

コンピュータアーキテクチャ (9)

坂井 修一

東京大学大学院 情報理工学系研究科 電子情報学専攻
東京大学 工学部 電子情報工学科 / 電気電子工学科

- ・ はじめに
- ・ 命令レベル並列処理 (2)

はじめに

- 本講義の目的
 - コンピュータアーキテクチャの基本を学ぶ
- 時間・場所
 - 火曜日 8:40 - 10:10 工学部2号館241
- ホームページ（ダウンロード可能）
 - url: <http://www.mtl.t.u-tokyo.ac.jp/~sakai/hard/>
- 教科書
 - 坂井修一『コンピュータアーキテクチャ』（コロナ社、電子情報レクチャーシリーズC-9）
 - 坂井修一『実践 コンピュータアーキテクチャ』（コロナ社）
 - 教科書通りやります
- 参考書
 - D. Patterson and J. Hennessy, Computer Organization & Design, 3rd Ed. (邦訳『コンピュータの構成と設計』(第3版)上下(日系BP))
 - 馬場敬信『コンピュータアーキテクチャ』(改訂2版)、オーム社
 - 富田眞治『コンピュータアーキテクチャ I』、丸善
- 予備知識： 論理回路
 - 坂井修一『論理回路入門』、培風館
- 成績
 - 試験+レポート+出席

講義の概要と予定 (1 / 2)

1. コンピュータアーキテクチャ入門

デジタルな表現、負の数、実数、加算器、ALU, フリップフロップ、レジスタ、計算のサイクル

2. データの流れと制御の流れ

主記憶装置、メモリの構成と分類、レジスタファイル、命令、命令実行の仕組み、実行サイクル、算術論理演算命令、シーケンサ、条件分岐命令

3. 命令セットアーキテクチャ

操作とオペランド、命令の表現形式、アセンブリ言語、命令セット、算術論理演算命令、データ移動命令、分岐命令、アドレッシング、サブルーチン、RISCとCISC

4. パイプライン処理 (1)

パイプラインの原理、命令パイプライン、オーバヘッド、構造ハザード、データハザード、制御ハザード

5. パイプライン処理 (2)

フォワードリング、遅延分岐、分岐予測、命令スケジューリング

6. キャッシュ

記憶階層と局所性、透過性、キャッシュ、ライトスルーとライトバック、ダイレクトマップ型、フルアソシアティブ型、セットアソシアティブ型、キャッシュミス

7. 仮想記憶

仮想記憶、ページフォールト、TLB、物理アドレスキャッシュ、仮想アドレスキャッシュ、メモリアクセス機構

講義の概要と予定 (2 / 2)

8. 基本CPUの設計

デジタル回路の入力、Verilog HDL、シミュレーションによる動作検証、アセンブラ、基本プロセッサの設計、基本プロセッサのシミュレーションによる検証

9. 命令レベル並列処理(1)

並列処理、並列処理パイプライン、VLIW、スーパースカラ、並列処理とハザード

10. 命令レベル並列処理(2)

静的最適化、ループアンローリング、ソフトウェアパイプラインニング、トレーススケジューリング

11. アウトオブオーダー処理

インオーダーとアウトオブオーダー、フロー依存、逆依存、出力依存、命令ウィンドウ、リザベーションステーション、レジスタリネーミング、マッピングテーブル、リオーダーバッファ、プロセッサの性能

12. 入出力と周辺装置

周辺装置、ディスプレイ、二次記憶装置、ハードウェアインタフェース、割り込みとポーリング、アービタ、DMA、例外処理

試験： 7月後半

10. 命令レベル並列処理（2）

■ 内容

– 静的最適化

- ・ 機械語プログラムと命令間依存性
- ・ ループアンローリング
- ・ ソフトウェアパイプライニング
- ・ トレーススケジューリング

機械語プログラムと命令間依存性

- 静的最適化 = 機械語プログラムの最適化
- ハザードを減らすための静的最適化
 - (1) 依存関係を解消したり減らしたりする
 - (2) 依存関係のある命令どうしをプログラムの中で離れた位置に置く

ループアンローリング

- ループアンローリング

- 小さなループを何周かまとめて一つのループとすることで、分岐命令によるハザードをなくす手法

例. 配列要素に定数を加えるプログラム

```
for (i=0; i<100; i++) a[i] = a[i] + 5;
```

図 6.5 配列要素に定数を加えるプログラム

```

addi r1, r0, 0    ; r1 を i を入れるレジスタとし、初期値 0 をセット
addi r2, r0, 100  ; r2 に 100 をセット
ForLoop: lw r4, 0(r3) ; r4 = a[i]; r3 は a[i] の番地を入れるレジスタとする
addi r4, 5, r4    ; r4 = r4 + 5;
sw r4, 0(r3)      ; a[i] = r4;
addi r1, r1, 1    ; i++;
addi r3, r3, 4    ; a[i] 番地の更新
blt r1, r2, ForLoop ; if (i < 100) goto ForLoop
    
```

図 6.6 配列要素に定数を加えるプログラム (アセンブリ言語)

ForLoop: lw r4, 0(r3)

addi r4, 5, r4

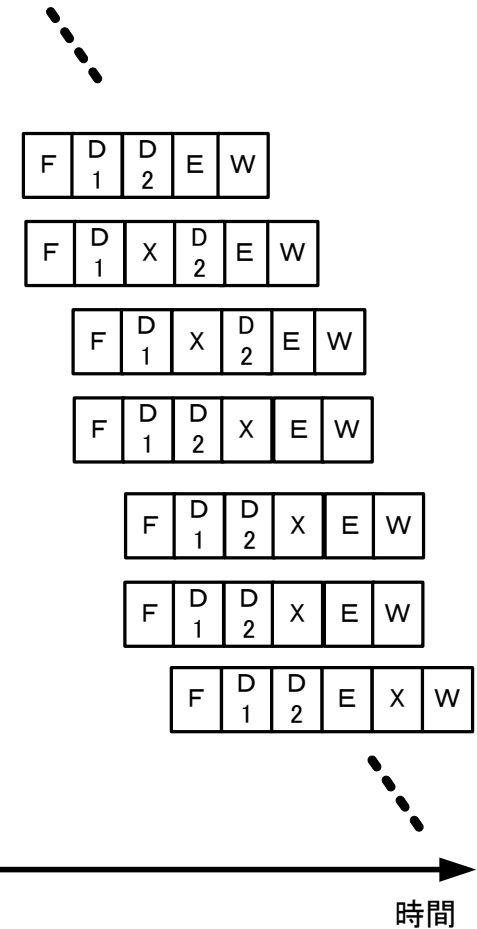
sw r4, 0(r3)

addi r1, r1, 1

addi r3, r3, 4

blt r1, r2, ForLoop

ForLoop: lw r4, 0(r3)



メモ: 正確にはFD1レジスタ、D1D2レジスタ、D2Eレジスタ、EWレジスタの構造ハザードも配慮すべき

- 上記を考慮すると、下記のようなタイミングになる
=>なぜこうなるか考えてみてください

lw r4, 0(r3)	F	D1	D2	E	W			
addi r4,5,r4	F	D1	D2	D2	E	W		
sw r4, 0(r3)		F	D1	D2	D2	E	W	
addi r1,r1,1		F	D1	D1	D2	E	W	
addi r3,r3,4			F	D1	D1	D2	E	W
blt r1,r2, Forloop				F	D1	D2	E	W

@Toshiyuki Nakata

回答

lw r4, 0(r3)	F	D1	D2	E	W					
addi r4,5,r4	F	D1	D2	D2	E	W				r4レジスタの値 E=>Eでフォワーディング
sw r4, 0(r3)		F	D1	D2	D2	E	W			r4レジスタの値 E=>Eでフォワーディング
addi r1,r1,1		F	D1	D1	D2	E	W			D1D2レジスタの値
addi r3,r3,4			F	D1	D1	D2	E	W		D1D2レジスタの値
blt r1,r2, Forloop				F	D1	D2	E	W		FD1レジスタの値

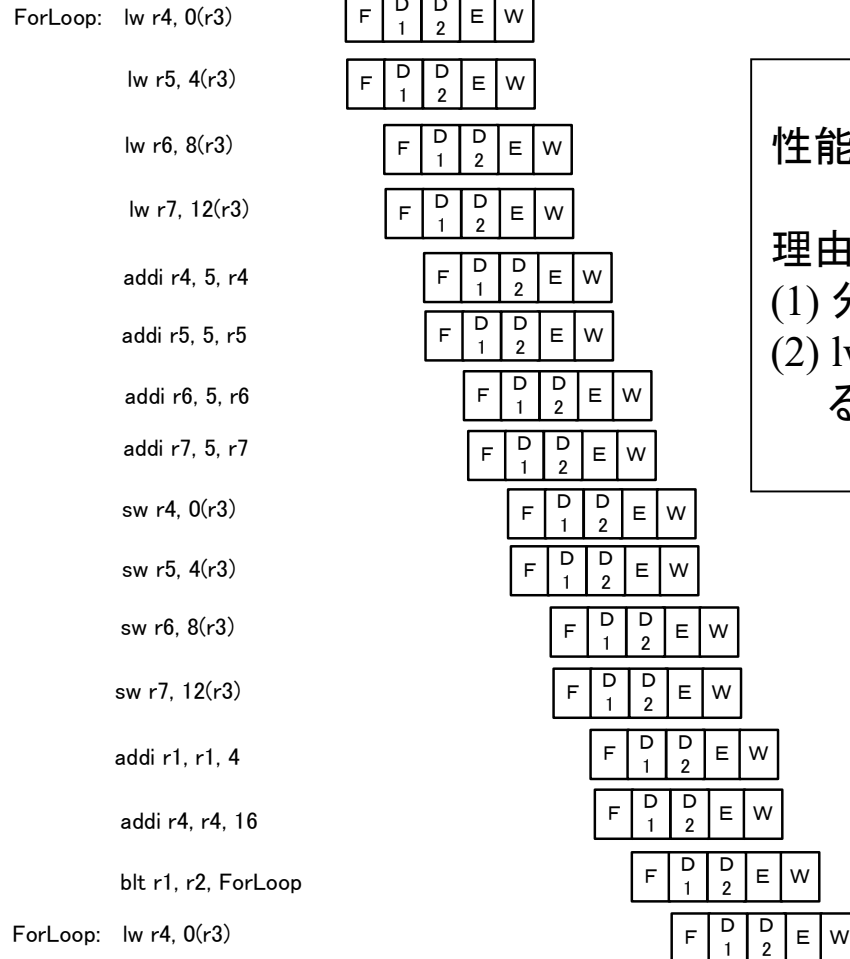
@Toshiyuki Nakata

ループアンローリングを施したプログラム

```
addi r1, r0, 0      ; r1 を  $i$  を入れるレジスタとし、初期値 0 をセット
addi r2, r0, 100    ; r2 に 100 をセット
ForLoop: lw r4, 0(r3) ;  $r4 = a[i]$ ; r3 は  $a[i]$  の番地を入れるレジスタとする
               lw r5, 4(r3) ;  $r5 = a[i+1]$ ;
               lw r6, 8(r3) ;  $r6 = a[i+2]$ ;
               lw r7, 12(r3) ;  $r7 = a[i+3]$ ;
               addi r4, 5, r4 ;  $r4 = r4 + 5$ ;
               addi r5, 5, r5 ;  $r5 = r5 + 5$ ;
               addi r6, 5, r6 ;  $r6 = r6 + 5$ ;
               addi r7, 5, r7 ;  $r7 = r7 + 5$ ;
               sw r4, 0(r3)   ;  $a[i] = r4$ ;
               sw r5, 4(r3)   ;  $a[i+1] = r5$ ;
               sw r6, 8(r3)   ;  $a[i+2] = r6$ ;
               sw r7, 12(r3)  ;  $a[i+3] = r7$ ;
               addi r1, r1, 4  ;  $i=i+4$ ;
               addi r3, r3, 16 ;  $b[i]$  番地の更新
               blt r1, r2, ForLoop ; if ( $i < 100$ ) goto ForLoop
```

図 6.8 ループアンローリングを施したプログラム（アセンブリ言語）

アンローリング後のパイプライン



性能は **2倍** になる

理由

- (1) 分岐命令の削減(bltが3つ減った)
- (2) lw, addi, swをそれぞれ4つずつまとめることによるデータハザードの解消

ソフトウェアパイプラインを施したプログラムの パイプライン実行

ForLoop: sw r5, 0(r3)

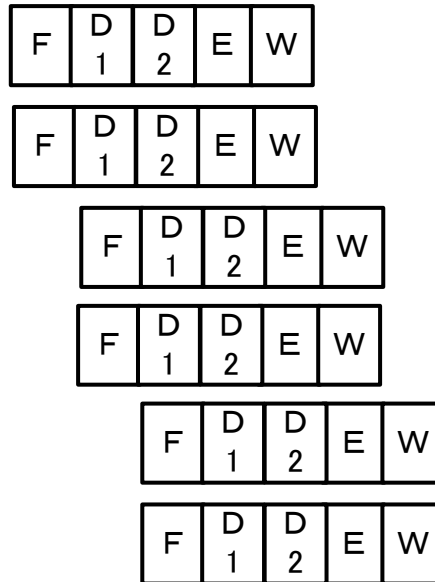
addi r5, r4, 5

lw r4, 8(r3)

addi r1, r1, 1

addi r3, r3, 4

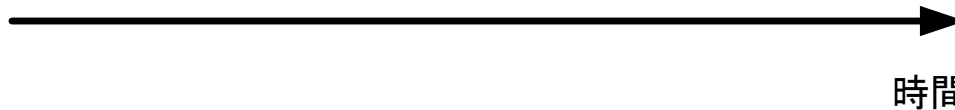
blt r1, r2, ForLoop



性能は **1.33** 倍になる

理由

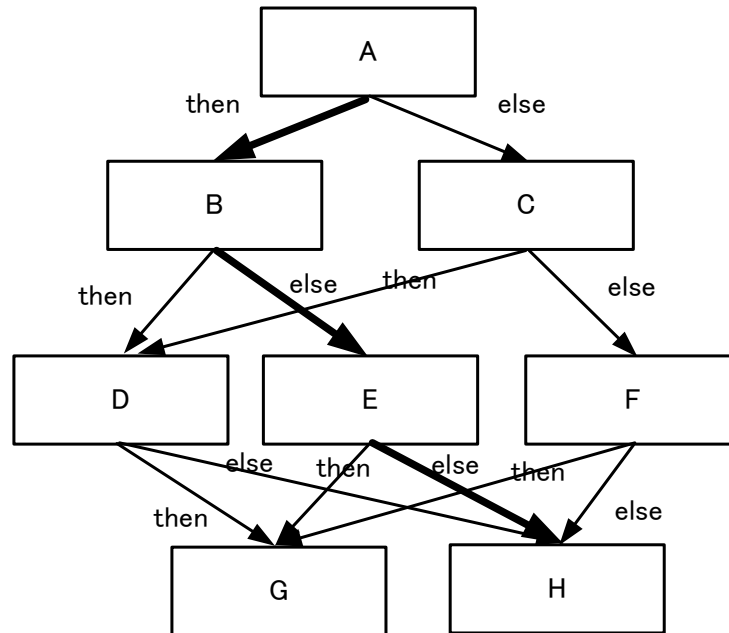
lw, swの距離を離すことによる
データハザードの解消



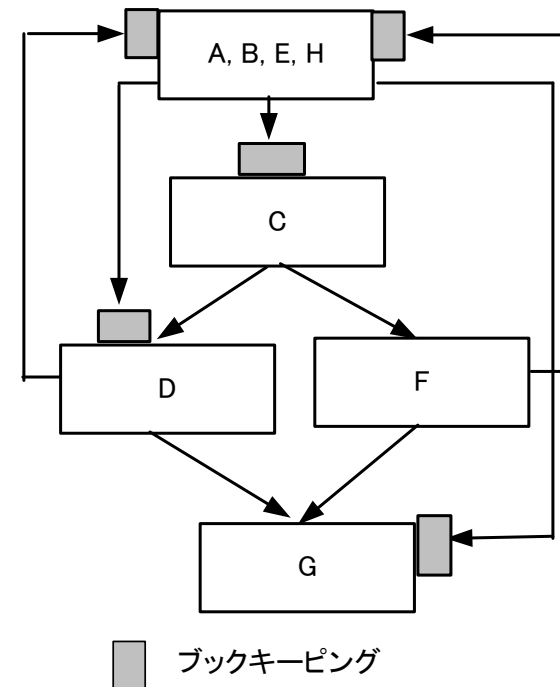
トレーススケジューリング

■ トレーススケジューリング

- 分岐予測をして複数の基本ブロックを統合し、制御依存を減らす手法
 - ・ 基本ブロック: ある分岐命令の直後から次の分岐命令までの命令列
 - ・ 制御フローグラフ: プログラムを基本ブロックの間の依存関係として表現したもの



太線が、最も確率の高い分岐パス



■ ブックキーピング

トレーススケジューリングの手順

- (1) 実行履歴(プロファイル、profile)などによって、実行される確率の最も高い分岐パターンを調べる。
- (2) (1)のパターンの上にある基本ブロックを統合する。これをトレース(trace)と呼ぶ。
- (3) トレースに命令スケジューリングを施すことで、トレース内の実行効率を高める。トレース内にも分岐命令はあるが、「分岐命令を超えた命令移動」も行う。
- (4) (3)によってトレース以外に分岐する場合に生じる不都合を防ぐため、分岐先の入り口に補正用のコードを入れる。これをブックキーピング(bookkeeping)と呼ぶ。
- (5) トレースを除いた制御フローグラフにおいて、(1)~(4)を繰り返す。