2010年 後学期

# 計算機アーキテクチャ 第二 (O)

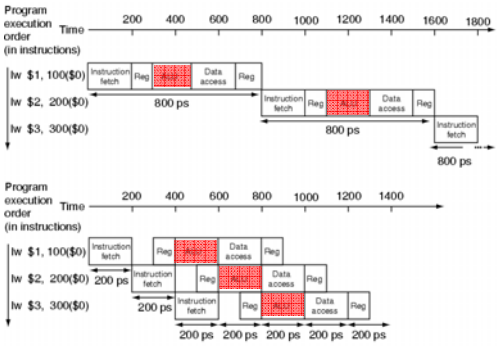
## 8. スーパースカラプロセッサ(1)

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



## パイプライン処理 (pipelining)とスカラプロセッサ

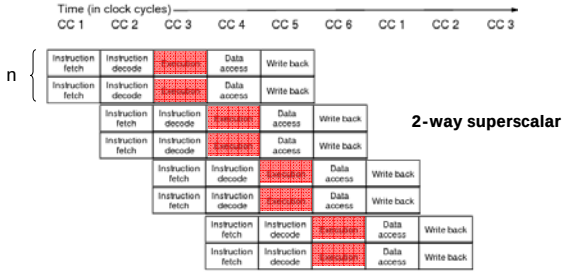
サイクル当たりの平均実行命令数，**IPC (instructions per cycle)** の上限は1




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，<u>複数の命令を並列に実行</u>
  - 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



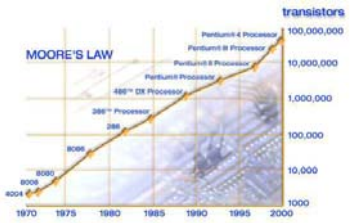

**2-way superscalar**

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。

| プロセッサ | 出荷年 | トランジスタ数 |
|---|---|---|
| 4004 | 1971 | 2,250 |
| 8008 | 1972 | 2,500 |
| 8080 | 1974 | 5,000 |
| 8086 | 1978 | 29,000 |
| 286 | 1982 | 120,000 |
| 386™ processor | 1985 | 275,000 |
| 486™ DX processor | 1989 | 1,180,000 |
| Pentium® processor | 1993 | 3,100,000 |
| Pentium II processor | 1997 | 7,500,000 |
| Pentium III processor | 1999 | 24,000,000 |
| Pentium 4 processor | 2000 | 42,000,000 |

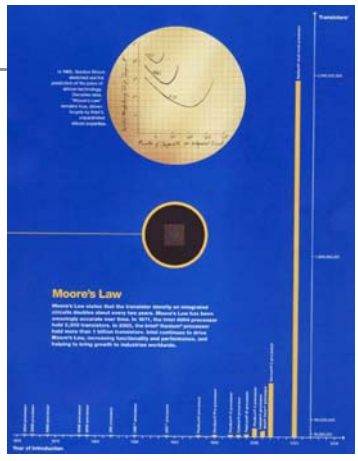



ムーアの法則に従ってトランジスタ数が増加してきた．今後も同様の増加が見込まれる．

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

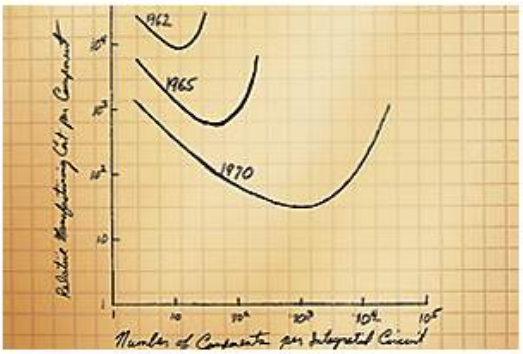
Adapted from *Computer Organization and Design*, Patterson & Hennessy, © 2005

## Moore's Law

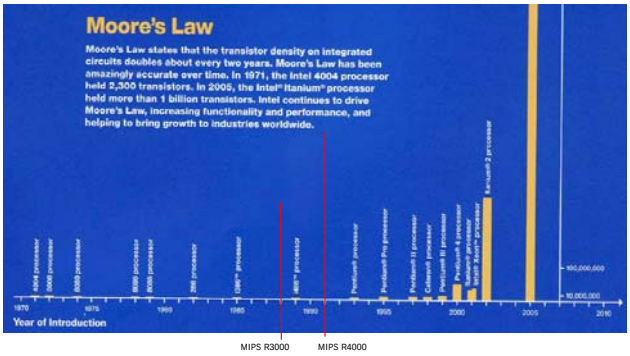

Adapted from *Computer Organization and Design*, Patterson & Hennessy, © 2005

## Moore's Law




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

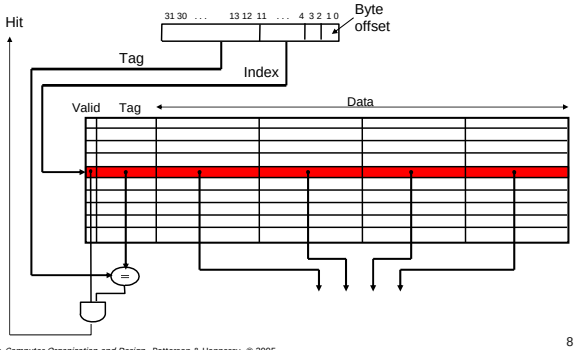
## Moore's Law






## 命令キャッシュの実装

- ラインサイズ 4ワード (16 Byte)





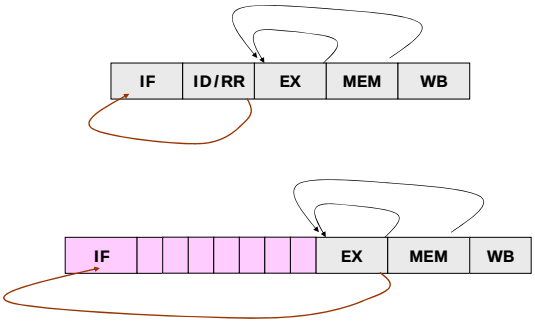

## 命令キャッシュの実装

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

class Icache {
    main_memory *mem;
    icache_line *buf;
public:
    int size;
    Icache(int, main_memory*);
    int fetch(data_t, data_t*);
};
```

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



## パイプラインと制約ループ



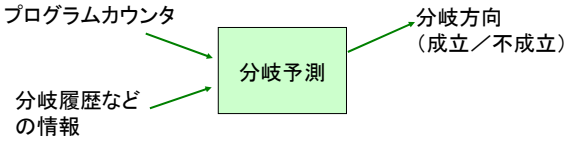



## 高いバンド幅の命令フェッチ

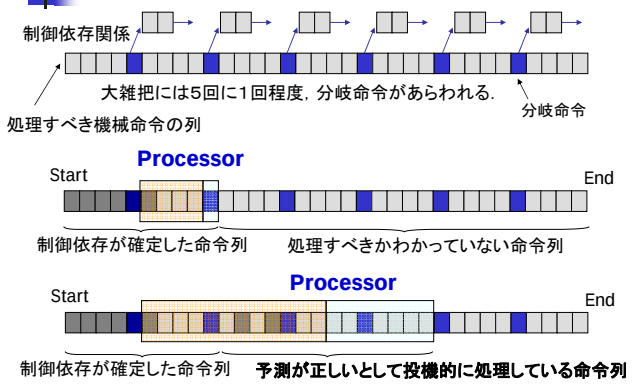
- パイプラインにバブルを生じさせないためには，条件分岐命令をフェッチした時に，次の3つを予測しなければならない．
  - **フェッチしている命令が分岐かどうか**
  - **分岐方向**
  - **分岐先アドレス**



## 分岐方向の予測（分岐予測）



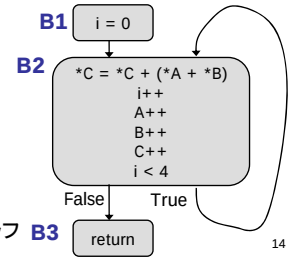
## 投機処理と分岐方向の予測（分岐予測）





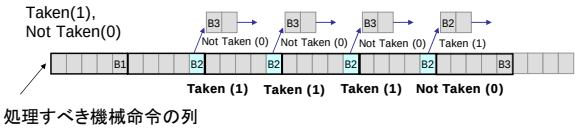

## サンプルプログラム Vector Add

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

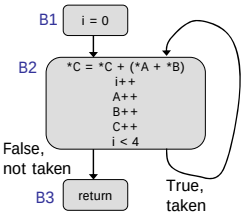





## シンプルな分岐予測 Branch Always




- 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．

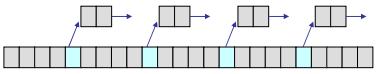




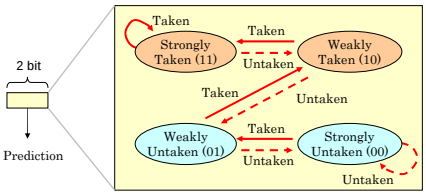

## シンプルな分岐予測 2ビットカウンタ方式

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 |
| 0040d7f9 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 |

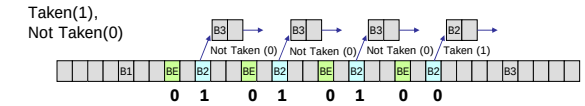


- 2ビットカウンタ方式
  - 大域的な偏り，局所性の利用
  - 2ビットカウンタの状態に応じて予測
  - 予測のためのメモリは2ビット
  - 状態の更新

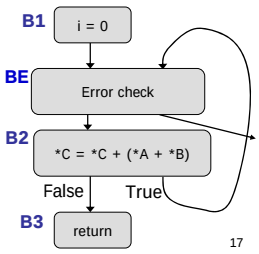





## サンプルプログラム Vector Add

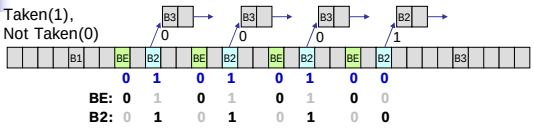



```
#define VSIZE 4
void vadd(long *A, long *B, long *C){
  for(i=0; i<VSIZE; i++) {
    if(A[i]<0) error_routine();
    C[i] += (A[i] + B[i]);
  }
}
```

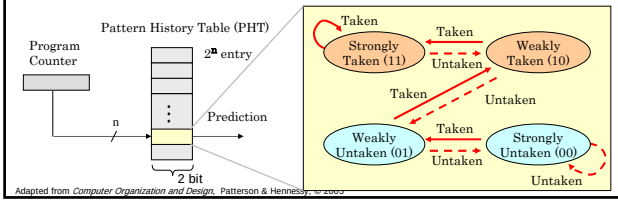




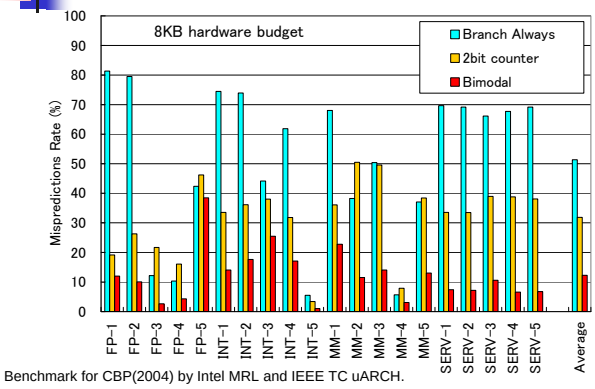

## Bimodal (ISCA 1981)




- 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．



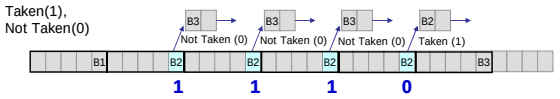
## シンプルな分岐予測の予測精度（予測ミス率）




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

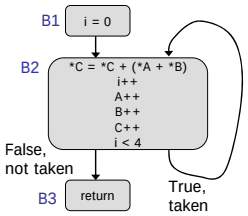


## 分岐履歴



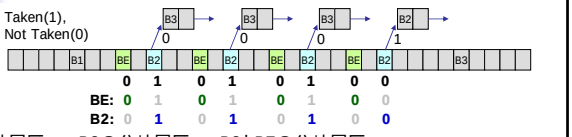

**B2の分岐履歴**

1110 ?
11101 ?
111011 ?
1110111 ?
11101110 ?






## ローカル，グローバル分岐履歴




BEの分岐履歴

0000 ?
00000 ?
000000 ?
0000000 ?
00000000 ?

**ローカル分岐履歴**

B2の分岐履歴

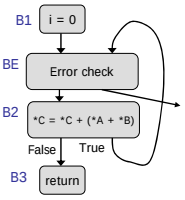
1110 ?
11101 ?
111011 ?
1110111 ?
11101110 ?

**ローカル分岐履歴**

B2とBEの分岐履歴

010101000 ?

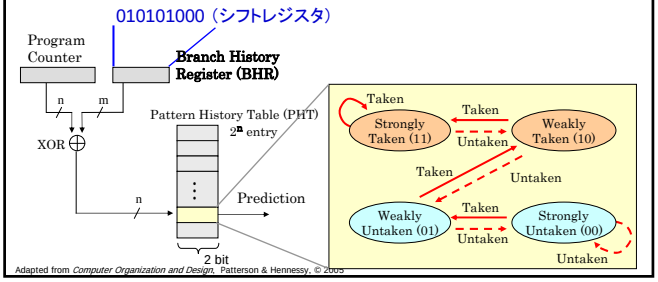
**グローバル分岐履歴**






## 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 {
    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){
    [redacted]
}

void Gshare::update(data_t pc, int taken){
    [redacted]
}
```



## Exercise

gshare の実装

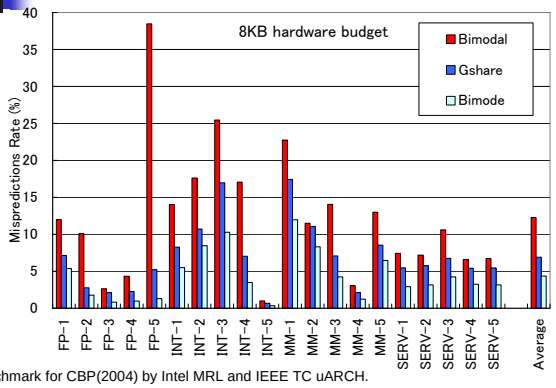
## Gshareの実装

```
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;
}
```



## 予測精度（予測ミス率）




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



## 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.