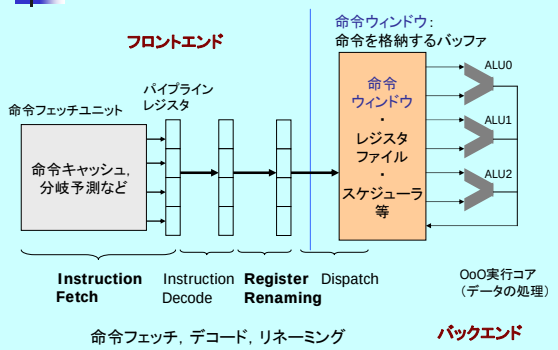
2009年 後学期

# 計算機アーキテクチャ 第二 (O)

## 9. アウトオブオーダ実行プロセッサ(2)

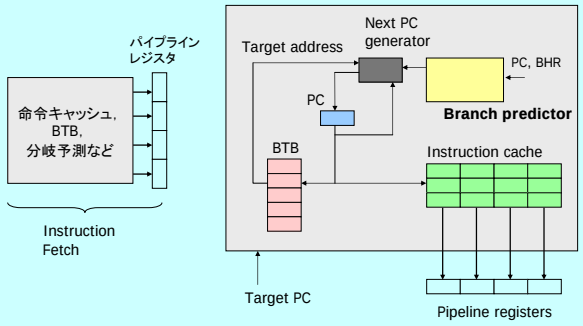


## アウトオブオーダ実行プロセッサの構成





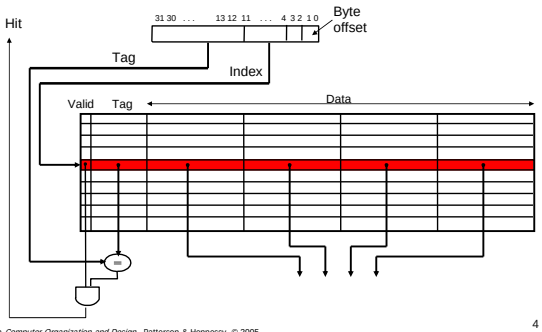

## 命令フェッチユニットの例






## 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

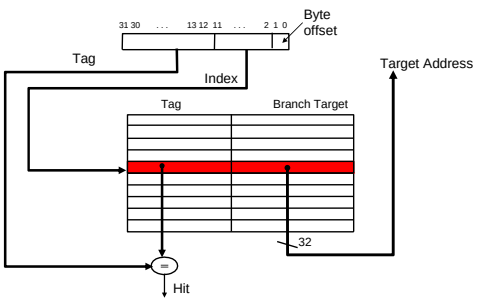
```
struct icache_line {
    data_t valid;
    data_t tag;
    data_t data[4];
} iline;

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; }
}
```



## 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

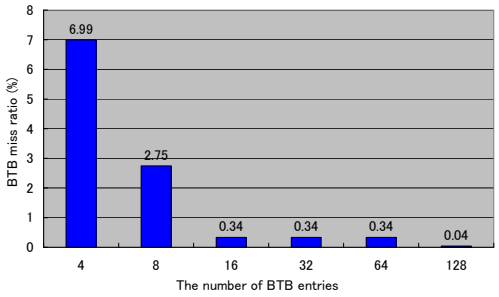
```
struct btb_line {
    data_t tag;
    data_t data;
};

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;
}
```



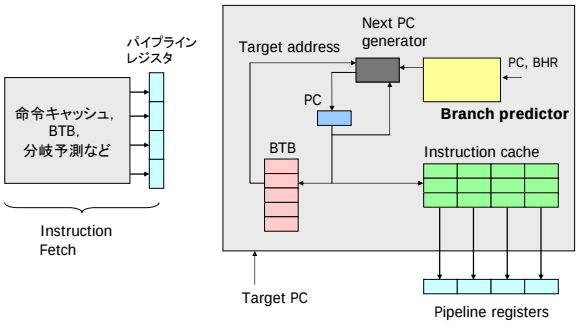
## Branch Target Buffer のミス率 (bench)

成立の分岐命令の場合のみ判定する点に注意





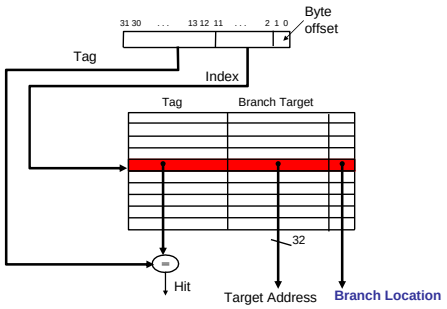

## 命令フェッチユニットの例






## 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 |

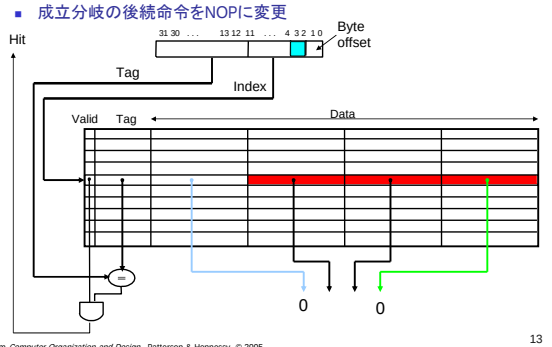
4命令デコーダの様子

命令のマージ

| | | | |
|---|---|---|---|
| S1 | S2 | S3 | S4 |
| S5 | T1 | T2 | T3 |
| T4 | | | |

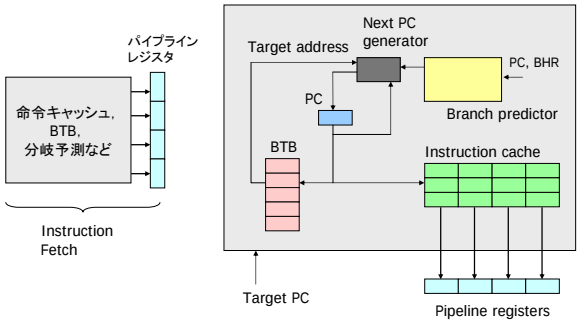
## 命令キャッシュの改良，フィルタリング

- 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





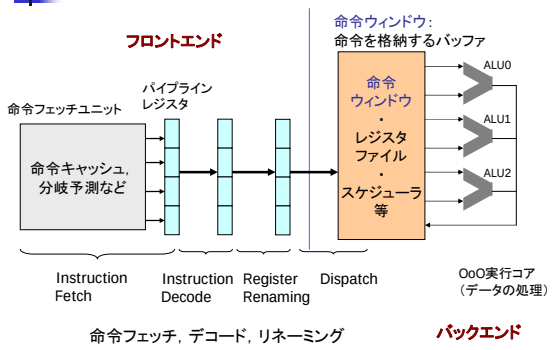

2009年 後学期

# 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のバックエンド



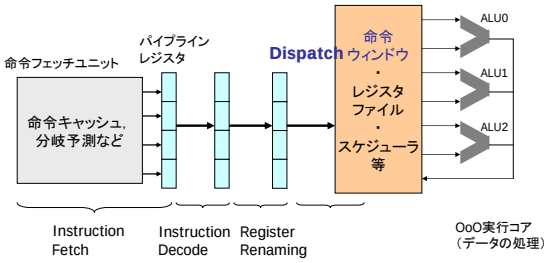
## アウトオブオーダ実行プロセッサの構成






## アウトオブオーダ実行プロセッサの構成

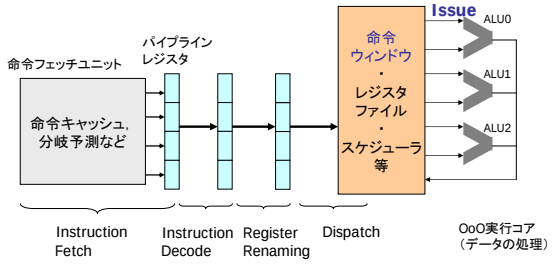
- ディスパッチ (dispatch)：命令ウィンドウに命令を格納する動作





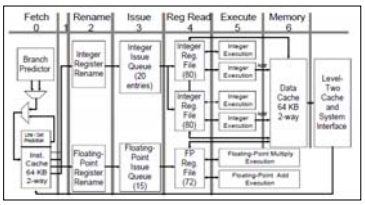

## アウトオブオーダ実行プロセッサの構成

- 発行 (issue, fire)：命令ウィンドウから，データ依存が解消された命令を機能ユニットに送り出す動作
  - この時に，レジスタファイルの値を読み出すことがある．



## 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



## 命令発行機構: Tomasuloのアプローチ

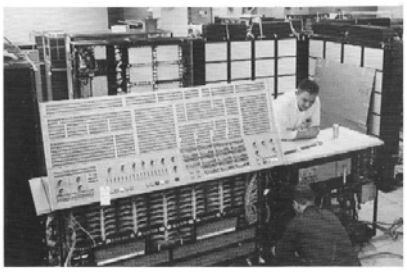
- IBM 360/91 の浮動小数点ユニットでは、アウトオブオーダ実行のための洗練された方式が採用されていた.
- Robert Tomasulo によって発明されたこの手法では
  - 命令が必要とするオペランドがいつ利用できるかを探知し, RAWハザードを最少化
  - レジスタリネーミングを導入してWAWハザードとWARハザードを回避
- 近年のプロセッサでは, この手法のさまざまなバリエーションが採用されているが, これら2つの重要な概念は共通の特徴



## 命令発行機構: Tomasuloのアプローチ

- IBM 360/91 の浮動小数点ユニットでは、アウトオブオーダ
- Ro
  - 
  - 
- 近
  シ
  通

The IBM 360/91



Installation of the IBM 360/91 in the Columbia Computer Center machine room in February or March 1969. Photo: AIS archive.



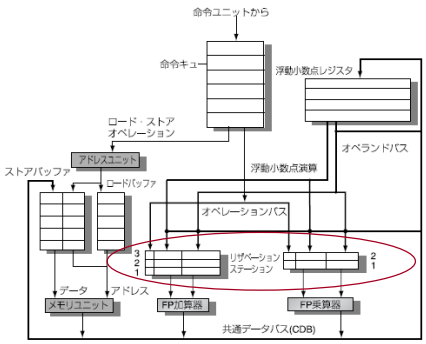
## Tomasuloのアプローチ

- (2)レジスタリネーミングを導入してWAWハザードとWARハザードを回避
- S と T という2つの一時レジスタが利用できると仮定

```
DIV.D  F0,F2,F4          DIV.D  F0,F2,F4
ADD.D  F6,F0,F8          ADD.D  S,F0,F8
S.D    F6,0(R1)          S.D    S,0(R1)
SUB.D  F8,F10,F14        SUB.D  T,F10,F14
MUL.D  F6,F10,F8         MUL.D  F6,F10,T
```



## Tomasuloのアプローチ






## Tomasuloのアプローチ

- レジスタリネーミングは発行を待つ命令のオペランドを蓄えるリザベーションステーション(命令ウィンドウの1つの実装方式)を用いて実現
  - オペランドの値をリザベーションステーションに格納することで, レジスタファイルを経由しないオペランドの受け渡しを実現.
  - 保留中の命令は, それらの入力を提供するリザベーションステーションを指定する.
  - 複数の同じレジスタへの書き込みが生じる場合には最後のデータのみをレジスタに書き込む.
  - 命令がディスパッチされる時, 保留中のオペランド用のレジスタ指示子はリザベーションステーションの名前にリネームする.

## Tomasuloのアプローチ

- 集中化されたレジスタファイルではなく、リザベーションステーションを使用することによる2つの重要な利点
- （1）ハザード検出および実行制御の分散化
  - 各機能ユニットはリザベーションステーションに保持された情報によって，そのユニットで命令がいつ実行を始めることができるかを決める．
- （2）実行結果がレジスタを経由するオーバヘッドを隠蔽
  - 実行結果が格納されているリザベーションステーションから機能ユニットに直接渡され，レジスタを経由する必要がない．
  - 実行結果は，共通の結果バスでバイパスされ，オペランドを待つユニットすべてが同時に値をロードする．
  - このバスは，IBM 360/91 で**共通データバス**（CDB：common data bus）と呼ばれる．
  - 複数の実行ユニットを備えたパイプラインでは2つ以上のバスが必要



## Tomasuloのアプローチ

- ロードバッファの3つの機能
  - 計算されるまでの間，実効アドレスの要素を保持
  - メモリからデータが到着するのを待っている処理中のロード命令を探知
  - 完了してCDBの利用を待っているロードの結果を保持
- ストアバッファの3つの機能
  - 計算されるまでの間，実効アドレスの要素を保持
  - データ値がストアされるのを待っている処理中のストア命令の書き込み先メモリアドレスを保持
  - メモリユニットが利用可能になるまで格納するアドレスおよび値を保持
- 浮動小数点機能ユニットとロードユニットの結果はすべてCDBを経由して，浮動小数点レジスタファイルやリザベーションステーションやストアバッファに送られる．



## Tomasuloのアプローチ

- 1. リザベーションステーションへの命令格納
  - 命令キュー（正確なデータフローを保証するためにFIFO で命令を格納している）のヘッド（先頭）から命令を取り出す．
    適切なリザベーションステーションに空きがある場合は，そこに命令を送る．
  - レジスタにオペランド値がある場合は，その値も同時にリザベーションステーションに送られる．
    空のリザベーションステーションがない場合，構造ハザードとなり，リザベーションステーションやバッファが解放されるまでストール．
  - オペランド値がレジスタにない場合は，オペランド値を生成する機能ユニットとリザベーションステーションを検出する．
    このステップがレジスタリネーミングに対応し，WARとWAWハザードを除去する。



## Tomasuloのアプローチ

- 2. 命令実行の開始と実行（execute）
  - オペランドの1つ以上がまだ利用できない場合は，値が送られてくるのを待ちながら共通データバスを監視する．オペランドが利用可能になった時に，それを待つリザベーションステーションに格納する．すべてのオペランドが利用可能になった時に，そのオペレーションは対応する機能ユニットで実行できる．
  - オペランドが利用可能になるまで命令の実行を遅らせることによって，RAWハザードを回避する．
  - 同じ機能ユニットを利用する複数の命令が同一のクロックサイクルにおいて実行可能となるかもしれない点に注意する．同じクロックサイクルにおいて，個々の機能ユニットは異なる命令の実行を開始することができるが，1つの機能ユニットに対して2つ以上の命令が実行可能であれば，ユニットはそれらの中から1つを選択する．
  - 整数演算，浮動小数点演算のリザベーションステーションについては，この選択は任意の方式で行うことができる．ロードとストアの場合には制約を考慮する必要がある．



## Tomasuloのアプローチ

- 2. 実行（execute）の続き
  - ロードとストアは2 段階の実行過程を必要とする．第1 段階では，ベースレジスタが利用可能な場合に実効アドレスを計算する．また，得られた実効アドレスをロード・ストアバッファに格納する．
    第2 段階では，メモリユニットが利用可能になるとすぐに，ロードバッファのロード命令を実行する．
  - ストアバッファのストア命令は，メモリユニットに送られる前に，ストアすべき値を待たなければならない．ロードとストアは実効アドレス計算を通じてプログラム順序を維持する．それによって、メモリを経由するハザードに対処できる．
  - 例外の振る舞いを維持するために，命令は，プログラム順序において先行する分岐がすべて完了するまで実行を始めてはいけない．この制約により、実行中に例外を引き起こす命令が実際に実行を完了するということが保証される．
  - 分岐予測を利用するプロセッサ（動的スケジューリングのすべてのプロセッサがそうであるが）では，分岐に続く命令の実行を始める前に，分岐予測が正しいことをプロセッサが知らなければならないことを意味する．



## Tomasuloのアプローチ

- 3. 結果書き込み（Write Result）
  - 結果が利用可能になったら，CDBに結果を流し，そこからレジスタ、およびこの結果を待っているすべてのリザベーションステーション（ストアバッファを含む）に書き込む．
  - ストアされる値およびストアするメモリのアドレスの両方が利用可能になるまで，ストア命令はストアバッファの中に保存され，メモリユニットが利用可能になるとすぐに結果が格納される．

## Tomasuloのアプローチ

- リザベーションステーションが保有する7つのフィールド
  - **Op**:ソースオペランドS1 およびS2 に対して行うオペレーション
  - **Qj、Qk**:対応するソースオペランド値を生成するリザベーションステーションの番号．値が0 の場合は，ソースオペランドがVj またはVkとしてすでに利用可能であるか，不必要であることを示す．
  - **Vj、Vk**:ソースオペランドの値．各オペランドについては，Vフィールドあるいは Qフィールドのどちらかが常に有効となる．ロード命令については，Vkフィールドはオフセットフィールドを保持するために利用される．
  - **A**:ロードあるいはストア命令がメモリアドレス計算の情報を保持するために利用する．最初に，命令の即値のフィールドがここに格納される．アドレス計算の後には，実効アドレスが格納される．
  - **Busy** :当該リザベーションステーション，および，対応する機能ユニットが占有されていることを示す．

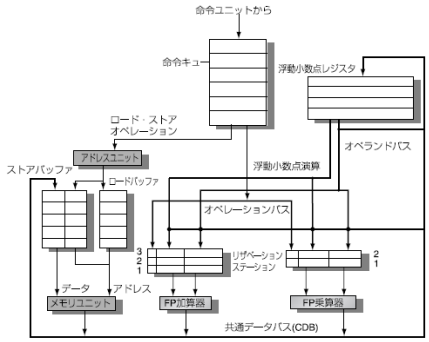


## Tomasuloのアプローチ

- レジスタファイルの各エントリにはQi フィールドを追加
  - **Qi** :その実行結果をレジスタへ格納する操作を含んでいるリザベーションステーションの番号．
    Qi の値がブランク(すなわち0)の場合は，現在，このレジスタに格納すべき結果を計算する命令が実行中でない．このため，このレジスタに格納されている内容がその値となる．



## Tomasuloのアプローチ






## 動的スケジューリングの例題

- 最初のロードだけが完了してその結果が書き戻されている時，次の命令列に対する状態テーブルはどのようになっているか？

1. L.D F6,32(R2)
2. L.D F2,44(R3)
3. MUL.D F0,F2,F4
4. SUB.D F8,F2,F6
5. DIV.D F10,F0,F6
6. ADD.D F6,F8,F2



命令状態

| 命令 | | 発行 | 実行 | 結果書き込み |
|---|---|---|---|---|
| L.D | F6,32(R2) | √ | √ | √ |
| L.D | F2,44(R3) | √ | √ | |
| MUL.D | F0,F2,F4 | √ | | |
| SUB.D | F8,F2,F6 | √ | | |
| DIV.D | F10,F0,F6 | √ | | |
| ADD.D | F6,F8,F2 | √ | | |

リザベーションステーション

| 名称 | Busy | Op | Vj | Vk | Qj | Qk | A |
|---|---|---|---|---|---|---|---|
| Load1 | no | | | | | | |
| Load2 | yes | Load | | | | | 44 + Regs[R3] |
| Add1 | yes | SUB | | Mem[32 + Regs[R2]] | Load2 | | |
| Add2 | yes | ADD | | | Add1 | Load2 | |
| Add3 | no | | | | | | |
| Mult1 | yes | MUL | | Regs[F4] | Load2 | | |
| Mult2 | yes | DIV | | Mem[32 + Regs[R2]] | Mult1 | | |

レジスタ状態

| フィールド | F0 | F2 | F4 | F6 | F8 | F10 | F12 | ... | F30 |
|---|---|---|---|---|---|---|---|---|---|
| Qi | Mult1 | Load2 | | Add2 | Add1 | Mult2 | | | |



## 動的スケジューリングの例題

- 例題2.5 と同じコードセグメントを利用して，MUL.D がその結果を書く準備ができている場合，状態テーブルがどのようになっているかを示せ．

1. L.D F6,32(R2)
2. L.D F2,44(R3)
3. MUL.D F0,F2,F4
4. SUB.D F8,F2,F6
5. DIV.D F10,F0,F6
6. ADD.D F6,F8,F2

命令状態

| 命令 | | 発行 | 実行 | 結果書き込み |
|---|---|---|---|---|
| L.D | F6,32(R2) | √ | √ | √ |
| L.D | F2,44(R3) | √ | √ | √ |
| MUL.D | F0,F2,F4 | √ | √ | |
| SUB.D | F8,F2,F6 | √ | √ | √ |
| DIV.D | F10,F0,F6 | √ | | |
| ADD.D | F6,F8,F2 | √ | √ | √ |

リザベーションステーション

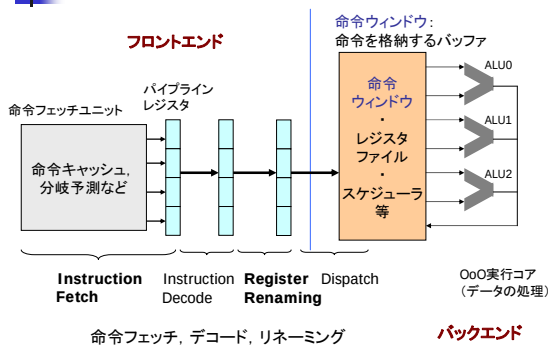
| 名称 | Busy | Op | Vj | Vk | Qj | Qk | A |
|---|---|---|---|---|---|---|---|
| Load1 | no | | | | | | |
| Load2 | no | | | | | | |
| Add1 | no | | | | | | |
| Add2 | no | | | | | | |
| Add3 | no | | | | | | |
| Mult1 | yes | MUL | Mem[45 + Regs[R3]] | | Regs[F4] | | |
| Mult2 | yes | DIV | | Mem[34 + Regs[R2]] | Mult1 | | |

レジスタ状態

| フィールド | F0 | F2 | F4 | F6 | F8 | F10 | F12 | ... | F30 |
|---|---|---|---|---|---|---|---|---|---|
| Qi | Mult1 | | | | | Mult2 | | | |

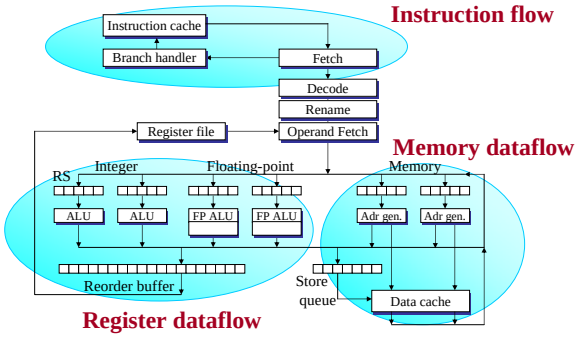


## アウトオブオーダ実行プロセッサの構成






## 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
- 講義用の計算機
  - 131.112.16.56 (情報工学科の演習室からは入れません)
  - ssh archo@131.112.16.56
    - mkdir myname
    - cd myname