工学部講義

コンピュータハードウェア（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 10:15 - 11:45、工3－31
- 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）

  教科書通りやります
- 参考書
  - D. Patterson and J. Hennessy, Computer Organization & Design、2nd Ed.（邦訳『コンピュータの構成と設計』(第２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（＋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
　ディジタルな表現、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）
　フォワーディング、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

講義の概要と予定（2／2）

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
8．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、スーパスカラ、並列処理とハザード
9．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ニング、トレーススケジューリング
10．アウトオブオーダ処理
　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、リオーダバッファ、プロセッサの性能
11．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

９．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

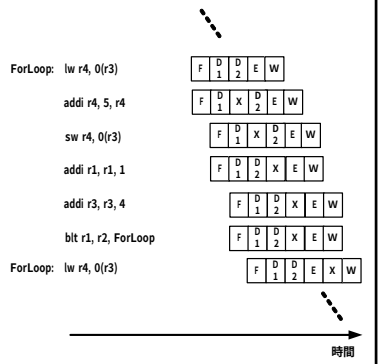
例．配列要素に定数を加えるプログラム

```
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）

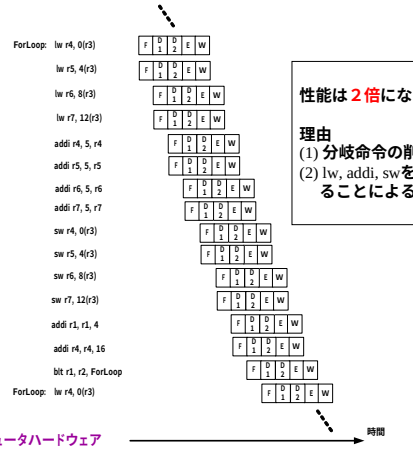


ループアンローリングを施したプログラム

```
            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

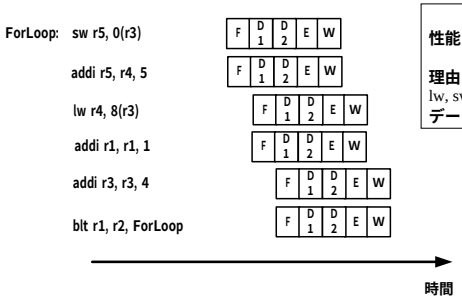
ソフトウェアパイプライニング

- ソフトウェアパイプライニング
  - ループ間にまたがって命令を移動し、依存関係のある命令どうしの距離を離すことで、ハザードを起こりにくくし、並列度をあげる技術
  - ソフトウェアパイプライニングとループアンローリングは組み合わせて用いることができる

```
            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
            lw r4, 0(r3)        ;  r4 = a[0] ; r3 は a[i] の番地を入れるレジスタとする
            addi r5, 5, r4      ;  r5 = r4 + 5;
            lw r4, 4(r3)        ;  r4 = a[1] ;
ForLoop:    sw r5 0(r3) ;       ;  a[i]  = r5;
            addi r5, r4, 5      ;  r5 = a[i+1]  + 5;
            lw r4, 8(r3)        ;  r4 = a[i+2] ;
            addi r1, r1, 1      ;  r1 = i + 1;
            addi r3, r3, 4      ;  r3 に a[i+1] 番地を入れる
            blt r1, r2, ForLoop ;  if (i < 100) goto ForLoop
```

図 6.10 ソフトウェアパイプライニングを施したプログラム（アセンブリ言語）

ソフトウェアパイプライニングを施したプログラムのパイプライン実行

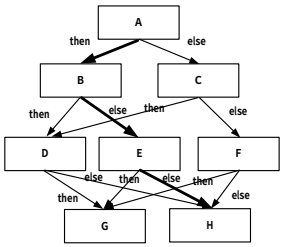


性能は１．３３倍になる

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

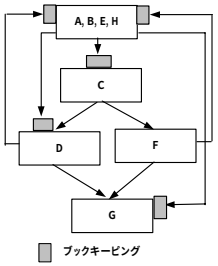
トレーススケジューリング

- 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

(a) 制御フローグラフ



ブックキーピング

(b) トレーススケジューリング(第一段階)

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。