Paralelismo ao nível das Instruções (ILP)

Arquitetura de Computadores

Lic. em Engenharia Informática

João Luís Sobral

Paralelismo ao nível das Instruções (ILP)

Conteúdos	**3.4 Processamento superescalar**
Resultados de Aprendizagem	R3.4 - Analisar e justificar o impacto de múltiplas unidades funcionais no desempenho da máquina

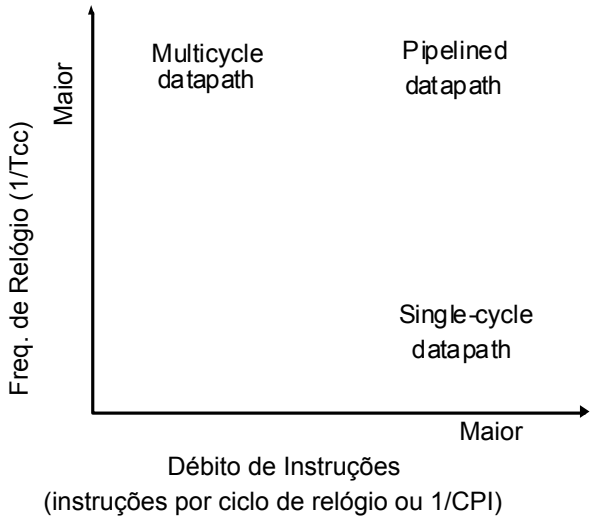
Variantes do *datapath*

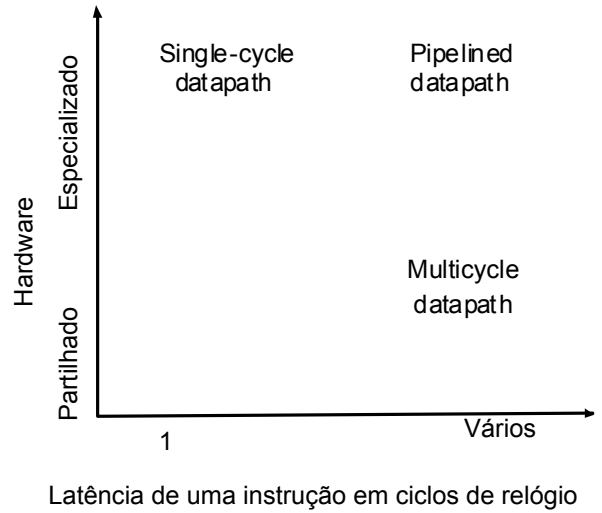
- *Single-cycle* (Y86-SEQ)- Todas as instruções são executadas num só ciclo
  - CPI = 1
  - Tcc = duração da instrução mais longa (pouco eficiente)
  - Latência de uma instrução = Tcc

- Multi-ciclo – Cada instrução demora vários ciclos
  - CPI = 1 … número máximo de ciclos (depende do mix)
  - Tcc = duração máxima de cada fase
  - Latência = Tcc x 1 .. *CPImax*

- Encadeada (Y86-PIPE) – Execução como numa linha de montagem
  - CPI = 1 + CPI*stalls*
  - Tcc = duração do estágio mais longo
  - Latência = Tcc x número de estágios

Variantes do *datapath*

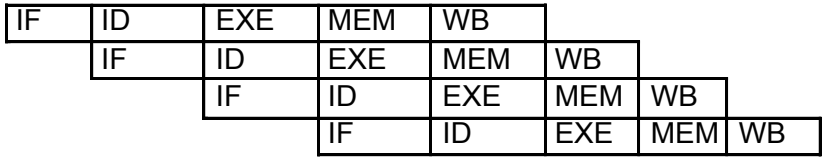




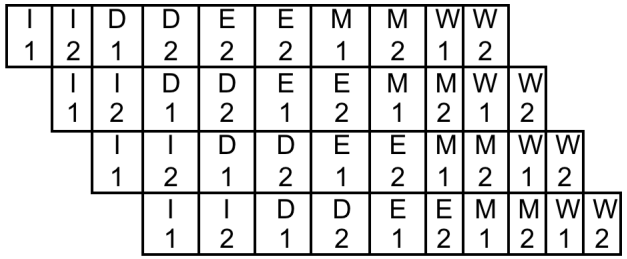
- O desempenho de uma dada arquitetura é determinado por vários factores
  - Texe = #I x CPI x Tcc
    - Decresce com 1/Tcc (frequência) e 1/CPI (instruções por ciclo de relógio - IPC)
  - A latência das instruções é um factor importante porque influencia o número de ciclos de *stall*, nomeadamente no caso dos saltos na execução encadeada

Alternativas de desenho

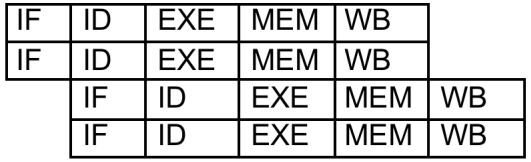
- Encadeamento – execução parcialmente sobreposta de instruções



- Super-encadeamento - aumentar o número de estágios para possibilitar um aumento da frequência interna de relógio

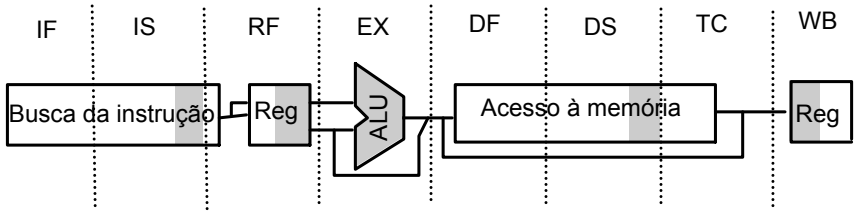


- Super-escalar – iniciar várias instruções por ciclo de relógio



Super-encadeamento

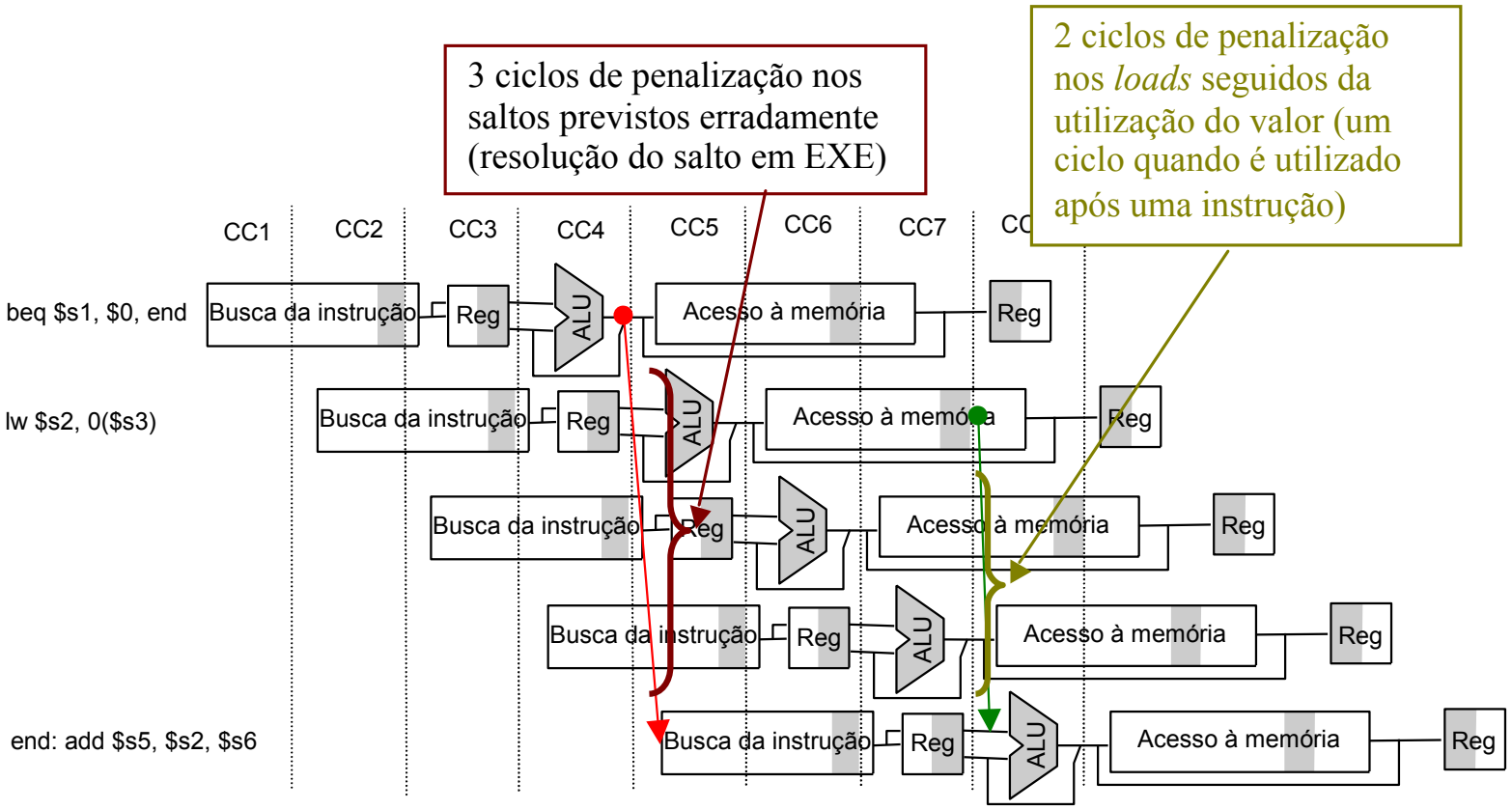
- Consiste em aumentar os estágios para assim conseguir aumentar a frequência
- Pode não diminuir a latência das instruções (pode mesmo aumentar) o que implica um aumento das penalizações devido a anomalias de dados e de controlo, se não forem introduzidos outros melhoramentos no DP
- Exemplo MIPS R4000 - 8 estágios, sendo os estágios adicionais utilizados para acessos à memória



IF – Primeira metade da busca da instrução (seleção do PC)
IS – Segunda metade da busca da instrução, completando o acesso
RF – descodificação e leitura do valor dos registos
EX – Execução, incluindo a resolução dos saltos
DF – Busca dos dados, primeira metade do acesso a *cache*
DS – segunda metade do acesso a *cache*
TC – *Tag check*, determinar se o acesso foi um *hit*
WB- Escrita dos resultados em registo para *load* e registo-registo

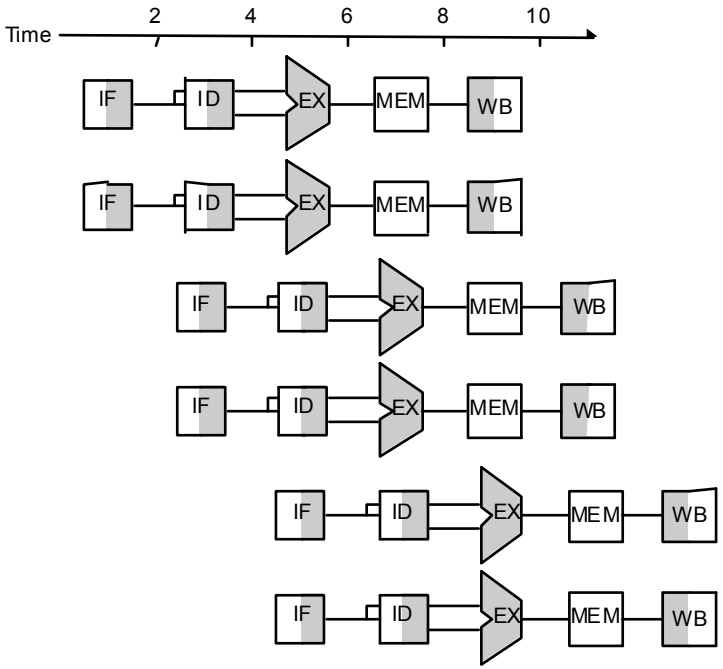
Super-encadeamento

- Exemplo MIPS R4000 – penalizações nos saltos e *loads*



Super-escalaridade

- Baseia-se na duplicação de unidades funcionais por forma a ser possível executar simultaneamente mais do que uma instrução em cada ciclo de relógio
- CPI < 1 (em situações ideais)
- Exemplo: grau 2



Super-escalaridade

- Implica a capacidade de efetuar a busca de duas instruções por ciclo (2 a 12 bytes no Y86), duas descodificações, etc
- Uma abordagem mais simples consiste em apenas duplicar algumas unidades funcionais, não suportando todas as combinações de instruções

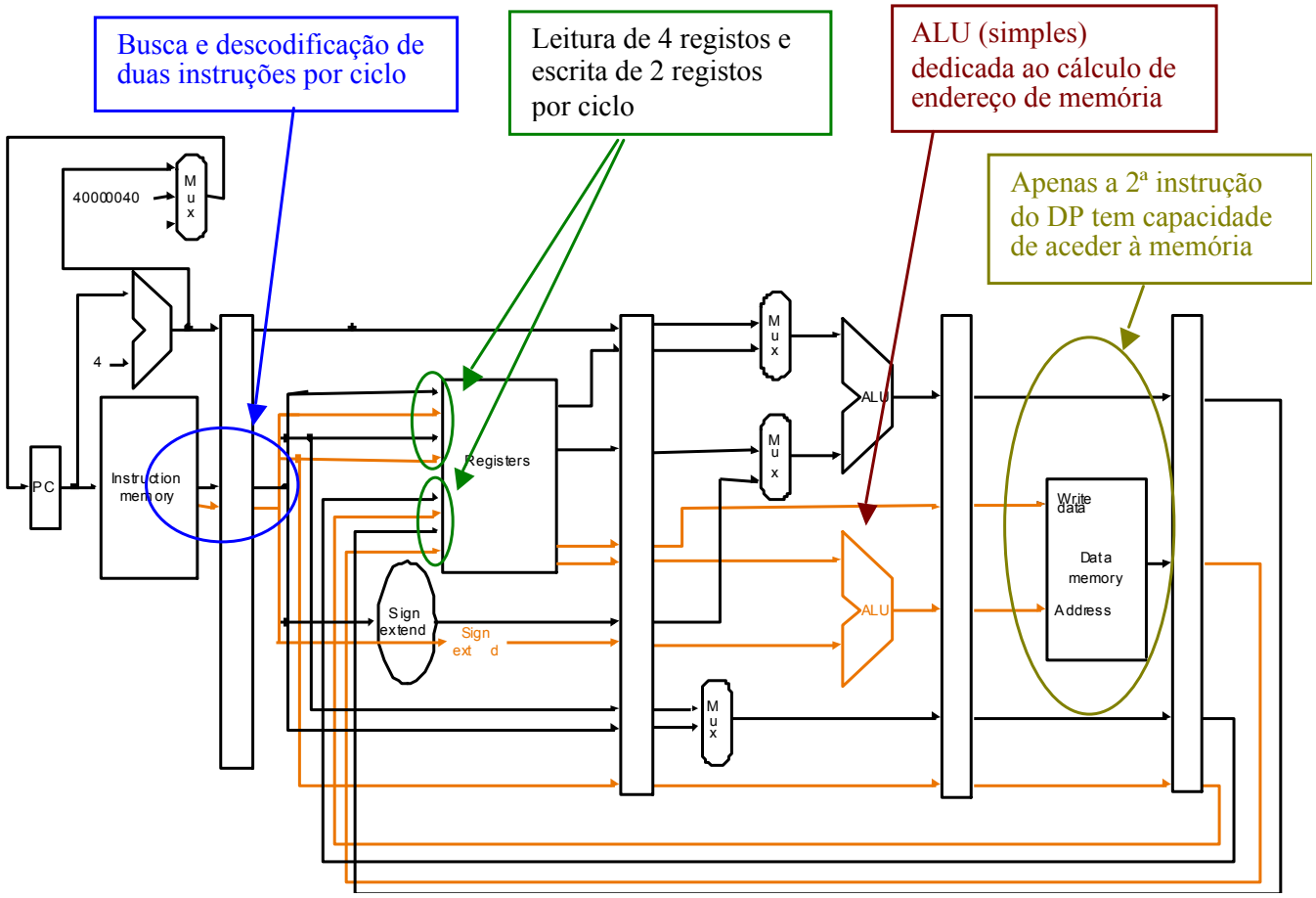
  - Implica anomalias estruturais: o hardware não suporta uma dada combinação de operações

  - Em Y86 a unidade de MEM apenas é utilizada por instruções de leitura e escrita na memória (mrmovel ou rmmovl), sendo, assim, mais simples conceber um processador super-escalar que suporta uma operação de salto/aritmética em simultâneo com um acesso à memória.

Tipo de instrução	Estágio						
ALU ou salto	IF	ID	EX	MEM	WB		
load ou *store*	IF	ID	EX	MEM	WB		
ALU ou salto		IF	ID	EX	MEM	WB	
load ou *store*		IF	ID	EX	MEM	WB	
ALU ou salto			IF	ID	EX	MEM	WB
load ou *store*			IF	ID	EX	MEM	WB

Super-escalaridade

- Exemplo de um *datapath* que suporta uma operação de salto/aritmética em simultâneo com um acesso à memória (arquitetura MIPS)



Super-escalaridade

- Desempenho de uma arquitetura superescalar (MIPS, grau 2)
  - Notação MIPS:
    - lw rd, v(rs) <=> R[rd] = MEM[R[rs]+V]
    - add rd, rs1, rs2 <=> rd = rs1 + rs2

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

	ALU || *branch*	*Load* || *store*	Ciclo
cic:		lw $t0, 0($s1)	1
	addi $s1, $s1, -4		2
	addu $t0, $t0, $s2		3
	bne $s1, $0, cic	sw $t0, 4($s1)	4

CPI ≅ número de ciclos / número de instruções = 4 / 5 = 0,8

- Desempenho desdobrando o ciclo 4x

	ALU || *branch*	*Load* || *store*	Ciclo
cic:	addi $s1, $s1, -16	lw $t0, 0($s1)	1
		lw $t1, 12($s1)	2
	addu $t0, $t0, $s2	lw $t2, 8($s1)	3
	addu $t1, $t1, $s2	lw $t3, 4($s1)	4
	addu $t2, $t2, $s2	sw $t0, 0($s1)	5
	addu $t3, $t3, $s2	sw $t1, 12($s1)	6
		sw $t2, 8($s1)	7
	bne $s1, $0, cic	sw $t3, 4($s1)	8

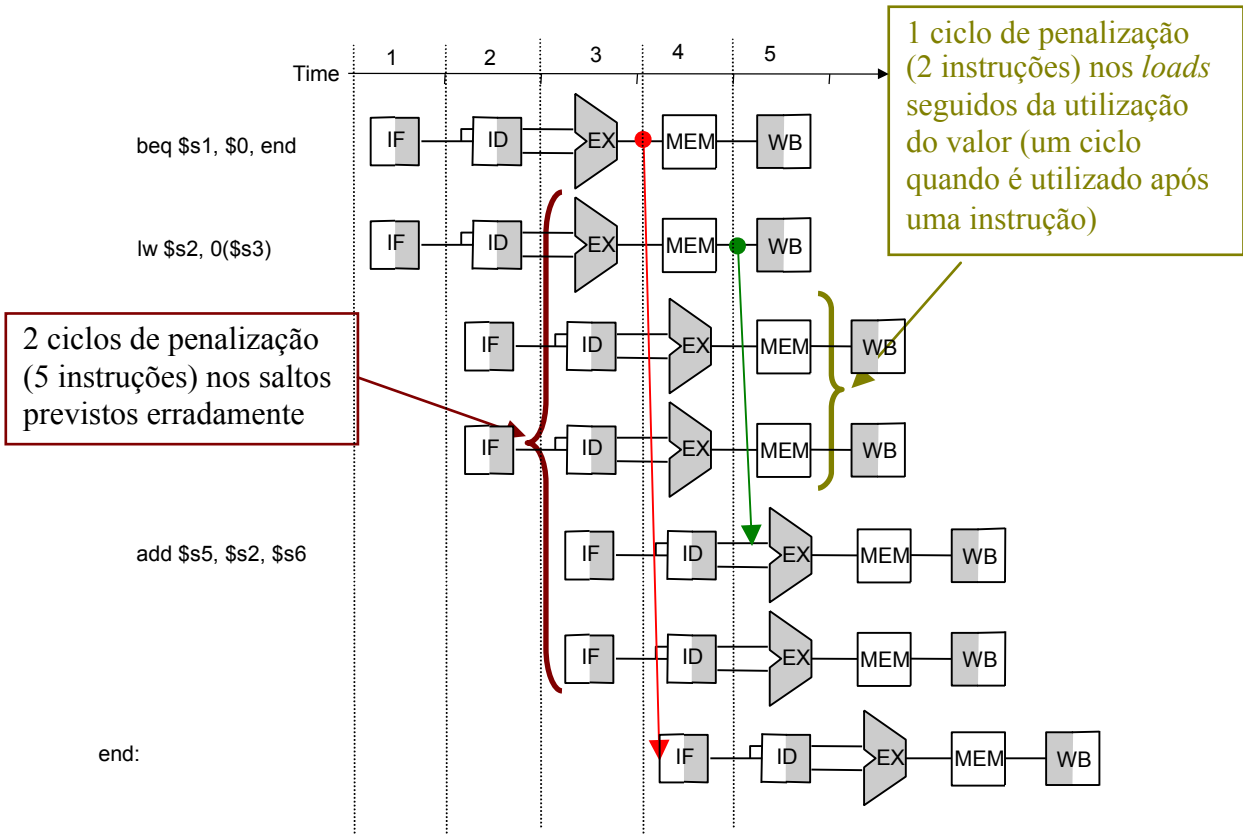
CPI ≅ 8/14 = 0,57

Utiliza mais registos:

$t1,$t2 e $t3

Super-escalaridade

- MIPS super-escalar – penalizações nos saltos e *loads*



Paralelismo ao nível das Instruções (ILP)

- Encadeamento e execução super-escalar são duas formas possíveis (e complementares) de exploração de paralelismo ao nível das instruções (ILP)
  - combinando das duas técnicas é possível executar centenas de instruções simultaneamente

- Qual o grau de paralelismo (ILP) explorável num dado programa?
  - O paralelismo explorável está limitado pelas dependências de controlo e dependências de dados do programa
    - Uma dependência pode gerar um ou vários ciclos de "stall"
      - » Depende da microarquitetura implementada
    - As dependências de dados impedem que as instruções sejam executadas em paralelo numa arquitetura super-escalar

- A possibilidade de executar centenas de instruções em paralelo implica a necessidade de efetuar o **escalonamento das instruções**: quais as instruções que podem executar em paralelo num dado ciclo de relógio?
  - **Hardware** – o processador inclui mecanismos para identificar as dependências e decidir as instruções que podem ser realizadas em paralelo
  - **Software** – o paralelismo é identificado durante o processo de compilação

Paralelismo ao nível das Instruções

Qual o desempenho de uma arquitetura?

CPI = CPI*ideal* + *stall* estruturais + *stall* dependências de dados + *stall* dependências de controlo

- As arquiteturas super-escalares conseguem obter CPI*ideal* < 1
  - Por vezes utiliza-se a métrica IPC – Instruções Por Ciclo (IPC = 1/ CPI)

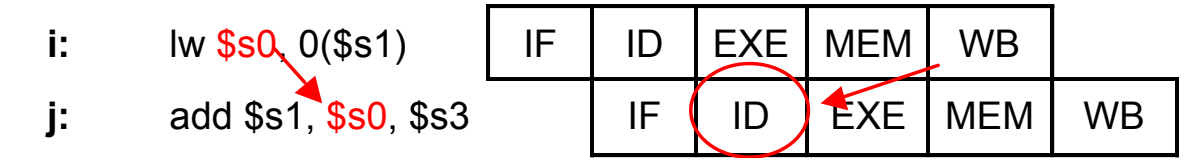
- Estratégias para lidar com as dependências de dados:
  - Encaminhamento; escalonamento/renomeação; especulação

- Estratégias para lidar com as dependências de controlo:
  - Escalonamento de saltos; previsão; especulação; desdobramento de ciclos

Paralelismo ao nível das Instruções

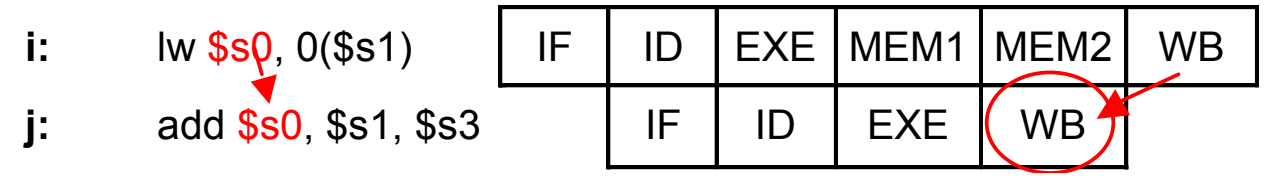
Análise detalhada dos tipos de dependências de dados:

- **RAW** (*Read After Write*) – uma instrução *j* usa um operando produzido (escrito) por uma instrução anterior *i*



  - Normalmente resolvidas com encaminhamento dos dados; pode originar ciclos de "stall"

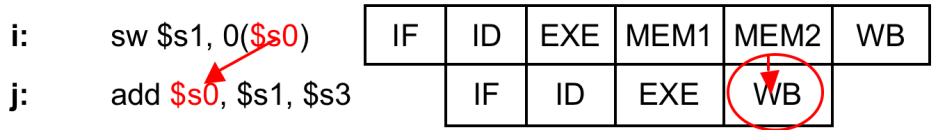
- **WAW** (*Write After Write*) – uma instrução *j* escreve um operando escrito também por uma instrução anterior *i*
  - surge em "pipeline" que escrevem valores em mais do que um estágio ou quando as instruções não são completadas na ordem do programa



Paralelismo ao nível das Instruções

Análise detalhada dos tipos de dependências de dados (cont.):

- **WAR** (*Write After Read*) – uma instrução *j* escreve um valor num operando que é usado por uma instrução anterior *i*
  - Originada quando a leitura de registos pode ocorrer após o WB da instrução seguinte, nomeadamente, quando as instruções não são completadas pela ordem do programa



- Estas dependências também podem ocorrer através de posições de memória
  - Exemplo:
    - sw %s1, 0($s0) - MEM[R[s0]+0] = R[s1]
    - lw %s2, 0($s0) - R[s2] = MEM[R[s0]+0]
  - A detecção da dependência é, neste casos, mais complexa:
    - 100 ($s0) = 20 ($s2) ?
    - 20 ($s2) = 20 ($s2) em iterações diferentes de um ciclo?

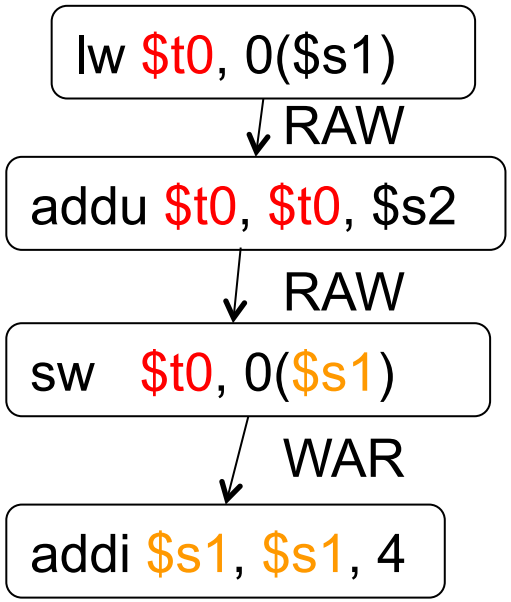
Paralelismo ao nível das Instruções

Nos processadores atuais as dependências de dados limitam a exploração de paralelismo ao nível da instrução (ILP)

Um **grafo de dependências de dados** permite representar/ visualizar essas dependências

- Importante para efetuar o escalonamento de instruções
- Exemplo (notação MIPS: addu rdestino, rsrc1, rsrc2, etc):

```
cic:    lw      $t0, 0($s1)
        addu    $t0, $t0, $s2
        sw      $t0, 0($s1)
        addi    $s1, $s1, 4
```



Paralelismo ao nível das Instruções

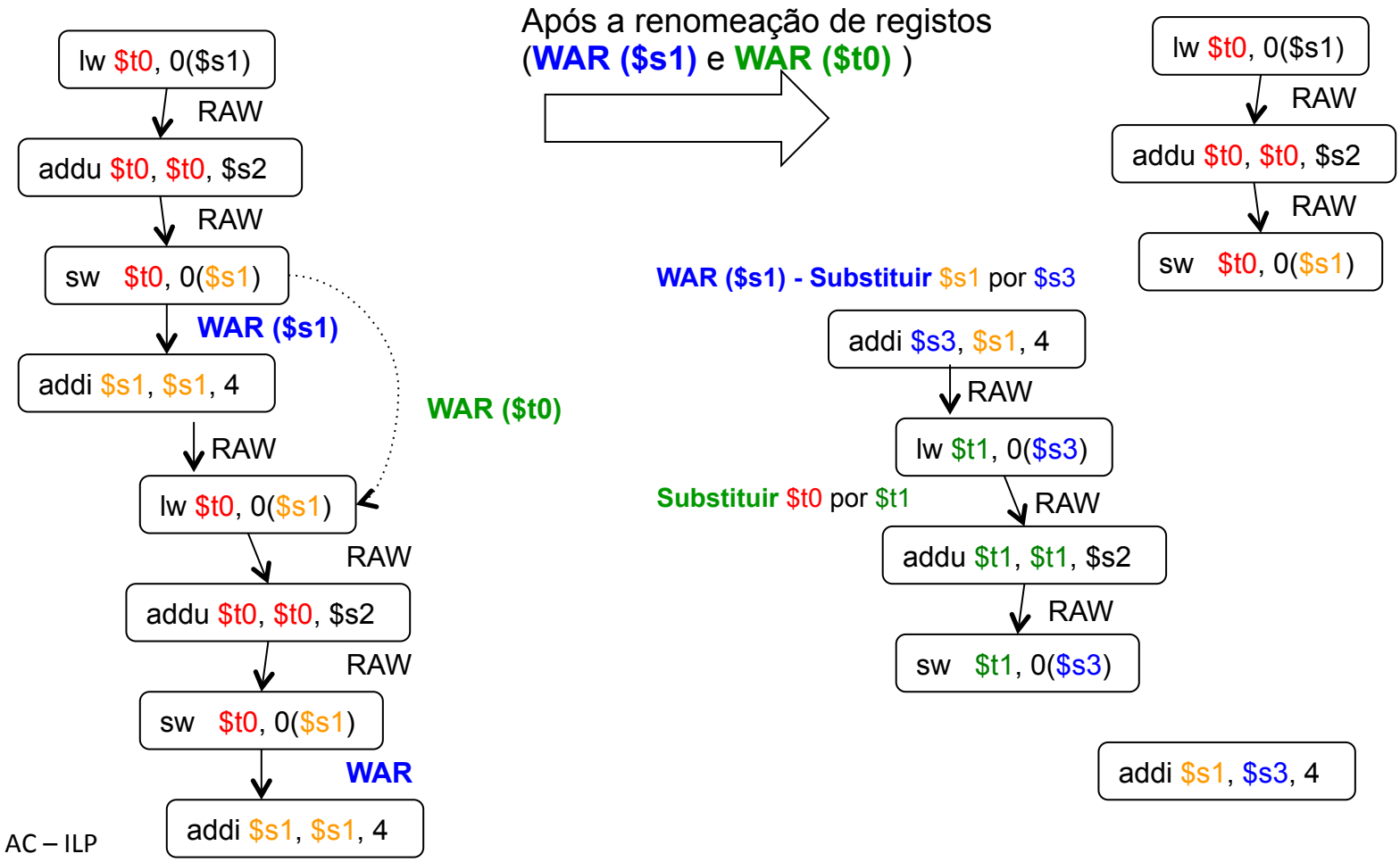
- A **renomeação de registos** permite remover as dependências WAR e WAW
  - São também designadas como “name dependencies” / “false dependencies”
  - Utiliza registos adicionais para remover as dependências
    - os registos podem não ser visíveis ao programador
  - Pode ser efectuada pelo compilador ou pelo processador
- Exemplo:

```
cic:  lw    $t0, 0($s1)        cic:  lw    $t0, 0($s1)
      addu  $t0, $t0, $s2              addu  $t0, $t0, $s2
      sw    $t0, 0($s1)              sw    $t0, 0($s1)
      addi  $s1, $s1, 4              addi  $s3, $s1, 4
      lw    $t0, 0($s1)              lw    $t1, 0($s3)
      addu  $t0, $t0, $s2            addu  $t1, $t1, $s2
      sw    $t0, 0($s1)              sw    $t1, 0($s3)
      bne   $s1,$0, cic              .... (addi $s1,$s3,0)
                                     bne   $s1,$0, cic
```

Paralelismo ao nível das Instruções

Renomeação de registos:

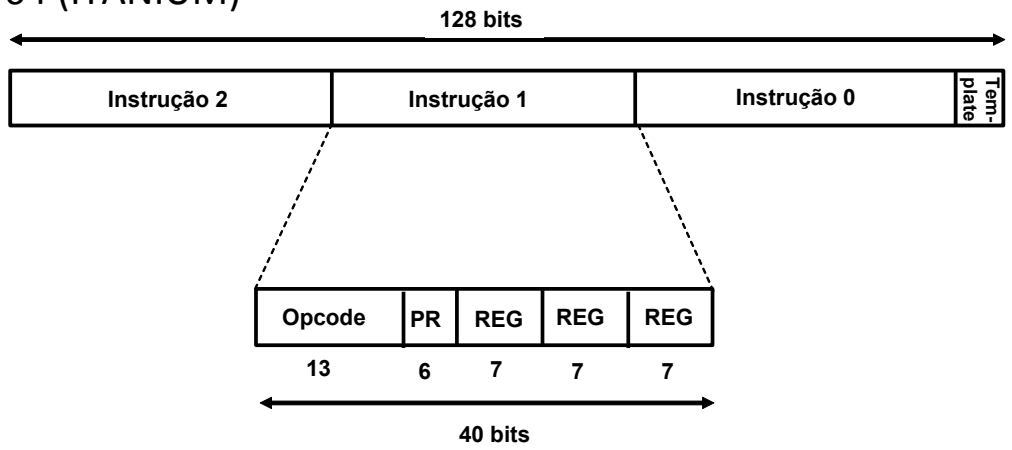
– Exemplo: grafo de dependências para duas iterações do ciclo



Paralelismo ao nível das Instruções (ILP)

Exemplo de escalonamento estático: arquiteturas VLIW (Very Long Instruction Word)

- O escalonamento de instruções incrementa de forma considerável a complexidade do hardware.
- VLIW efetua um escalonamento estático, sendo o compilador responsável por indicar as instruções que podem ser realizadas em paralelo.
- O formato de instrução indica as operações que são realizadas em paralelo por cada unidade funcional.
- Exemplo IA-64 (ITANIUM)



Paralelismo ao nível das Instruções (ILP)

Escalonamento estático: arquiteturas VLIW (cont.)

- Limitações de VLIW
  - O código gerado tende a ser de maior dimensão, porque é necessário inserir *nop* nos campos da instrução não preenchidos.
  - Compatibilidade de código entre gerações do mesmo processador
    - tende a expor a arquitetura interna do processador
  - É mais penalizado com *stalls* que o escalonamento dinâmico

- EPIC – IA-64 / Itanium
  - 64 registos de inteiros + 64 registos FP, ambos com 64 bits
  - 3 instruções em 128 bits (LIW?)
    - menos bits que VLIW clássico, produzindo código mais compacto
    - possibilidade de ligação entre os vários grupos de instruções
  - Verificação de dependências em HW => compatibilidade de código

Paralelismo ao nível das Instruções (ILP)

Escalonamento dinâmico de instruções

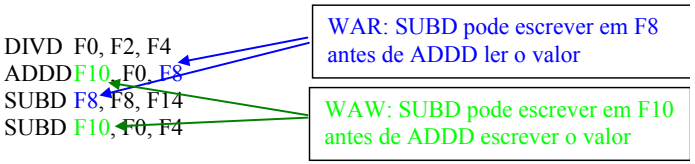
- O processador reorganiza o ordem das instruções para diminuir os *stalls*
  - instruções podem ser iniciadas e/ou completadas fora de ordem

- Diminui a complexidade dos compiladores e a dependência do código gerado de uma arquitetura específica relativamente ao escalonamento estático

- Aumenta fortemente a complexidade do hardware

- Elimina *stalls* devidos a dependências de dados através da execução fora de ordem de instruções:

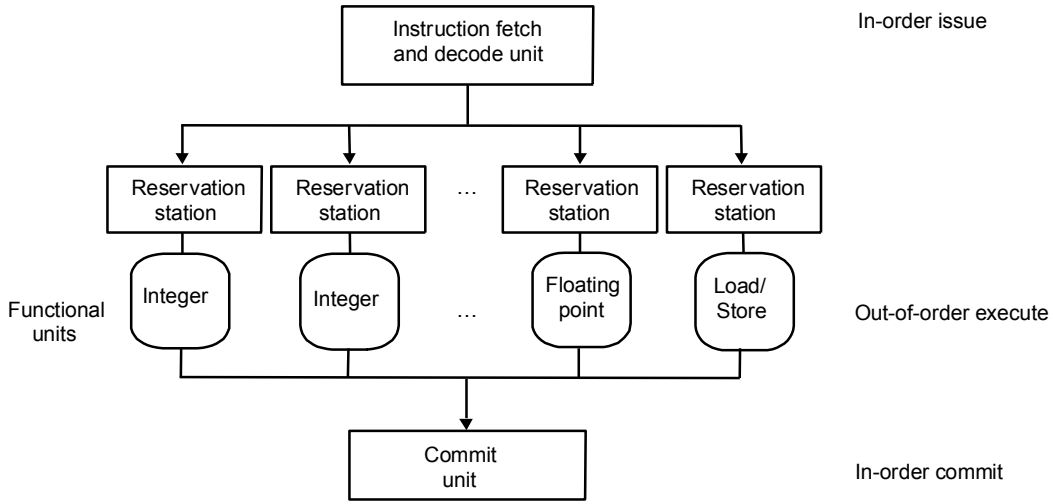
- A execução fora de ordem pode originar anomalias WAR e WAW (em MIPS)



Paralelismo ao nível das Instruções (ILP)

Estrutura de uma arquitetura atual com escalonamento dinâmico

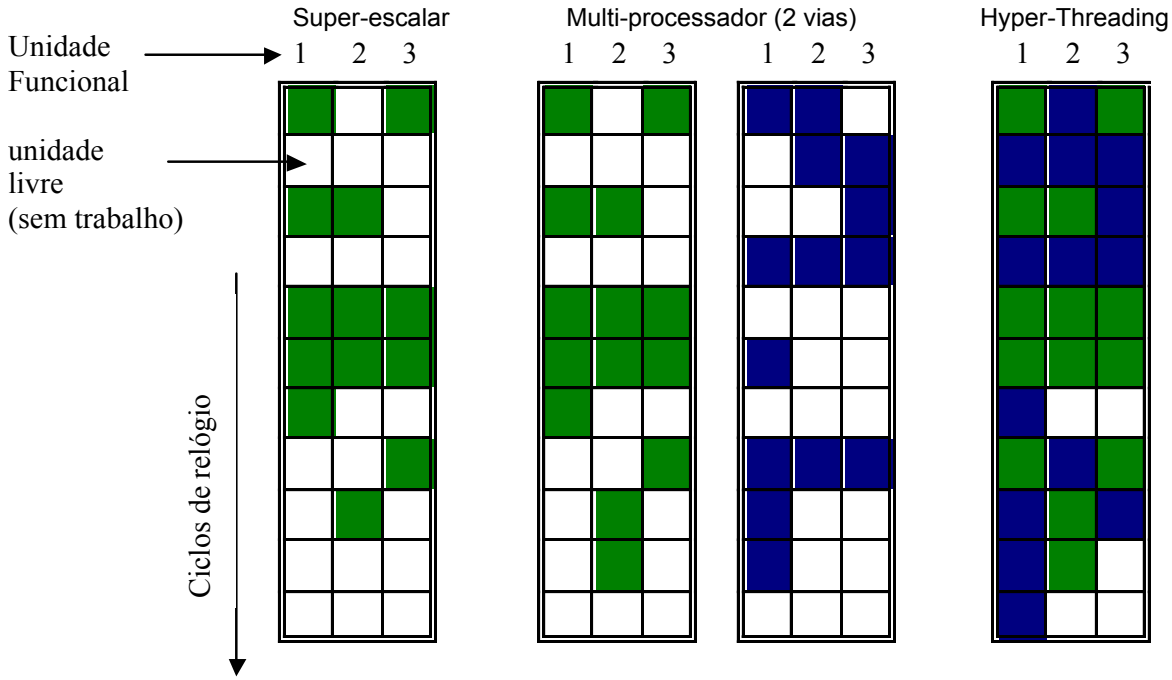
- A *pipeline* é dividida em três grupos de unidades:
  1. busca de instruções / descodificação / iniciação (*issue*)
  2. unidades de execução (fora de ordem)
  3. unidade de conclusão (*commit*)
- Busca e descodificação
  - descodifica as instruções e envia-as para a unidade de execução correspondente; faz a renomeação dos registos
- Execução das instruções
  - A instrução é executada quando as unidade funcional está livre e os operandos estão disponíveis
- Conclusão das instruções
  - Atualiza o estado visível (e.g. registos) quando é seguro tornar os resultados da instrução visíveis (ex. especulação na previsão de saltos)



Paralelismo ao nível das Instruções (ILP)

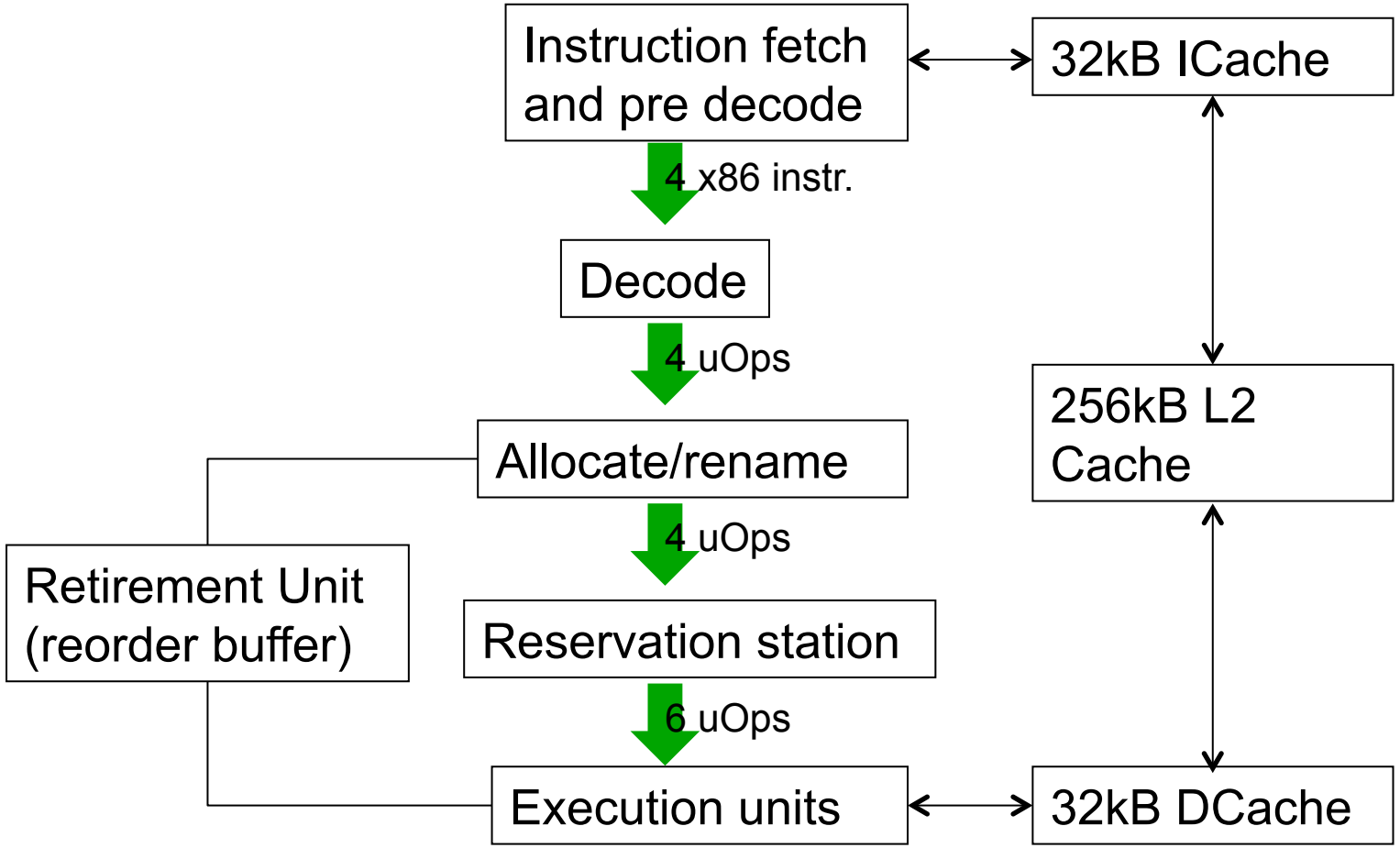
Hyper-threading

- Permite a eliminação de *stalls* intercalado instruções de várias linhas de execução



Paralelismo ao nível das Instruções (ILP)

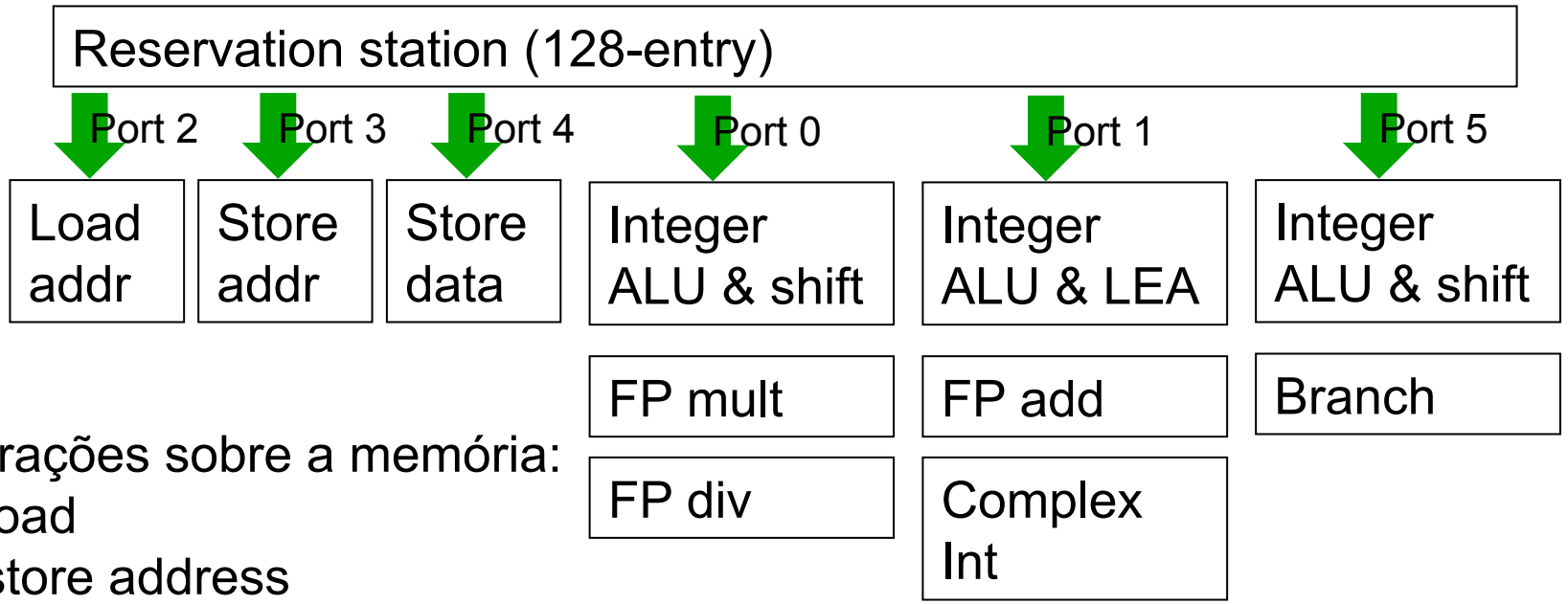
Arquitetura i7 (um núcleo)



Paralelismo ao nível das Instruções (ILP)

Arquitetura i7

Detalhe das unidades de execução



3 operações sobre a memória:

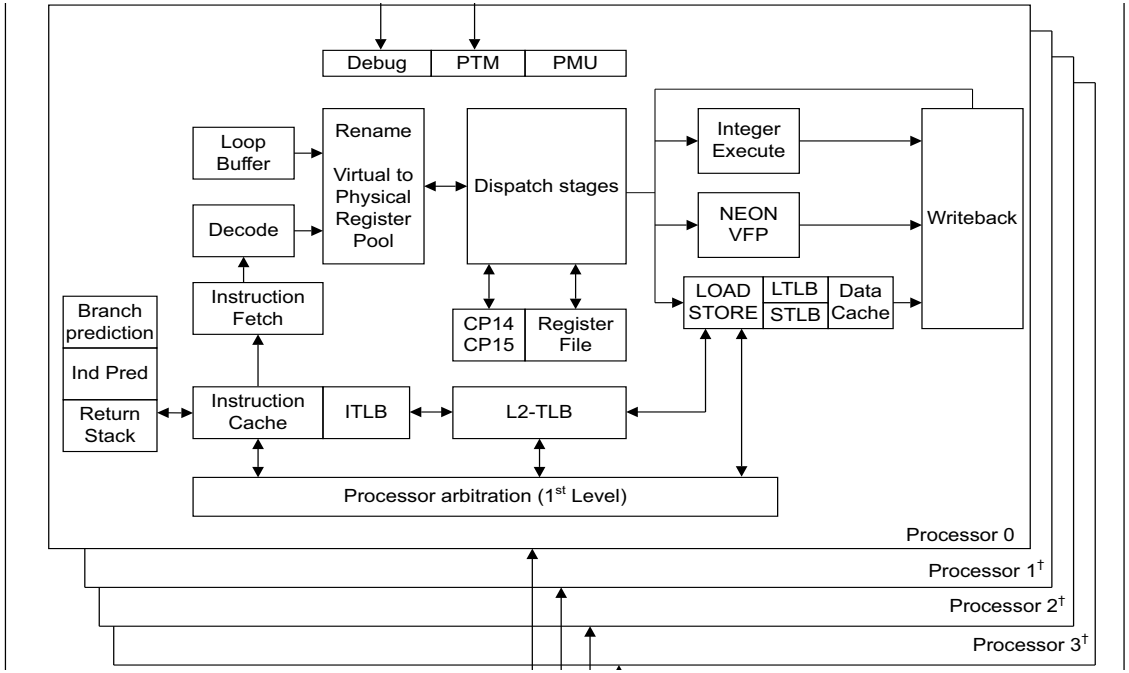
- 1 load
- 1 store address
- 1 store data

3 operações “computacionais”

Apple A6X (Ipad)

- SoC baseada no ISA da ARM

- ARM Cortex A15



Paralelismo ao nível das Instruções (ILP)

CPU	80286	80386	80486	Pentium	Pentium Pro	Pentium 4	Core i7
Ano	1982	1985	1989	1993	1997	2001	2010
Tipo	16 bits	32 bits	pipelined 5-stage Cache L1	Super-escalar 2-way	OOO 3-way	OOO super-pipelined Cache L2	OOO 4-way Cache L3
Freq.	12,5 MHz	16 MHz	25 MHz	66 MHz	200 MHz	1,5 GHz	3,3 GHz
Core/ chip	1	1	1	1	1	1	4
MIPS	2	6	25	132	600	4500	50000