Aula 11

Organização de Computadores

Processadores Paralelos (VLIW x Superscalar)

Profa. Débora Matos

Arquiteturas paralelas

- É importante notar que nem todas as aplicações podem se beneficiar do paralelismo, já que:
  - paralelismo multiprocessado adiciona custos (sincronização de processos, dentre outros)

- Se uma aplicação **não é sensível a solução paralela, não vale a pena**, em termos de custo, **portá-la para uma arquitetura paralela multiprocessada**.

- Embora o paralelismo possa resultar em significativa aceleração, essa aceleração nunca será perfeita.
  Porque??

Tendo-se *n* processadores executando em paralelo, isso não significa (e nunca será) que a aceleração será de *n* vezes.

Arquiteturas paralelas

- A aceleração perfeita não é possível, pois se 2 componentes de processamento funcionam em velocidade diferentes, a velocidade mais baixa irá dominar;

- Independente de quanto uma aplicação seja serializada, sempre existirá uma pequena parte do trabalho feita por um processador que precisa ser feita serialmente.

- Nesses casos, a inclusão de mais processadores não mudará a aceleração da aplicação, já que nada poderá realizar além de aguardar até que o processamento serial se complete;

- A premissa é que cada algoritmo tem uma parte sequencial que limita a aceleração . Quanto maior for o processamento sequencial, menor será a vantagem de adotar uma arquitetura paralela multiprocessada (em custo).

Arquiteturas paralelas

Que técnicas um projetista de HW pode utilizar para melhorar o desempenho?

Paralelismo em nível de instrução

- A técnica de pipeline explora o paralelismo entre as instruções chamado de **paralelismo em nível de instrução (ILP – Instruction Level Parallelism).**
- **Existem 2 métodos principais:**
  1. Aumenta a profundidade do pipeline para sobrepor mais instruções;
     - Ciclo de clock pode ser reduzido, aumentando a frequência de operação.
  2. Despacho múltiplo: Replica os componentes internos do computador de modo que várias instruções possam ser iniciadas em cada estágio do pipeline.

Paralelismo em nível de instrução

- No entanto, tanto para *pipelines* profundos como para despachos múltiplos, ocorrem ainda mais problemas de *hazards* e restrições quanto ao tipo de instruções que podem ser executadas simultaneamente;
- Funções do pipeline de despacho múltiplo:
  1. Empacotar as instruções em slots de despacho. Esse procedimento é tratado parcialmente pelo compilador;
  2. Lidar com hazard de dados e de controle. Em processadores de despacho estático, quase todas as consequências dos hazards de dados e controle são tratadas estaticamente pelo compilador.

Paralelismo em nível de instrução

- Como o processador de despacho múltiplo estático normalmente restringe o mix de instruções que podem ser iniciadas em um determinado ciclo de clock, é útil pensar no pacote de despacho como uma única instrução;

- Essa estratégia definiu uma outra técnica: VLIW (Very Long Instruction Word – palavra de instrução muito longa).

- As responsabilidades do compilador VLIW podem incluir previsão estática de desvios e escalonamento de código para reduzir ou impedir hazards.

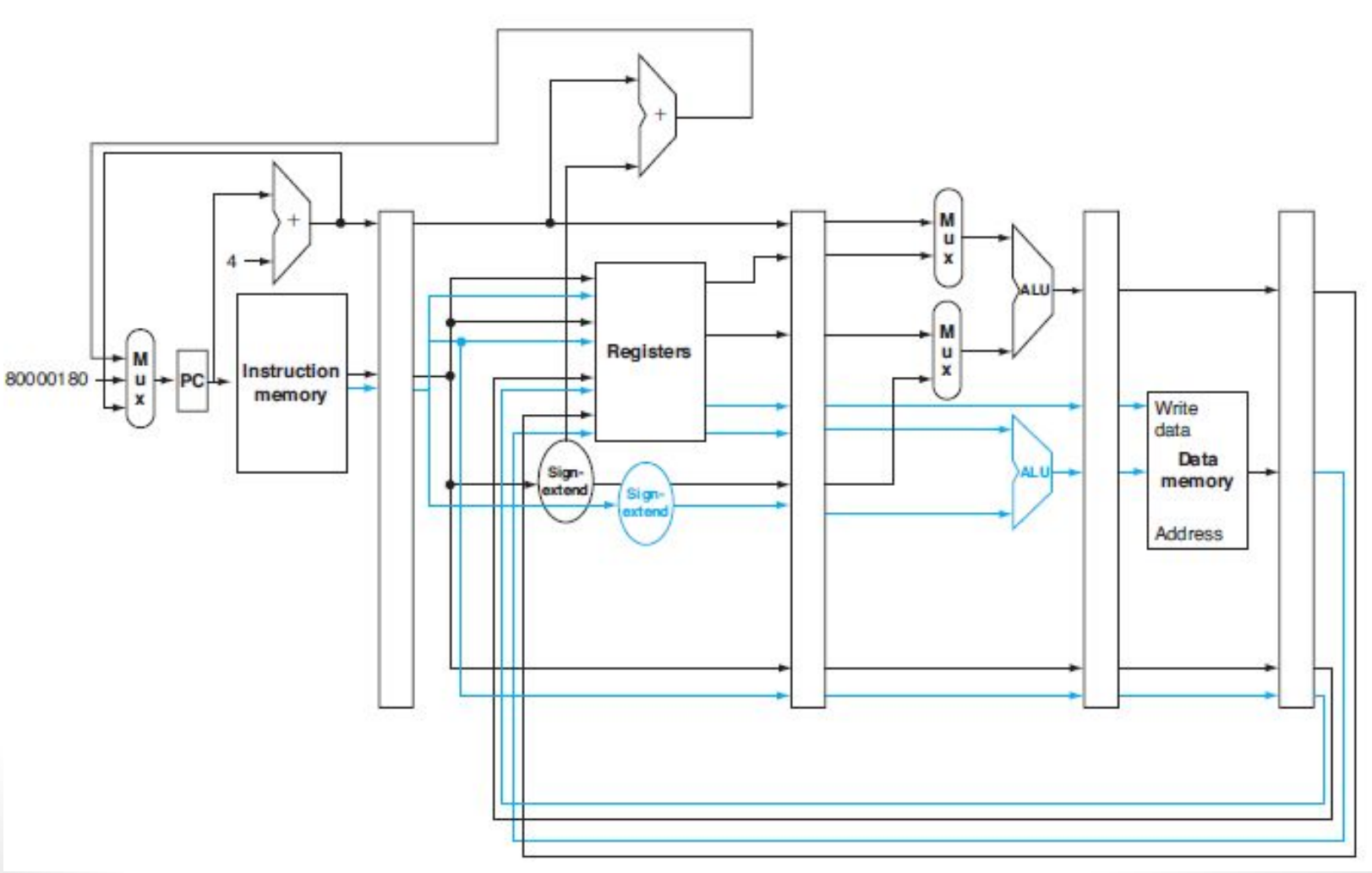
Introdução ao paralelismo em nível de instrução

- Exemplo de despacho múltiplo estático considerando o MIPS:
  - Despache de 2 instruções por ciclo
  - Uma instrução de acesso à memória e a outra com operação na ULA
  - Se uma instrução do par não puder ser executada, ela é substituída por um *nop*

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

Introdução ao paralelismo em nível de instrução

- **Implementação de despacho duplo no MIPS:**



Arquiteturas paralelas

- Existem diversas técnicas de processamento paralelo:
  - Paralelismo em nível de instrução (ILP – Instruction Level Parallelism)
    - superpipelining
    - VLIW (Very Long Instruction Word)
    - superescalar
  - Paralelismo em nível de thread (TLP – Thread Level Parallelism)

Arquiteturas paralelas

REVISÃO:

- Pipeline é uma técnica que visa aumentar o nível de paralelismo de execução de instruções
  - ILP (**I**nstruction-**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

Arquiteturas paralelas

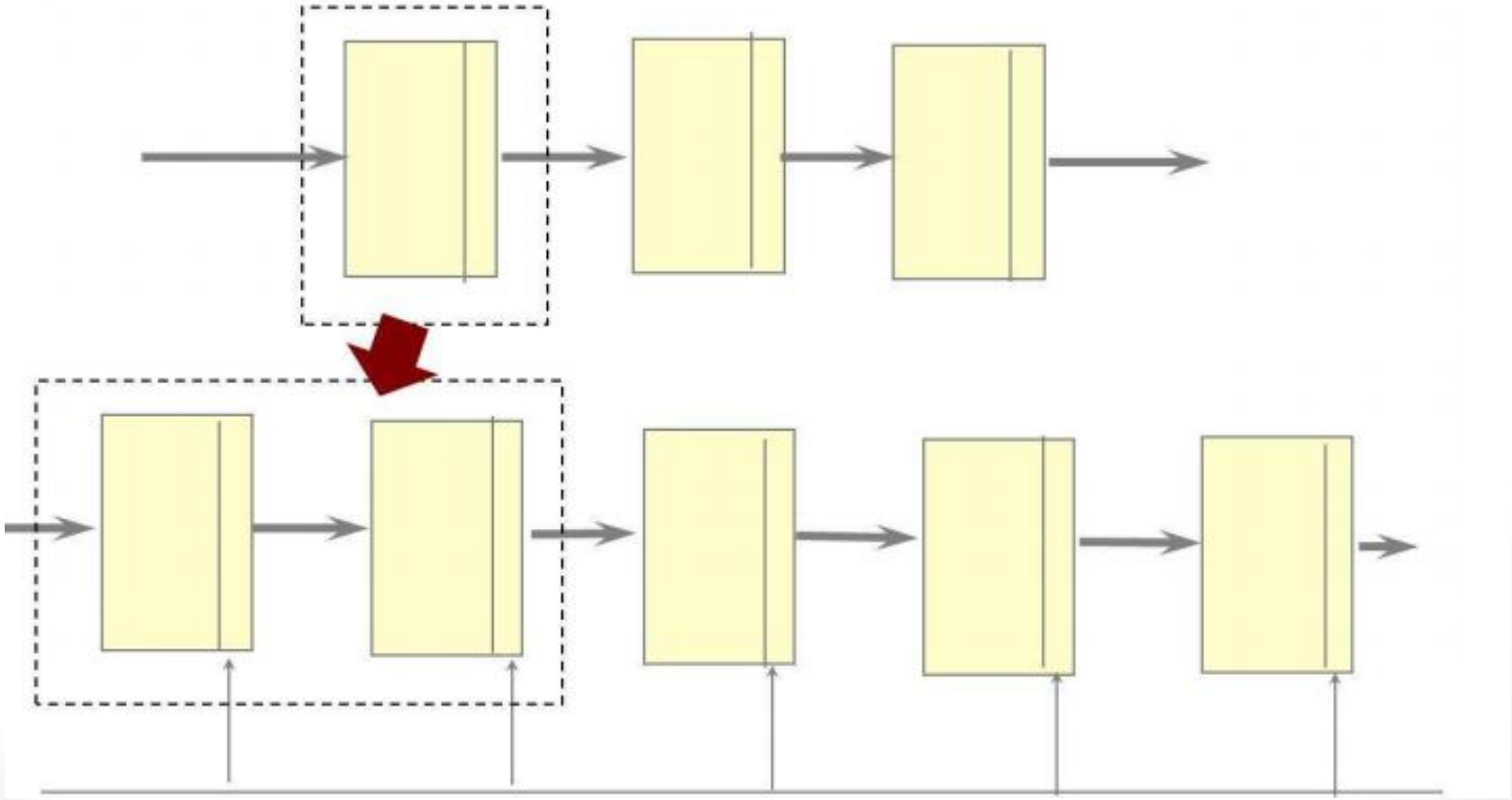
Como melhorar o desempenho do pipeline:

- 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

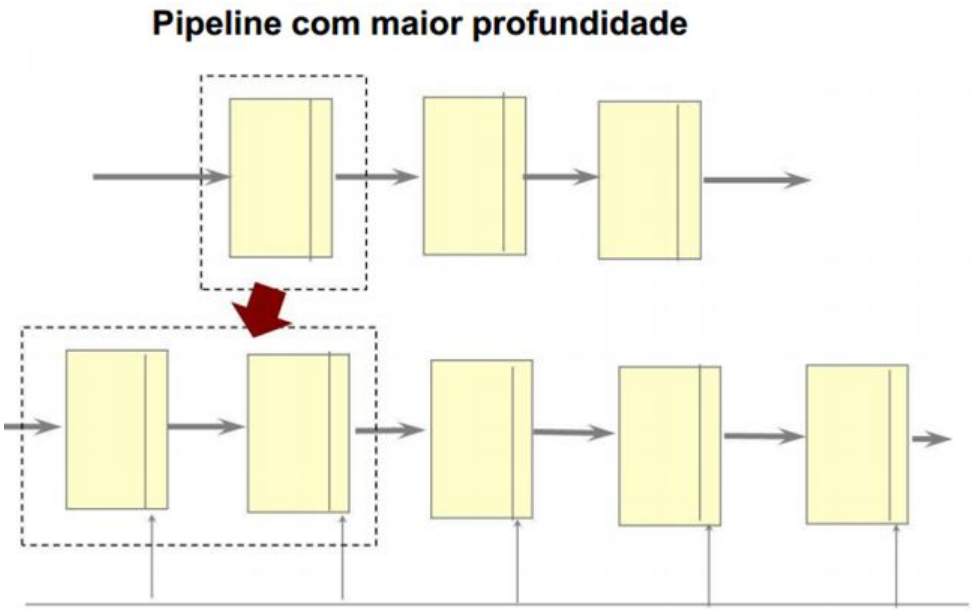
- Quebra estágios em subestágios (estágios menores)
  - Cada subestágio faz menos trabalho que estágio original
  - **Pipeline com maior profundidade**



Superpipeline

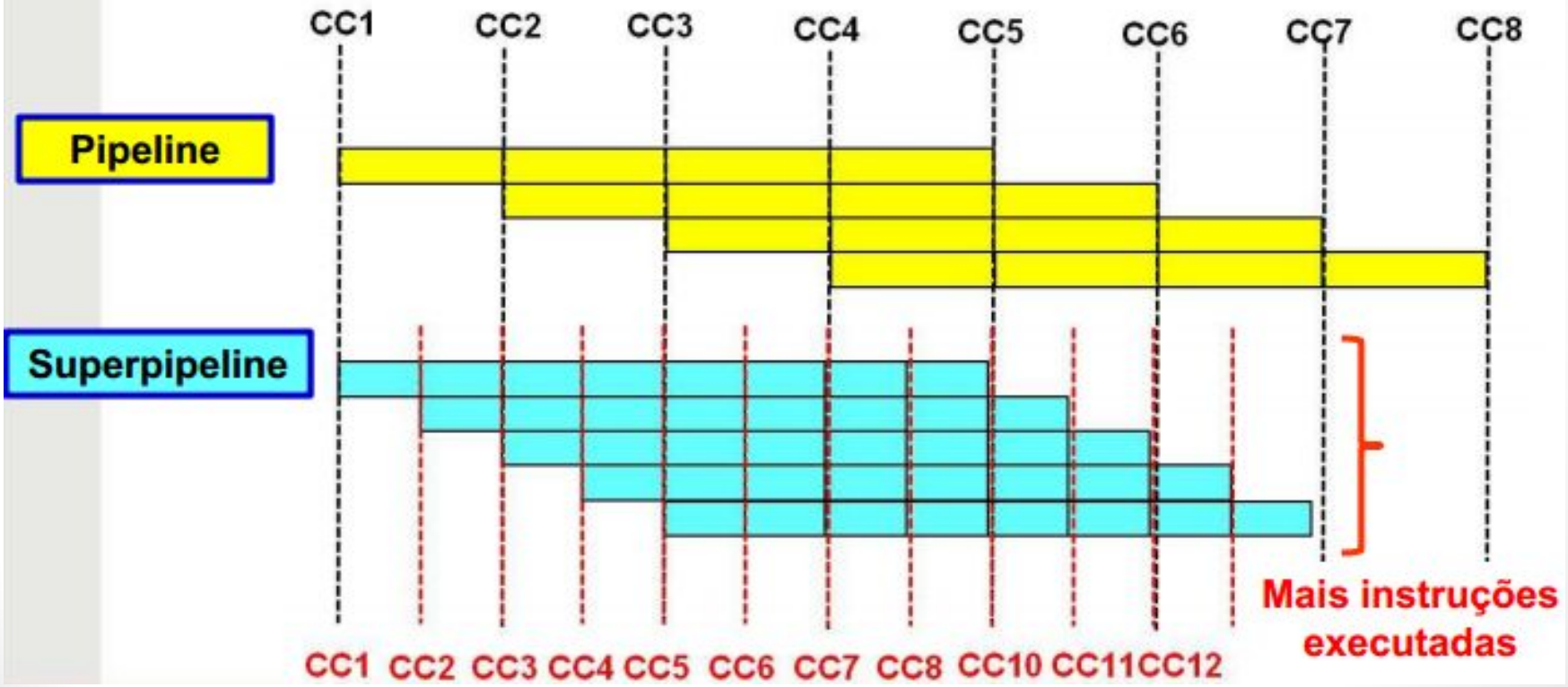
Superpepiline:

- um estágio é menor do que um ciclo de clock;
- pode ser considerado um clock interno, executando, por exemplo, com o dobro da velocidade do clock externo, completando 2 instruções por ciclo de clock;



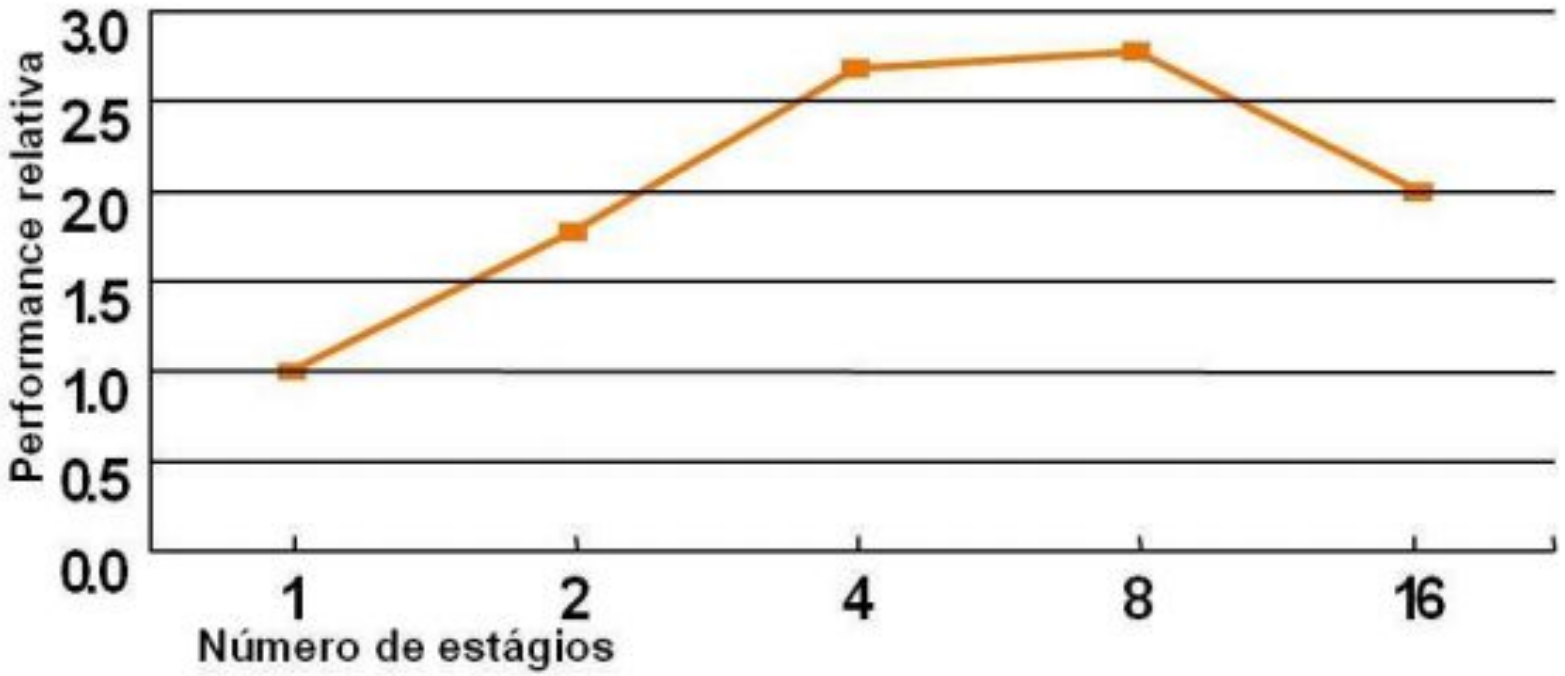
Pipeline x Superpipeline

- 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

Analisando 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

Exploração de paralelismo pelo compilador

- Para manter o pipeline cheio, o paralelismo entre instruções deve ser explorado, realocando sequências de instrução nao-relacionadas que possam ser sobrepostas no pipeline;

- Uma das técnicas utilizadas para aumentar o paralelismo é chamada de *loop unrolling*;

Loop unroling

- É uma técnica que explora o paralelismo em laços.

Exemplo:

```
for (i=1000; i>0; i=i-1)
        x[i] = x[i] + s;
```

Assembly:

```
Loop:   L.D     F0,0(R1)     ;F0=array element
        ADD.D   F4,F0,F2     ;add scalar in F2
        S.D     F4,0(R1)     ;store result
        DADDUI  R1,R1,#-8    ;decrement pointer
                             ;8 bytes (per DW)
        BNE     R1,R2,Loop   ;branch R1!=zero
```

Loop unroling

Aplicando a técnica de *loop unroling*. Na solução, foram implementadas 4 cópias do corpo do loop:

```
Loop:  L.D     F0,0(R1)
       ADD.D   F4,F0,F2
       S.D     F4,0(R1)
       L.D     F6,-8(R1)
       ADD.D   F8,F6,F2
       S.D     F8,-8(R1)
       L.D     F10,-16(R1)
       ADD.D   F12,F10,F2
       S.D     F12,-16(R1)
       L.D     F14,-24(R1)
       ADD.D   F16,F14,F2
       S.D     F16,-24(R1)
       DADDUI  R1,R1,#-32
       BNE     R1,R2,Loop
```

Onde obteve-se ganho com essa implementação?

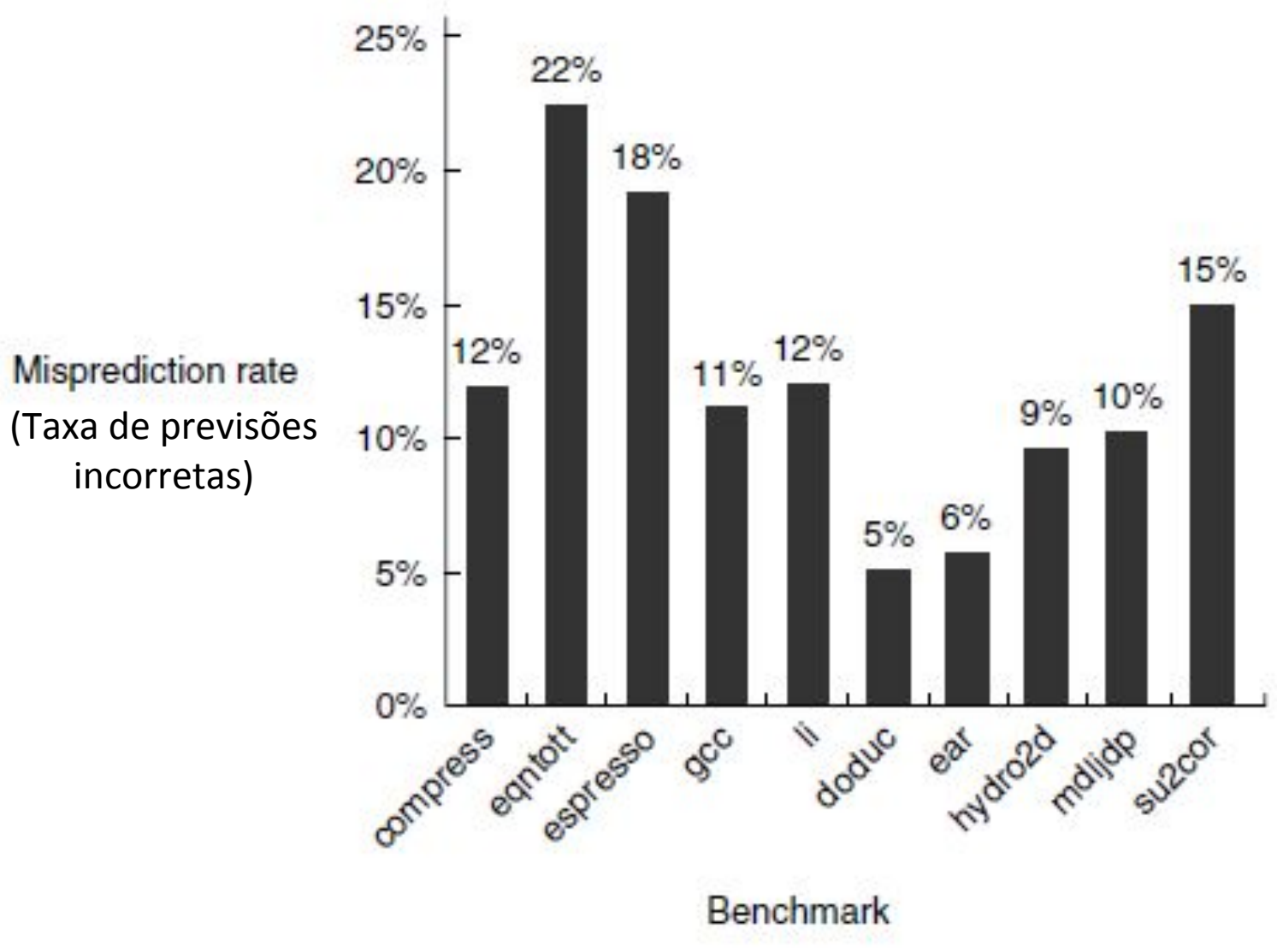
Elimina-se 6 instruções no total, 3 instruções de decremento do ponteiro (DADDUI) e 3 instruções de teste de desvio (BNE)

Previsão de desvio estático

- É utilizado em processadores em que o comportamento de desvios seja altamente previsível em tempo de compilação;
- A previsão estática também pode ser utilizada para auxiliar na previsão dinâmica;
- Podem ser definidos desvios como seguidos e desvios como não-seguidos (para os programas SPEC, 34% na média são de desvios incorretos para definição de desvios seguidos);
- Outra possibilidade é definir desvios de acordo com informações coletadas de execuções anteriores;

Previsão de desvio estático

- Taxa de previsões incorretas:



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

Emissão múltipla estática: abordagem VLIW

- VLIWs se baseiam na tecnologia de compiladores que tem por função:
  - ✓minimizar as paradas potenciais de conflitos de dados (dependências);
  - ✓ formatar as instruções em um pacote de emissão (instruções sem dependência);

- Tem como vantagem a abordagem de um HW mais simples (quando comparada a outras opções de processamento paralelo);

- Processadores VLIW possuem instruções de 64, 128 ou mais bits;

Emissão múltipla estática: abordagem VLIW

- Os VLIWs utilizam várias unidades funcionais independentes;
- No entanto, o processamento é visto como uma única instrução ao invés de diversas instruções independentes;
- Todas as operações são reunidas em uma única instrução muito longa, embora não haja muita diferença entre as duas abordagens (comparando com despacho de múltiplas instruções);
- Nem sempre é possível alocar o número de operações aceitável pelo VLIW, sendo necessário, às vezes, deixar campos de operações vazios.

Emissão múltipla estática: abordagem VLIW

- Os primeiros processadores VLIW eram bastante rígidos em seus formatos de instruções, exigindo a recompilação de programas para diferentes hardwares de processadores;
- Embora seja função do compilador fazer o escalonamento das unidades funcionais para evitar paradas, é difícil prever quais acessos de dados encontrarão uma parada de cache;
- Quando ocorre uma falha na cache, todas as unidades funcionais são paradas;
- Em processadores mais recentes, as unidades funcionais operam com maior independência.

Processadores VLIW

- Processadores VLIW se apoiam inteiramente no compilador:
  - as instruções independentes são empacotadas em uma única instrução longa;
  - visto que o compilador tem uma visão geral das dependências do código, esta abordagem pode apresentar um melhor desempenho;
  - à medida que o compilador VLIW cria instruções muito longas, ele também arbitra todas as dependências;

Processadores VLIW

- Nos processadores VLIW:
  - as instruções que são fixadas durante a compilação, geralmente contém de quatro (4) a oito (8) instruções;

    Quão complexo precisa ser o HW de um processador VLIW?

  - A tecnologia de processadores VLIW pode ter um HW simplificado, já que a complexidade é repassada para o compilador.

Emissão múltipla estática: abordagem VLIW

- A compatibilidade de HW é um dos problemas dos processadores VLIW:
  - Diferentes números de unidades funcionais e latências de unidades exigem versões diferentes de código (ponto flutuante, inteiros, acesso à memória…);
  - Uma solução para esse caso é a conversão ou emulação do código-objeto (ao invés de compilar para cada caso);

Detecção e otimização do paralelismo em nível de loop

- A análise em nível de loop envolve a determinação das dependências entre os operandos de um loop
- *Dependência transportada por loop*: é quando os acessos a dados em iterações posteriores são dependentes de valores de dados produzidos em iterações anteriores. Ex.:

```
for (i=1; i<=100; i=i+1) {
        A[i+1] = A[i] + C[i];    /* S1 */
        B[i+1] = B[i] + A[i+1]; /* S2 */
}
```

Quais são as dependências existentes nesse *for*?

Detecção e otimização do paralelismo em nível de loop

```
for (i=1; i<=100; i=i+1) {
        A[i+1] = A[i] + C[i];   /* S1 */
        B[i+1] = B[i] + A[i+1]; /* S2 */
}
```

✓ S1 utiliza um valor calculado por S1 em uma iteração anterior. O mesmo acontece com S2;

✓ S2 utiliza o valor A[i+1] calculado por S1 na mesma iteração.

- No primeiro caso, a dependência força a execução em série de iterações sucessivas desse loop;
- No segundo caso, a dependência está dentro de uma iteração. Assim, as iterações podem ser executadas em paralelo, desde que seja mantida a ordem;

Detecção e otimização do paralelismo em nível de loop

Exemplo 2:

```
for (i=1; i<=100; i=i+1) {
        A[i] = A[i] + B[i];    /* S1 */
        B[i+1] = C[i] + D[i]; /* S2 */
}
```

E agora quais são as dependências? Esse loop é paralelo?

Detecção e otimização do paralelismo em nível de loop

```
for (i=1; i<=100; i=i+1) {
        A[i] = A[i] + B[i];    /* S1 */
        B[i+1] = C[i] + D[i]; /* S2 */
}
```

- ✓S1 utiliza o valor atribuido na instrução anterior pela instrução S2, sendo assim, existe uma dependência transportada por loop. No entanto, não é uma dependência circular (S2 não depende de S1);

- ✓ Na primeira iteração do loop, a instrução S1 depende do valor de B[1] calculado antes da inicialização do loop;

Detecção e otimização do paralelismo em nível de loop

- Reescrevendo o código, tem-se:

```
A[1] = A[1] + B[1];
for (i=1; i<=99; i=i+1) {
        B[i+1] = C[i] + D[i];
        A[i+1] = A[i+1] + B[i+1];
 }
 B[101] = C[100] + D[100];
```

✓ Dessa forma, elimina-se a dependência existente anteriormente de forma que iterações do loop possam ser sobrepostas, desde que as instruções em cada iteração sejam mantidas na ordem.

A arquitetura IA-64 da Intel e o processador Itanium

- O IA-64 apresenta um conjunto de instruções registrador-registrador no estilo RISC mas com suporte à exploração do ILP baseada no compilador;
- Possui 128 registradores de uso geral de 64 bits;
- Possui 128 registradores de ponto flutuante de 82 bits;
- 64 registradores de predicados de 1 bit;
- 8 registradores de desvio de 64 bits, usados para desvios indiretos;
- outros registradores utilizados para controle do sistema.

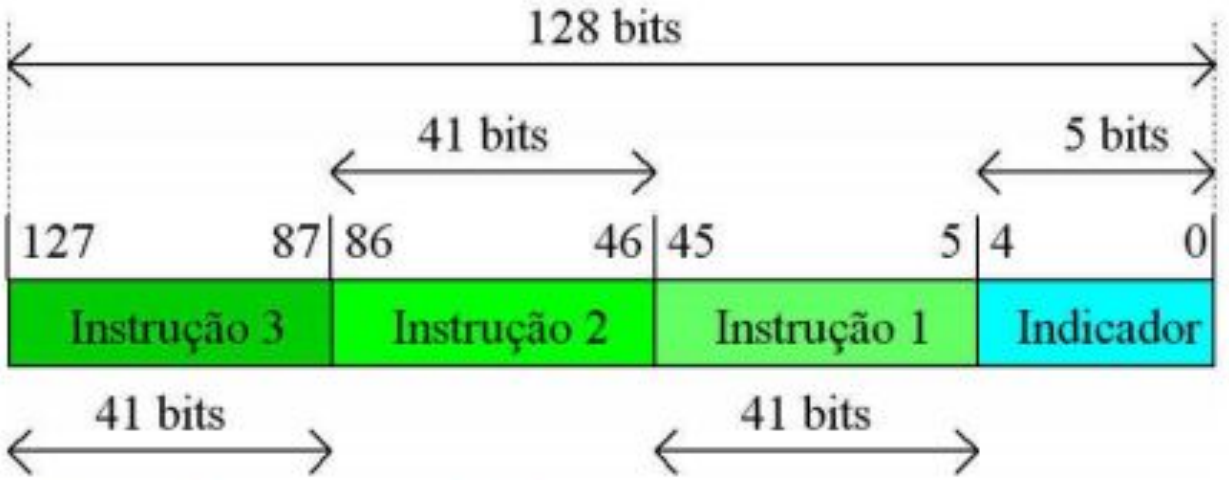
A arquitetura IA-64 da Intel e o processador Itanium

Slots de unidades de execução da arquitetura IA-64:

Execution Unit Slot	Instruction type	Instruction Description	Example Instructions
I-unit	A	Integer ALU	add, subtract, and, or, compare
	I	Non-ALU Integer	integer and multimedia shifts, bit tests, moves
M-unit	A	Integer ALU	add, subtract, and, or, compare
	M	Memory access	Loads and stores for integer/FP registers
F-unit	F	Floating point	Floating point instructions.
B-unit	B	Branches	Conditional branches, calls, loop branches
L+X	L+X	Extended	Extended immediates, stops and no-ops.

A arquitetura IA-64 da Intel e o processador Itanium

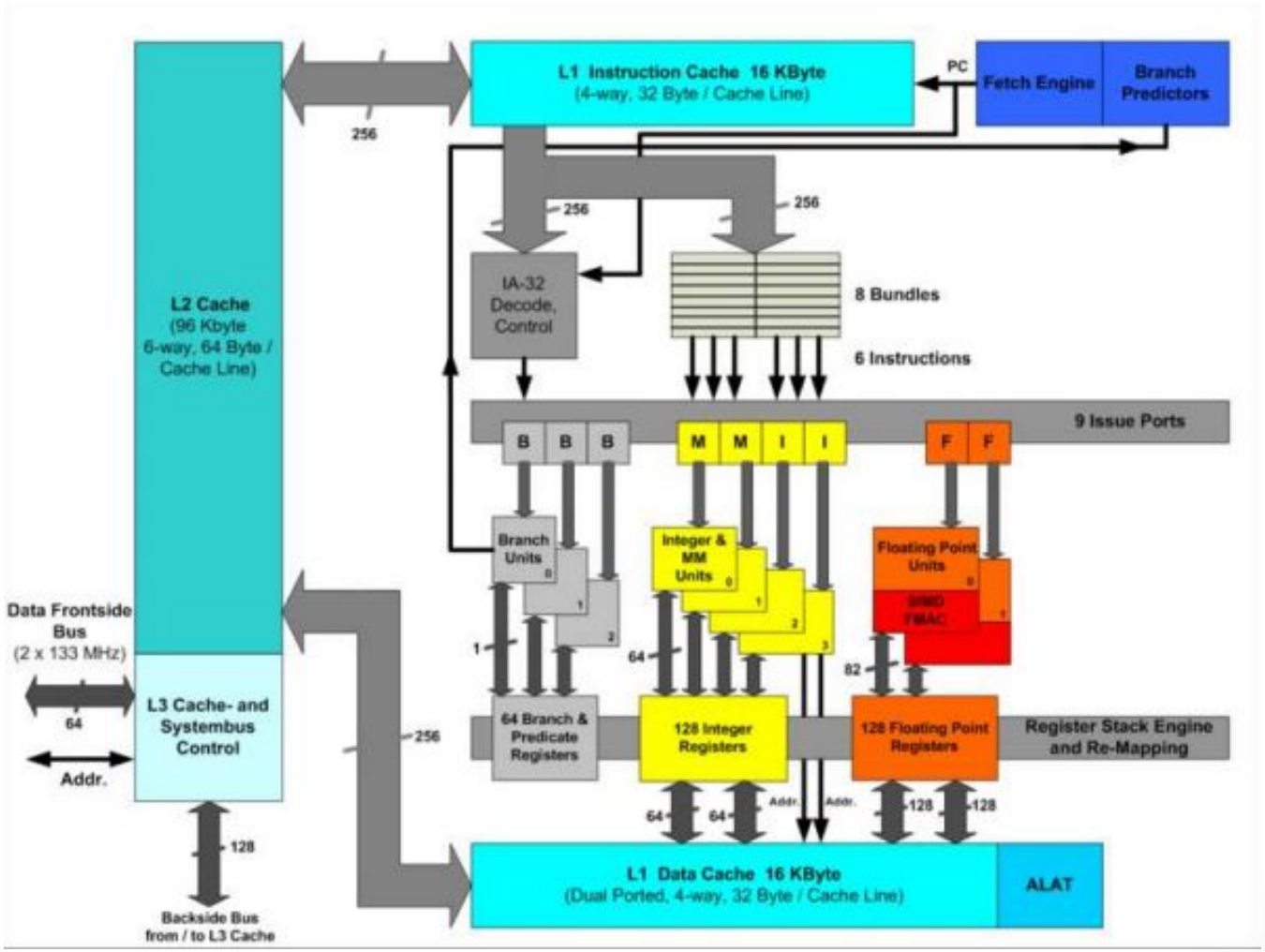
Exemplos de processador VLIW: Itanium da Intel



- Instruções são empacotadas
  - Cada pacote (*bundle*) contém 3 instruções e 128 bits
  - Cada instrução tem 41 bits
  - 5 bits são utizados para informar quais unidades funcionais serão usadas pelas instruções

A arquitetura IA-64 da Intel e o processador Itanium

Exemplos de processador VLIW: Itanium da Intel

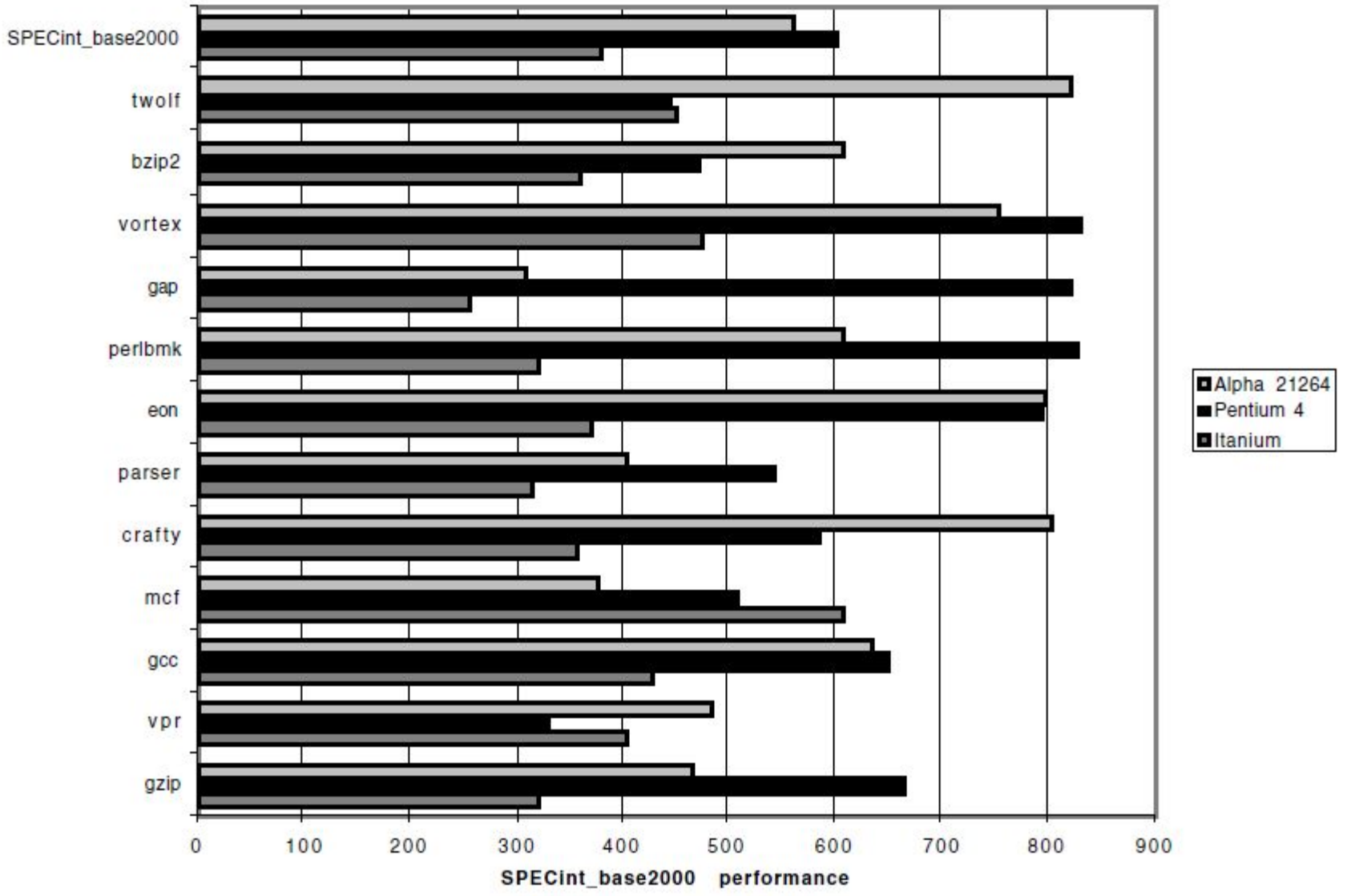


A arquitetura IA-64 da Intel e o processador Itanium

- O processador Itanium é a primeira implementação da arquiteutra IA-64.
- O núcleo do processador é capaz de até seis emissões por clock, com até 3 desvios e 2 referências de memória;
- A hierarquia de memória consiste de uma cache de 3 níveis (L1 com cache de instruções e dados separadas);
- Há 9 unidades funcionais: 2 unidades I, 2 unidades M, 3 unidades B e 2 unidades F;
- Todas as unidades funcionais possuem pipeline;

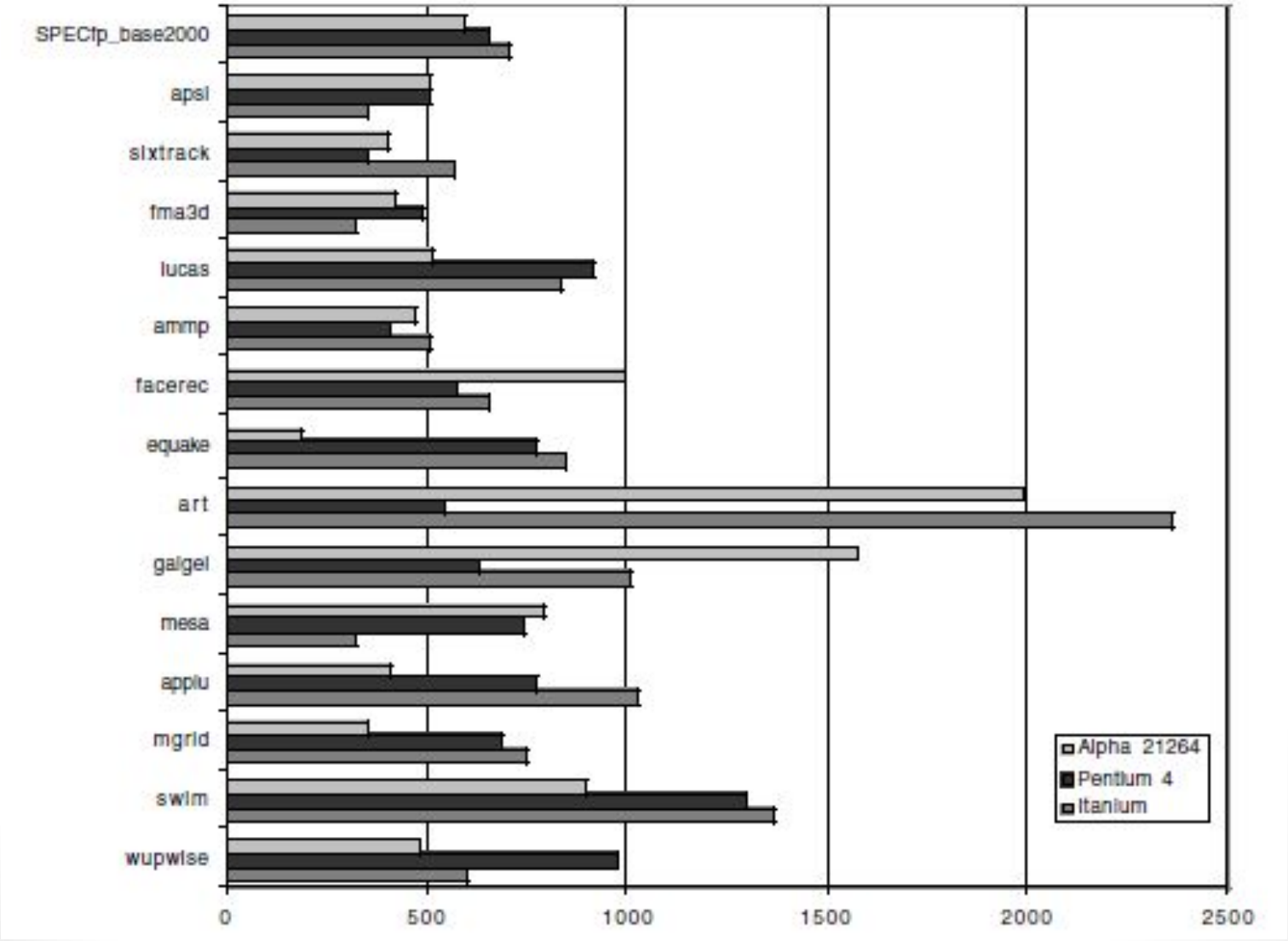
Processador Itanium

Operações com Inteiros:



Processador Itanium

Operações com Ponto-flutuante:



Outros processadores VLIW

- Exemplos de processadores com abordagens interessantes para sistemas embutidos são:
  - ✓ Trimedia
  - ✓ Crusoe

- O processador Crusoe utiliza conversão de SW da arquitetura X86 para um processador VLIW, obtendo menor energia que os processadores X86 – característica importante em aplicações móveis;

Outros processadores VLIW

- A CPU da Trimedia é uma arquitetura VLIW clássica: toda a instrução contém 5 operações e o escalonamento do processador é completamente estático;
- A CPU da Crusoe é um processador VLIW para o mercado de baixa potência
  - possui instruções de 2 tamanhos: 64 e 128 bits;
  - apresenta 4 slots de operações

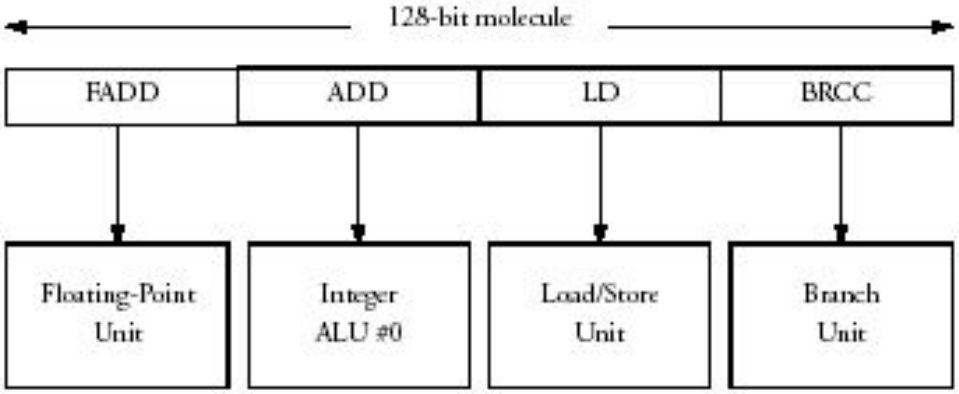
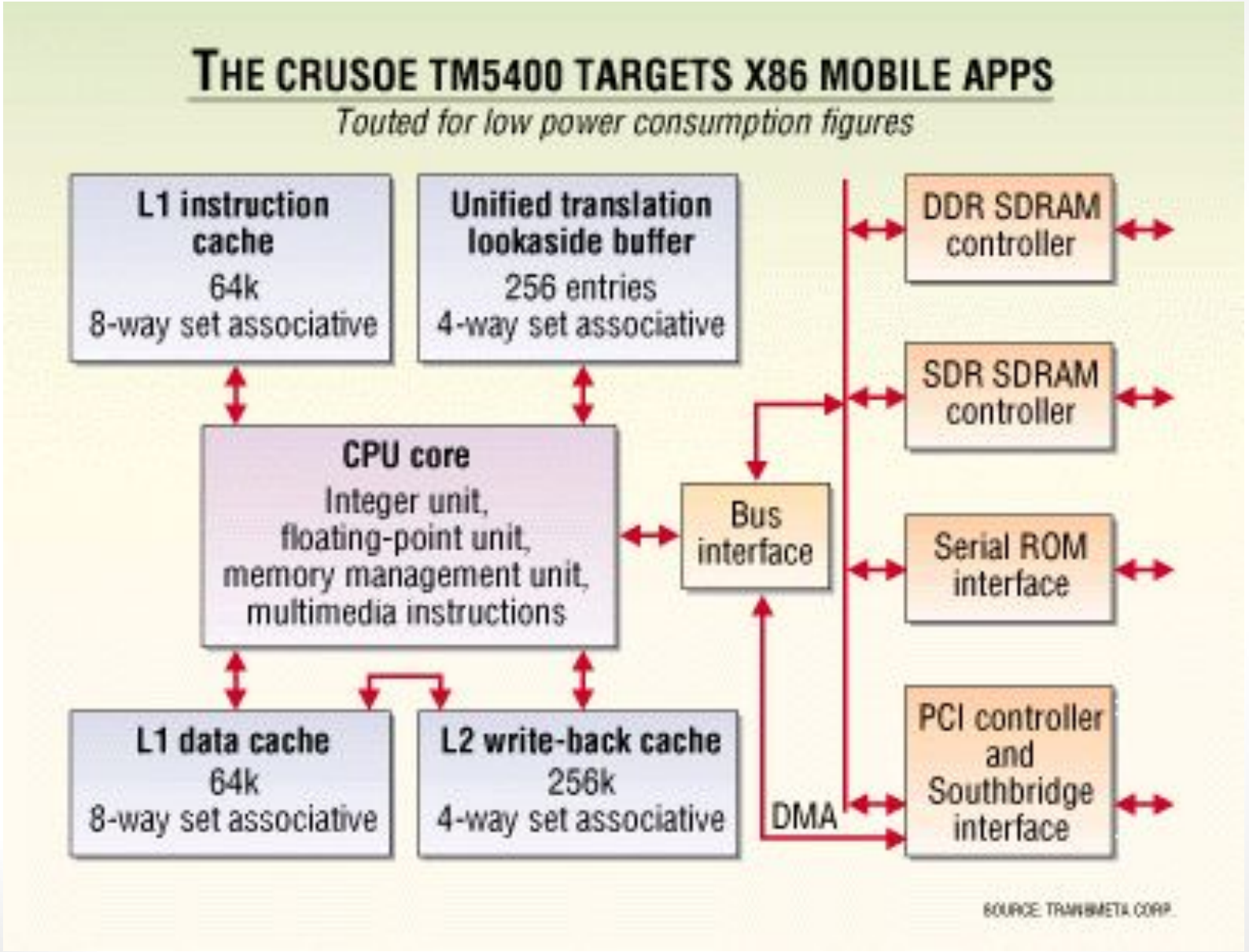
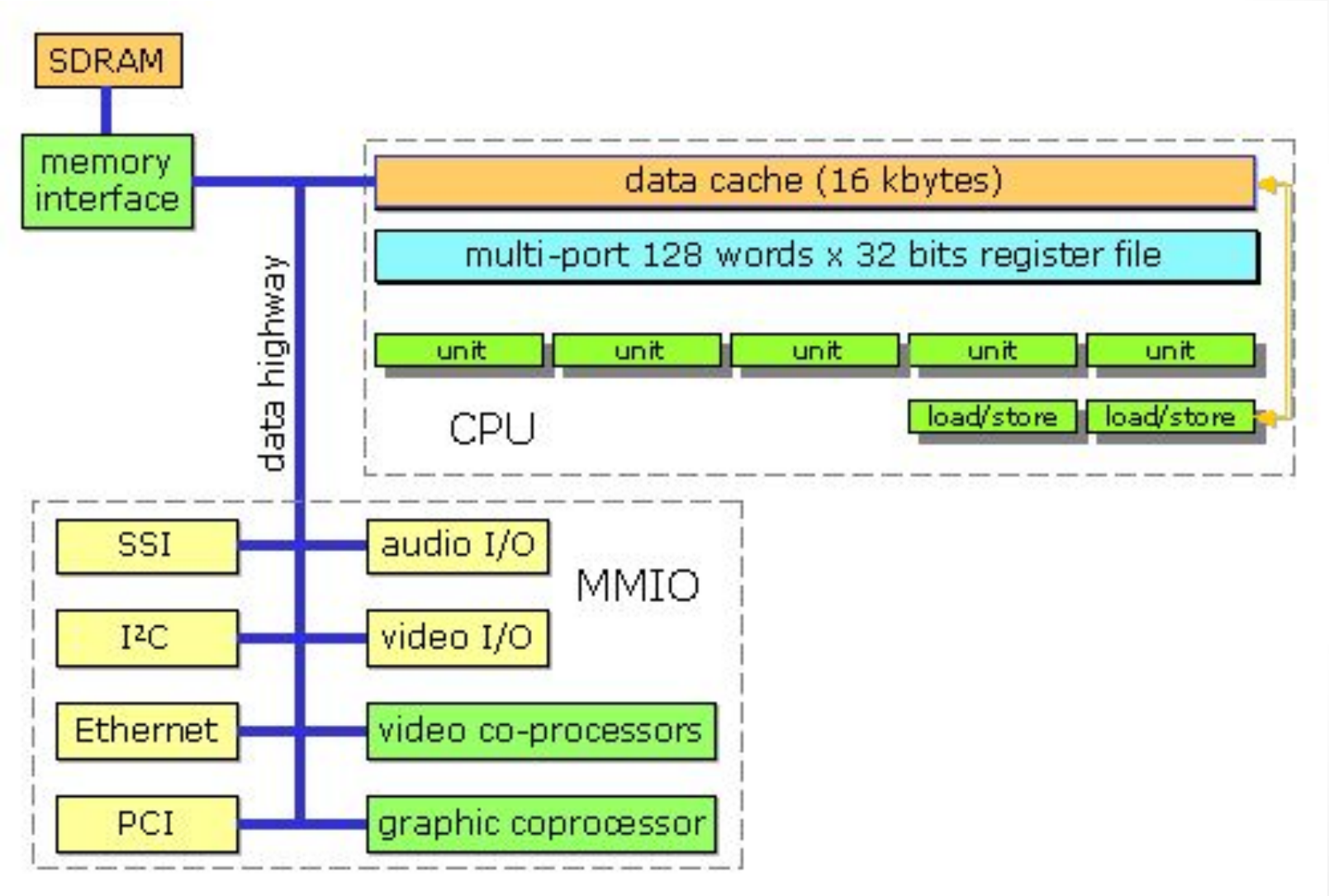


Figure 1. A molecule can contain up to four atoms, which are executed in parallel.

Outros processadores VLIW



Outros processadores VLIW



Resumo VLIW

Analisando 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