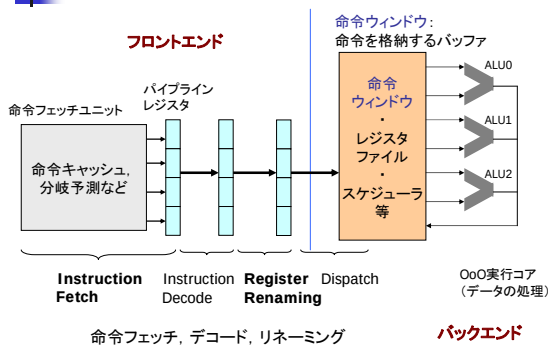
計算機アーキテクチャ特論
(Advanced Computer Architectures)

プロセッサフロントエンド

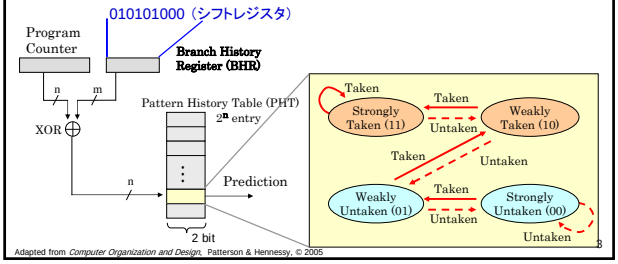
吉瀬 謙二 計算工学専攻
kise _at_ cs.titech.ac.jp www.arch.cs.titech.ac.jp
W832 講義室 金曜日 13:20 – 14:50

アウトオブオーダ実行プロセッサの構成



Gshare (TR-DEC 1993)の実装

- 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



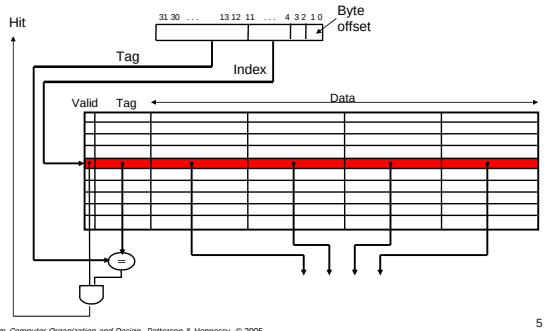
Gshareの実装

```
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){
    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;
}
```


命令キャッシュの実装

- ラインサイズ 4ワード (16 Byte)

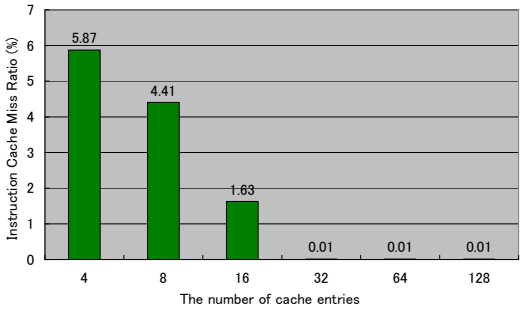


命令キャッシュの実装

```
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*);
};
```

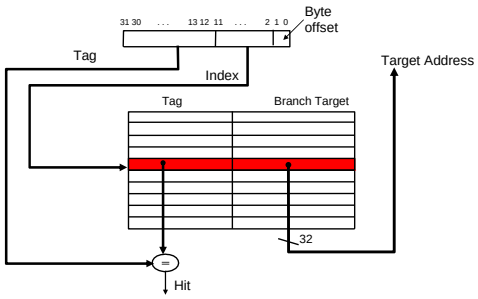
```
Icache::Icache(int icache_size, main_memory *m){
    mem = m;
    size = icache_size;
    buf  = (icache_line *)calloc(size, sizeof(iline));
}

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; }
}
```

命令キャッシュのミス率 (bench, シンプルな合成ベンチマーク)



Branch Target Buffer (BTB)の実装

- 分岐成立の場合にのみ，分岐先アドレスを登録する．
- Validビットは利用しない．



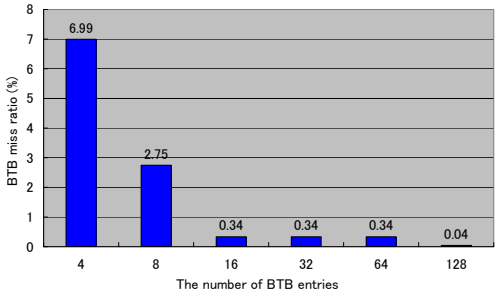
Branch Target Buffer (BTB)の実装

```
struct btb_line {
    data_t tag;
    data_t data;
};

class BTB {
    btb_line *buf;
public:
    int size;
    BTB(int);
    void fetch(data_t, data_t*);
    void regist(data_t, data_t);
};
```

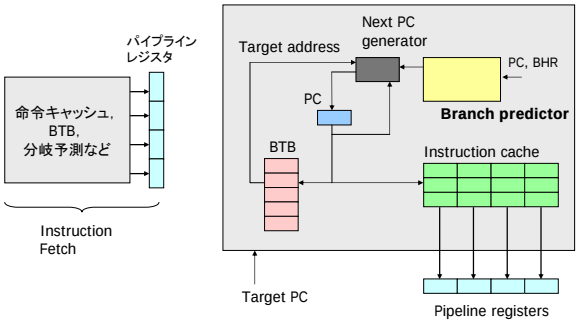
```
BTB::BTB(int btb_size){
    size = btb_size;
    buf  = (btb_line *)calloc(size, sizeof(btb_line));
}

void BTB::fetch(data_t pc, data_t *target){
    int index  = (pc >> 2) % size;
    data_t tag = (pc >> 2);
    if(buf[index].tag==tag) *target=buf[index].data;
    else *target = 0;
}

void BTB::regist(data_t pc, data_t target){
    int index  = (pc >> 2) % size;
    data_t tag = (pc >> 2);
    buf[index].tag  = tag;
    buf[index].data = target;
}
```


Branch Target Buffer のミス率 (bench)

成立の分岐命令の場合のみ判定する点に注意

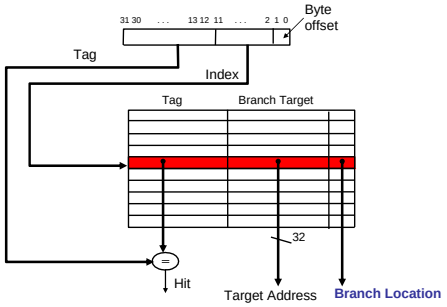


命令フェッチユニットの例



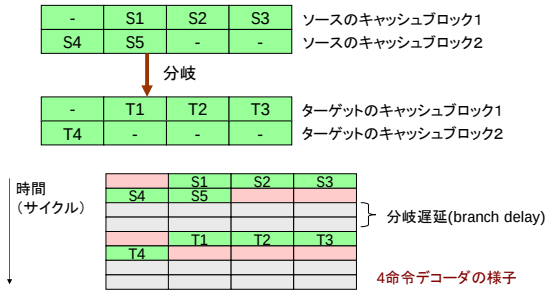
Branch Target Buffer (BTB)の改良

- キャッシュラインに1つの分岐のみを許す



命令キャッシュにおけるミスアラインメント

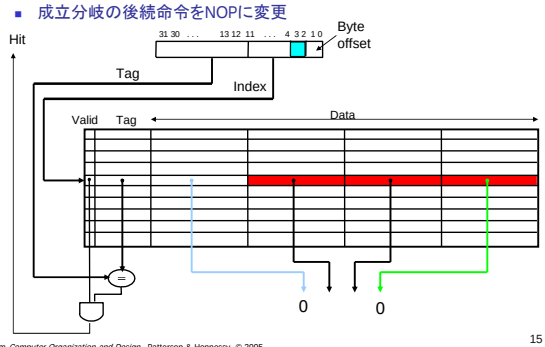
- 分岐命令S5の飛び先をT1とする.



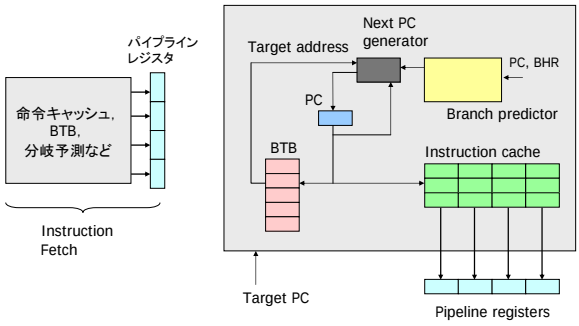
命令の整列化およびマージ

命令キャッシュの改良，フィルタリング

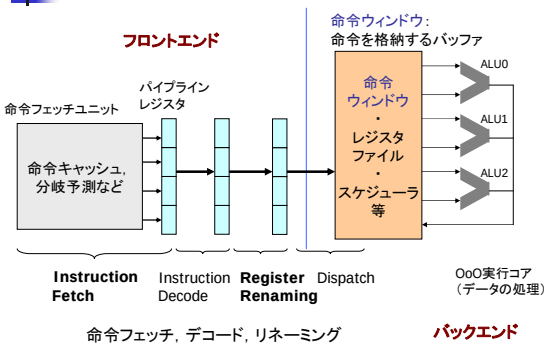
- PCが指し示す以前の命令をNOPに変更
- 成立分岐の後続命令をNOPに変更



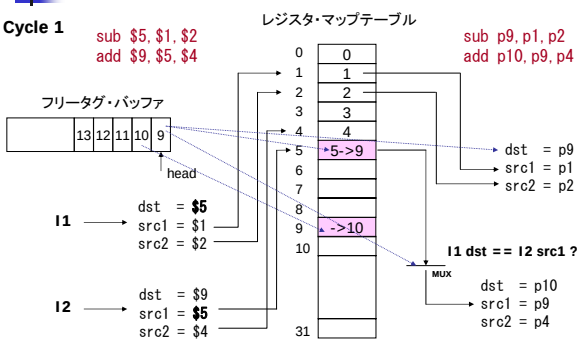
命令フェッチユニットの例



アウトオブオーダ実行プロセッサの構成



2命令のレジスタ・リネーミング



レジスタ データ依存(data dependence)

- **真のデータ依存 (true data dependence)**
  - **RAW, read after write**
- 出力依存 (output dependence)
  - WAW, write after write
- 逆依存 (antidependence)
  - WAR, write after read

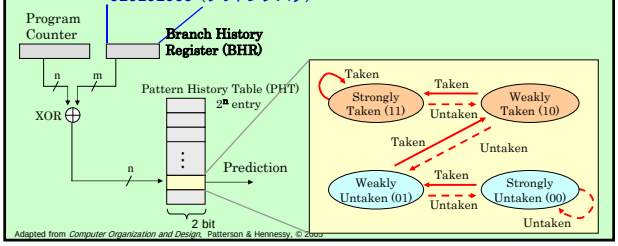
偽のデータ依存

- RAR ?, read after read

Gshare (TR-DEC 1993)

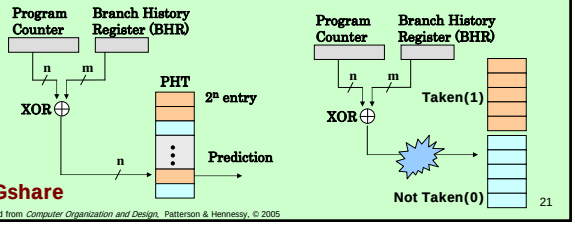
- グローバル分岐履歴と分岐アドレスとの排他的論理和によりパターン履歴表へのインデックスを作成
  - パターン履歴表は2ビット飽和型カウンタの配列で，選択された2ビットカウンタの値により分岐方向を予測（bimodalと同じ）
  - 分岐結果を用いて，予測に利用したカウンタを更新

010101000 (シフトレジスタ)



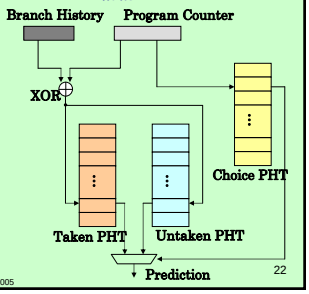
Gshare の改良

- PHTの競合が発生して性能が低下
- PCとBHRによって特定される予測（成立，不成立）には偏りが存在するので，これらを別のテーブルに格納することで競合の悪影響を緩和
  - 分岐成立に偏っているもの
  - 分岐不成立に偏っているもの



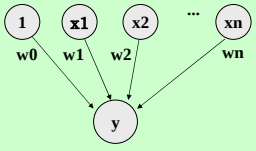
Bimode (MICRO 1997)

- 偏りを利用して競合の悪影響を緩和
  - 分岐成立に偏っているものをTaken PHTに格納
  - 分岐不成立に偏っているものをUntaken PHTに格納
- Choice PHT の内容で，どちらのテーブルを利用するか選択
- インデックスを工夫
  - Choice PHT は命令アドレス
  - Taken PHT, Untaken PHT は命令アドレスと分岐履歴



パーセプトロン

- パーセプロロンモデル
  - x_1からx_nまでのnビットの分岐履歴を入力とする.
  - y を計算する.
    - w は符号付きの整数で表現
  - y の値がある閾値より高い場合に成立と予測する.



Perceptron Model

$$y = w_0 + \sum_{i=1}^{n} x_i w_i.$$

レポート(1)：分岐予測の実装と評価

- **Bimode分岐予測**を実装し，その予測ミス率を測定せよ．また，Gshare分岐予測との予測精度の比較を示せ．
  - ハードウェア量を 2KB，4KB，8KB，16KB，32KB，64KBとしてグラフを描け．
- **Bimode分岐予測**に工夫を施し（あるいは，ことなる方式の予測を実装し），予測ミス率を測定せよ．
  - ハードウェア量を 2KB，4KB，8KB，16KB，32KB，64KBとしてグラフを描け．
- 次回の講義（１２月１７日）の開始時点にレポートを提出
  - コードの説明（コードは少ないほどベター），工夫した点
  - **ハードウェア量の計算方法を明示**
  - ミス率のグラフ（表ではないので注意）
  - 考察と感想

レポート(1):分岐予測の実装と評価

- **トレースデータ，命令アドレスと分岐結果の系列**

```
/***** BPKit 0.5 trace file *****/
//trace_name_______: CBP1-IT1
//total_branches___: 4184792
//total_instructions: 29499987
004058fb 0
00405910 0
0040591c 0
00405925 0
0040592e 0
0040593a 0
00405944 0
0040594b 0
0040492d 1
0040494f 0
```

```
while(!gzeof(gzfp)){
        gzgets(gzfp, buf, BUFSIZE);
        sscanf(buf, "%x %d", &pc, &taken);

        bp_predict(pc, NULL, &pred); /* prediction */
        bp_regist(pc, taken, NULL);  /* update storage */

        if(pred==taken) hit++; else miss++;
}
```


レポート(1):分岐予測の実装と評価

```
window-1
[advance@sc440 ~/kise]$ tar xfz /home/advance/bpkit.tgz
[advance@sc440 ~/kise]$ ls
bpkit
[advance@sc440 ~/kise]$ ls bpkit/
Core  Trace  base
[advance@sc440 ~/kise]$
```


レポート(1):分岐予測の実装と評価

```
window-1
[advance@sc440 ~/kise]$ tar xfz /home/advance/bpkit.tgz
[advance@sc440 ~/kise]$ ls
bpkit
[advance@sc440 ~/kise]$ ls bpkit/
Core  Trace  base
[advance@sc440 ~/kise]$ cd bpkit/base/always/
[advance@sc440 always]$ make
make predictor
make[1]: ディレクトリ '/home/advance/kise/bpkit/base/always' に入ります
g++ -Wall -O2 -Iz ../../Core/main.cc bpred.cc -o predictor
make[1]: ディレクトリ '/home/advance/kise/bpkit/base/always' から出ます
[advance@sc440 always]$ ls
Makefile  bpred.cc  log  predictor
[advance@sc440 always]$
```


レポート(1):分岐予測の実装と評価

```
window-1
[advance@sc440 ~/kise]$ tar xfz /home/advance/bpkit.tgz
[advance@sc440 ~/kise]$ ls
bpkit
[advance@sc440 ~/kise]$ ls bpkit/
Core  Trace  base
[advance@sc440 ~/kise]$ cd bpkit/base/always/
[advance@sc440 always]$ make
make predictor
make[1]: ディレクトリ '/home/advance/kise/bpkit/base/always' に入ります
g++ -Wall -O2 -Iz ../../Core/main.cc bpred.cc -o predictor
make[1]: ディレクトリ '/home/advance/kise/bpkit/base/always' から出ます
[advance@sc440 always]$ ls
Makefile  bpred.cc  log  predictor
[advance@sc440 always]$ make run
make predictor
make[1]: ディレクトリ '/home/advance/kise/bpkit/base/always' に入ります
make[1]: 'predictor' は更新済みです
make[1]: ディレクトリ '/home/advance/kise/bpkit/base/always' から出ます
predictor -o ../../Trace/CBP1/FP-1.txt.gz
logfile: CBP1-FP1.log
predictor -o ../../Trace/CBP1/FP-2.txt.gz
logfile: CBP1-FP2.log
```


講義用の計算機の使い方

- ユーザ名 advance で 131.112.16.56 にログイン
  - linuxなど
    - ssh archo@131.112.16.56
    - 講義時に伝えたパスワードでログイン
- 学籍番号でディレクトリを作成して，そこで作業する．
  - mkdir myname
  - cd myname
- 参考ファイルをコピーして実行
  - tar …
  - maker run
  - make cat

アウトオブオーダ実行プロセッサの構成