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

8. スーパースカラプロセッサ
アウトオブオーダプロセッサ

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

レポート(1): 分岐予測の実装と評価

- **gshare分岐予測**を実装し，その予測ミス率を測定せよ．また，bimodal分岐予測との予測精度(20本のベンチマークのミス率の算術平均)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．
- **gshare分岐予測**に工夫を施し（あるいは，異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．
  - **予測ミス率が低い（性能が高い）と高得点．**
- １月６日の講義の開始時にレポートを提出
  - 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]$
```

bpkit03.tgz を使う

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 -lz ../../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]$
```

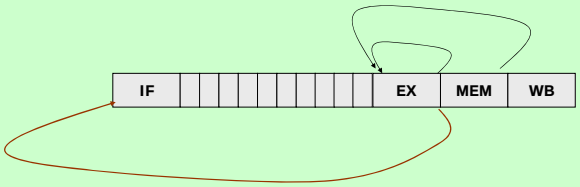

レポート(1):分岐予測の実装と評価

```
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 -lz ../../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
```

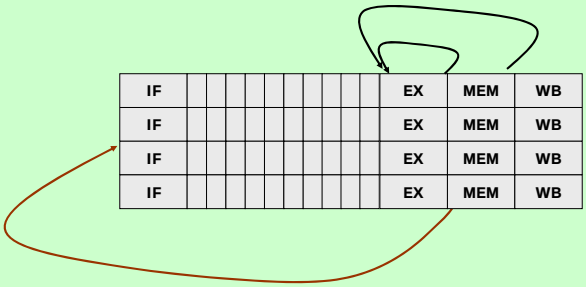
講義用の計算機の使い方

- 講義用の計算機のIPアドレスが変わりました.
- ユーザ名 advance で serv.arch.cs.titech.ac.jp にログイン
  - linuxなど
    - ssh advance@serv.arch.cs.titech.ac.jp
    - 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

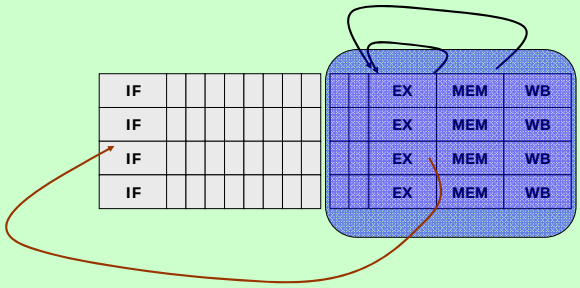
スカラ・プロセッサ（パイプラインの長い，スーパーパイプライン），パイプラインと制約ループ



スーパースカラプロセッサ（パイプラインの長い，スーパーパイプライン），パイプラインと制約ループ



アウトオブオーダ実行のスーパースカラプロセッサ

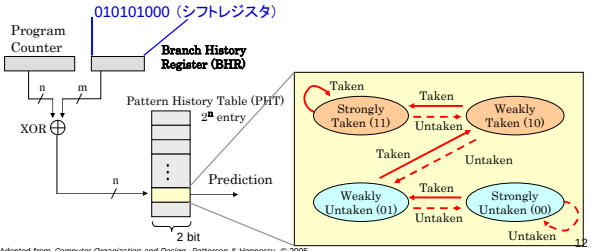


高いバンド幅の命令フェッチ

- 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(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（分岐予測を用いる）**
  - **分岐先アドレス**

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);
};
```

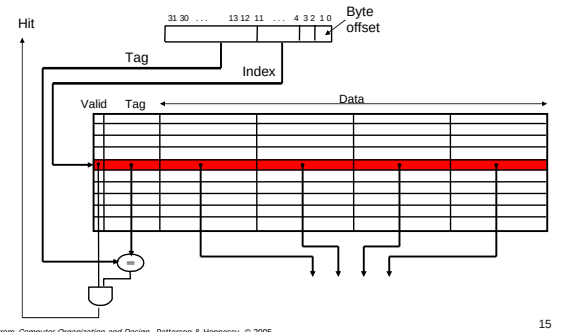
```
Gshare::Gshare(int bpred_size){
    size = bpred_size;
    buf  = (data_t *)calloc(size, sizeof(data_t));
    for(int i=0; i<size; i++) buf[i] = 2;
}

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;
}
```


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(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 （分岐予測を用いる）**
  - **分岐先アドレス**

命令キャッシュの実装

- ラインサイズ 4ワード (16 Byte)

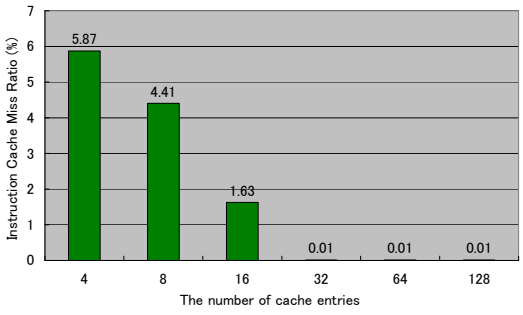


命令キャッシュの実装

```
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*);
};
```

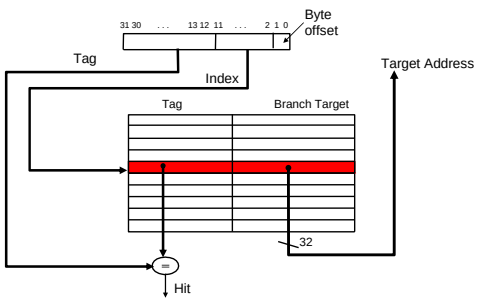
```
Icache::Icache(int icache_size, main_memory *m){
    mem = m;
    size = icache_size;
    buf  = (icache_line *)calloc(size, sizeof(iline));
}

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．



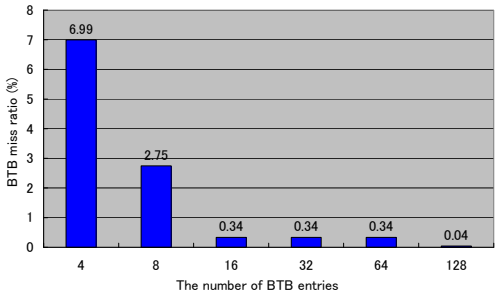
Branch Target Buffer (BTB)の実装

```
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);
};
```

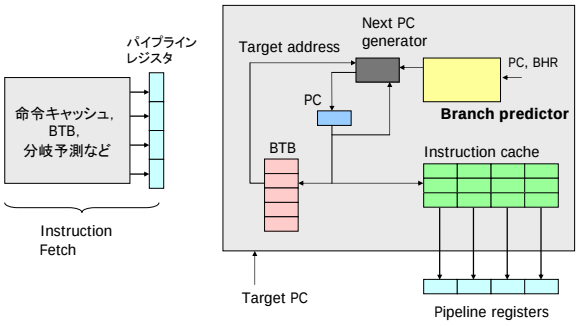
```
BTB::BTB(int btb_size){
    size = btb_size;
    buf  = (btb_line *)calloc(size, sizeof(btb_line));
}

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

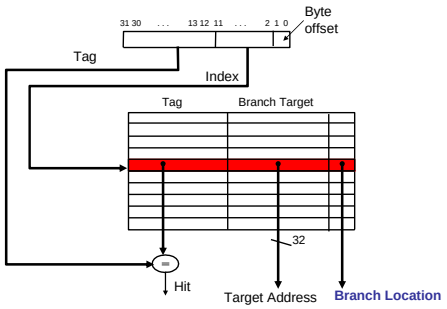


命令フェッチユニットの例



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.

-	S1	S2	S3	ソースのキャッシュブロック1
S4	S5	-	-	ソースのキャッシュブロック2

分岐

-	T1	T2	T3	ターゲットのキャッシュブロック1
T4	-	-	-	ターゲットのキャッシュブロック2

時間
（サイクル）

	S1	S2	S3
S4	S5		
	T1	T2	T3
T4			

分岐遅延(branch delay)

4命令デコーダの様子

命令の整列化およびマージ

時間
（サイクル）

	S1	S2	S3
S4	S5		
	T1	T2	T3
T4			

分岐遅延(branch delay)

4命令デコーダの様子

4命令デコーダの様子

命令の整列化

S1	S2	S3	S4
S5			
T1	T2	T3	T4

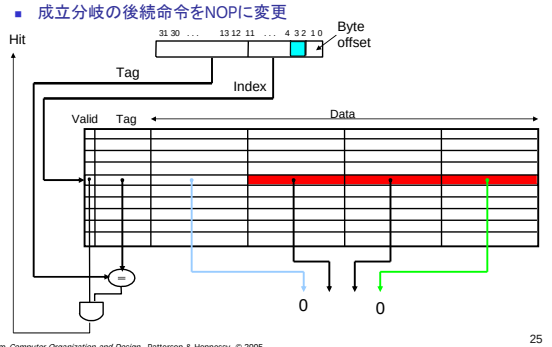
4命令デコーダの様子

命令のマージ

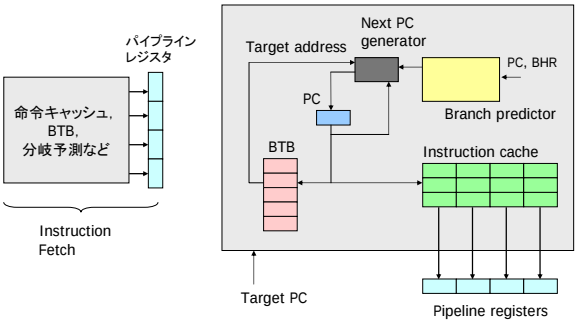
S1	S2	S3	S4
S5	T1	T2	T3
T4			

命令キャッシュの改良，フィルタリング

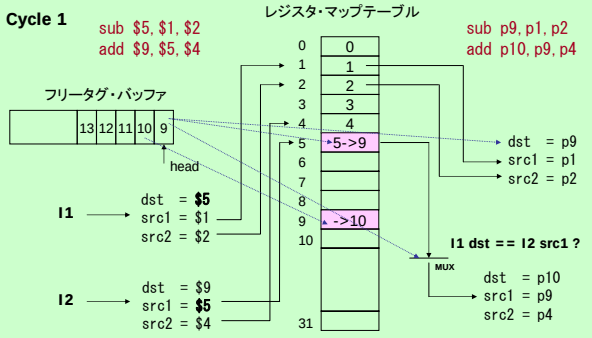
- PCが指し示す以前の命令をNOPに変更
- 成立分岐の後続命令をNOPに変更



命令フェッチユニットの例



2命令のレジスタ・リネーミング



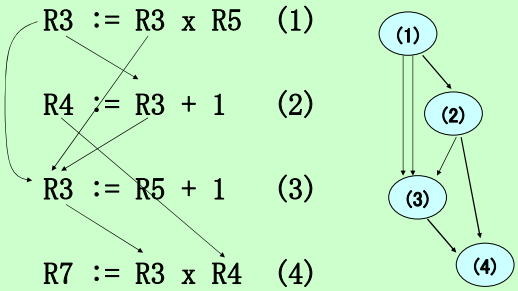
データ依存関係 (data dependence)

- 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

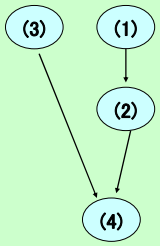
データ依存関係の例



命令レベル並列性は？

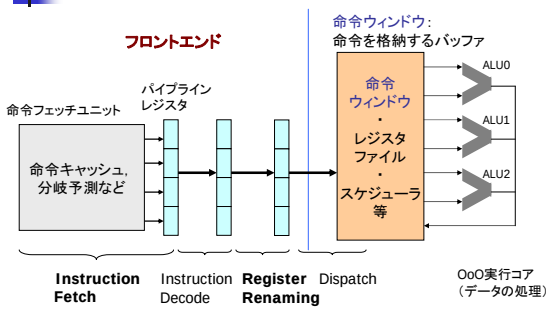
レジスタ名前換え，レジスタ・リネーミング

```
R3 := R3 x R5    (1)
R4 := R3 + 1     (2)
R3 := R5 + 1     (3)
R7 := R3 x R4    (4)

R8 := R3 x R5    (1)
R9 := R8 + 1     (2)
R10:= R5 + 1     (3)
R11:= R10 x R9   (4)
```



命令レベル並列性は？

アウトオブオーダ実行プロセッサの構成



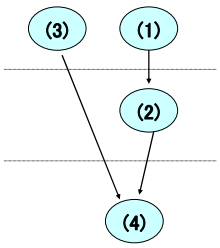
動的スケジューリング（アウトオブオーダ実行）

(1) DIV.D F0, F2, F4
(2) ADD.D F10, F0, F8
(3) SUB.D F12, F8, F14

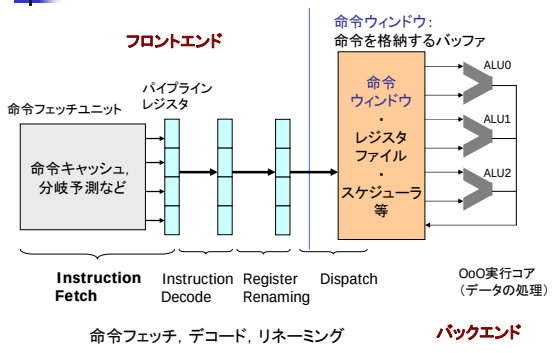
- DIV.D とADD.Dの依存がパイプラインをストールさせ，SUB.D 命令の実行を阻害
- SUB.D はパイプラインのどの命令にもデータ依存しない
- プログラム順序に従って命令を実行するという制約を取り除くことで，この制限を解消

レジスタ名前換え，レジスタ・リネーミング

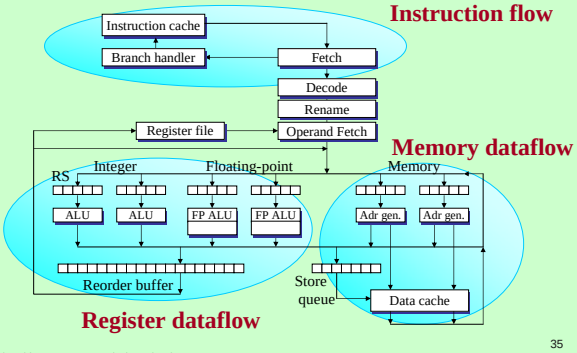
```
P8 := P3 x P5      (1)
P9 := P8 + 1       (2)
P10:= P5 + 1       (3)
P11:= P10 x P9     (4)

P8 := P3 x P5      (1)
P9 := P8 + 1       (2)
P10:= P5 + 1       (3)
P11:= P10 x P9     (4)
```



アウトオブオーダ実行プロセッサの構成



Out-of-orderスーパースカラ・プロセッサ



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