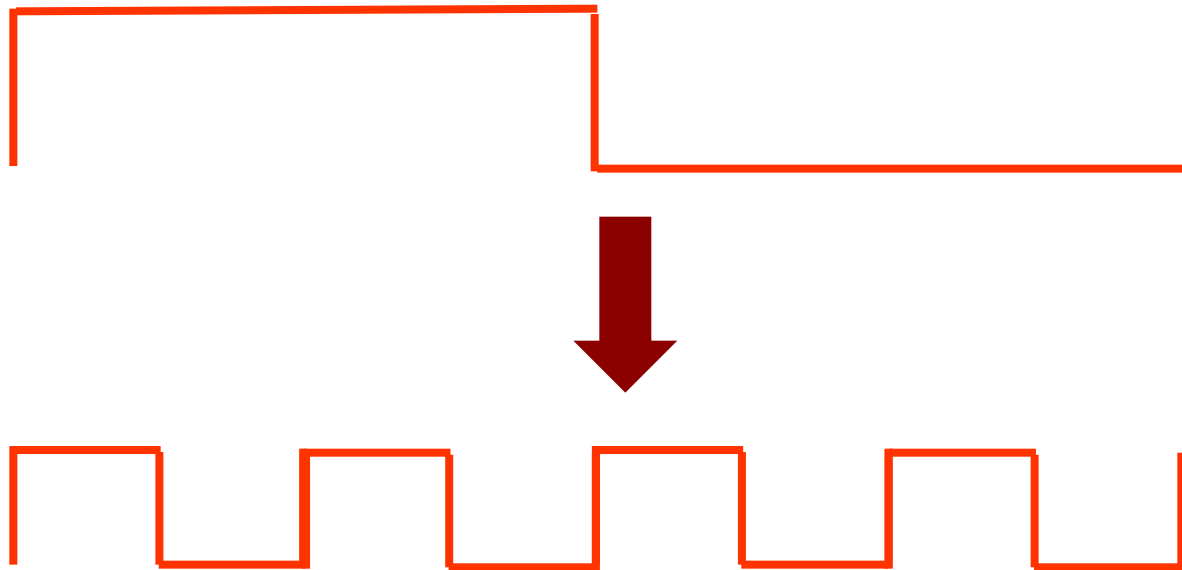
- Quebra estágios em subestágios (estágios menores)
Cada subestágio faz menos trabalho que estágio original
Pipeline com maior profundidade



Frequência do Clock em Superpipeline

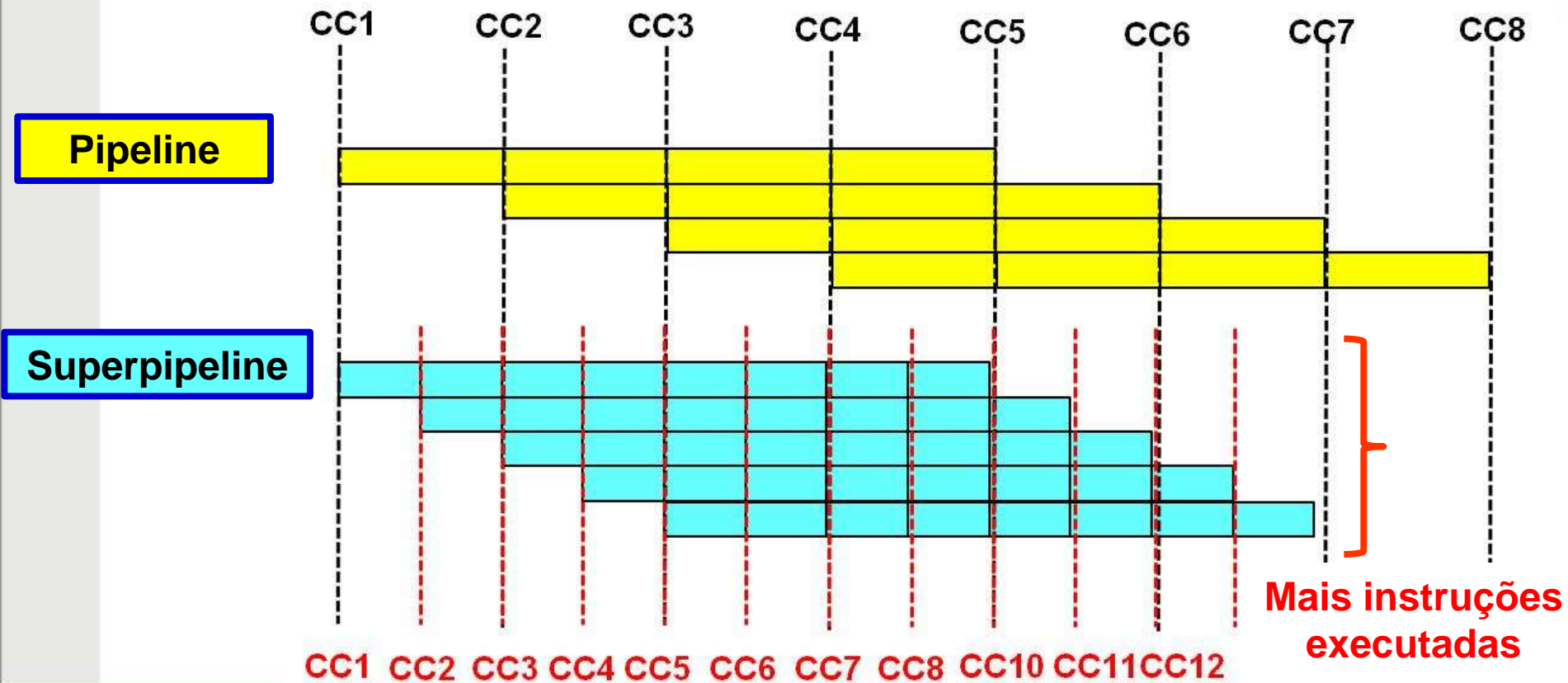
- Estágios menores demandam menos tempo para serem executados

Período menor \leftrightarrow Frequência maior



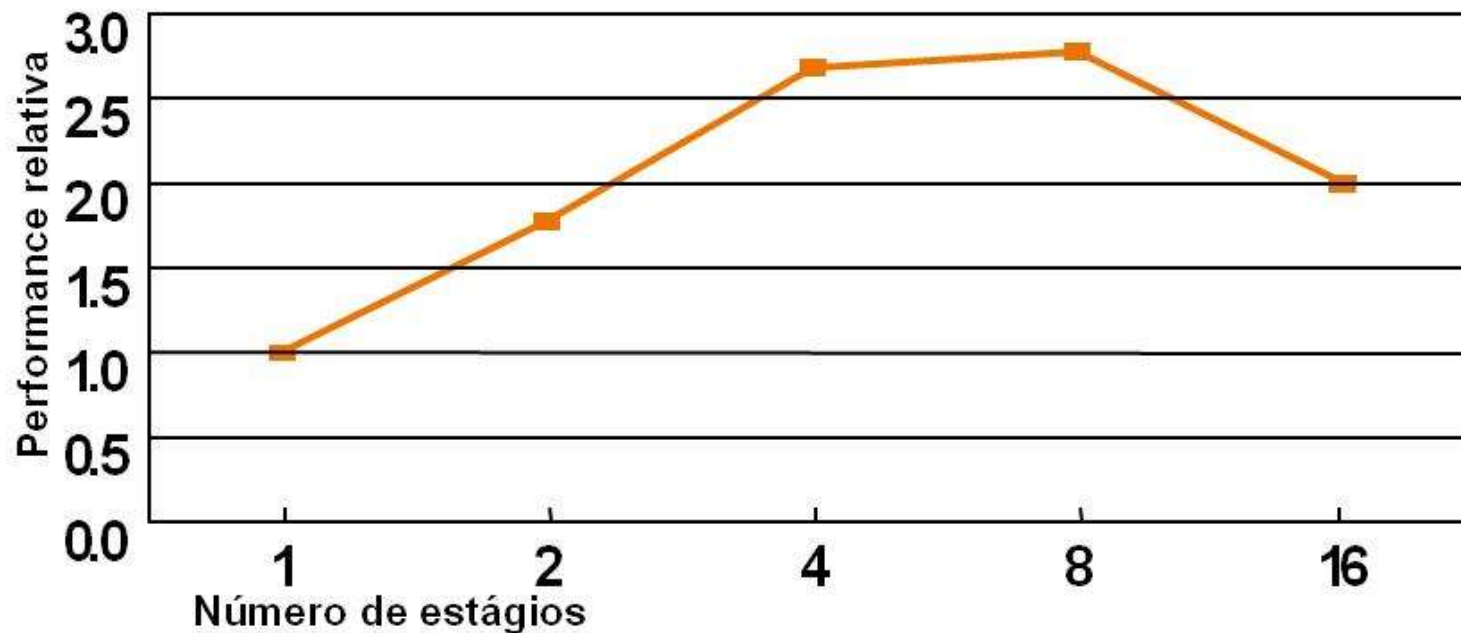
Pipeline x Superpipeline

- Superpipeline: Maior throughput → Melhor desempenho
Maior números de estágios, frequência maior de clock
Mais instruções podem ser processadas simultaneamente



Desempenho Relativo ao Número de Estágios

- Aumentar profundidade do pipeline nem sempre vai melhorar desempenho



Mais Sobre Superpipeline

- Superpipeline visa diminuir tempo de execução de um programa

Dependências degradam desempenho

- Número de estágios excessivos penalizam desempenho

Conflito de dados

- pipeline maior → mais dependências → mais retardos

Conflito de controle

- pipeline maior → mais estágios para preencher

Tempo dos registradores do pipeline (entre estágios)

- Limita tempo mínimo por estágio

- Maior custo de hardware

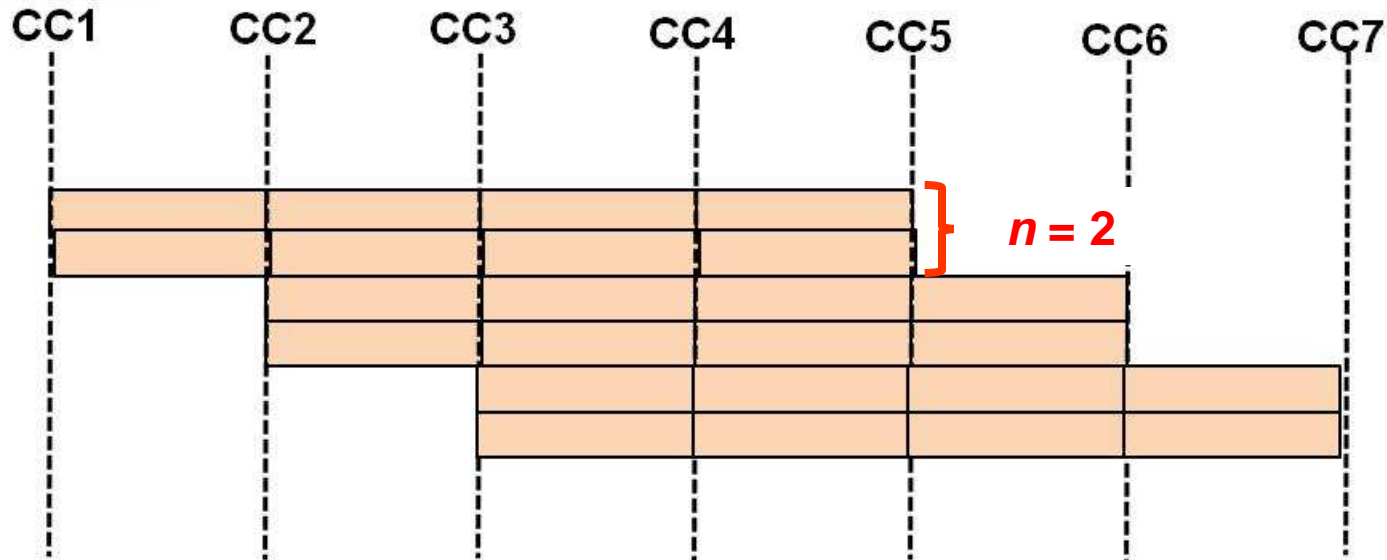
Superescalar

- Processador com n pipelines de instrução replicados
 n dá o grau do pipeline superescalar
- Instruções diferentes podem iniciar a execução ao mesmo tempo
- Requer replicação de recursos de HW
- Aplicável a arquiteturas RISC e CISC
 - RISC : melhor uso efetivo
 - CISC : implementação mais difícil

Idéia Geral de Processadores Superescalares

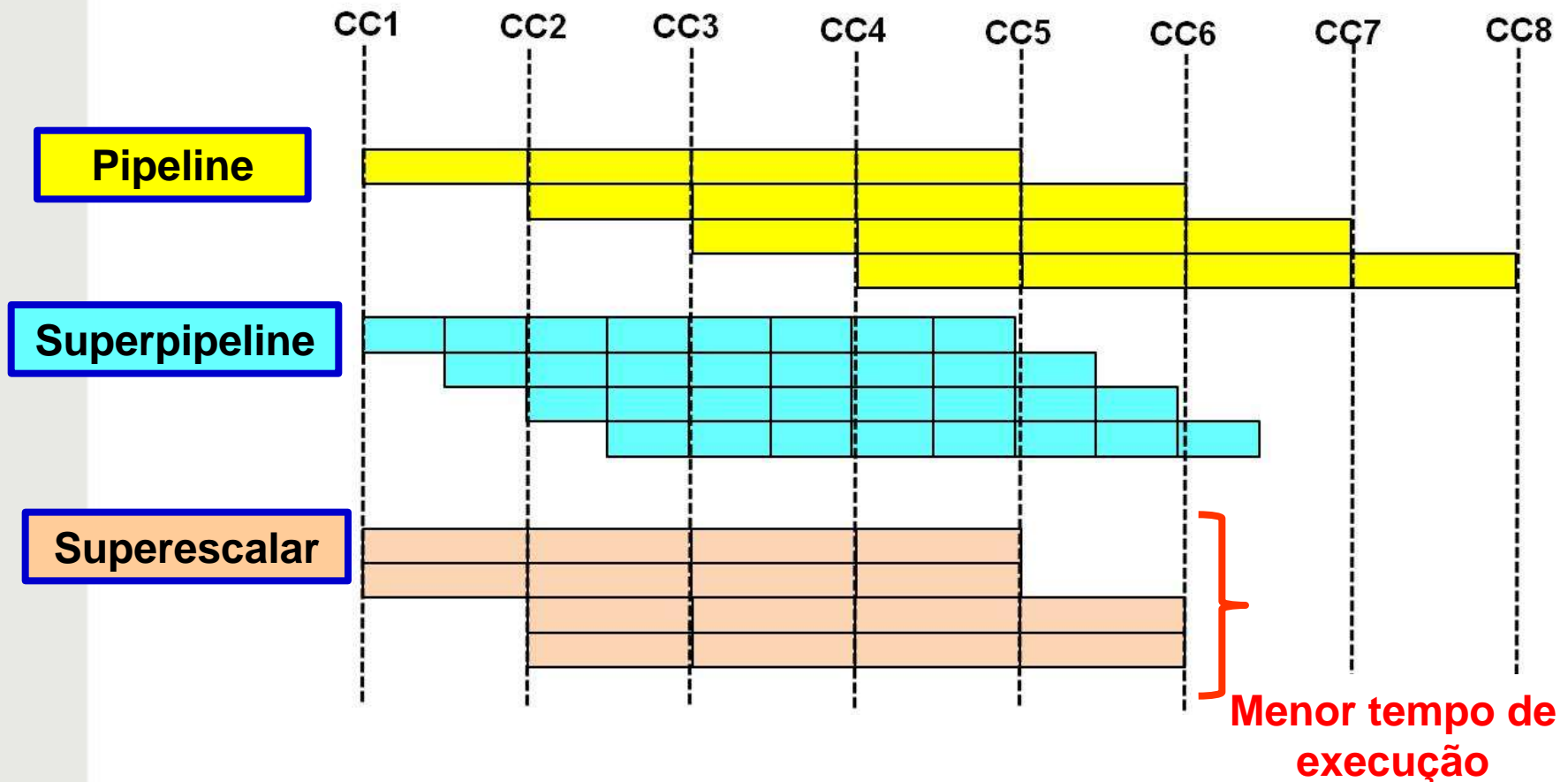
- Grupos de instruções podem ser executados ao mesmo tempo
Número n de instruções por grupo define o grau do pipeline superescalar

Superescalar

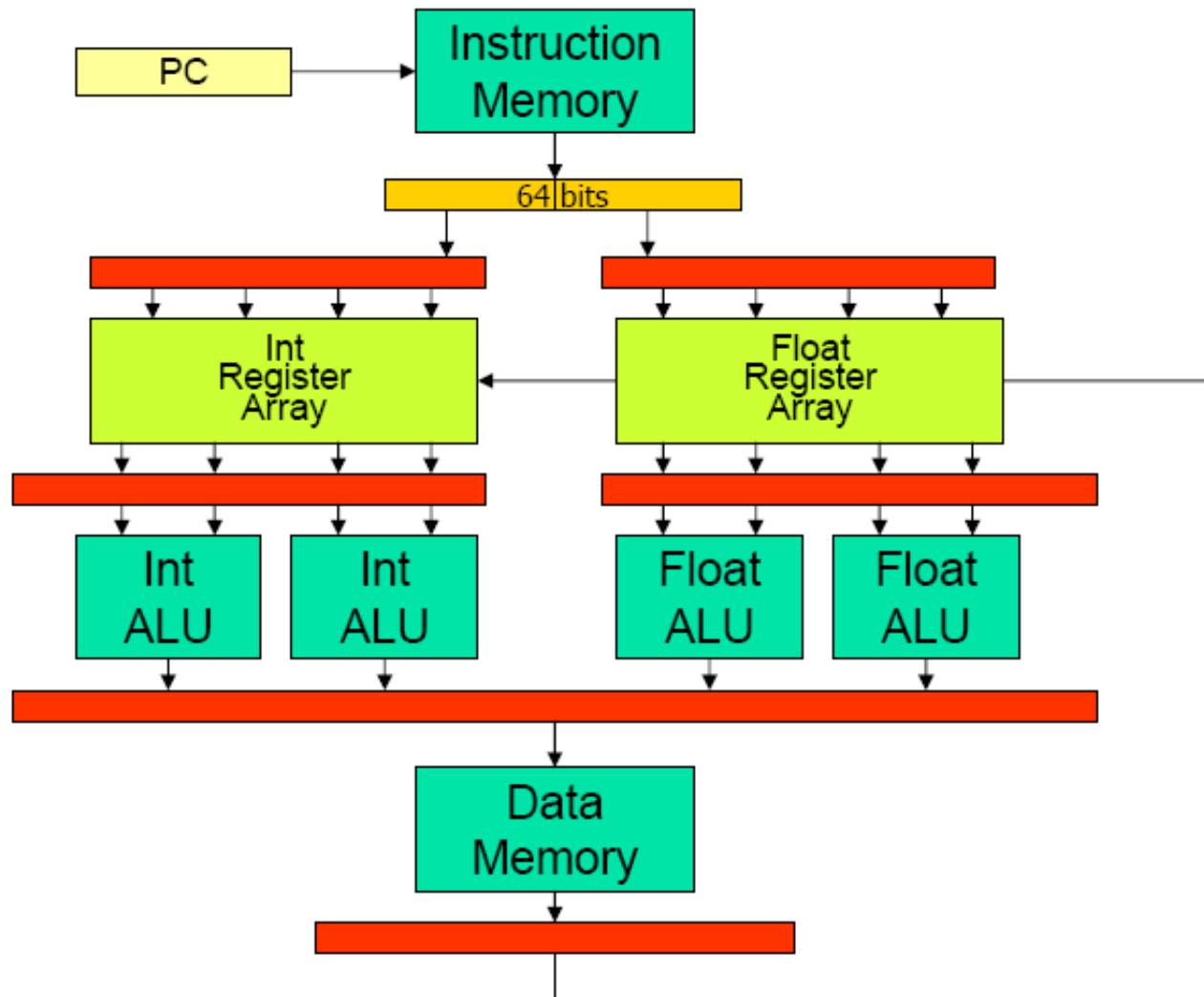


Pipeline x Superpipeline x Superescalar

- Superescalar: Paralelismo real → Maior throughput → Melhor desempenho



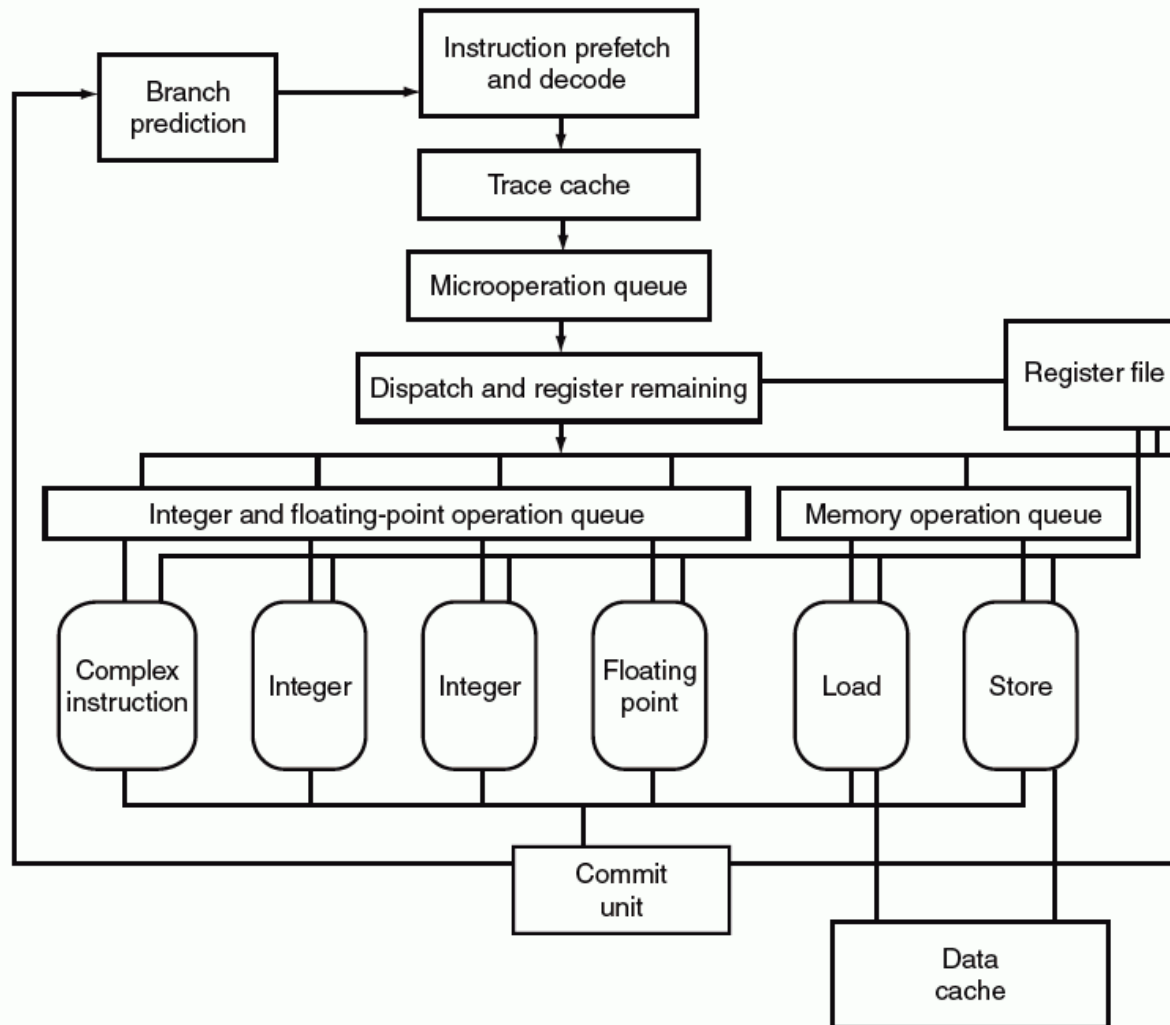
Replicação de Recursos de HW



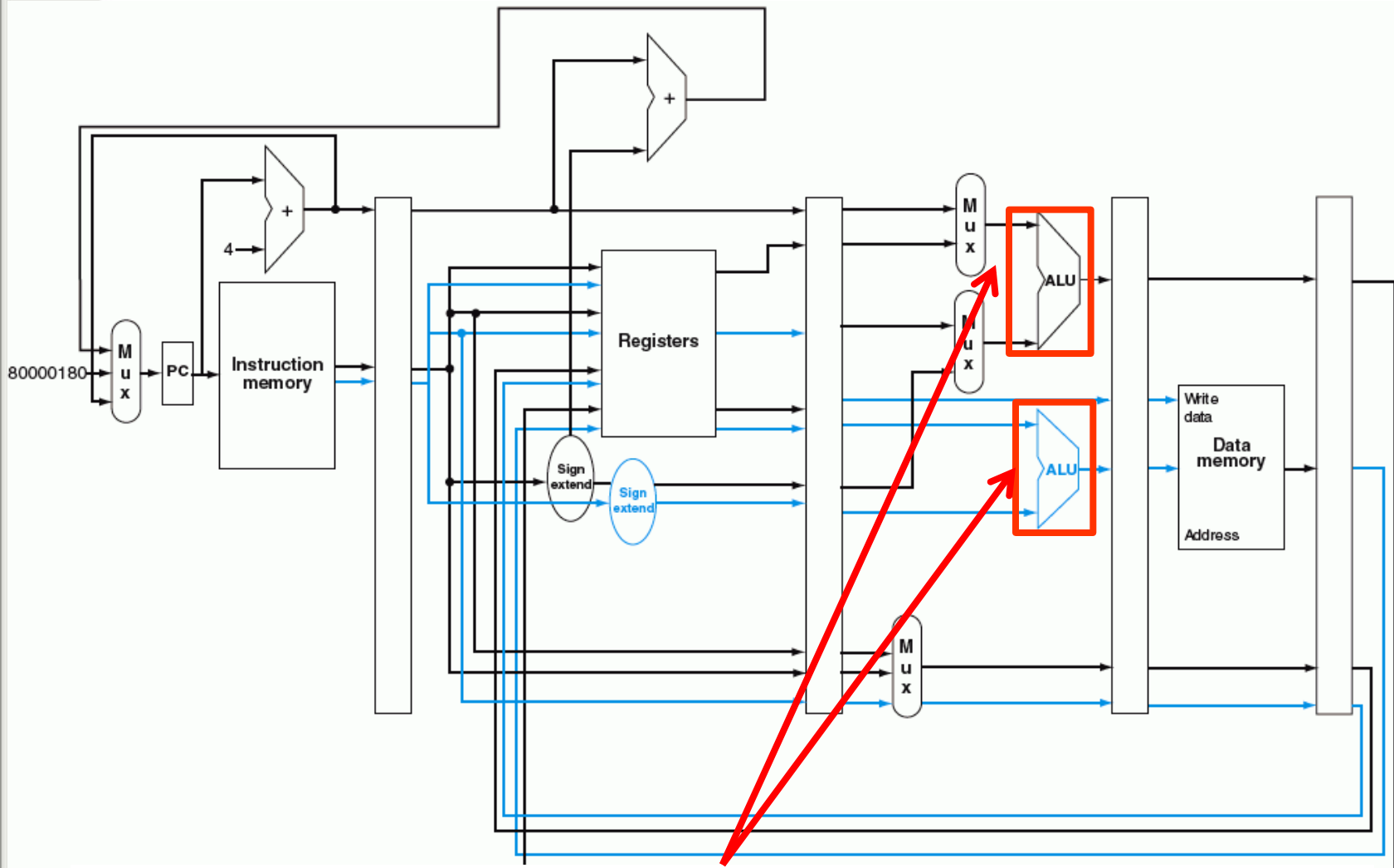
Exemplo de Superescalar: Pentium 4

■ Micro-arquitetura

Micro-instruções(operações) entram no pipeline superescalar

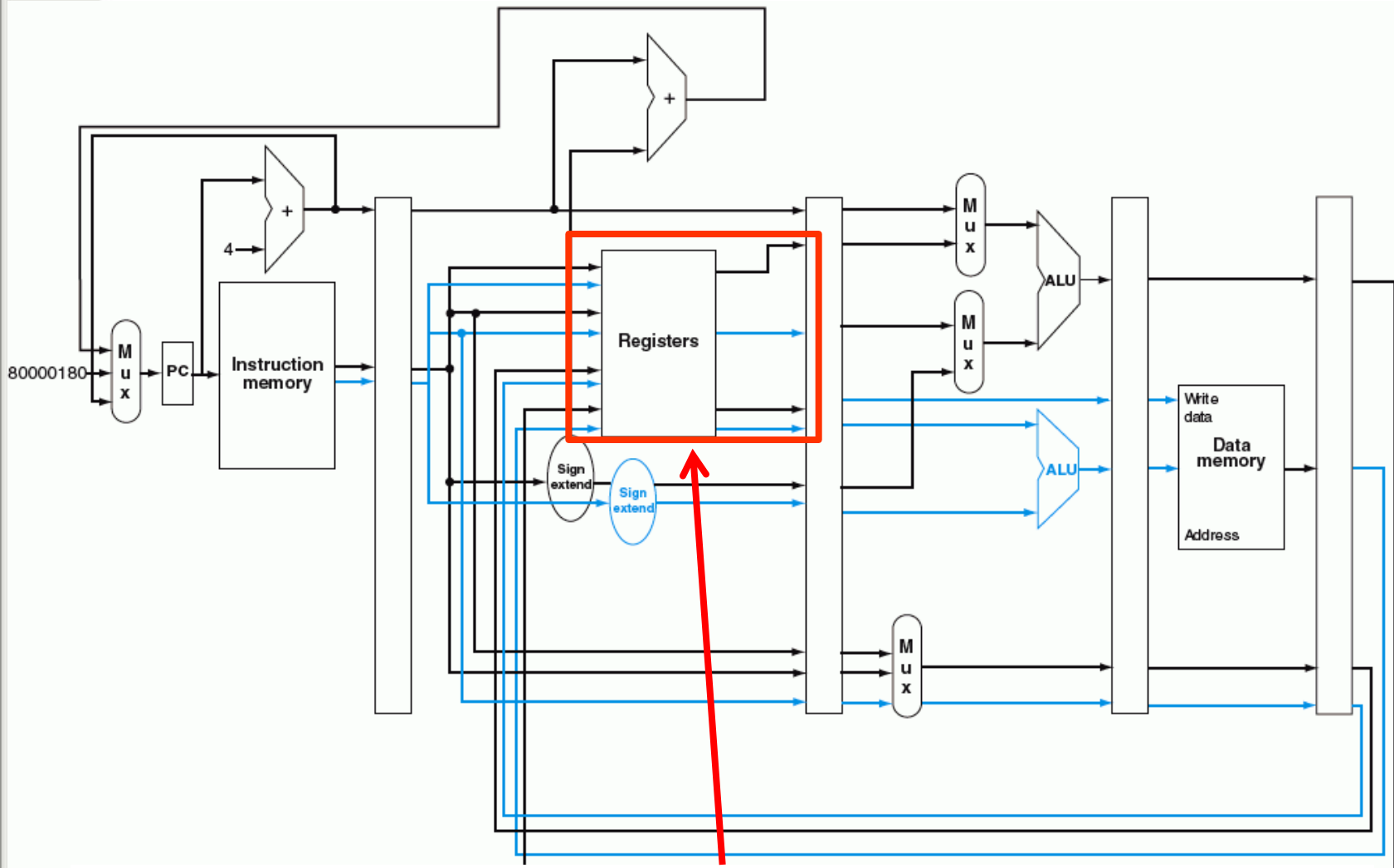


Processador Superscalar com ISA do MIPS



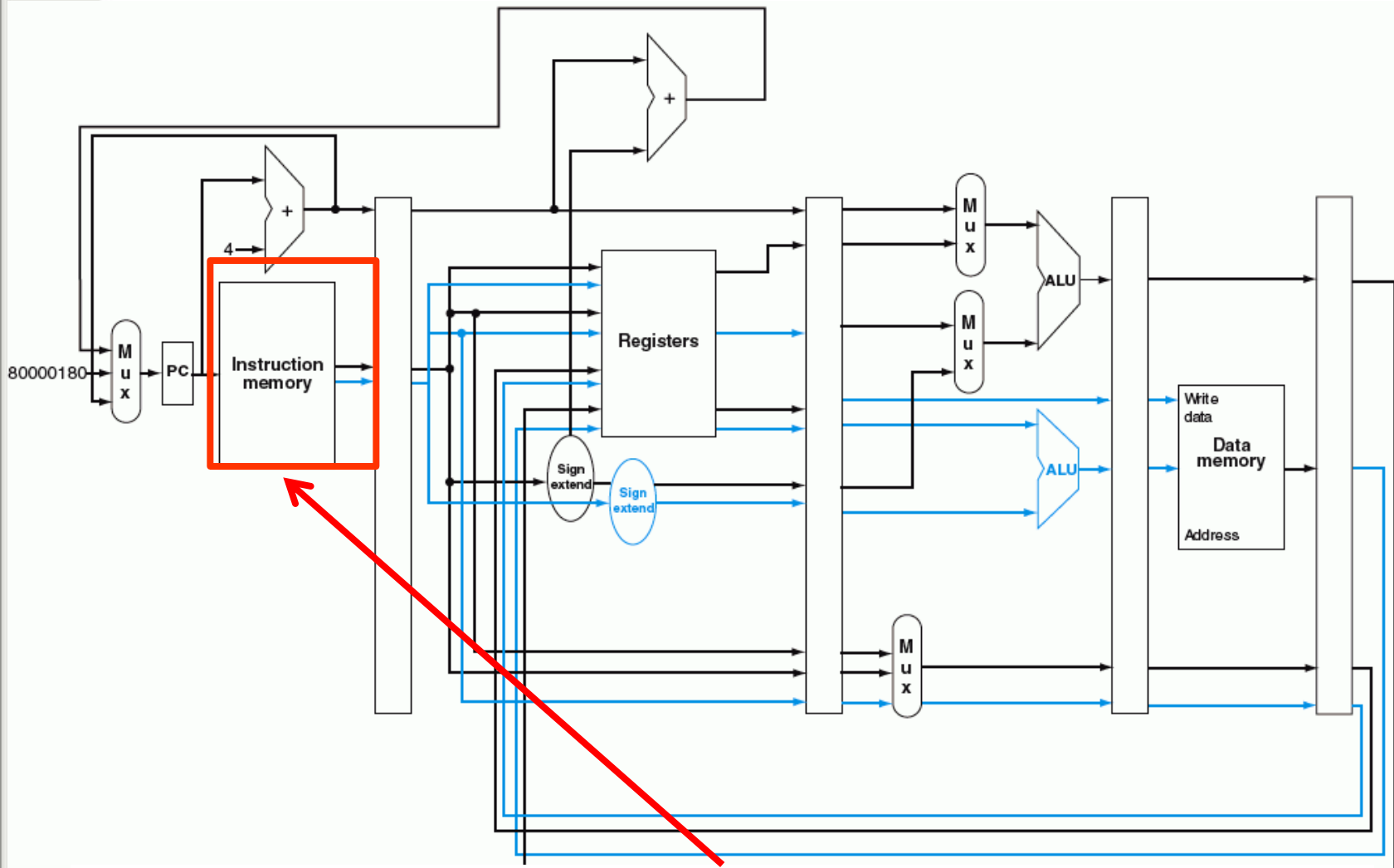
Replicação de ALU – Capacidade para realizar operações aritméticas/branches e de acesso a memória simultaneamente

MIPS Superescalar – Banco de Registradores



Banco de registradores capazes de fazer 4 leituras e 2 escritas

MIPS Superescalar – Memória de Instrução



Memória de Instrução permite leitura de 64 bits

Operando com MIPS Superescalar

- Leitura da instrução de memória deve ser de 64 bits
Execução das instruções aos pares
Primeiros 32 bits, instrução aritmética/branch, e os outros 32 bits, instrução load/store

Instruction type	Pipe stages							
ALU or branch instruction	IF	ID	EX	MEM	WB			
Load or store instruction	IF	ID	EX	MEM	WB			
ALU or branch instruction		IF	ID	EX	MEM	WB		
Load or store instruction		IF	ID	EX	MEM	WB		
ALU or branch instruction			IF	ID	EX	MEM	WB	
Load or store instruction			IF	ID	EX	MEM	WB	
ALU or branch instruction				IF	ID	EX	MEM	WB
Load or store instruction				IF	ID	EX	MEM	WB

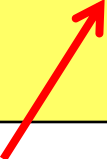
Executando um Programa com o MIPS Superescalar

Programa lê elementos do array na ordem inversa e soma valor contido em um registrador a cada elemento do array

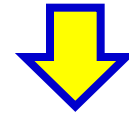
```
Loop: lw $t0,0($s1) # $t0 = elemento do array  
      add $t0,$t0,$s2 # soma valor contido em $s2  
      sw $t0,0($s1) # armazena resultado  
      addi $s1,$s1,-4 # decrementa ponteiro  
      bne $s1,$zero,Loop # desvia se $s1 != 0
```

Mapeando o Programa para o MIPS Superescalar

```
Loop: lw $t0,0($s1) #$t0 = elemento do array
      add $t0,$t0,$s2 # soma valor contido em $s2
      sw $t0,0($s1) # armazena resultado
      addi $s1,$s1,-4 # decrementa ponteiro
      bne $s1,$zero,Loop # desvia se $s1 != 0
```



Dependência de dados obriga a reorganização do código



	ALU ou Branch	Load ou Store	Clock
Loop	nop	lw \$t0, 0(\$s1)	1
	addi \$s1, \$s1, -4	nop	2
	add \$t0, \$t0, \$s2	nop	3
	bne \$s1,\$zero,Loop	sw \$t0, 4(\$s1)	4

Desempenho do Programa com o MIPS Superescalar

	ALU ou Branch	Load ou Store	Clock
Loop	nop	lw \$t0, 0(\$s1)	1
	addi \$s1, \$s1, -4	nop	2
	add \$t0, \$t0, \$s2	nop	3
	bne \$s1,\$zero,Loop	sw \$t0, 4(\$s1)	4

- 4 ciclos por iteração (desprezando os ciclos para finalizar a execução da última instrução na última iteração)
- 4 ciclos para as 5 instruções
 $CPI = 4/5 = 0,8$
- Aproveita pouco o fato do processador ser superescalar
Muitos nops

Otimizando o Desempenho do Programa

- Compiladores possuem técnicas avançadas de otimização
 - Loop Unrolling
 - Replica-se corpo do laço para reduzir número de iterações
- Supondo que índice de laço fosse múltiplo de 4, pode-se buscar 4 elementos do array a cada iteração
- Utilização mais eficiente do pipeline superescalar
 - Porém, código fica maior

Otimizando o Desempenho do Programa

```
Loop: lw $t0,0($s1)
      add $t0,$t0,$s2
      sw $t0,0($s1)
      addi $s1,$s1,-4
      bne $s1,$zero,Loop
```

Loop Unrolling



```
Loop: addi $s1,$s1,-16
      lw $t0,0($s1)
      lw $t1,12($s1)
      add $t0,$t0,$s2
      lw $t2,8($s1)
      add $t1,$t1,$s2
      lw $t3,4($s1)
      add $t2,$t2,$s2
      sw $t0,16($s1)
      add $t3,$t3,$s2
      sw $t1,12($s1)
      sw $t2,8($s1)
      sw $t3,4($s1)
      bne $s1,$zero,Loop
```

Carregando 4
elementos por
iteração

Mapeando o Programa Otimizado para o MIPS Superescalar

	ALU ou Branch	Load ou Store	Clock
Loop	<code>addi \$s1, \$s1, -16</code>	<code>lw \$t0, 0(\$s1)</code>	1
	<code>nop</code>	<code>lw \$t1, 12(\$s1)</code>	2
	<code>add \$t0, \$t0, \$s2</code>	<code>lw \$t2, 8(\$s1)</code>	3
	<code>add \$t1, \$t1, \$s2</code>	<code>lw \$t3, 4(\$s1)</code>	4
	<code>add \$t2, \$t2, \$s2</code>	<code>sw \$t0, 16(\$s1)</code>	5
	<code>add \$t3, \$t3, \$s2</code>	<code>sw \$t1, 12(\$s1)</code>	6
	<code>nop</code>	<code>sw \$t2, 8(\$s1)</code>	7
	<code>bne \$s1, \$zero, Loop</code>	<code>sw \$t3, 4(\$s1)</code>	8

Desempenho do Programa Otimizado

	ALU ou Branch	Load ou Store	Clock
Loop	<code>addi \$s1, \$s1, -16</code>	<code>lw \$t0, 0(\$s1)</code>	1
	<code>nop</code>	<code>lw \$t1, 12(\$s1)</code>	2
	<code>addu \$t0, \$t0, \$s2</code>	<code>lw \$t2, 8(\$s1)</code>	3
	<code>addu \$t1, \$t1, \$s2</code>	<code>lw \$t3, 4(\$s1)</code>	4
	<code>addu \$t2, \$t2, \$s2</code>	<code>sw \$t0, 16(\$s1)</code>	5
	<code>addu \$t3, \$t3, \$s2</code>	<code>sw \$t1, 12(\$s1)</code>	6
	<code>nop</code>	<code>sw \$t2, 8(\$s1)</code>	7
	<code>bne \$s1, \$zero, Loop</code>	<code>sw \$t3, 4(\$s1)</code>	8

- 8 ciclos para 4 iterações ou 2 ciclos por iteração

- 14 instruções

CPI = $8/14$ ou 0,57

Escolhendo as Instruções que são Executadas em Paralelo

- Escolha de quais instruções serão executadas em paralelo pode ser feita por hardware ou software
- Hardware
 - Lógica especial deve ser inserida no processador
 - Decisão em tempo de execução (escolha dinâmica)
- Software
 - Compilador
 - Rearruma código e agrupa instruções
 - Decisão em tempo de compilação (escolha estática)

Escolha pelo HW

- O termo **Superescalar** é mais associado a processadores que utilizam o hardware para fazer esta escolha
- CPU decide se 0, 1, 2, ... instruções serão executadas a cada ciclo
 - Escalonamento de instruções
 - Evitando conflitos
- Evita a necessidade de escalonamento de instruções por parte do compilador
 - Embora o compilador possa ajudar
 - Semântica do código é preservada pela CPU

Escalonamento Dinâmico pelo HW

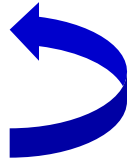
- Permite que CPU execute instruções fora de ordem para evitar retardos

Escrita nos bancos de registradores para completar instrução é feita em ordem

- Encontram-se processadores que permitem escrita fora de ordem também

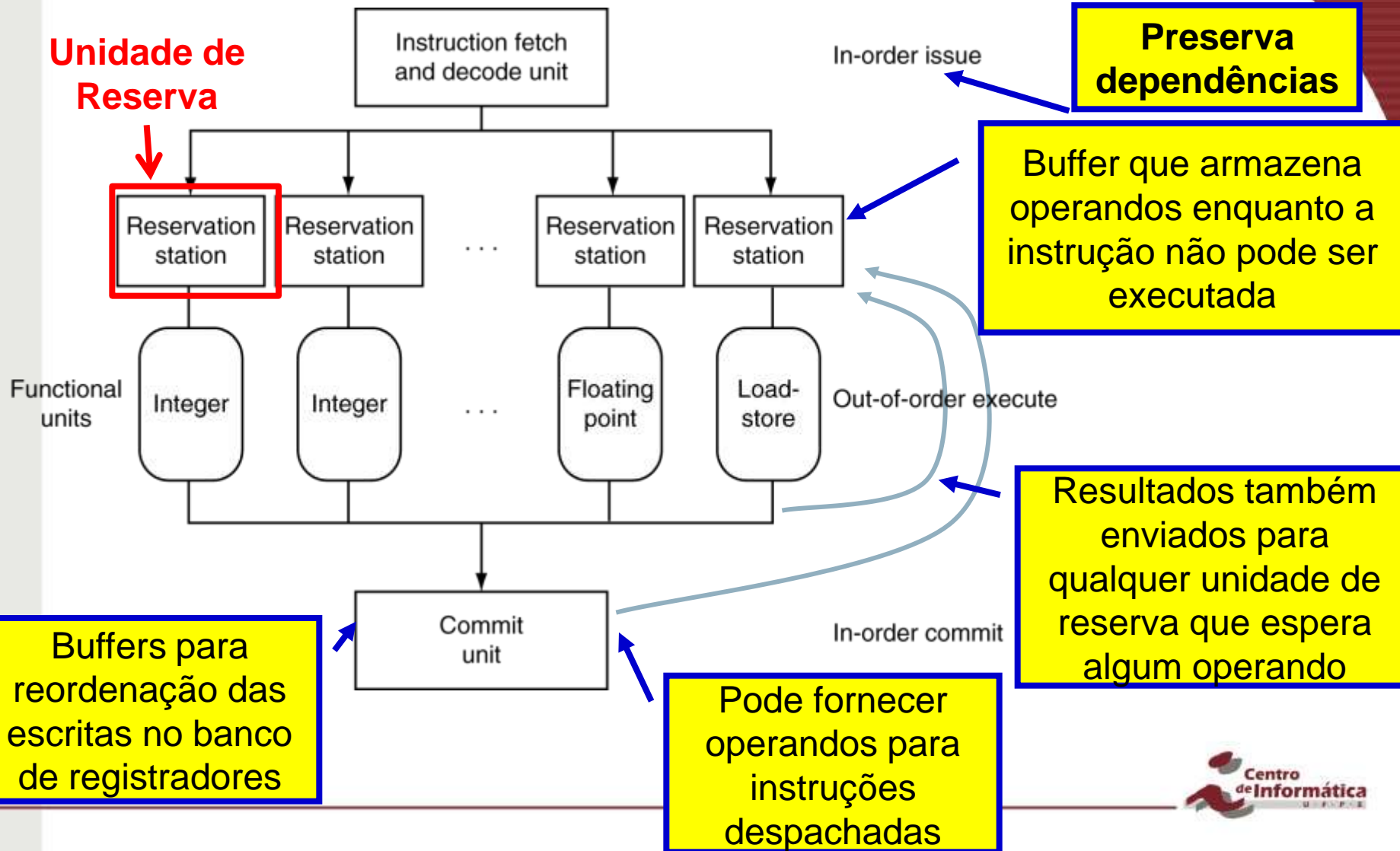
- Exemplo

```
lw      $t0, 20($s2)
add     $t1, $t0, $t2
sub     $s4, $s4, $t3
slti    $t5, $s4, 20
```



sub pode começar enquanto **add** está esperando por **lw**

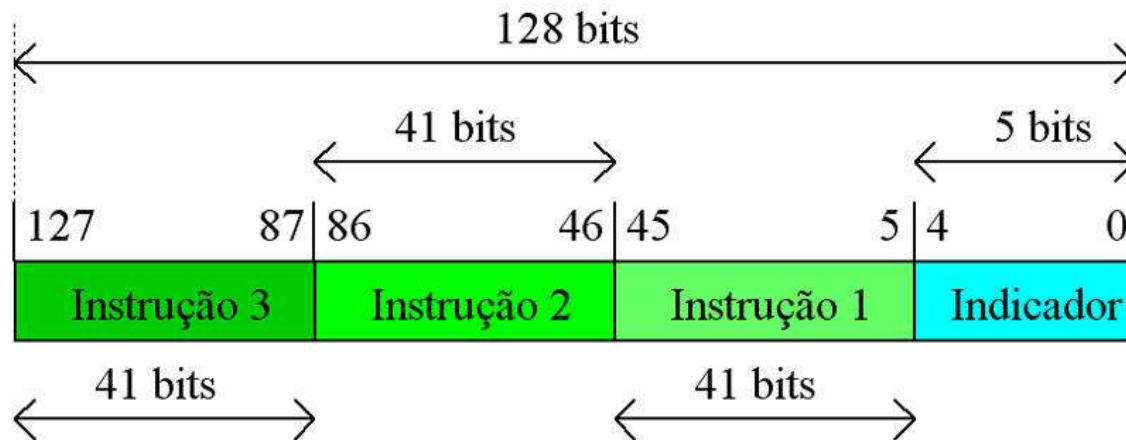
Implementação de Escalonamento Dinâmico



Escolha de Instruções pelo SW (Processadores VLIW)

- O termo **VLIW (Very Long Instruction Word)** é associado a processadores parecidos com superescalares mas que dependem do software(compilador) para fazer esta escolha
- O compilador descobre as instruções que podem ser executadas em paralelo e as agrupa formando uma longa instrução **(Very Long Instruction Word)** que será despachada para a máquina
 - Escalonamento de instruções
 - Evitando conflitos

Exemplo de VLIW: Intel Itanium



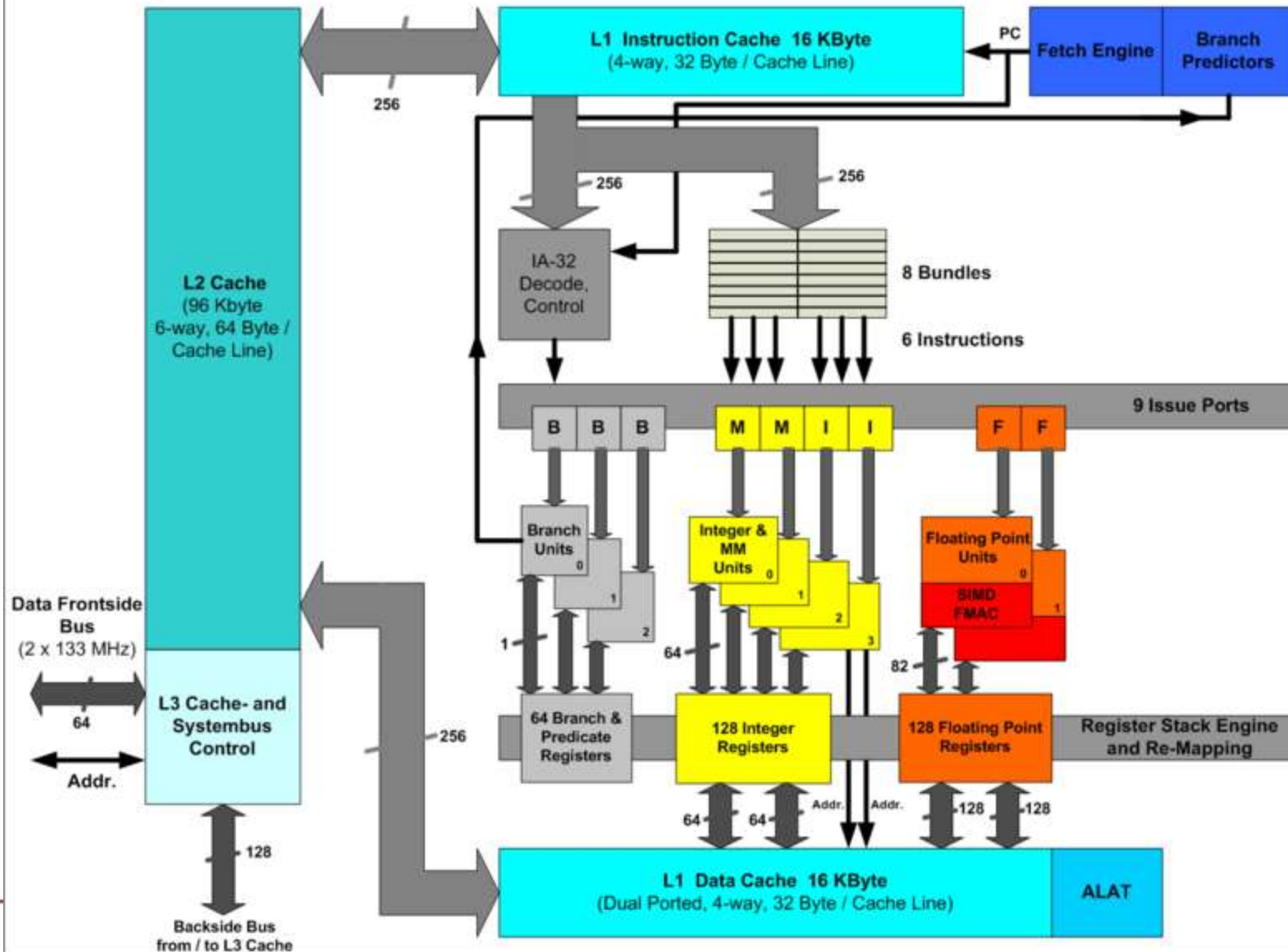
- Instruções são empacotadas

Cada pacote (*bundle*) contém 3 instruções e 128 bits

Cada instrução tem 41 bits

5 bits são utilizados para informar quais unidades funcionais serão usadas pelas instruções

Arquitetura do Intel Itanium



Revendo Dependências de Dados

$r3 := r0 \text{ op}_1 r5$	(i1)
$r4 := r3 \text{ op}_2 1$	(i2)
$r3 := r5 \text{ op}_3 1$	(i3)
$r7 := r3 \text{ op}_4 r4$	(i4)

- Dependência Verdadeira (*Read-After-Write – RAW*)
i2 e i1, i4 e i3, i4 e i2
- Anti-dependência (*Write-After-Read - WAR*)
i3 não pode terminar antes de i2 iniciar
- Dependência de Saída (*Write-After-Write – WAW*)
i3 não pode terminar antes de i1

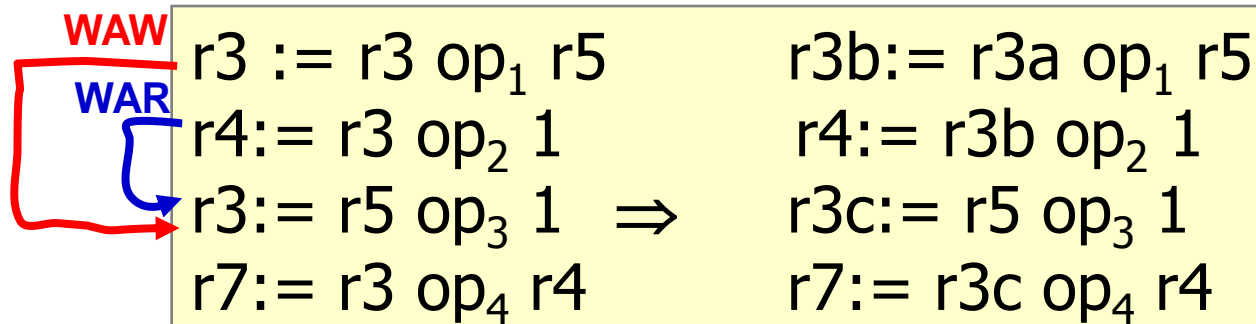
Tipos de Dependências de Dados em Pipeline

$r3 := r0 \text{ op}_1 r5$	(i1)
$r4 := r3 \text{ op}_2 1$	(i2)
$r3 := r5 \text{ op}_3 1$	(i3)
$r7 := r3 \text{ op}_4 r4$	(i4)

- Único tipo de dependência que causa problemas em um pipeline é a Dependência Verdadeira (*RAW*)
- Anti-dependência (*WAR*) não causa nenhum problema em um pipeline , pois instrução que escreve sempre o faz em um estágio posterior ao da leitura do registrador pela instrução anterior
Idem para *WAW*

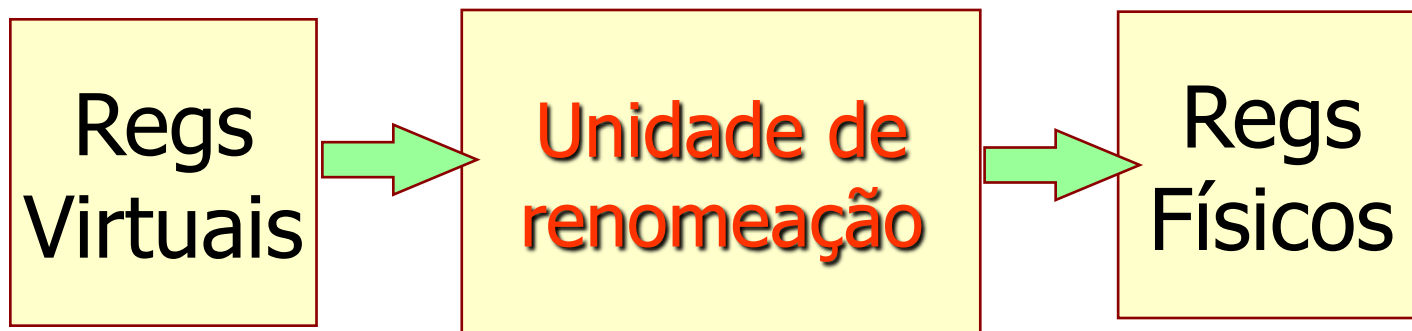
Resolvendo Dependências WAR e WAW

- Mesmo não causando nenhum problema para o pipeline, estas dependências podem “iludir” o compilador ou HW, devendo portanto ser eliminadas
- Solução: **Renomeação de registradores**



Renomeação de Registradores

- Pode ser feita tanto pelo compilador (SW) ou pelo HW



Renomeação de Registradores por SW

```
Loop: lw $t0,0($s1)
      addu $t0,$t0,$s2
      sw $t0,0($s1)
      addi $s1,$s1,-4
      bne $s1,$zero,Loop
```

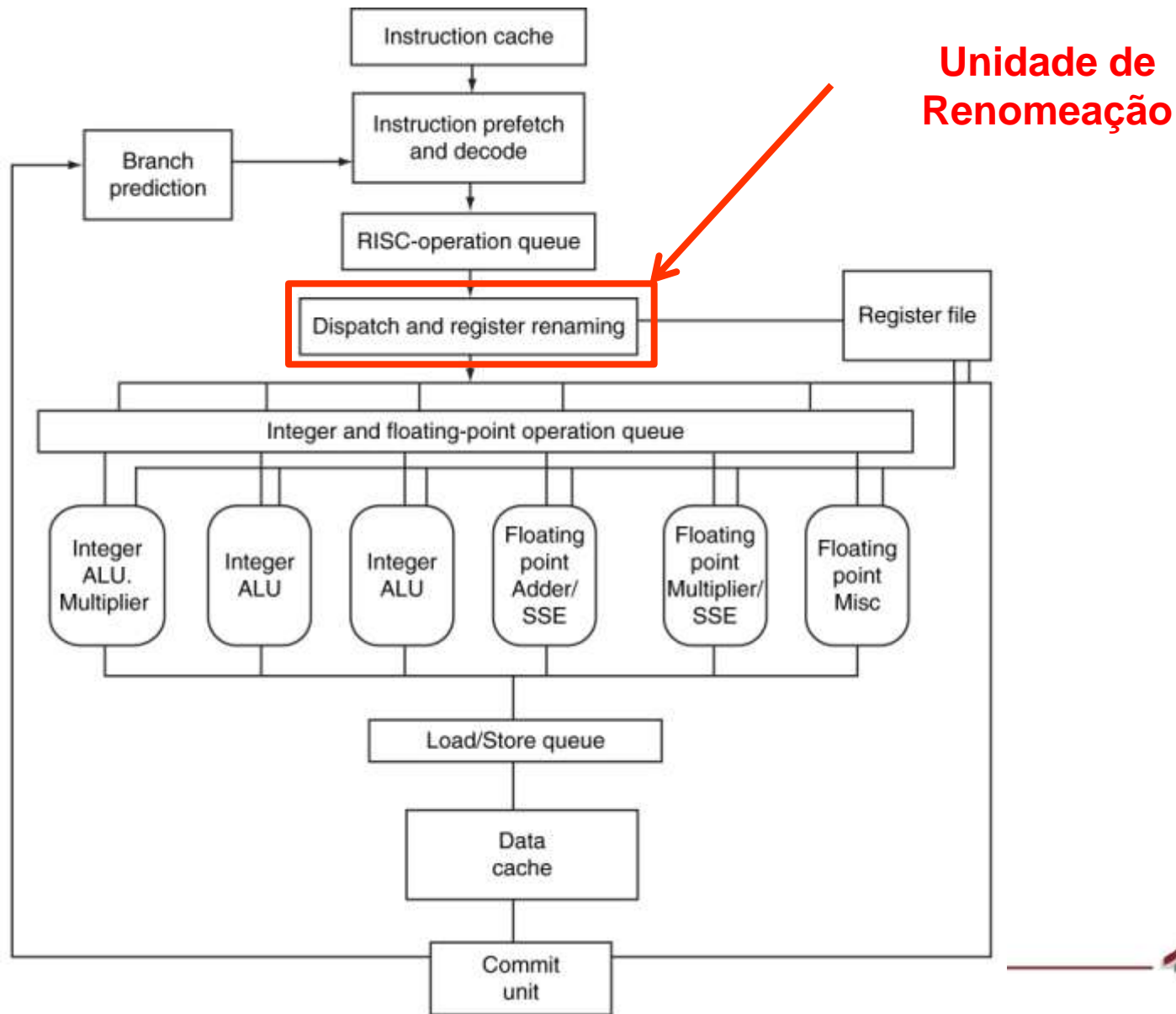
Loop Unrolling



```
Loop: addi $s1,$s1,-16
      lw $t0,0($s1)
      lw $t1,4($s1)
      addu $t0,$t0,$s2
      lw $t2,8($s1)
      addu $t1,$t1,$s2
      lw $t3,12($s1)
      addu $t2,$t2,$s2
      sw $t0,0($s1)
      addu $t3,$t3,$s2
      sw $t1,4($s1)
      sw $t2,8($s1)
      sw $t3,12($s1)
      bne $s1,$zero,Loop
```

Renomeação para
carregar 4
elementos
independentes

Renomeação por HW: AMD Opteron X4



Analizando o Superpipeline

- Superpipeline visa diminuir tempo de execução de um programa
 - Aumento da frequência do clock
 - Dependências degradam desempenho
- Número de estágios excessivos podem penalizar desempenho
 - Conflito de dados, controle
 - Overhead de passagem de estágios
- Maior custo de hardware
 - Mais registradores, mais lógica

Analizando o Superescalar...

- Visa reduzir o número médio de ciclos por instrução (CPI)
CPI < 1
Instruções podem iniciar ao mesmo tempo
- HW encarregado de escalonar instruções e detectar conflitos
Permite execução fora de ordem de instruções
Considera aspectos dinâmicos de execução
Lógica em HW mais complexa
- Maior custo de HW
Unidades de Reserva, lógica de escalonamento

Analizando o VLIW

- Simplifica HW transferindo para o compilador a lógica de detecção do paralelismo
 - Circuitos mais simples
 - Clock mais rápido
- Aspectos dinâmicos não são analisados pelo compilador, o que pode causar retardos inesperados
 - Exemplo: Se dado não estiver na cache
- Mudanças no pipeline de um VLIW requer mudanças no compilador