

*Instruction Level Parallelism*  
Super Escalaridade

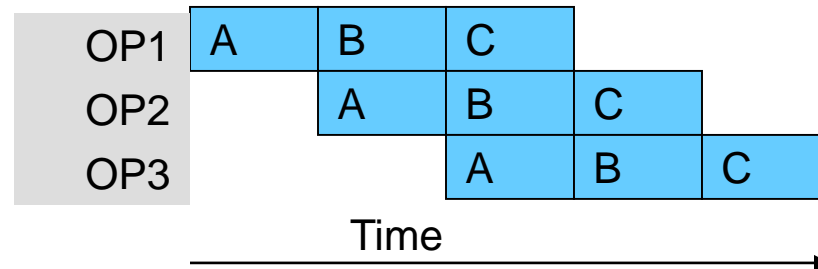
Arquitetura de Computadores  
Mestrado Integrado em  
Engenharia Informática

# Material de Apoio

- “*Computer Organization and Design: The Hardware / Software Interface*”  
David A. Patterson, John L. Hennessy; 5th Edition, 2013
  - Secção 4.10 (pags. 332 .. 344) – Parallelism via instructions
  - Secção 4.11 (pags. 344 .. 351) – Real Stuff
- “*Computer Systems: a Programmer's Perspective*”; Randal E. Bryant, David R. O'Hallaron--Pearson (2nd ed., 2011)
  - Secção 5.7 (pags. 496 .. 500) – Understanding modern processors
  - ADVANCED MATERIAL:  
Secção 5.7.2 .. 5.9 (pags. 523 .. 446) – Understanding modern processors

# Pipelining

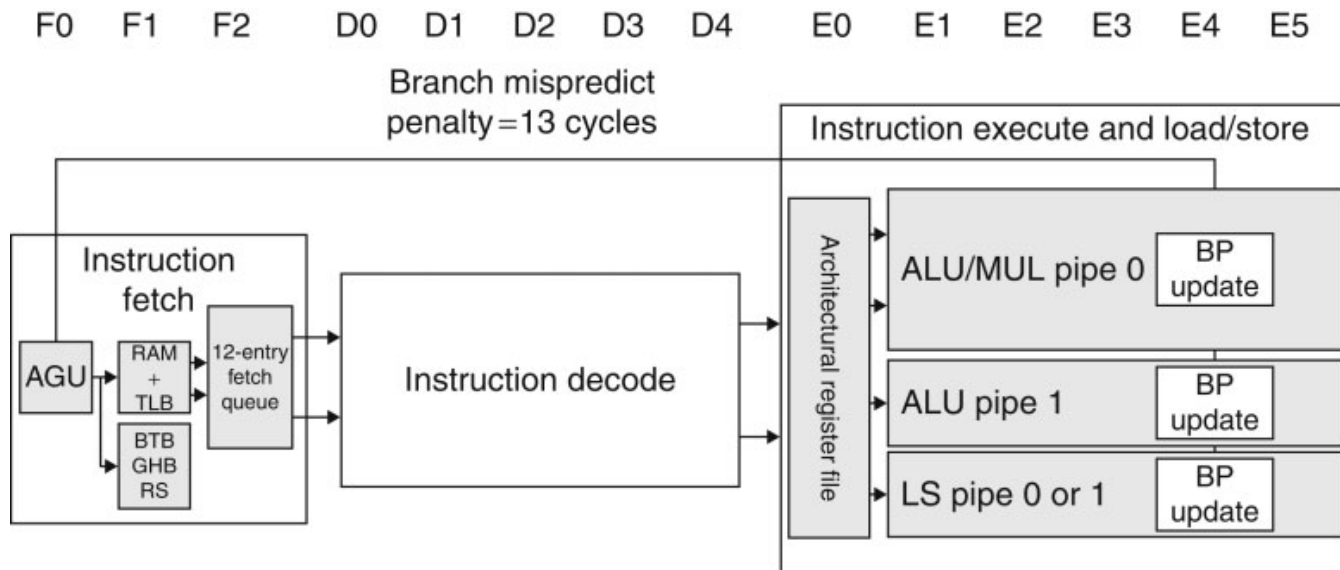
- Explora ILP (*Instruction Level Parallelism*) por permitir a execução simultânea de múltiplas instruções em diferentes estágios



- Permite aumentar a frequência, relativamente a organizações de ciclo único
- O CPI ideal é 1, mas difícil de manter devido a
  - dependências de dados;
  - dependências de controlo;
  - atrasos nos acessos à memória

# Multiple Issue

- Uma abordagem complementar consiste em ter múltiplos *pipelines*, permitindo a execução simultânea de múltiplas instruções -> *multiple issue*



[ FIGURE 4.75 – Patterson & Hennessy; Computer Organization & Design – 5<sup>th</sup> Edition, Elsevier, 2013  
--- The A8 pipeline]

# *Multiple Issue*

- As múltiplas unidades funcionais permitem o lançamento (*issue*) de mais do que uma instrução por ciclo, logo, potencialmente podemos ter  $CPI < 1$
- Um processador com 4 unidades funcionais, cada organizada como um *pipeline* de 5 estágios, pode ter 20 instruções a ser executadas simultaneamente em cada ciclo
- Os processadores actuais mais avançados tentam lançar (*issue*) 3 a 6 instruções por ciclo

# *Multiple Issue: Limitações*

- Existem no entanto limitações ao número de instruções que podem executar em simultâneo:
  - dependências de dados e/ou controlo
  - disponibilidade de recursos: número e tipo de unidades funcionais, latência de diferentes instruções, atrasos no acesso à memória
- Quem determina quais as instruções que podem ser lançadas simultaneamente?

<i>static issue</i>	<i>dynamic issue</i>
compilador	processador
	Embora o compilador possa ordenar as instruções por forma a facilitar o <i>multiple issue</i>

# *Multiple Issue: execução especulativa*

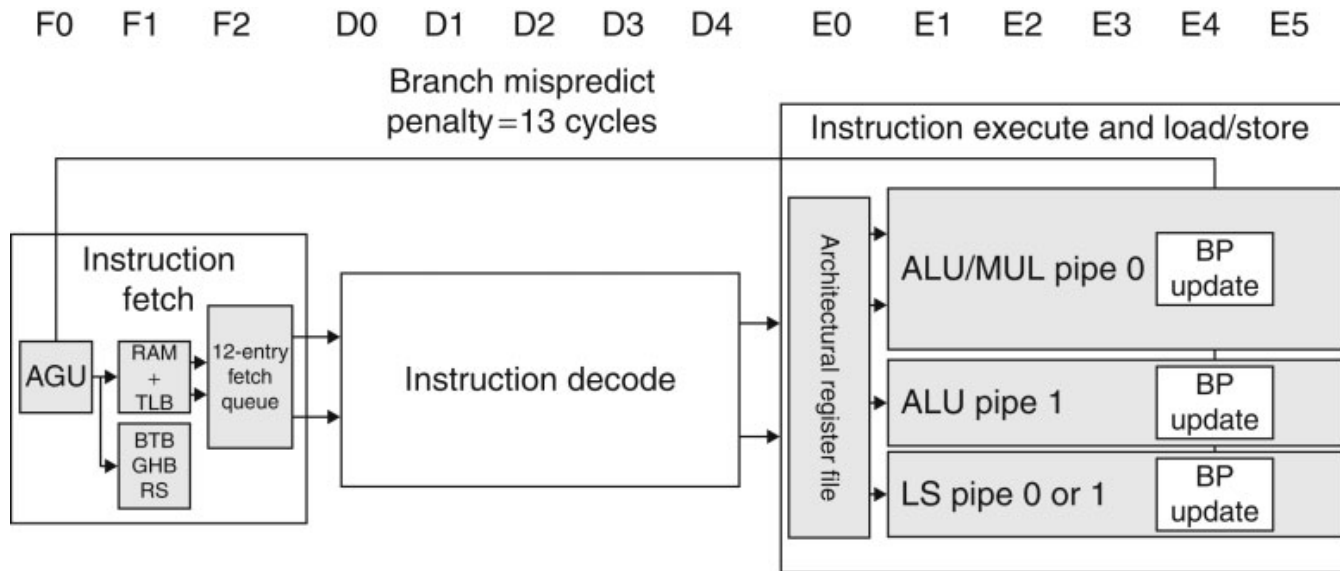
- O processador **especula** sobre o resultado de uma instrução para permitir o lançamento de outras instruções dependentes desta.
  - Ex.: especular se um salto é tomado ou não
  - Ex.: especular que um `store` que precede um `load` não se refere ao mesmo endereço, podendo executar o `load` antes, ou simultaneamente, com o `store`
- A especulação requer um mecanismo de recuperação, no caso da previsão ser errada
  - *static issue* (compilador): código adicional para verificação e correcção
  - *dynamic issue* (processador): *buffering* dos resultados especulativos, que apenas são escritos (*committed*) quando confirmados

# *Static Multiple Issue*

- O compilador agrupa em pacotes as instruções que serão lançadas (*issued*) em cada ciclo do relógio
- Estes pacotes de instruções concatenadas podem ser vistos como uma única instrução com múltiplas operações:  
*VLIW – Very Long Instruction Word*
- Uma VLIW comporta tantas instruções quantos os *pipelines* (unidades funcionais) disponíveis
- O compilador é responsável por:
  - garantir que não há dependências dentro de uma VLIW
  - que as instruções integradas numa VLIW podem ser executadas pelas unidades funcionais
  - resolver dependências de controlo ou dependências de dados entre diferentes VLIW



# Static Multiple Issue – Exemplo: ARM Cortex A8



Pipe 0	Pipe 1	Válido ?
movl \$10, %r1	movl (%r2), %r3	
movl \$10, %r1	addl %r1, %r3	
movl (%r2), %r3	movl (%r4), %r5	

# Static Multiple Issue – Exemplo: MIPS

- Considere uma versão com *static multiple issue* do MIPS, com 2 *pipelines*:
  - Pipe 0 : ALU ou *branch*
  - Pipe 1 : *load / store*

Execução fora de ordem

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

CC	Pipe 0 (ALU/Branch)	Pipe 1 (LS)
1	addi \$s1, \$s1, -4	lw \$t0, 0(\$s1)
2	addu \$t0, \$t0, \$s2	<b>nop</b>
3	bne \$s1, \$zero, Loop	sw \$t0, 4(\$s1)
CPI	<b>3 / 5 = 0.6</b>	

# Loop Unrolling– Exemplo: MIPS

Loop:

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

Loop:

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

anti-dependência -> *register renaming*

CC	Pipe 0 (ALU/Branch)	Pipe 1 (LS)
1	addi \$s1,\$s1,-8	lw \$t0, 0(\$s1)
2	addu \$t0,\$t0,\$s2	lw \$t1, 4(\$s1)
3	addu \$t1,\$t1,\$s2	sw \$t0, 8(\$s1)
4	bne \$s1,\$zero,Loop	sw \$t1, 4(\$s1)
CPI		4 / 8 = 0.5

# Loop Unrolling

- Consiste em fazer múltiplas cópias do corpo de um ciclo (normalmente feito pelo compilador)

```
for (i=0 ; i<N ; i++) {  
    a[i] = a[i] + var;  
}
```

```
for (i=0 ; i<N ; i+=3) {  
    a[i] = a[i] + var;  
    a[i+1] = a[i+1] + var;  
    a[i+2] = a[i+2] + var;  
}
```

- O seu efeito no desempenho nem sempre é positivo
- Contribui para:
  - disponibilizar mais instruções independentes para aumentar o ILP, e consequentemente reduzir o CPI
  - reduzir o número de instruções executadas

## *Static Multiple Issue: Limitação*

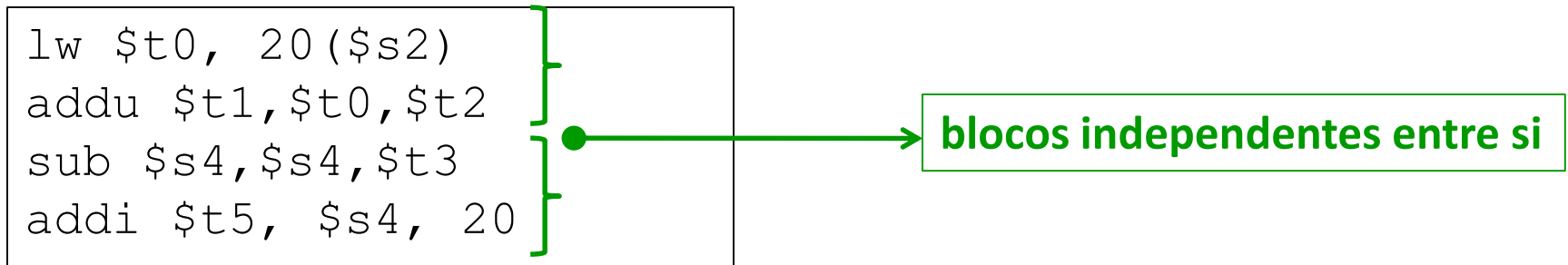
- O código gerado depende da organização do processador (número, tipo e desempenho das unidades funcionais)
- A mudança para diferentes implementações da mesma arquitectura:
  - frequentemente exige recompilação para que o código execute;
  - noutros casos, exige recompilação para manter um desempenho aceitável

# *Dynamic Multiple Issue: SuperEscalaridade*

- **SuperEscalaridade:** o processador selecciona dinamicamente, em tempo de execução, quais as instruções a executar em cada *pipeline* (unidade funcional)
- Diminui a dependência da compilação, relativamente à abordagem estática, pois é o *hardware* que selecciona as instruções de acordo com a sua própria organização

# SuperEscalaridade: *static in-order scheduling*

- ***static in-order scheduling***: instruções seleccionadas na ordem com que aparecem no programa, baseado na disponibilidade dos operandos e de unidades funcionais apropriadas



CC	Pipe 0 (ALU/Branch)	Pipe 1 (LS)
1	nop	lw \$t0, 20(\$s2)
2	addu \$t1, \$t0, \$s2	nop
3	sub \$s4, \$s4, \$t3	nop
4	addi \$t5, \$s4, 20	nop
CPI	4 / 4 = 1	

# SuperEscalaridade: *dynamic out-of-order scheduling*

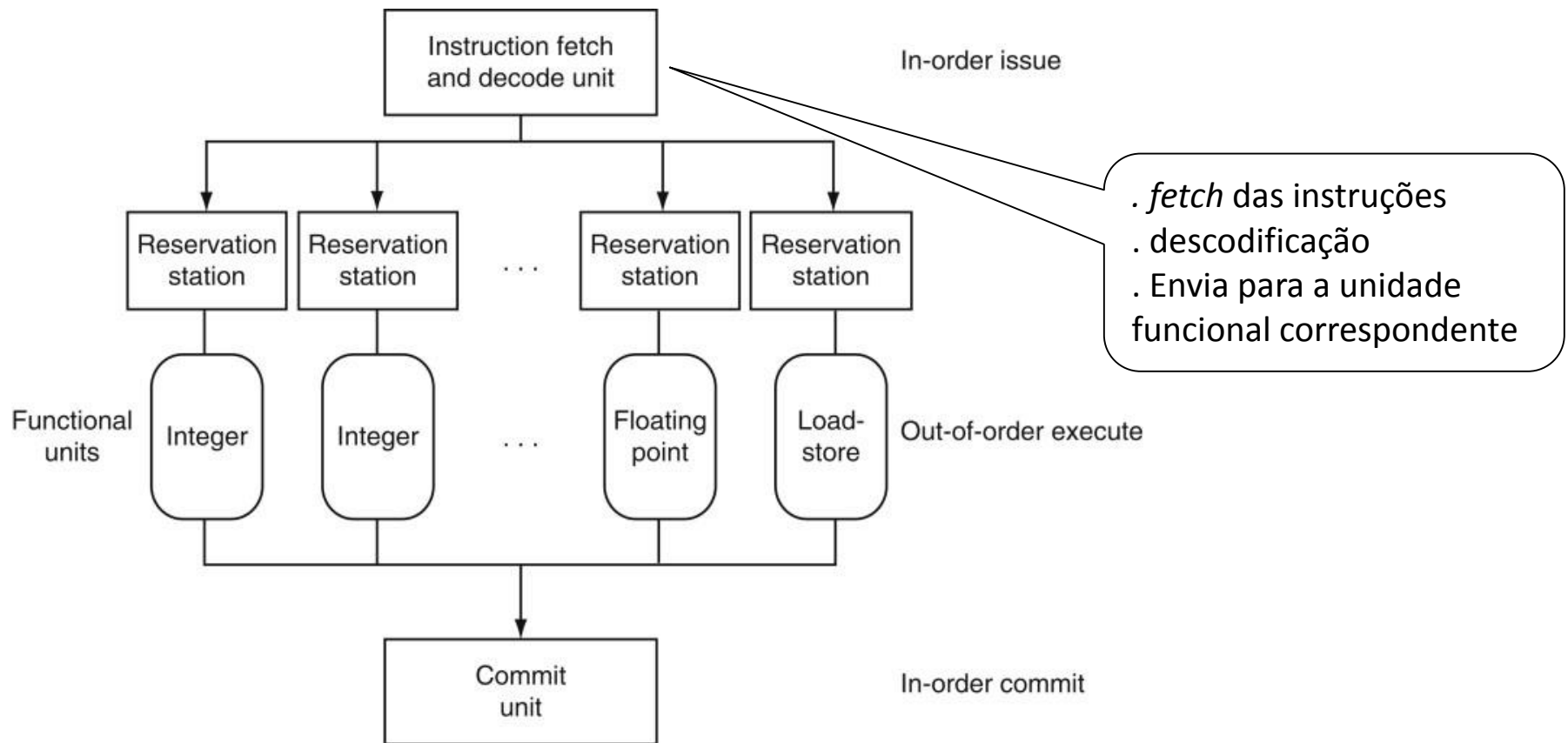
- ***dynamic out-of-order scheduling***: as instruções são reordenadas pelo processador para maximizar o CPI, garantindo a correcta execução do programa

```
lw $t0, 20($s2)
addu $t1,$t0,$t2
sub $s4,$s4,$t3
addi $t5, $s4, 20
```

CC	Pipe 0 (ALU/Branch)	Pipe 1 (LS)
1	sub \$s4,\$s4,\$t3	lw \$t0, 20(\$s2)
2	addu \$t1,\$t0,\$s2	nop
3	addi \$t5, \$s4, 20	nop
<b>CPI</b>	<b>3 / 4 = 0.75</b>	

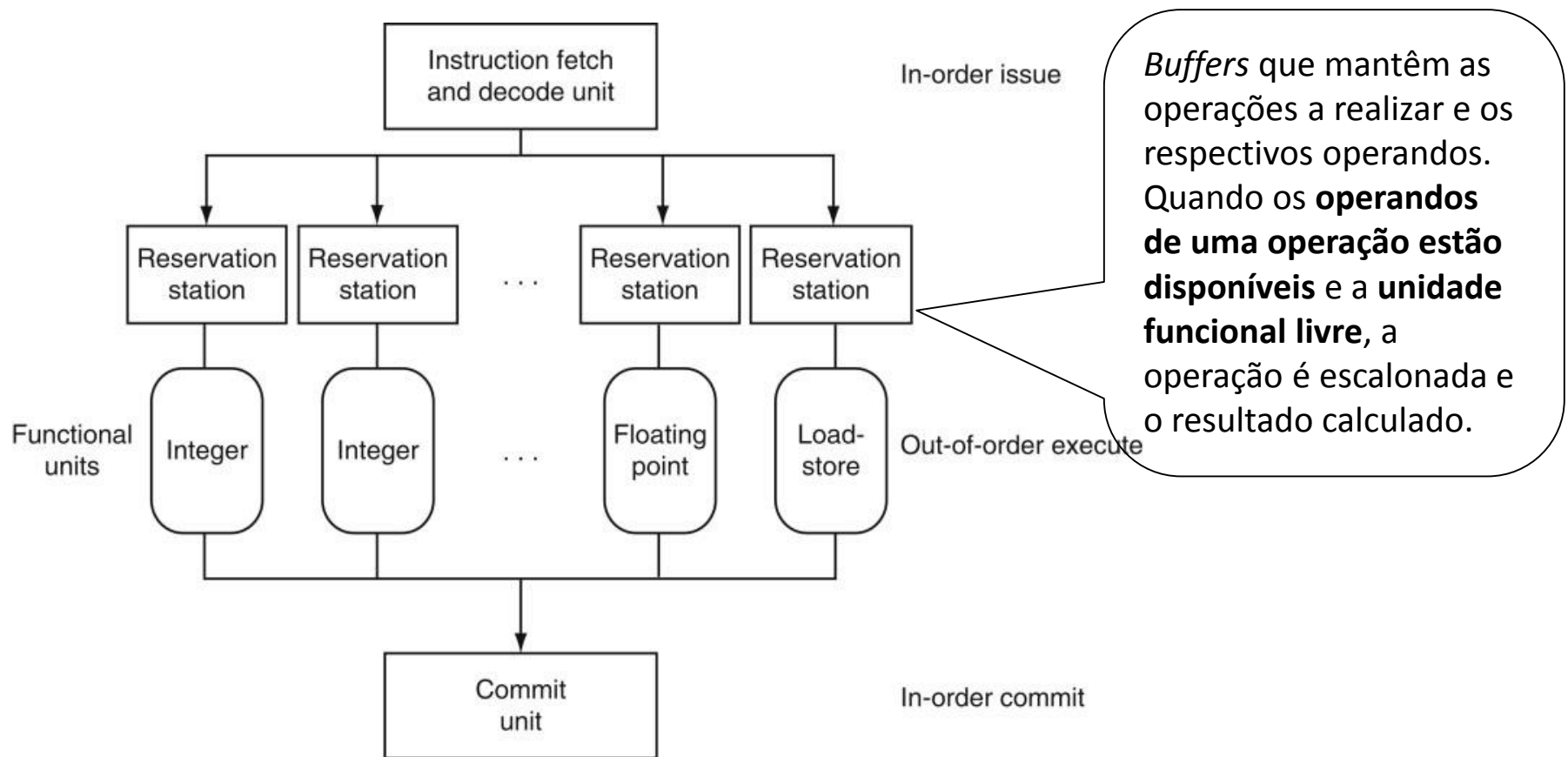


# SuperEscalaridade: organização



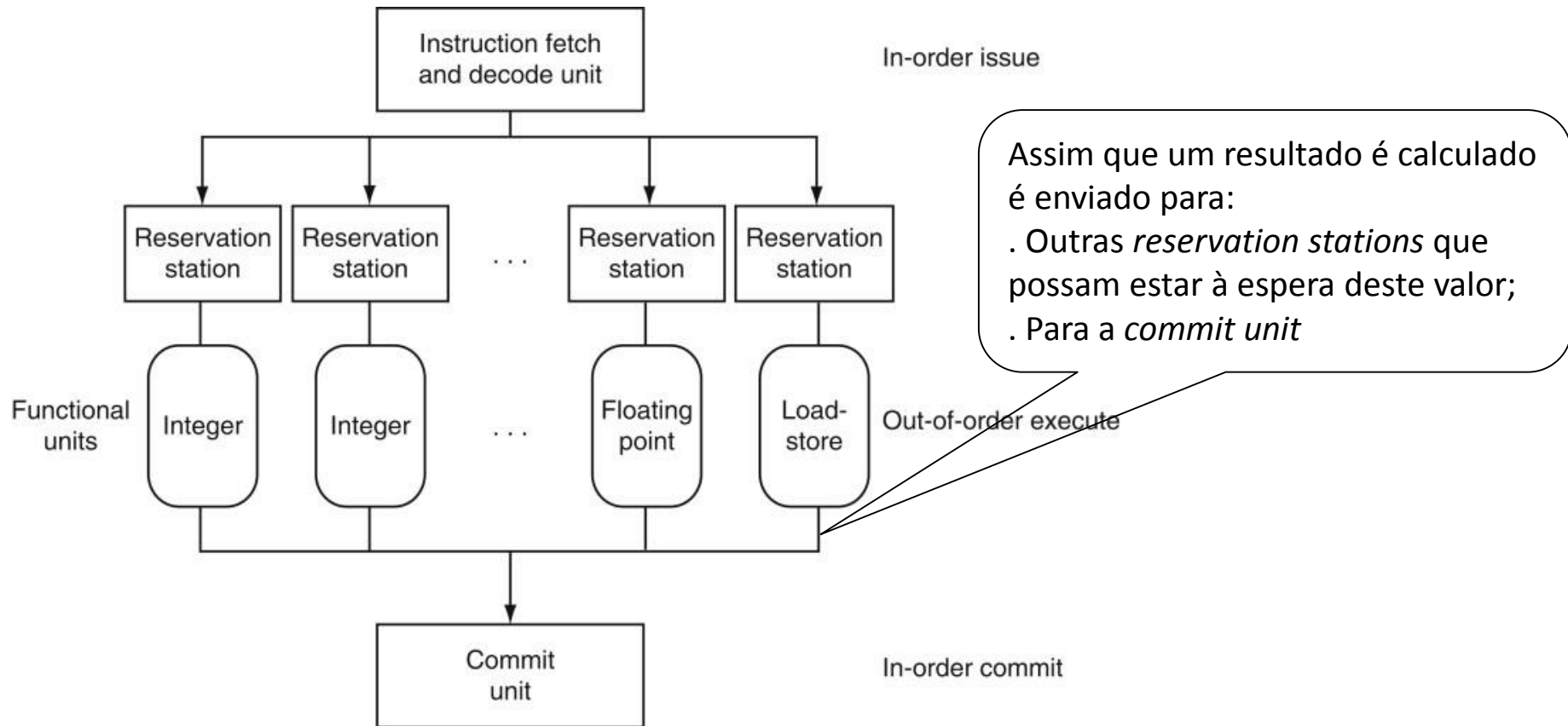
[ FIGURE 4.72 – Patterson & Hennessy; Computer Organization & Design – 5<sup>th</sup> Edition, Elsevier, 2013 --- As 3 unidades primárias para *dynamic pipeline scheduling*]

# SuperEscalaridade: organização



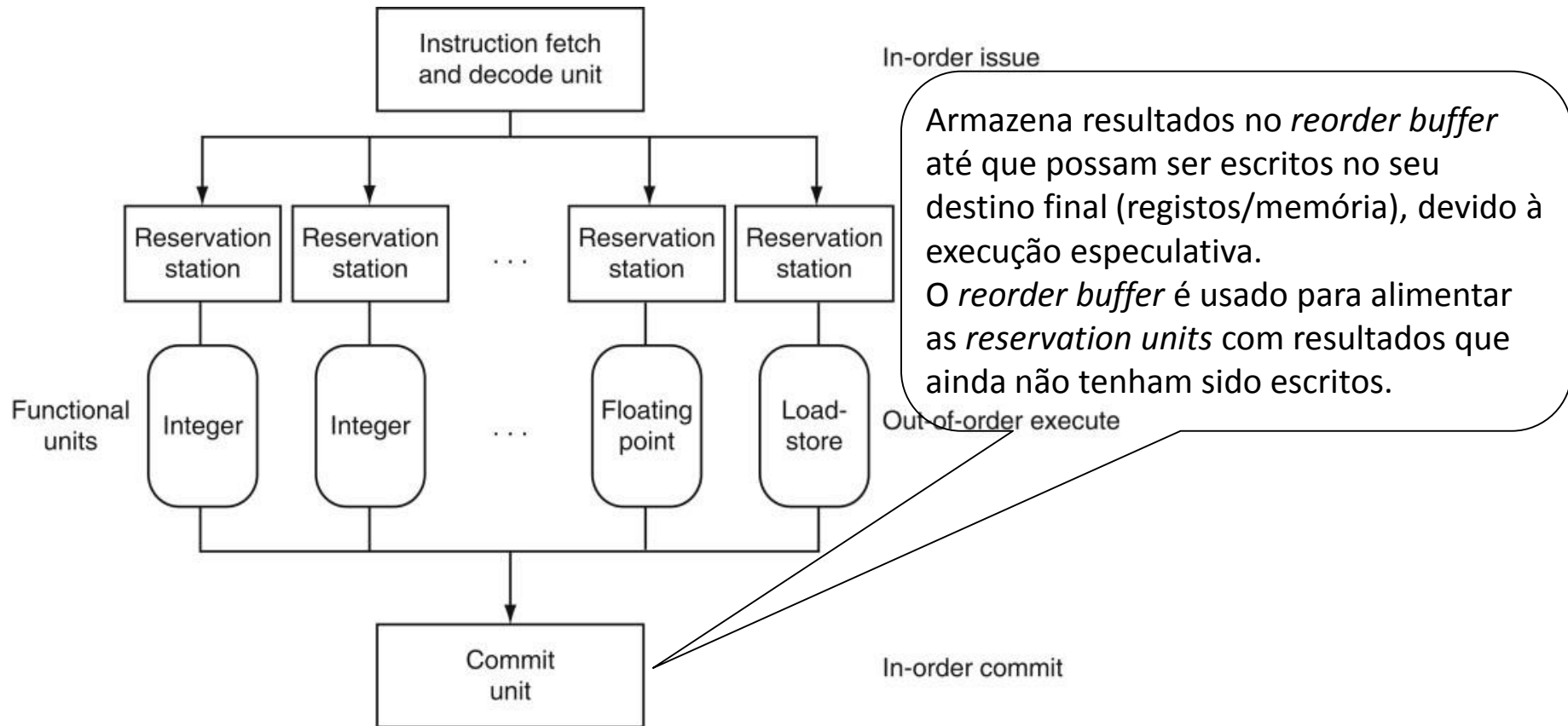
[ FIGURE 4.72 – Patterson & Hennessy; Computer Organization & Design – 5<sup>th</sup> Edition, Elsevier, 2013 --- As 3 unidades primárias para *dynamic pipeline scheduling*]

# SuperEscalaridade: organização



[ FIGURE 4.72 – Patterson & Hennessy; Computer Organization & Design – 5<sup>th</sup> Edition, Elsevier, 2013 --- As 3 unidades primárias para *dynamic pipeline scheduling*]

# SuperEscalaridade: organização



[ FIGURE 4.72 – Patterson & Hennessy; Computer Organization & Design – 5<sup>th</sup> Edition, Elsevier, 2013 --- As 3 unidades primárias para *dynamic pipeline scheduling*]

# ILP: limitações

- *Pipelining* e *multiple issue* são usados para explorar *Instruction Level Parallelism* (ILP)
- No entanto, na maior parte das aplicações é difícil lançar mais do que duas instruções por ciclo:
  1. Existem dependências de controlo e dados no código que limitam o paralelismo disponível;
  2. Penalizações devido ao acesso à memória limitam a capacidade de manter os múltiplos *pipelines* constantemente ocupados.

# Eficiência Energética

- O aumento do número de estágios dos *pipelines* juntamente com o controlo associado à execução especulativa resultam numa perda de eficiência energética (performance per joule).
- Processadores mais simples, embora exibam menor desempenho, apresentam melhores *ratios* performance per joule.

Microprocessor	Year	Clock Rate	Pipeline Stages	Issue Width	Out-of-Order/ Speculation	Cores/ Chip	Power	
Intel 486	1989	25 MHz	5	1	No	1	5	W
Intel Pentium	1993	66 MHz	5	2	No	1	10	W
Intel Pentium Pro	1997	200 MHz	10	3	Yes	1	29	W
Intel Pentium 4 Willamette	2001	2000 MHz	22	3	Yes	1	75	W
Intel Pentium 4 Prescott	2004	3600 MHz	31	3	Yes	1	103	W
Intel Core	2006	2930 MHz	14	4	Yes	2	75	W
Intel Core i5 Nehalem	2010	3300 MHz	14	4	Yes	1	87	W
Intel Core i5 Ivy Bridge	2012	3400 MHz	14	4	Yes	8	77	W

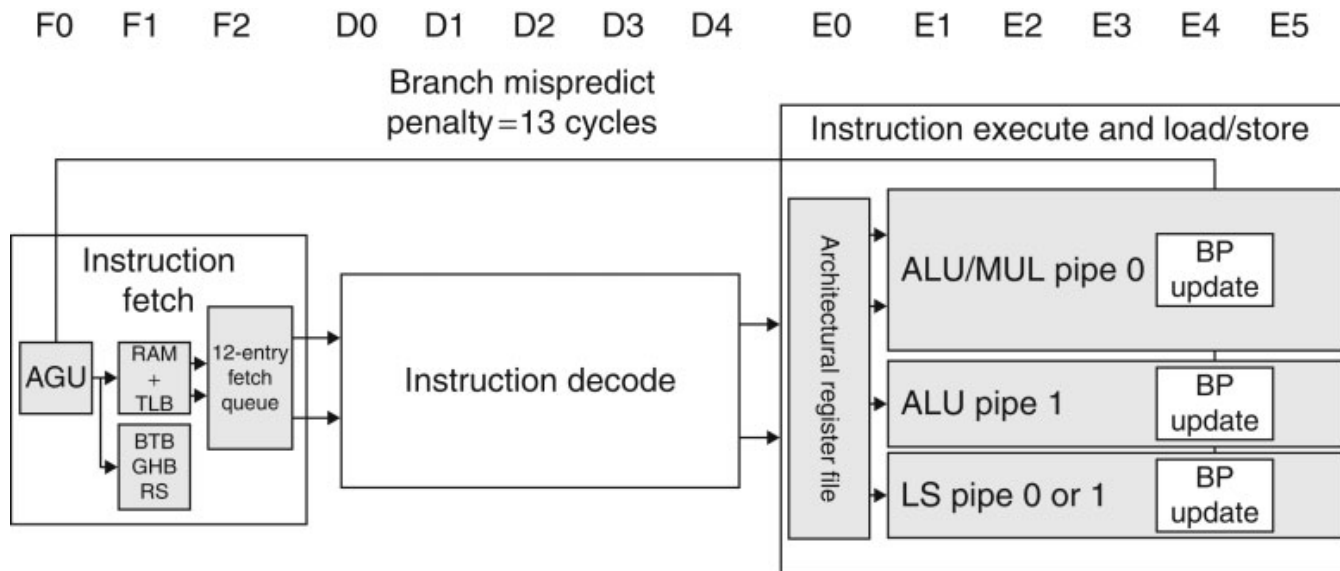
[ FIGURE 4.73 – Patterson & Hennessy; Computer Organization & Design – 5<sup>th</sup> Edition, Elsevier, 2013 ---The Pentium 4 pipelines do not include the commit stages. If included the pipelines would be deeper. ]

# ARM Cortex-A8 vs Intel Core i7

Processor	ARM A8	Intel Core i7 920
Market	Personal Mobile Device	Server, Cloud
Thermal design power	2 Watts	130 Watts
Clock rate	1 GHz	2.66 GHz
Cores/Chip	1	4
Floating point?	No	Yes
Multiple Issue?	Dynamic	Dynamic
Peak instructions/clock cycle	2	4
Pipeline Stages	14	14
Pipeline schedule	Static In-order	Dynamic Out-of-order with Speculation
Branch prediction	2-level	2-level
1st level caches / core	32 KiB I, 32 KiB D	32 KiB I, 32 KiB D
2nd level cache / core	128–1024 KiB	256 KiB
3rd level cache (shared)	–	2–8 MiB

[ FIGURE 4.74 – Patterson & Hennessy; Computer Organization & Design – 5<sup>th</sup> Edition, Elsevier, 2013 ]

# ARM Cortex A8



[ FIGURE 4.75 – Patterson & Hennessy; Computer Organization & Design – 5<sup>th</sup> Edition, Elsevier, 2013 --- The A8 pipeline]



# ARM Cortex A8 - Desempenho

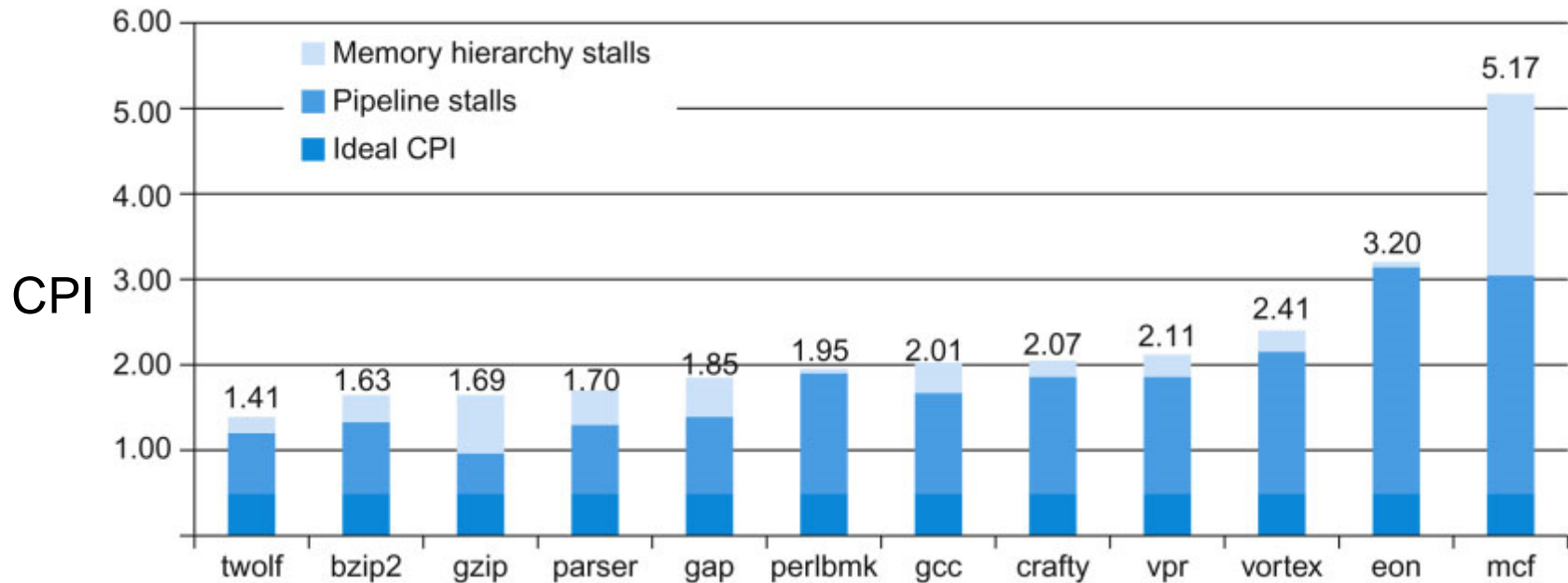
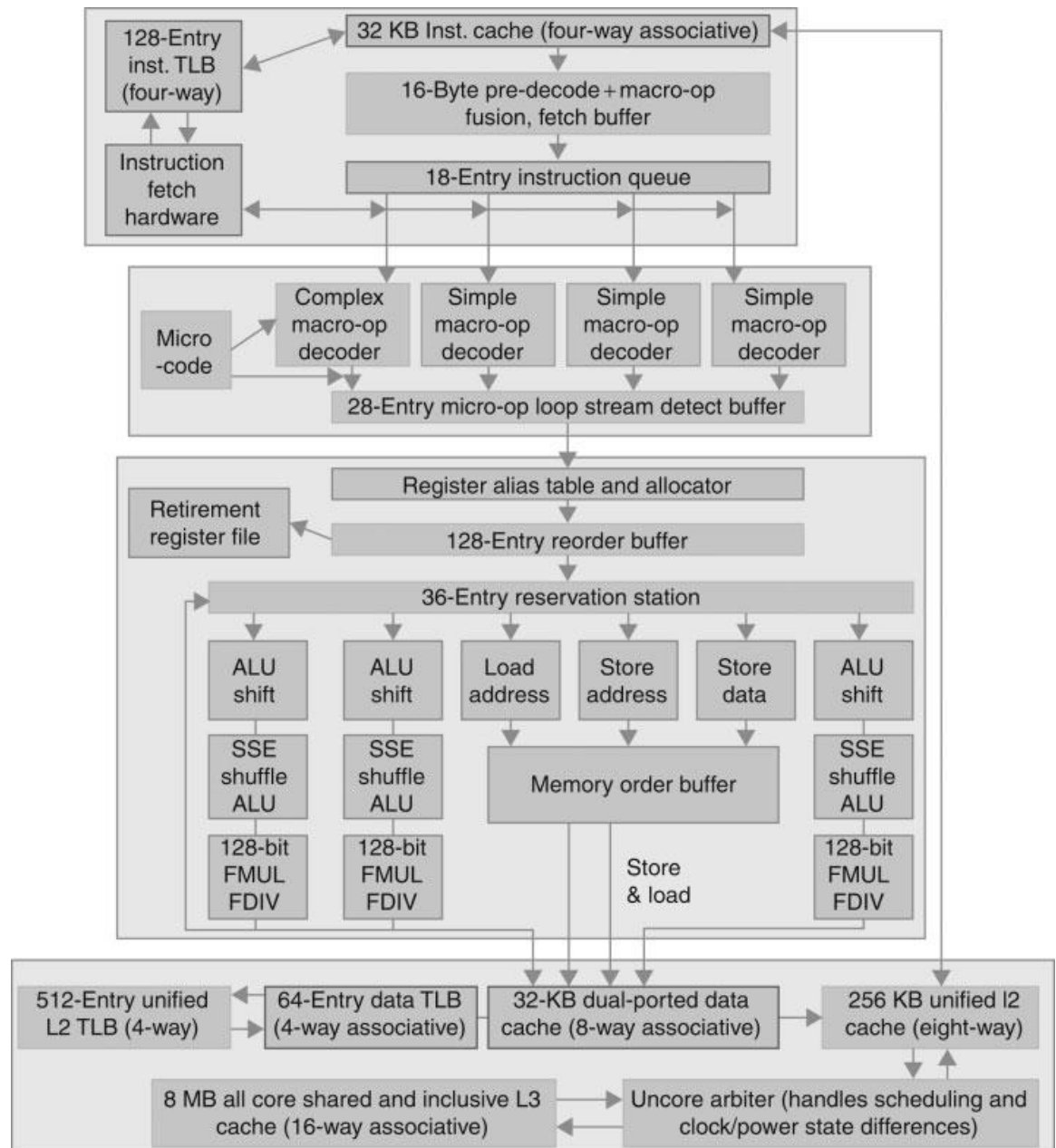


FIGURE 4.76 Patterson & Hennessy; Computer Organization & Design – 5<sup>th</sup> Edition, Elsevier, 2013  
CPI on ARM Cortex A8 for the Minnespec benchmarks, which are small versions of the SPEC2000 benchmarks.

# Intel Core i7 920

[ FIGURE 4.77 – Patterson & Hennessy;  
Computer Organization & Design  
5<sup>th</sup> Edition, Elsevier, 2013 ]



# Intel Core i7 920 - Desempenho

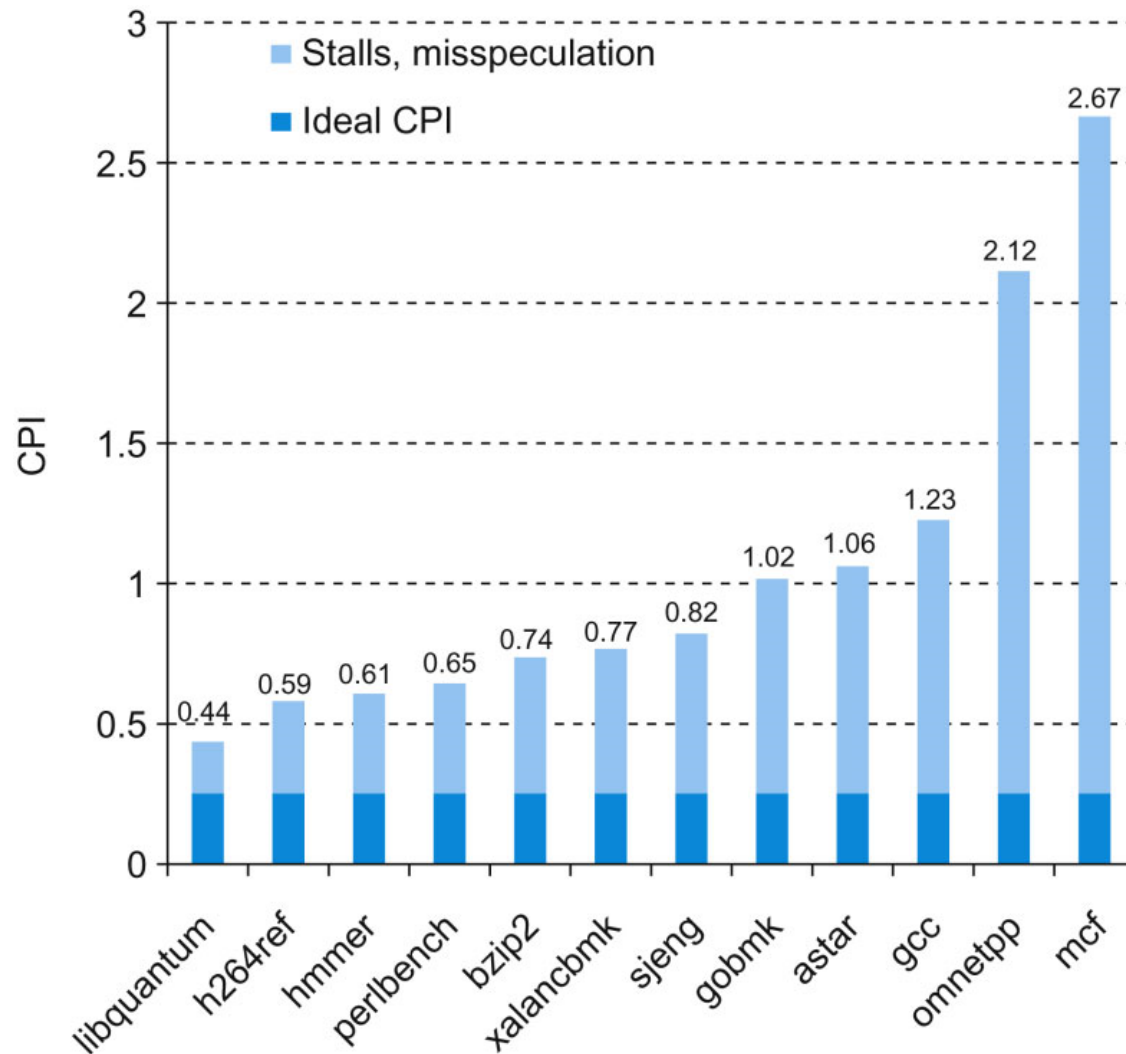


FIGURE 4.78 CPI of Intel Core i7 920 running SPEC2006 integer benchmarks.

# Intel Core i7 920 - Desempenho

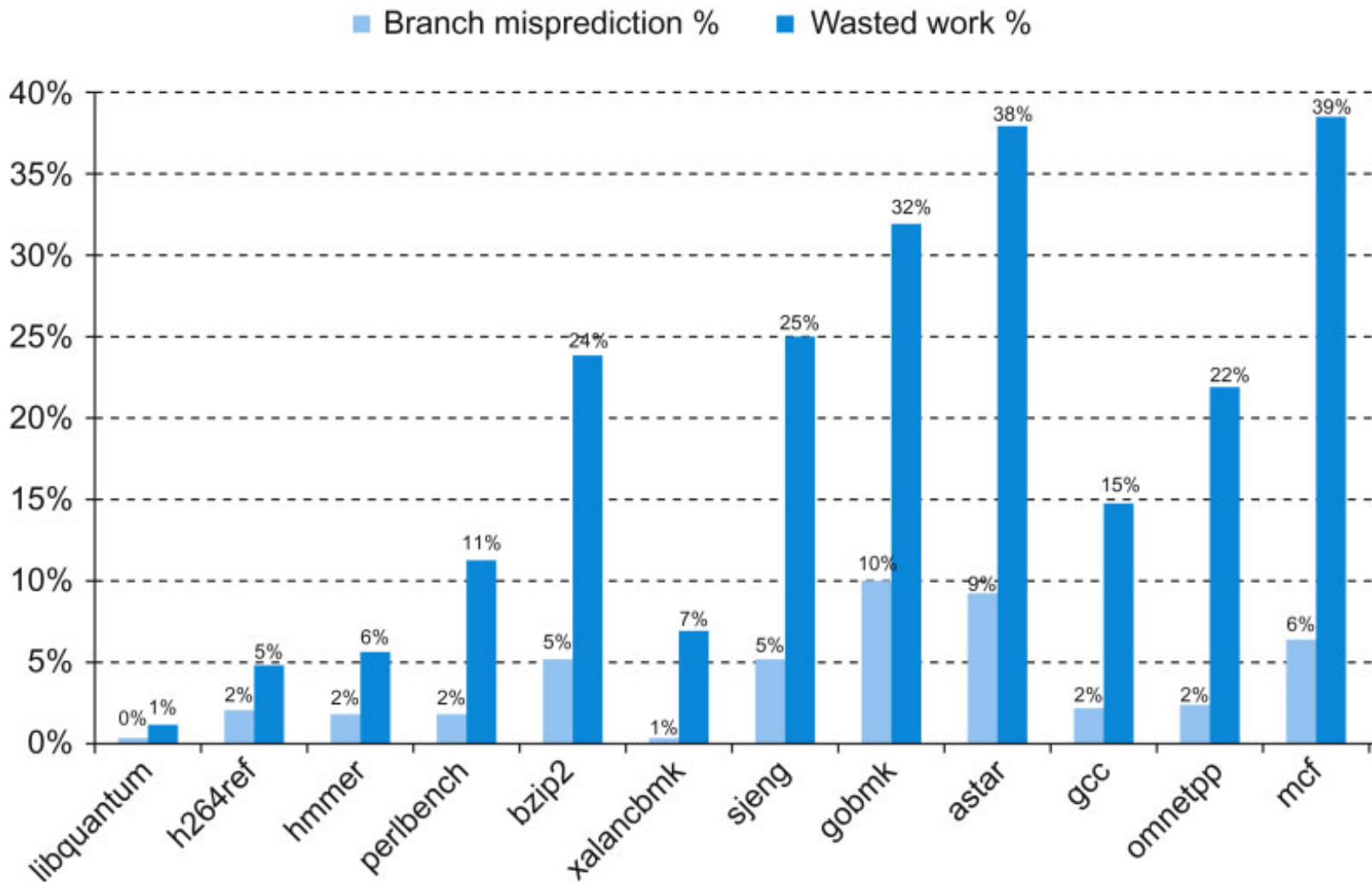


FIGURE 4.79 Percentage of branch mispredictions and wasted work due to unfruitful speculation of Intel Core i7 920 running SPEC2006 integer benchmarks.