

計算機アーキテクチャ 第二 (O)

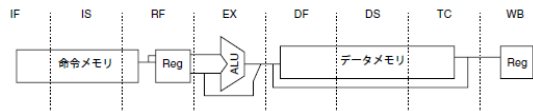
8. パイプライン制御とスーパーパイプライン

大学院情報理工学専攻 計算工学専攻
吉瀬謙二 kise_at_cs.titech.ac.jp
S321講義室 月曜日 5, 6時限 13:20-14:50

1

補足: MIPS R4000 パイプライン

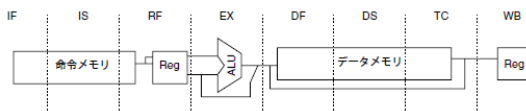
- 64ビットのマイクロプロセッサ
- 整数パイプラインを8段とすることで、クロック周波数を向上させる。
 - 100MHz / 150MHz (1991/10)
- キャッシュアクセスの時間が厳しいため、ここに追加のパイプラインを割り当てる。
- 深いパイプラインは、**スーパーパイプライン**とよばれることがある。



Adapted from Computer Organization and Design, Patterson & Hennessy, © 2005.

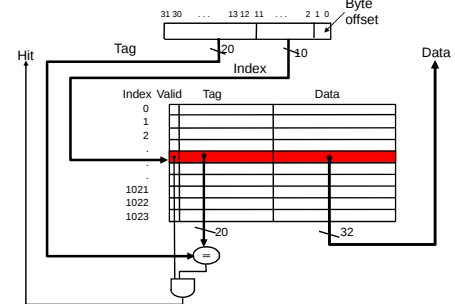
R4000のステージ構成

- IF: 命令フェッチの前半, PC の選択を行い、命令キャッシュアクセスを開始
- IS: 命令フェッチの後半, 命令キャッシュアクセスを完了
- RF: 命令デコードとレジスタファイルアクセス、ハザードチェック、そして命令キャッシュのヒット検出
- EX: 実行
- DF: データフェッチ、データキャッシュアクセスの前半
- DS: データフェッチの後半、データキャッシュアクセスの完了
- TC: タグのチェック、データキャッシュアクセスヒットの決定
- WB: ロード演算とレジスタ-レジスタ間演算の結果をレジスタファイルに格納



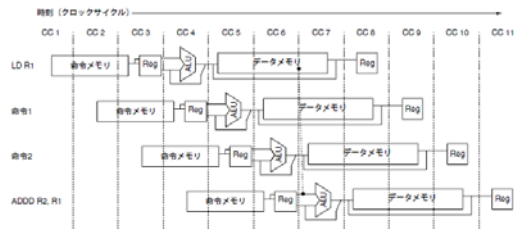
Adapted from Computer Organization and Design, Patterson & Hennessy, © 2005.

MIPS Direct Mapped Cache Example



Adapted from Computer Organization and Design, Patterson & Hennessy, © 2005.

MIPS R4000: 2サイクルのロード遅延



- データ値はDSステージで利用可能
- TCステージでのタグ比較によりミスが判明すると、1サイクル後退する。

Adapted from Computer Organization and Design, Patterson & Hennessy, © 2005.

計算機アーキテクチャ 第二 (O)

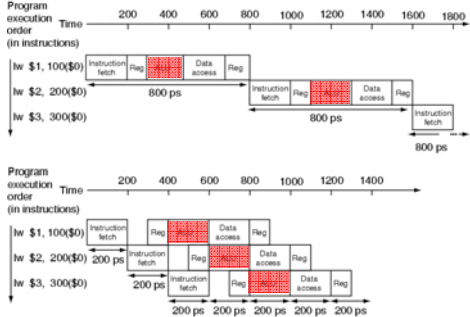
スーパースカラプロセッサ(1) 分岐予測

大学院情報理工学専攻 計算工学専攻
吉瀬謙二 kise_at_cs.titech.ac.jp
S321講義室 月曜日 5, 6時限 13:20-14:50

6

パイプライン処理 (pipelining)とスカルプロセッサ

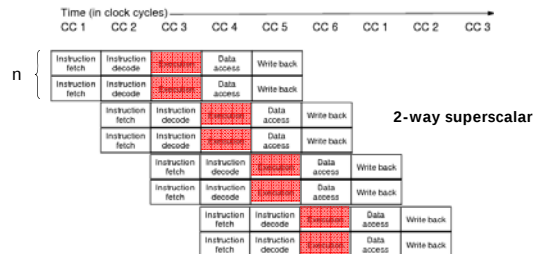
サイクル当たりの平均実行命令数, IPC (instructions per cycle) の上限は1



Adapted from Computer Organization and Design, Patterson & Hennessy, © 2005

スーパースカルプロセッサと命令レベル並列性

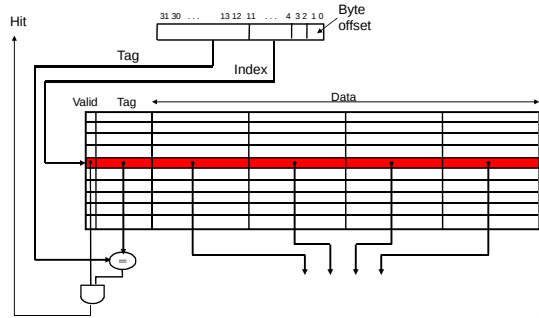
- 複数のパイプラインを利用して IPC (instructions per cycle) を1以上に引き上げる, 複数の命令を並列に実行
 - n-way スーパースカル
- ハザードの積極的な解消, ストールの隠蔽が重要



Adapted from Computer Organization and Design, Patterson & Hennessy, © 2005

命令キャッシュの実装

- ラインサイズ 4ワード (16 Byte)



Adapted from Computer Organization and Design, Patterson & Hennessy, © 2005

9

命令キャッシュの実装

```

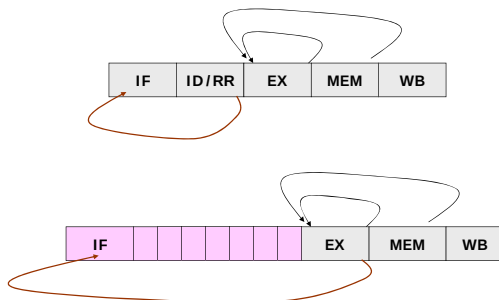
struct icache_line {
    int valid;
    int tag;
    int data[4];
} iline;

int icache::fetch(int pc, int *ir) {
    int index = (pc >> 4) % size;
    int tag = (pc >> 4);
    if (buf[index].valid && buf[index].tag == tag) { /* hit */
        for (int i=0; i<4; i++) ir[i] = buf[index].data[i];
        return 1;
    } else { /* cache miss */
        buf[index].valid = 1;
        buf[index].tag = tag;
        for (int i=0; i<4; i++) {
            int ir_t;
            mem->ld_4byte(pc+4*i, &ir_t);
            buf[index].data[i] = ir_t;
        }
        return 0;
    }
}
    
```

Adapted from Computer Organization and Design, Patterson & Hennessy, © 2005

10

パイプラインと制約ループ



Adapted from Computer Organization and Design, Patterson & Hennessy, © 2005

11

高いバンド幅の命令フェッチ

- パイプラインにバブルを生じさせないためには, 条件分岐命令をフェッチした時に, 次の3つを予測しなければならない.
 - フェッチしている命令が分岐かどうか
 - 分岐方向
 - 分岐先アドレス

Adapted from Computer Organization and Design, Patterson & Hennessy, © 2005

12

分岐方向の予測(分岐予測)

The diagram illustrates the inputs to a branch predictor. On the left, two text labels are connected by green arrows to a central light green box labeled "分岐予測" (Branch Prediction). The top label is "プログラムカウンタ" (Program Counter), and the bottom label is "分岐履歴などの情報" (Information such as branch history). A green arrow points from the "分岐予測" box to the text "分岐方向 (成立／不成立)" (Branch Direction (Success / Failure)) on the right.

13

Adapted from *Computer Organization and Design*, Patterson & Hennessy, © 2005

13

投機処理と分岐方向の予測 (分岐予測)

制御依存関係

大雑把には5回に1回程度、分岐命令があらわれる。

分岐命令

処理すべき機械命令の列

Processor

Start End

制御依存が確定した命令列

処理すべきかわかっていない命令列

Processor

Start End

制御依存が確定した命令列

予測が正しいとして投機的に処理している命令列

Adapted from Computer Organization and Design, Patterson & Hennessy, © 2005

Adapted from *Computer Organization and Design*, Patterson & Hennessy, © 200

19

サンプルプログラム Vector Add

```
#define VSIZE 4
void vadd(long *A, long *B, long *C) {
    for(i=0; i<VSIZE; i++)
        C[i] += (A[i] + B[i]);
}
```

```

graph TD
    B1["B1  
i = 0"] --> B2["B2  
*C = *C + (*A + *B)  
i++  
A++  
B++  
C++  
i < 4"]
    B2 -- False --> Return["return"]
    B2 -- True --> B2

```

制御フローグラフ B3

Adapted from *Computer Organization and Design*, Patterson & Hennessy, © 2005

15

Adapted from *Computer Organization and Design*, Patterson & Hennessy, © 2005

15

シンプルな分岐予測 Branch Always

Taken(1),
Not Taken(0)

Taken (1) Taken (1) Taken (1) Not Taken (0)

処理すべき機械命令の列

- **Branch Always**: 常に分岐が成立すると予測する。
 - 上の例では、予測成功率は75%、ミス率25%
 - 予測のためのメモリを必要としない。
 - 予測とよぶほどのものではない。

```
graph TD
    B1[B1  
i = 0] --> B2[B2  
*C = *C + (*A + *B)  
i++  
A++  
B++  
C++  
i < 4]
    B2 -- True, taken --> B3[B3  
return]
    B2 -- False, not taken --> B3
```

Adapted from *Computer Organization and Design*, Patterson & Hennessy, © 2005

16

Adapted from *Computer Organization and Design*, Patterson & Hennessy, © 200

16

シンプルな分岐予測 2ビットカウンタ方式

トレースデータ
(分岐アドレス, 分岐結果)

0040d6c5	1	0040d89c	1
0040d6b8	1	0040d89c	1
0040d6bc	0	0040d89c	1
0040d6c5	0	0040d89b	1
0040d6df	0	0040d89c	1
0040d71e	0	0040d89c	1
0040d736	0	0040d89c	0
0040d7ab	0	0040d8a2	0
0040d7cd	0	0040d8c0	1
0040d7f9	0	0040d8c4	0
0040d81e	1	0040d8cd	1
0040d7f9	1	0040d8c0	0
0040d81e	1	0040d8c4	1
0040d7f9	0	0040d8c0	0
0040d81e	0	0040d8c4	0
0040d83d	0	0040d8c0	0
0040d86d	1	0040d8c0	0
0040d86d	1	0040d8e7	0
0040d86d	1	0040d923	1
0040d86d	1	0040d7ab	0
0040d86d	1	0040d7cd	0
0040d86d	1	0040d7f9	0

■ 2ビットカウンタ方式

- 大域的な偏り, 局所性の利用
- 2ビットカウンタの状態に応じて予測
- 予測のためのメモリは2ビット
- 状態の更新

2 bit

Prediction

Strongly Taken (11)

Weakly Taken (10)

Weakly Untaken (01)

Strongly Untaken (00)

Taken

Untaken

Adapted from *Computer Organization and Design*, Patterson & Hennessy, © 2005

サンプルプログラム Vector Add

Taken(1),
Not Taken(0)

```
#define VSIZE 4
void vadd(long *A, long *B, long *C){
  for(i=0; i<VSIZE; i++) {
    if(A[i]<0) error_routine();
    C[i] += (A[i] + B[i]);
  }
}
```

制御フローグラフ

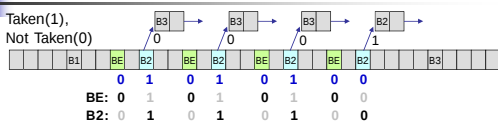
```
graph TD
    B1[B1: i = 0] --> BE[BE: Error check]
    BE -- True --> BE
    BE -- False --> B2[B2: *C = *C + (*A + *B)]
    B2 -- True --> BE
    B2 -- False --> B3[B3: return]
```

18

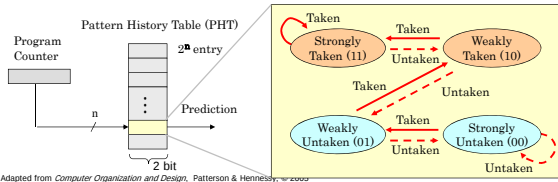
Adapted from *Computer Organization and Design*, Patterson & Hennessy, © 200

18

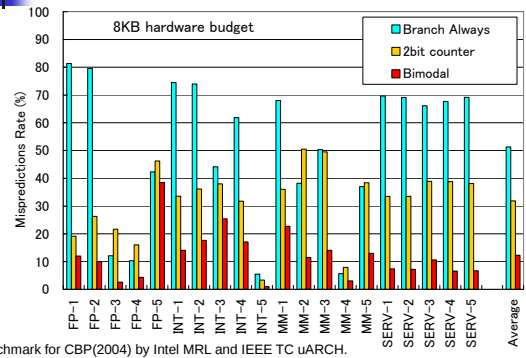
Bimodal (ISCA 1981)



- 分岐アドレス(プログラムカウンタ)毎に履歴を切り替える
- 分岐アドレスによりパターン履歴表(PHT)のインデックスを作成
- パターン履歴表は2ビットカウンタの配列。

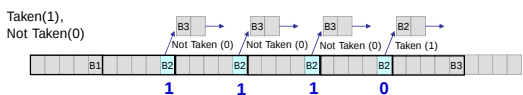


シンプルな分岐予測の予測精度(予測ミス率)



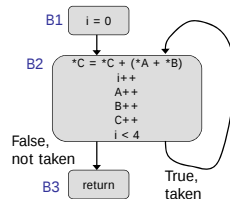
Adapted from Computer Organization and Design, Patterson & Hennessy, © 2005

分岐履歴



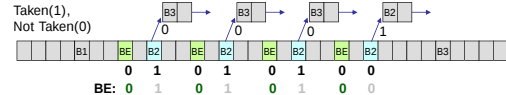
B2の分岐履歴

1110 ?
11101 ?
111011 ?
1110111 ?
11101110 ?
111011100 ?



Adapted from Computer Organization and Design, Patterson & Hennessy, © 2005

ローカル, グローバル分岐履歴



BEの分岐履歴

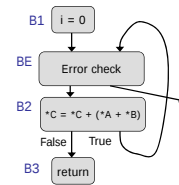
0000 ?
00000 ?
000000 ?
0000000 ?
00000000 ?

B2の分岐履歴

1110 ?
11101 ?
111011 ?
1110111 ?
11101110 ?
111011100 ?

B2とBEの分岐履歴

010101000 ?

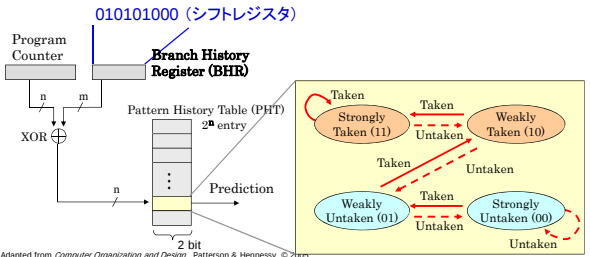


ローカル分岐履歴 ローカル分岐履歴 グローバル分岐履歴

Adapted from Computer Organization and Design, Patterson & Hennessy, © 2005

Gshare (TR-DEC 1993)

- グローバル分岐履歴と分岐アドレスとの排他的論理和によりパターン履歴表へのインデックスを作成
- パターン履歴表は2ビット飽和型カウンタの配列で、選択された2ビットカウンタの値により分岐方向を予測(bimodalと同じ)
- 分岐結果を用いて、予測に利用したカウンタを更新



Gshareの実装

```
class Gshare {
    int bhr;
    int *buf;
public:
    int size;
    Gshare(int):
    int predict(int);
    void update(int, int);
};

Gshare::Gshare(int bpred_size) {
    size = bpred_size;
    buf = (data_t *)calloc(size, sizeof(data_t));
    for (int i=0; i<size; i++) buf[i] = 2;
}

int Gshare::predict(int pc) {
    // ...
}

void Gshare::update(int pc, int taken) {
    // ...
}
```

Adapted from Computer Organization and Design, Patterson & Hennessy, © 2005

Exercise

gshare の実装



Adapted from Computer Organization and Design, Patterson & Hennessy, © 2005

25

Gshareの実装

```
class Gshare {
public:
    int bhr;
    int *buf;
    int size;
    Gshare(int):
    {
        predict(int);
        update(int, int);
    }

    Gshare::Gshare(int bpred_size) {
        size = bpred_size;
        buf = (data_t *)calloc(size, sizeof(data_t));
        for(int i=0; i<size; i++) buf[i] = 2;
    }

    int Gshare::predict(int pc) {
        int index = ((pc >> 2) ^ bhr) % size;
        return (buf[index]>1);
    }

    void Gshare::update(int pc, int taken) {
        // ...
    }
}
```

Adapted from Computer Organization and Design, Patterson & Hennessy, © 2005

26

Gshareの実装

```
class Gshare {
public:
    int bhr;
    int *buf;
    int size;
    Gshare(int):
    {
        predict(int);
        update(int, int);
    }

    Gshare::Gshare(int bpred_size) {
        size = bpred_size;
        buf = (data_t *)calloc(size, sizeof(data_t));
        for(int i=0; i<size; i++) buf[i] = 2;
    }

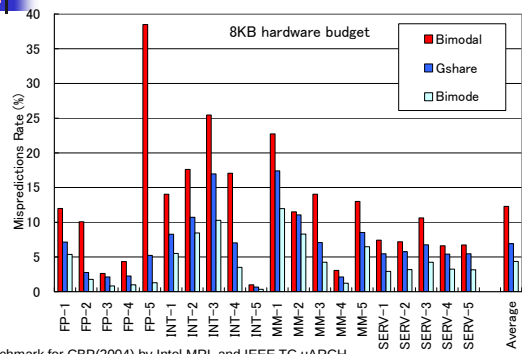
    int Gshare::predict(int pc) {
        int index = ((pc >> 2) ^ bhr) % size;
        return (buf[index]>1);
    }

    void Gshare::update(int pc, int taken) {
        int index = ((pc >> 2) ^ bhr) % size;
        if(taken!=0 && buf[index]<3) buf[index]++;
        if(taken==0 && buf[index]>0) buf[index]--;
        bhr = (bhr << 1) | taken;
    }
}
```

Adapted from Computer Organization and Design, Patterson & Hennessy, © 2005

27

予測精度(予測ミス率)



Benchmark for CBP(2004) by Intel MRL and IEEE TC uARCH.

Adapted from Computer Organization and Design, Patterson & Hennessy, © 2005

28

アナウンス

- 講義スライド, 講義スケジュール
 - www.arch.cs.titech.ac.jp

Adapted from Computer Organization and Design, Patterson & Hennessy, © 2005

29