

計算機アーキテクチャ 第二 (O)

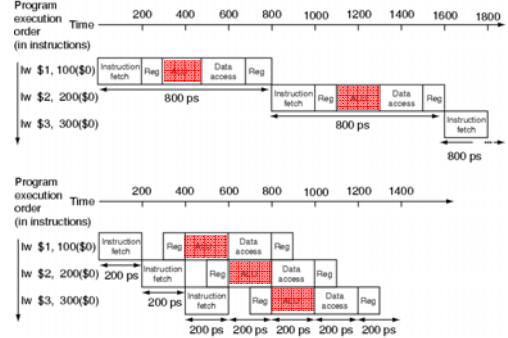
8. スーパースカラプロセッサ(1)

大学院情報理工学研究科 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

パイプライン処理 (pipelining)とスカラプロセッサ

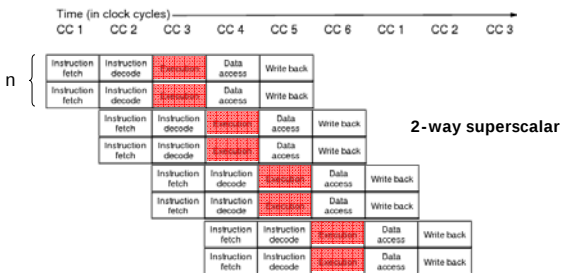
サイクル当たりの平均実行命令数, IPC (instructions per cycle) の上限は1



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

スーパースカラ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 スーパースカラ
- 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

ムーアの法則によるトランジスタ数の増加

ムーアの法則

チップで利用できるトランジスタの数は2年間で2倍に増加する。

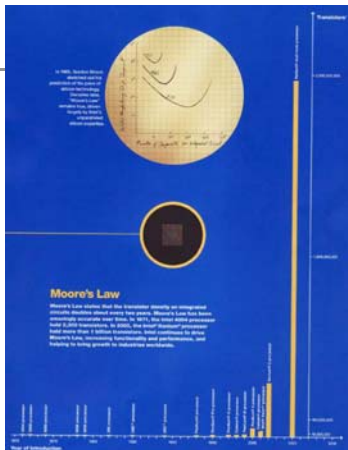


ムーアの法則に従ってトランジスタ数が増加してきた。今後も同様の増加が見込まれる。

出典: Intel社, <http://www.intel.com/research/silicon/mooreslaw.htm>

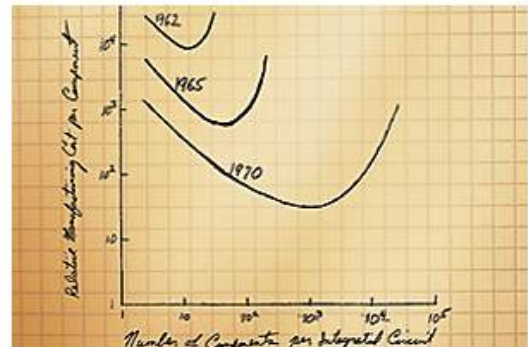
Adapted from Computer Organization and Design, Patterson & Hennessy, © 2005

Moore's Law



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

Moore's Law



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

Moore's Law

Moore's Law

Moore's Law states that the transistor density on integrated circuits doubles about every two years. Moore's Law has been amazingly accurate over time. In 1971, the Intel 4004 processor held 2,300 transistors. In 2005, the Intel® Itanium® processor held more than 1 billion transistors. Intel continues to drive Moore's Law, increasing functionality and performance, and helping to bring growth to industries worldwide.

The chart illustrates the exponential growth of transistor density over time, following Moore's Law. The x-axis represents the year of introduction, and the y-axis represents the number of transistors. Key milestones include the Intel 4004 (1971, 2,300 transistors), Intel 8080 (1974, 6,000 transistors), Intel 8085 (1976, 6,000 transistors), Intel 286 (1982, 2.75 million transistors), Intel 386 (1985, 2.75 million transistors), Intel 486 (1989, 12 million transistors), Pentium (1992, 3.1 million transistors), Pentium II (1997, 7.5 million transistors), Celeron (1998, 9.5 million transistors), Pentium III (1999, 9.5 million transistors), Pentium 4 processor (2000, 9.5 million transistors), Itanium processor (2001, 200 million transistors), Intel® Itanium® processor (2005, 1 billion transistors), and Intel® Itanium® processor (2016, 1 billion transistors). The chart also shows the doubling of transistor density every two years, with a vertical line at 2005 marking the 1 billion transistor milestone.

Year of Introduction	Processor	Transistor Count
1971	Intel 4004 processor	2,300
1974	Intel 8080 processor	6,000
1976	Intel 8085 processor	6,000
1982	Intel 286 processor	2.75 million
1985	Intel 386 processor	2.75 million
1989	Intel 486 processor	12 million
1992	Pentium processor	3.1 million
1997	Pentium II processor	7.5 million
1998	Celeron processor	9.5 million
1999	Pentium III processor	9.5 million
2000	Pentium 4 processor	9.5 million
2001	Itanium processor	200 million
2005	Intel® Itanium® processor	1 billion
2016	Intel® Itanium® processor	1 billion

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

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)

The diagram illustrates a 4-word (16-byte) command cache. It features a Hit signal, a 32-bit address (split into a 16-bit Tag and a 16-bit Index), and a cache array with 4 rows and 4 words per row. The selected row's Data is output, and its Valid bit is checked against the Hit signal. A 4-to-1 multiplexer selects the output data from the four rows.

8

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

命令キャッシュの実装

```

        icache:(int icache_size, main_memory *m){
            mem = m;
            size = icache_size;
            buf = (icache_line *)calloc(size, sizeof(iline));
        }

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

int icache::fetch(data_t pc, data_t *ir){
    int index = (pc >> 4) % size;
    data_t 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++){
            data_t ir_t;
            mem->ld_4byte(pc+4*i, &ir_t);
            buf[index].data[i] = ir[i] = ir_t;
        }
        return 0;
    }
}

```

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

9

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

パイプラインと制約ループ

The diagram illustrates two pipeline architectures. The top pipeline is a 5-stage pipeline with stages labeled IF, ID/RR, EX, MEM, and WB. It features a feedback loop from the EX stage back to the ID/RR stage, and a longer feedback loop from the WB stage back to the IF stage. The bottom pipeline is a 6-stage pipeline with a first stage labeled IF, followed by four unlabeled stages, and then EX, MEM, and WB stages. It also features a feedback loop from the EX stage back to the first stage, and a longer feedback loop from the WB stage back to the first stage.

10

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

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

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

11

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

The diagram illustrates the inputs and output of a branch prediction unit. A central light green box is labeled "分岐予測" (Branch Prediction). Two inputs on the left are "プログラムカウンタ" (Program Counter) and "分岐履歴などの情報" (Information such as branch history), both with arrows pointing to the box. An output arrow on the right points to "分岐方向 (成立／不成立)" (Branch direction (taken/not taken)).

分岐方向の予測(分岐予測)

プログラムカウンタ

分岐履歴などの情報

分岐予測

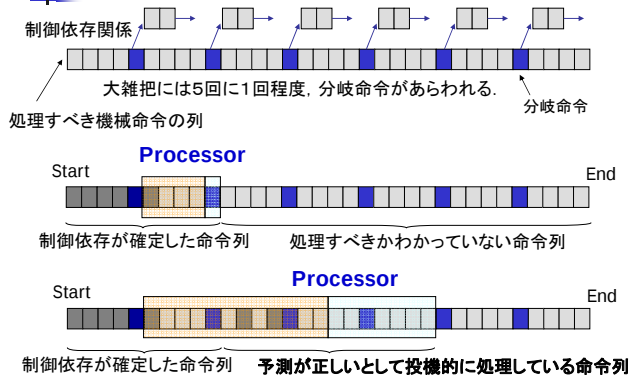
分岐方向
(成立／不成立)

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

12

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

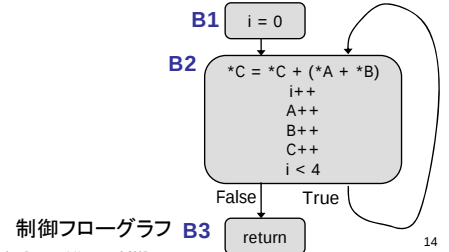
投機処理と分岐方向の予測(分岐予測)



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

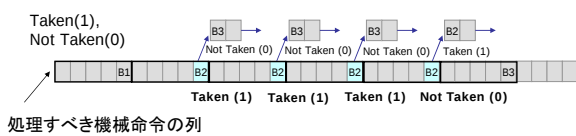
サンプルプログラム Vector Add

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



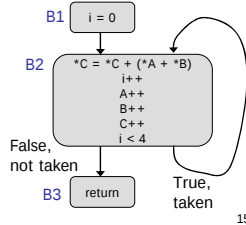
Adapted from Computer Organization and Design, Patterson & Hennessy, © 2005

シンプルな分岐予測 Branch Always



- 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。



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

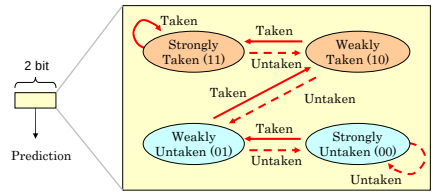
シンプルな分岐予測 2ビットカウンタ方式

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	0040d89c	1
0040d6df	0	0040d89c	1
0040d71f	0	0040d89c	1
0040d736	0	0040d89c	0
0040d7ab	0	0040d8a2	0
0040d7cd	0	0040d8c0	1
0040d7f9	0	0040d8c4	0
0040d81e	1	0040d8cd	1
0040d81e	1	0040d8c0	0
0040d81e	1	0040d8c4	1
0040d7f9	0	0040d8cd	1
0040d81e	0	0040d8c0	1
0040d83d	0	0040d8c4	0
0040d86d	1	0040d8cd	0
0040d86d	1	0040d8e7	0
0040d86d	1	0040d923	1
0040d86d	1	0040d7ab	0
0040d86d	1	0040d7cd	0
0040d86d	1	0040d7f9	0

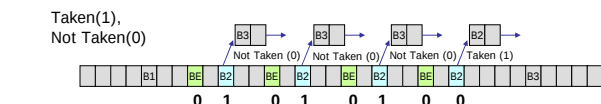
2ビットカウンタ方式

- 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



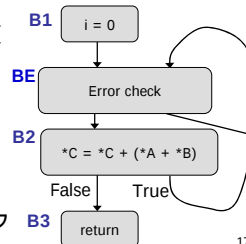
Adapted from Computer Organization and Design, Patterson & Hennessy, © 2005

サンプルプログラム Vector Add



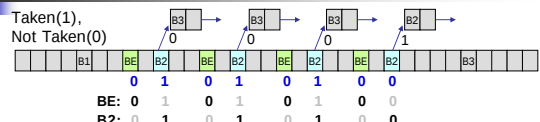
```
#define VSIZE 4
void vadd(long *A, long *B, long *C){
    for(i=0; i<VSIZE; i++){
        if(A[i]<0) error_routine();
        C[i] += (A[i] + B[i]);
    }
}
```

制御フローグラフ

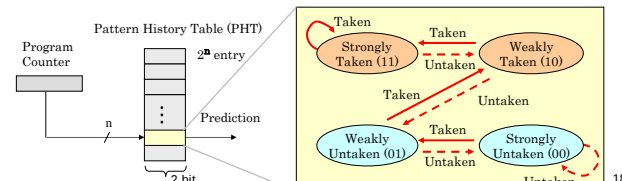


Adapted from Computer Organization and Design, Patterson & Hennessy, © 2005

Bimodal (ISCA 1981)

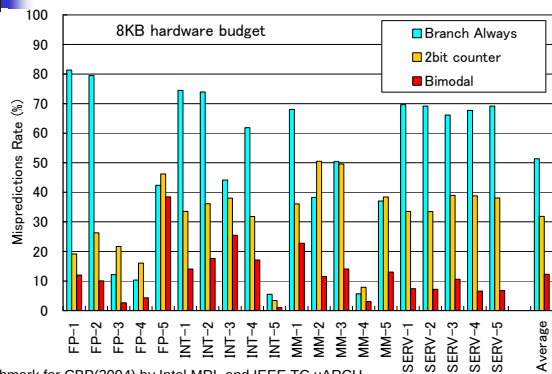


- 分岐アドレス(プログラムカウンタ)毎に履歴を切り替える
- 分岐アドレスによりパターン履歴表(PHT)のインデックスを作成
- パターン履歴表は2ビットカウンタの配列。



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

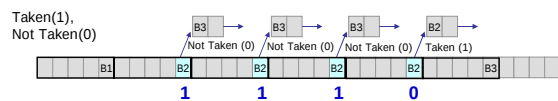
シンプルな分岐予測の予測精度(予測ミス率)



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

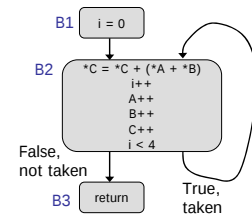
19

分岐履歴



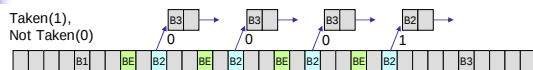
B2の分岐履歴

1110 ?
11101 ?
111011 ?
1110111 ?
11101110 ?



20

ローカル, グローバル分岐履歴



BEの分岐履歴

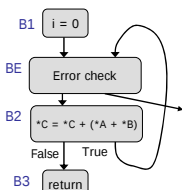
0000 ?
00000 ?
000000 ?
0000000 ?
00000000 ?

B2の分岐履歴

1110 ?
11101 ?
111011 ?
1110111 ?
11101110 ?

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

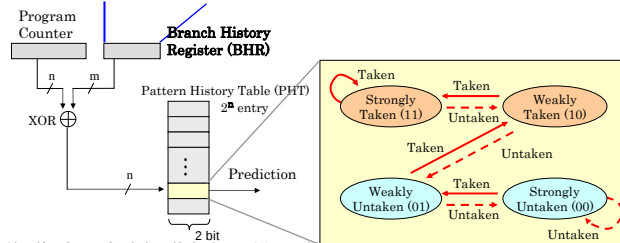
21

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

010101000 (シフトレジスタ)



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

Gshareの実装

```
class Gshare {
    data_t bhr;
    data_t *buf;
public:
    int size;
    Gshare(int);
    int predict(data_t);
    void update(data_t, 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(data_t pc) {
    // Prediction logic
}

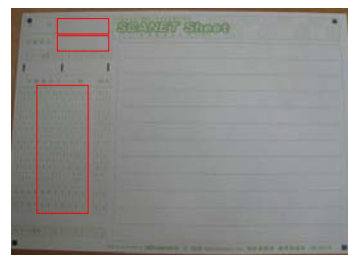
void Gshare::update(data_t pc, int taken) {
    // Update logic
}
```

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

23

Exercise

gshare の実装



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

24

Gshareの実装

```

class Gshare {
    data_t bhr;
    data_t *buf;
public:
    int size;
    Gshare(int);
    int predict(data_t);
    void update(data_t, 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(data_t pc) {
    int index = ((pc >> 2) ^ bhr) % size;
    return (buf[index]>1);
}

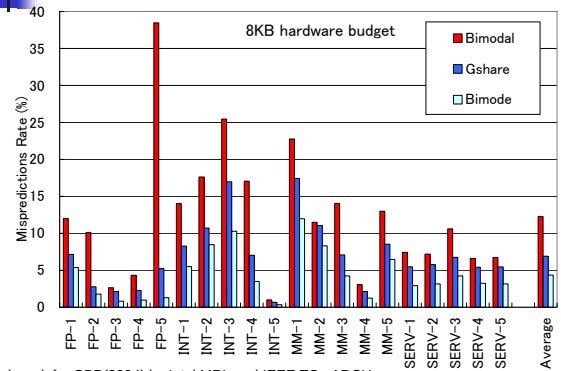
void Gshare::update(data_t pc, int taken) {
    int index = ((pc >> 2) ^ bhr) % size;
    if (taken!=0 && buf[index]<3) buf[index]++;
    if (taken==0 && buf[index]>0) buf[index]--;
    bhr = (bhr << 1) | taken;
}

```

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

25

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

26

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
- 2010年12月6日(月) 休講です。

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

27