ARQUITETURA DE COMPUTADORES

VLIW (Very Long Instruction Word)

Introdução

- Máquinas que exploram paralelismo no nível das instruções
- Várias operações são executadas em paralelo em diferentes unidades funcionais, como em máquinas superescalares
- A diferença está no controle do despacho e da terminação das operações
- Superescalares: as dependências são resolvidas em tempo de execução por um hardware dedicado
- VLIW: as dependências são resolvidas em tempo de compilação pelo compilador
- Teoricamente tanto processadores superescalares como VLIW conseguem explorar o mesmo paralelismo existente nos programas, apenas com mecanismos e custos diferentes

Introdução

- Em uma máquina VLIW, várias instruções independentes são codificadas em uma mesma palavra longa
- A posição de cada operação dentro da palavra VLIW determina a unidade funcional que será usada
- Se não há uma instrução para ser executada numa das unidades funcionais numa dada palavra, a posição na palavra é ocupada por um NOP
  - Maior gasto de memória de programa
- O hardware de despacho é simples

3

VLIW (resumo 1)

- Máquinas que exploram paralelismo no nível das instruções
- Várias operações são executadas em paralelo em diferentes unidades funcionais, como em máquinas superescalares
  - A diferença está no controle do despacho e da terminação das operações
  - Superescalares: as dependências são resolvidas em tempo de execução por um hardware dedicado
  - VLIW: as dependências são resolvidas em tempo de compilação pelo compilador
- Teoricamente, tanto processadores superescalares quanto VLIW conseguem explorar o mesmo paralelismo existente nos programas, apenas com mecanismos e custos diferentes

4

VLIW (resumo 2)

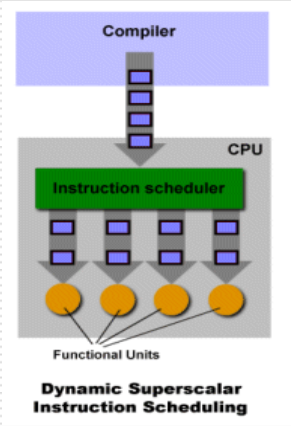
- Em uma máquina VLIW, várias instruções independentes são codificadas em uma mesma palavra longa
- A posição de cada operação dentro da palavra VLIW determina a unidade funcional que será usada
- Se não há uma instrução para ser executada numa das unidades funcionais numa dada palavra, a posição na palavra é ocupada por um NOP
  - Maior gasto de memória de programa
- O hardware de despacho é simples

EPIC – Explicit Parallelism Instruction set Computers

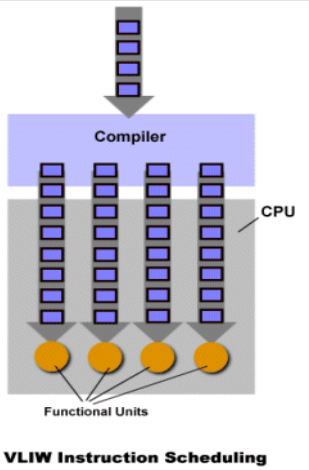
5

Comparação entre arquiteturas

Superescalar

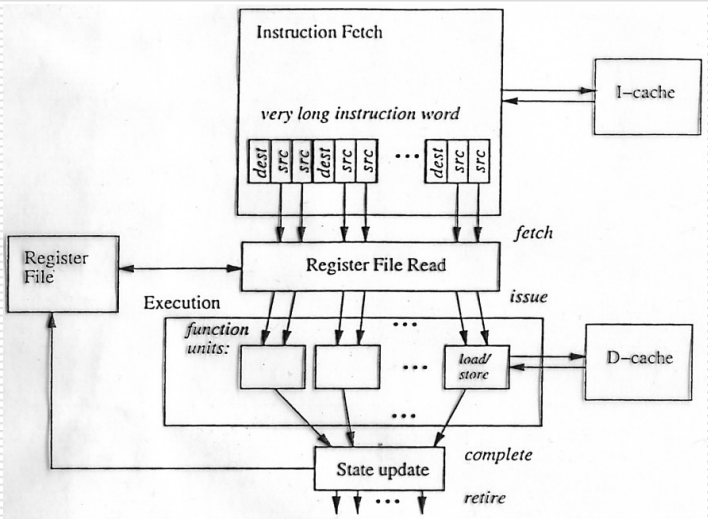


VLIW



6

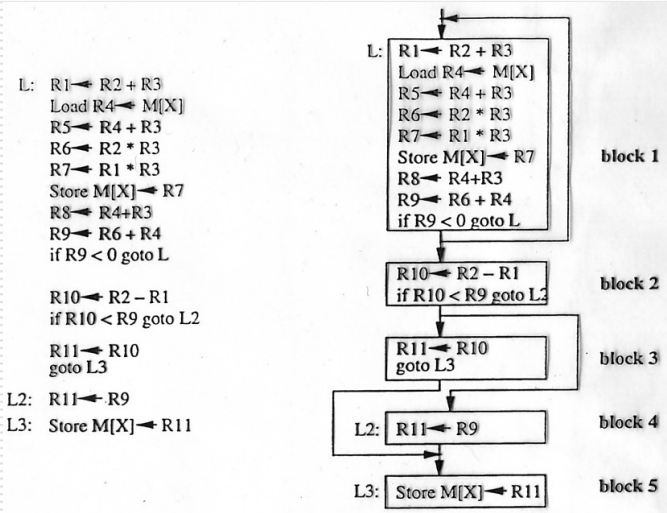
Organização VLIW



Programas e Fluxos (compiladores)

- A compilação de um programa é (normalmente) feita em três fases
  - *Parsing* da linguagem de entrada para uma descrição (estrutura) intermediária
  - Otimização da estrutura intermediária
  - Geração do código para a arquitetura-alvo
- A estrutura de dados intermediária representa:
  - Fluxo de controle – as diferentes seqüências de execução das instruções que formam o programa
  - Fluxo de dados – as dependências de dados que existem entre as várias operações
- Blocos básicos: grupos de instruções que são SEMPRE executadas em seqüência

Blocos básicos

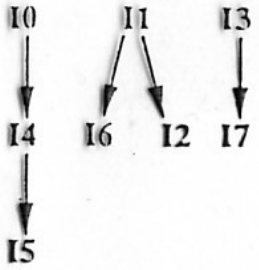


Escalonamento

- O escalonamento das operações consiste em determinar as operações que serão executadas em paralelo
- Em uma máquina VLIW o compilador é responsável por esta tarefa
- Operações que são executadas em paralelo são atribuídas à mesma palavra de instrução
- Em um mesmo bloco básico (seqüencial) estas instruções podem ser escalonadas com base no fluxo de dados
  - Escalonamento ASAP (*as soon as possible*)
- A transição da fronteira entre blocos exige técnicas mais elaboradas
  - Escalonamento acíclico: programas com desvios, sem ciclos
  - Escalonamento cíclico: programas com ciclos e laços

Escalonamento dentro de um bloco básico

I0 R1 ← R2 + R3
I1 Load R4 ← M[X]
I2 R5 ← R4 + R3
I3 R6 ← R2 * R3
I4 R7 ← R1 * R3
I5 Store M[X] ← R7
I6 R8 ← R4+R3
I7 R9 ← R6 + R4



Fluxo de dados para o código exemplo

Codificação da instrução

I0 R1 ← R2 + R3
I1 Load R4 ← M[X]
I2 R5 ← R4 + R3
I3 R6 ← R2 * R3
I4 R7 ← R1 * R3
I5 Store M[X] ← R7
I6 R8 ← R4+R3
I7 R9 ← R6 + R4

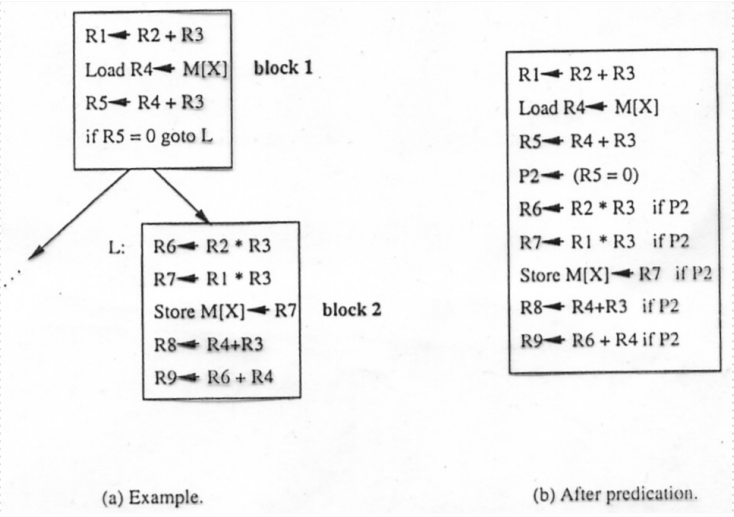
+			+			*			Load	Store	
dest	*src*	*src*	*dest*	*src*	*src*	*dest*	*src*	*src*	*dest*	*address*	*address*	*dest*
R1	R2	R3				R6	R2	R3	R4	X		
						R7	R1	R3				
R5	R4	R3	R8	R4	R3							
R9	R6	R4										
											X	R7

Escalonamento acíclico

- Exemplo de técnica: escalonamento baseado em predicados
- Operações que se seguem a um *if* são associadas com um predicado
  - valor do predicado é definido em função do resultado do teste do *if*
- A palavra VLIW deve ser acrescida de um registrador de predicado para cada instrução
- A máquina deve ser acrescida de uma unidade operativa para implementar operador de definição de predicados
- Uma operação ...
  - é completada se o predicado é verdadeiro
  - não é completada se o predicado é falso
- Esta técnica é conhecida como *if-conversion*

13

Predicação



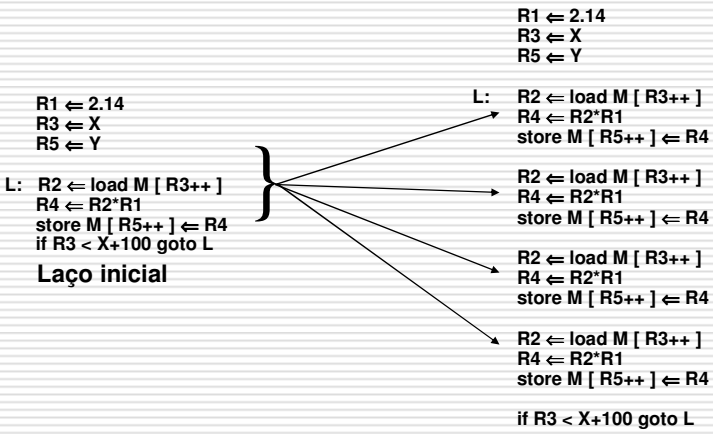
14

Escalonamento Cíclico

- O código do ciclo inicial é reescrito seqüencialmente até se conseguir um padrão de instruções que se repetem ao longo do tempo
  - *loop unrolling*
- Este padrão repetido ao longo do tempo é chamado de *kernel* do laço
- O kernel do laço será realizado por uma (ou mais) instruções VLIW
- Cada instrução VLIW pode conter operações de diferentes ciclos antes da duplicação (isto é representado com índices)

15

Escalonamento cíclico: exemplo



16

Escalonamento cíclico: exemplo

```
        R1 ⇐ 2.14
        R3 ⇐ X
        R5 ⇐ Y

L:      R2 ⇐ load M[R3++]
        R4 ⇐ R2*R1
        store M[R5++] ⇐ R4

        R2 ⇐ load M[R3++]
        R4 ⇐ R2*R1
        store M[R5++] ⇐ R4

        R2 ⇐ load M[R3++]
        R4 ⇐ R2*R1
        store M[R5++] ⇐ R4

        R2 ⇐ load M[R3++]
        R4 ⇐ R2*R1
        store M[R5++] ⇐ R4

        if R3<X+100 goto L
```

Laço reescrito 4 vezes seqüencialmente

```
R1 ⇐ 2.14
R3 ⇐ X
R5 ⇐ Y

R2_1 ⇐ load M[R3++]
R2_2 ⇐ load M[R3++]   R4_1 ⇐ R2_1*R1
R2_3 ⇐ load M[R3++]   R4_2 ⇐ R2_2*R1   store M[R5++] ⇐ R4_1
R2_4 ⇐ load M[R3++]   R4_3 ⇐ R2_3*R1   store M[R5++] ⇐ R4_2
R2_5 ⇐ load M[R3++]   R4_4 ⇐ R2_4*R1   store M[R5++] ⇐ R4_3
R2_6 ⇐ load M[R3++]   R4_5 ⇐ R2_5*R1   store M[R5++] ⇐ R4_4
R2_7 ⇐ load M[R3++]   R4_6 ⇐ R2_6*R1   store M[R5++] ⇐ R4_5
R2_8 ⇐ load M[R3++]   R4_7 ⇐ R2_7*R1   store M[R5++] ⇐ R4_6
R2_9 ⇐ load M[R3++]   R4_8 ⇐ R2_8*R1   store M[R5++] ⇐ R4_7
R2_10 ⇐ load M[R3++]  R4_9 ⇐ R2_9*R1   store M[R5++] ⇐ R4_8
                      R4_10 ⇐ R2_10*R1  store M[R5++] ⇐ R4_9
                                        store M[R5++] ⇐ R4_10
if R3<X+100 goto L
```

Ao longo do tempo estas três operações podem sempre ser executadas em paralelo

Escalonamento cíclico: exemplo

Ao longo do tempo estas três operações podem sempre ser executadas em paralelo

```
R2_3 ⇐ load M[R3++]
R2_4 ⇐ load M[R3++]
R2_5 ⇐ load M[R3++]
R2_6 ⇐ load M[R3++]
R2_7 ⇐ load M[R3++]
R2_8 ⇐ load M[R3++]
R2_9 ⇐ load M[R3++]
R2_10 ⇐ load M[R3++]
```

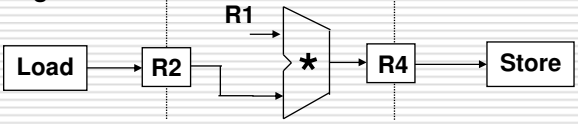
1º estágio

```
R4_2 ⇐ R2_2*R1
R4_3 ⇐ R2_3*R1
R4_4 ⇐ R2_4*R1
R4_5 ⇐ R2_5*R1
R4_6 ⇐ R2_6*R1
R4_7 ⇐ R2_7*R1
R4_8 ⇐ R2_8*R1
R4_9 ⇐ R2_9*R1
```

2º estágio

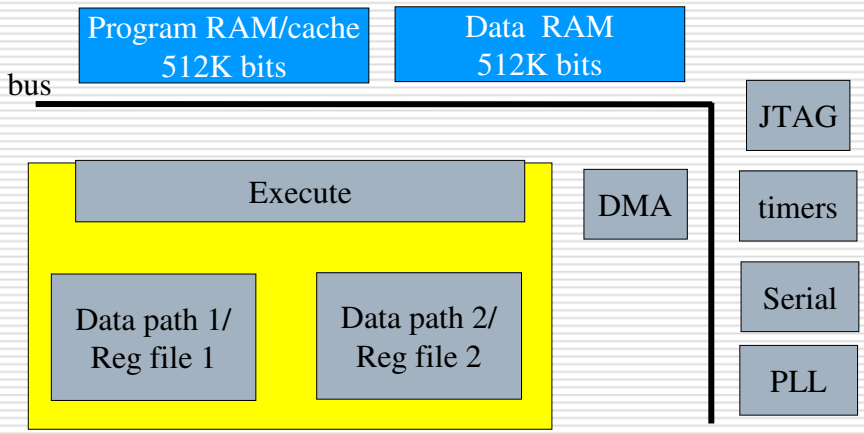
```
store M[R5++] ⇐ R4_1
store M[R5++] ⇐ R4_2
store M[R5++] ⇐ R4_3
store M[R5++] ⇐ R4_4
store M[R5++] ⇐ R4_5
store M[R5++] ⇐ R4_6
store M[R5++] ⇐ R4_7
store M[R5++] ⇐ R4_8
```

3º estágio



Na prática o compilador usou as unidades operativas para configurar (via software) um pipeline de 3 estágios dedicado à execução deste laço

Exemplo: TMS 320C6x (diagrama de blocos)



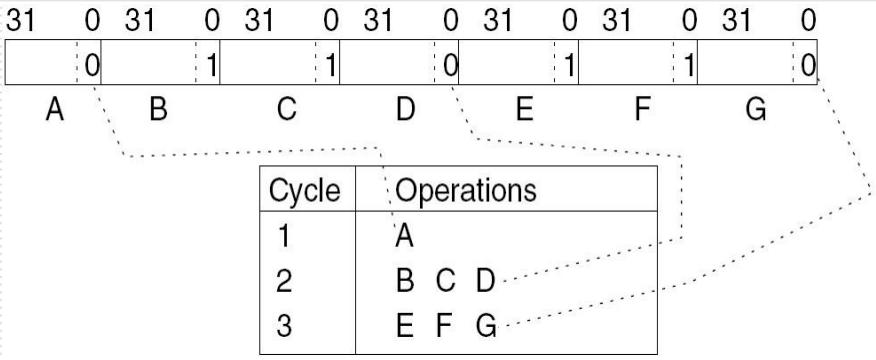
19

Caminho de dados do 320C6x

- Registradores de propósito geral (A and B, 16 words each).
- Oito unidades funcionais:
  - .L1, .L2, .S1, .S2, .M1, .M2, .D1, .D2
- Duas unidades load (LD1, LD2).
- Duas unidades store (ST1, ST2).
- Dois registradores *cross paths* (1X and 2X).
- Dois operadores dado-endereço (DA1 and DA2).

20

320C6x – formato de instrução



Cycle	Operations
1	A
2	B C D
3	E F G

Literatura

- *Patterson & Hennessy – cap 6.9, 6.10*
- *Stallings – cap 13*
- *Slides do prof. Gabriel P. Silva (DCC-UFRJ)*
- *Sites de fabricantes*
  - *Itanium*- Intel
  - *Crusoe* – Transmeta
- http://www.gdhpress.com.br/hmc/leia/index.php?p=cap2-34
- http://www.hardwareanalysis.com/content/article/1237.2/

Resumo

De tanto ver triunfar as nulidades, de tanto ver prosperar a desonra, de tanto ver crescer a injustiça, de tanto ver agigantarem-se os poderes nas mãos dos maus, o homem chega a desanimar da virtude, a rir-se da honra, a ter vergonha de ser honesto...

Rui Barbosa de Oliveira, 1914

Um povo cuja fé se petrificou, é um povo cuja liberdade se perdeu.

Rui Barbosa de Oliveira

25

Questões adicionais...

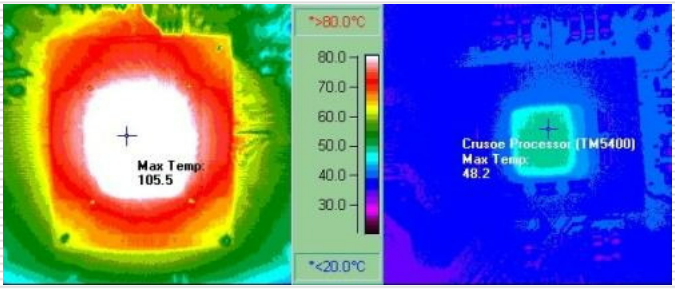
- Pesquise o significado do *code morphing* usado no processador Crusoe

26

Processador Crusoe

- processador simples
  - poucas unidades funcionais
  - parte de controle simplificada
    - economia de potência
- compatível com o conjunto de instruções "x86"
- realiza tradução dinâmica de código, no nível do conjunto de instruções (*tradução binária*)
  - aplicações são escritas com código "assembly" para o conjunto de instruções do "x86"
- implementa um *compromisso* entre HW e SW na execução de algoritmos
  - SW → decodificação de instruções "x86" e escalonamento destas em instruções VLIW
  - HW → processador VLIW

27



Pentium III Coppermine e Crusoe.
Ambos exibindo um filme em DVD, sem refrigeração especial.

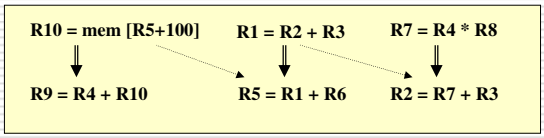
28

Exercício

I1: R1 = R2 + R3
I2: R10 = mem[R5+100]
I3: R5 = R1 + R6
I4: R7 = R4 * R8
I5: R2 = R7 + R3
I6: R9 = R4 + R10

Somador tem latência de 1 ciclo
Multiplicador e memória têm latência de 2 ciclos

Dependências de dados ⟹ verdadeiras
⇢ falsas



+	+	*	load / store
R1 = R2 + R3	nop	R7 = R4 * R8	R10 = mem […]
R5 = R1 + R6	nop	nop	nop
R9 = R4 + R10	R2 = R7 + R3	nop	nop

VLIW gasta mais memória de programa que superescalar

29