2011年 後学期

計算機アーキテクチャ 第二 (O)

9. スーパースカラプロセッサ

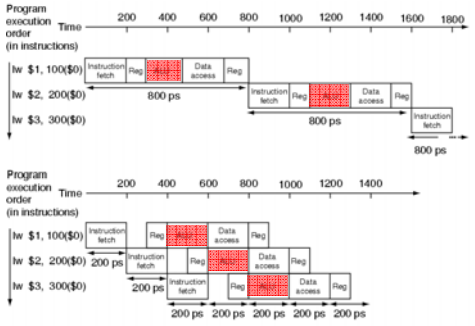
大学院情報理工学研究科 計算工学専攻
吉瀬謙二 kise _at_ cs.titech.ac.jp
S321講義室 月曜日 5, 6時限 13:20−14:50

プロセッサのマイクロアーキテクチャ

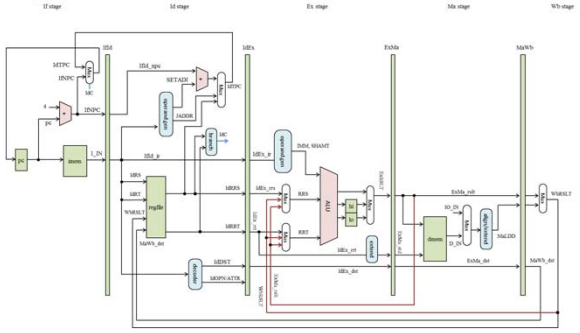
- マルチサイクルプロセッサ
- シングルサイクルプロセッサ
- パイプライン
  - スカラープロセッサ
  - スーパースカラプロセッサ（インオーダ）
  - スーパースカラプロセッサ（アウトオブオーダ）

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

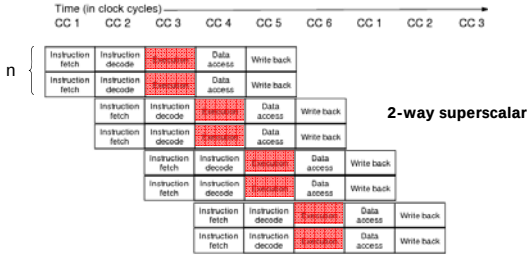


5段パイプラインの標準的なスカラプロセッサ

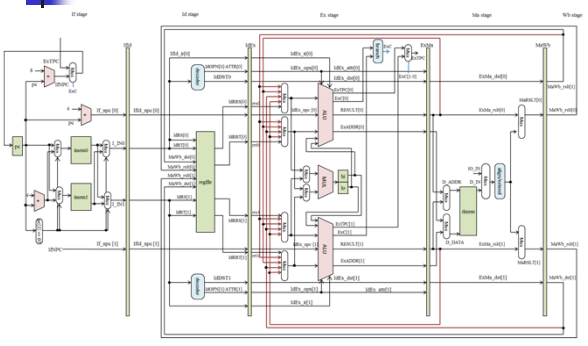


スーパースカラプロセッサと命令レベル並列性

- 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



5段パイプラインのインオーダ・スーパースカラプロセッサ インタリーブ命令メモリ版

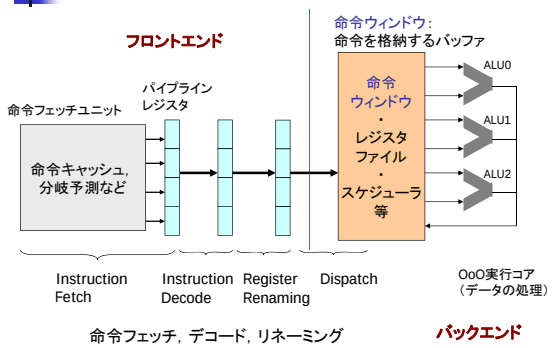


スーパースカラプロセッサにおける動的スケジューリング（アウトオブオーダ実行）

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

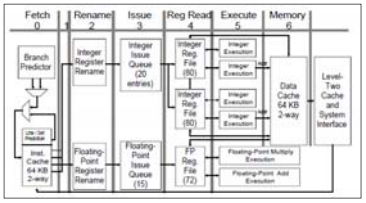
- DIV.D とADD.Dの依存がパイプラインをストールさせ，SUB.D 命令の実行を阻害
- SUB.D はパイプラインのどの命令にもデータ依存しない
- プログラム順序に従って命令を実行するという制約を取り除くことで，この制限を解消

アウトオブオーダ実行プロセッサの構成



アウトオブオーダ実行プロセッサの命令パイプライン

Instruction Fetch	Decode	Rename	Dispatch				
				Issue	Register Read	Execute	Commit



The Alpha 21264 Microprocessor Architecture
R. E. Kessler, E. J. McLellan, and D. A. Webb, Compaq Computer Corporation

2011年 後学期

計算機アーキテクチャ 第二 (O)

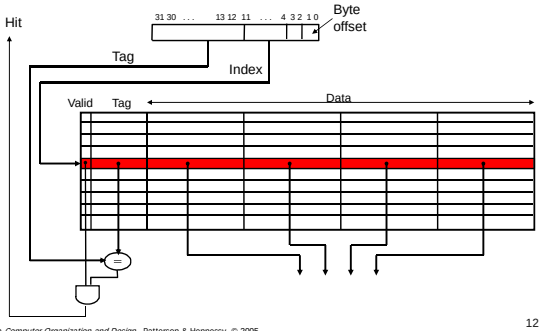
アウトオブオーダ実行プロセッサとフロントエンド

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**

命令キャッシュの実装

- ラインサイズ 4ワード (16 Byte)



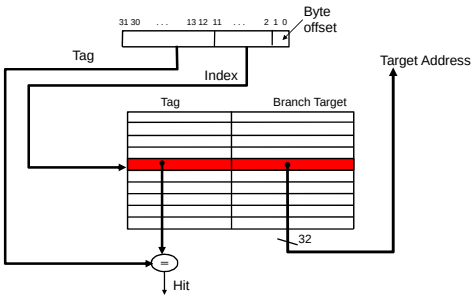
命令キャッシュの実装

```
struct icache_line {
    int valid;
    int tag;
    int 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; }
}
```


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ビットは利用しなくてもよい．

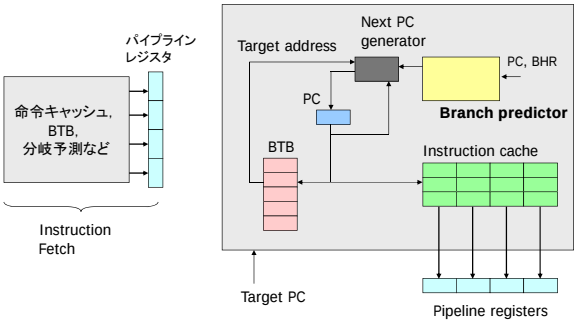


Branch Target Buffer (BTB)の実装

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

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

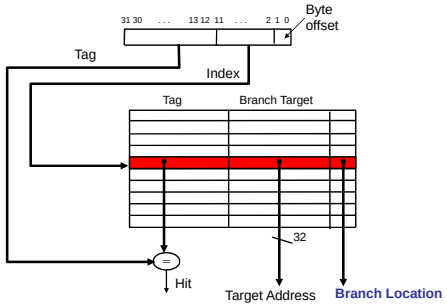
```
BTB::BTB(int btb_size){
    size = btb_size;
    buf  = (btb_line *)calloc(size, sizeof(btb_line));
}

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;
}
```


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



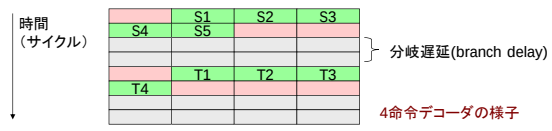
命令キャッシュにおけるミスアラインメント

- 分岐命令S5の飛び先をT1とする．

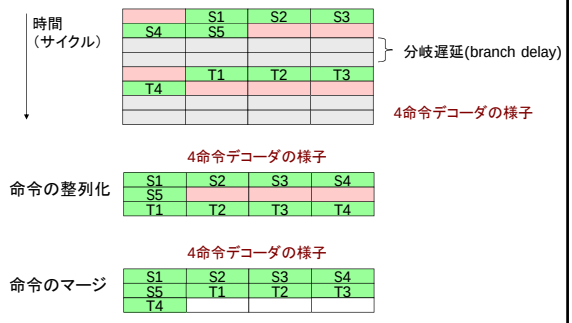
-	S1	S2	S3	ソースのキャッシュブロック1
S4	S5	-	-	ソースのキャッシュブロック2

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

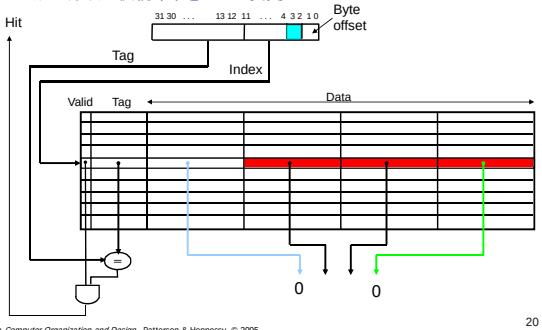


命令の整列化およびマージ

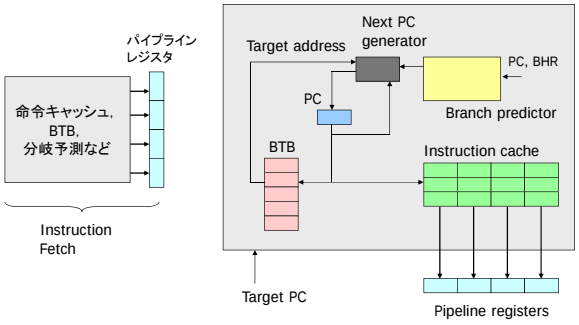


命令キャッシュの改良，フィルタリング

- 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

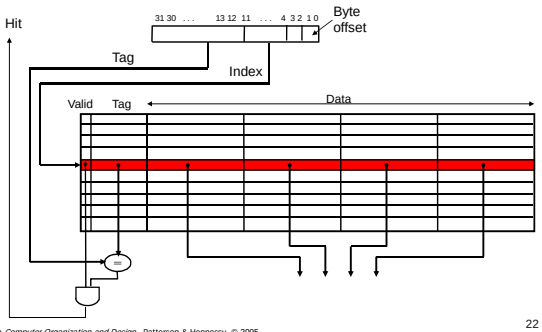


命令フェッチユニットの例



命令キャッシュの実装

- ラインサイズ 4ワード (16 Byte)



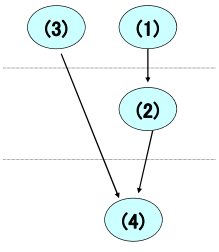
命令キャッシュの実装

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

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

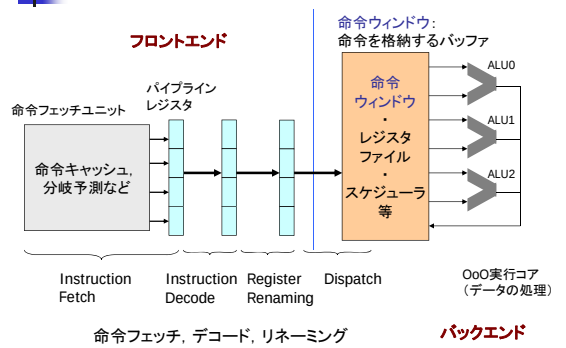
```
Icache::Icache(int icache_size, main_memory *m){
    mem = m;
    size = icache_size;
    buf  = (icache_line *)calloc(size, sizeof(iline));
}

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[i] = ir_t;
        }
        return 0; }
}
```


スーパースカラプロセッサにおける 動的スケジューリング（アウトオブオーダ実行）

```
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



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