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

9. アウトオブオーダプロセッサ
ステートと例外回復

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

例外 (exception)，割込み (interrupt)

- （希に）発生する好ましくないイベント
  - 分岐予測ミス
    - 頻度
  - 算術演算オーバフロー／アンダーフロー
  - ページ・フォールト（HDDへのアクセス）
  - メモリ保護違反
  - 未定義命令の使用
  - 入出力(I/O)からのリクエスト
  - ブレークポイント（プログラマが設定する割込み）
  - …

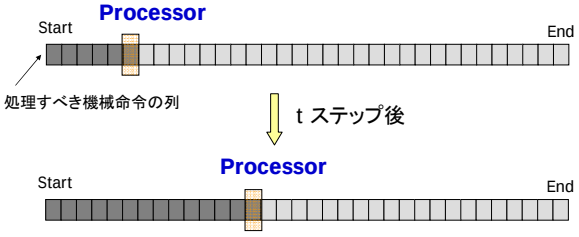
投機的実行 (speculative execution)

- 投機的実行
  - 割込みが発生しないという仮定のもとで，命令をフェッチおよび実行することで高い性能を得る．
- 回復 (recovery, repair)
  - 誤った仮定に基づいて実行された命令の影響を取り消す処理．
- 再開 (restart)
  - 回復の後に，正しい命令列を作り直す処理．

- 正確な割込み
  - 割込みを発生させた命令より前の命令列は完了
  - 割込みを発生させた命令以降を回復，再開

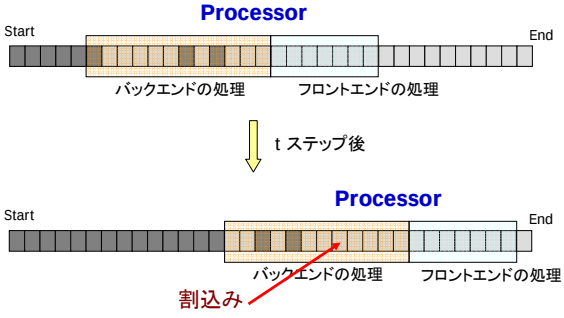
古典的なプロセッサの実行モデル

- 1サイクルに1命令を処理する古典的なプロセッサの実行モデル
- 正確な割込み
  - 割込みを発生させた命令より前の命令列は完了
  - 割込みを発生させた命令以降を回復，再開



高性能プロセッサの実行モデル

- 多数の命令を並列処理する高性能プロセッサの実行モデル



MIPS R3000 Instruction Set Architecture (ISA)

- Instruction Categories
  - Computational
  - Load/Store
  - Jump and Branch
  - Floating Point
    - coprocessor
  - Memory Management
  - Special

Registers

R0 - R31
PC
HI
LO

3 Instruction Formats: all 32 bits wide

OP	rs	rt	rd	sa	funct	R format
OP	rs	rt	immediate			I format
OP	jump target					J format

効率のよい回復とステート情報

- 論理レジスタのみの状態(ステート)を考える.
  - 主記憶は書き込み頻度が低いので異なる技術を利用
- 効率よい回復のため, 少なくとも2組のステート情報が必要
  - 計算に必要なステートの集合
  - 例外発生時に利用するステートの集合

イン・オーダー・ステート (in-order state)

- (5, 7, 8) の命令は処理中, その他は完了している.
- 完了命令だけから成るもっとも長い連続命令列による最新の代入操作によって構成される状態

```
R3 := (1)
R7 := (2)
R8 := (3)
R7 := (4)
R4 := (5)
R3 := (6)
R8 := (7)
R3 := (8)
```

命令列

```
R3 := (1)

R8 := (3)
R7 := (4)
```

イン・オーダー・ステート

先見ステート (lookahead state)

- 最初の未完了命令から命令列の末尾に至るまでの代入操作のすべてから構成されるステート

```
R3 := (1)
R7 := (2)
R8 := (3)
R7 := (4)
R4 := (5)
R3 := (6)
R8 := (7)
R3 := (8)
```

命令列

```
R3 := (1)

R8 := (3)
R7 := (4)
```

イン・オーダー・ステート

```
R4 := (5)
R3 := (6)
R8 := (7)
R3 := (8)
```

先見ステート

アーキテクチャ・ステート (architectural state)

- 現在実行中の命令列の末尾からみて各レジスタに対する最新の代入操作から構成されるステート

```
R3 := (1)
R7 := (2)
R8 := (3)
R7 := (4)
R4 := (5)
R3 := (6)
R8 := (7)
R3 := (8)
```

命令列

```
R3 := (1)

R8 := (3)
R7 := (4)
```

イン・オーダー・ステート

```
R4 := (5)
R3 := (6)
R8 := (7)
R3 := (8)
```

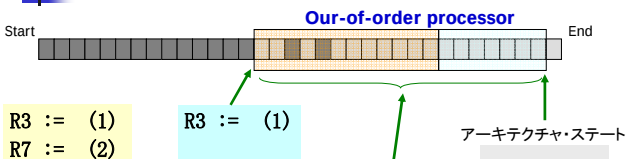
先見ステート

```
R7 := (4)
R4 := (5)

R8 := (7)
R3 := (8)
```

アーキテクチャ・ステート

3つのステートの関係




```
R3 := (1)
R7 := (2)
R8 := (3)
R7 := (4)
R4 := (5)
R3 := (6)
R8 := (7)
R3 := (8)
```

命令列

```
R3 := (1)

R8 := (3)
R7 := (4)
```

イン・オーダー・ステート

```
R4 := (5)
R3 := (6)
R8 := (7)
R3 := (8)
```

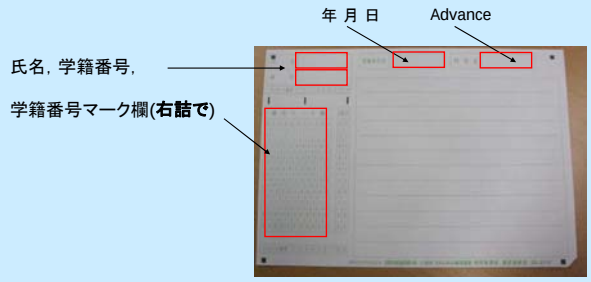
先見ステート

```
R7 := (4)
R4 := (5)

R8 := (7)
R3 := (8)
```

アーキテクチャ・ステート

Exercise



イン・オーダー・ステート，先見ステート，アーキテクチャ・ステートを求めよ

```
R3 :=  (1)
R7 :=  (2)
R8 :=  (3)
R7 :=  (4)
R4 :=  (5)
R3 :=  (6)
R2 :=  (7)
R2 :=  (8)
R7 :=  (9)
```

命令列　　赤色の命令はまだ完了していない

イン・オーダー・ステート，先見ステート，アーキテクチャ・ステートを求めよ

命令列	イン・オーダー・ステート	先見ステート	アーキテクチャ・ステート
R3 := (1)	R3 := (1)		R8 := (3)
R7 := (2)	R7 := (2)		
R8 := (3)	R8 := (3)		
R7 := (4)		R7 := (4)	
R4 := (5)		R4 := (5)	R4 := (5)
R3 := (6)		R3 := (6)	R3 := (6)
R2 := (7)		R2 := (7)	
R2 := (8)		R2 := (8)	R2 := (8)
R7 := (9)		R7 := (9)	R7 := (9)

分岐予測ミスからの回復の例

```
R3 :=  (1)
R7 :=  (2)
R8 :=  (3)
R7 :=  (4)
R4 :=  (5)
R3 :=  (6)
Branch
```

予測ミスしたパス

```
R8 :=  (7)
R3 :=  (8)
```

訂正前

先見ステート	アーキテクチャ・ステート
R4 := (5)	R7 := (4)
R3 := (6)	R4 := (5)
R8 := (7)	R8 := (7)
R3 := (8)	R3 := (8)

訂正後

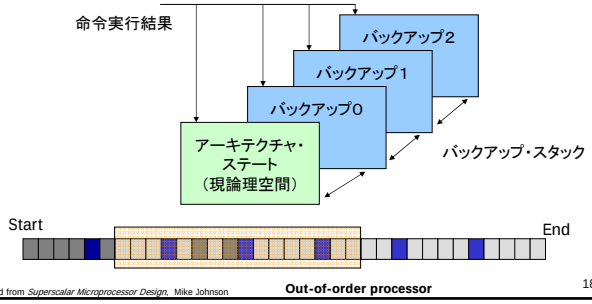
先見ステート	アーキテクチャ・ステート
R4 := (5)	R8 := (3)
R3 := (6)	R7 := (4)
~~R8 := (7)~~	R4 := (5)
~~R3 := (8)~~	R3 := (6)

例外回復のためのバッファリング手法

- チェックポイント回復 (checkpoint repair)
- ヒストリ・バッファ (history buffer)
- リオーダ・バッファ (reorder buffer)
- フューチャ・ファイル (future file)

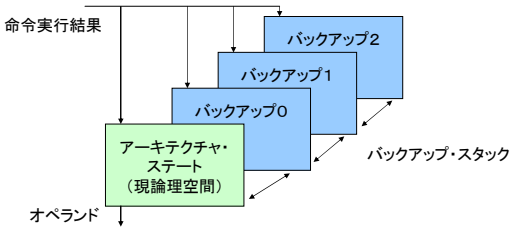
チェックポイント回復 (checkpoint repair)

- 分岐命令のたびにアーキテクチャ・ステートのバックアップを生成
- 命令が完了するたびにすべての空間を更新，バックアップのすべてのレジスタが更新されるとインオーダ・ステートとなる．
- 例外が発生した場合には，バックアップの内容をコピーして再開



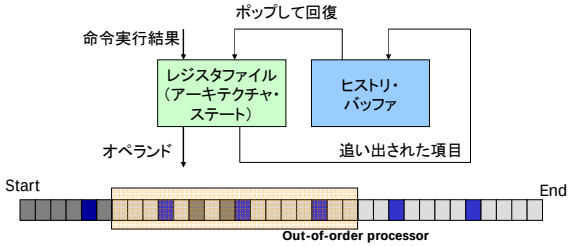
チェックポイント回復 (checkpoint repair)

- 短所
  - ステートのコピー(バックアップの作成)に長い時間を要する.
  - 記憶量が多い.
  - 先見能力に比例してバックアップの数が増大する.



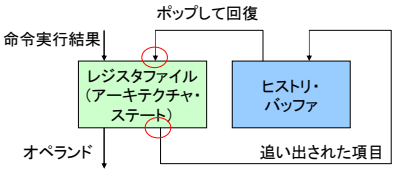
ヒストリ・バッファ (history buffer)

- レジスタファイルにはアーキテクチャ・ステートを保存
- スタックの構造を持つヒストリバッファ（先見ステートを保持）
  - 命令がデコードされるとプッシュされる.
  - 例外時に，例外命令までポップして回復
  - インオーダ・ステートに含まれる命令はバッファから削除



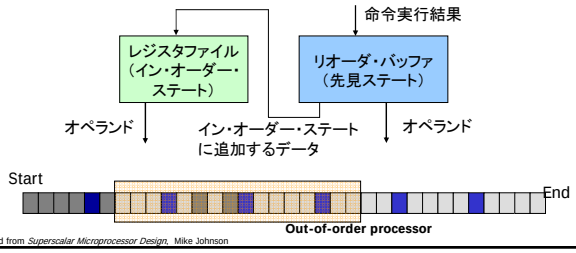
ヒストリ・バッファ (history buffer)

- 短所
  - レジスタファイルのポート数の増加（性能向上に寄与しない）
  - 回復するための時間が長い．先見能力の向上に伴ってサイクル数が増加する.



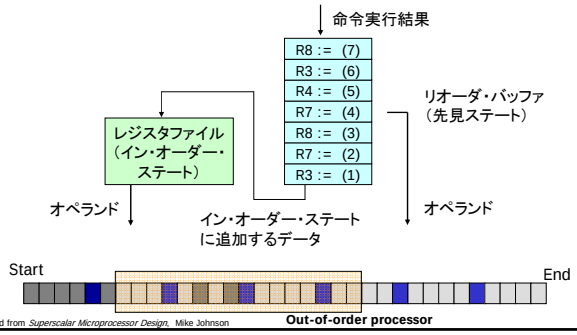
リオーダ・バッファ (reorder buffer, ROB)

- イン・オーダー・ステートと先見ステートを結合して，アーキテクチャ・ステートを得る.
- FIFOキュー構造をもつリオーダ・バッファ
  - 命令がデコードされるとエンキューする.
  - インオーダ・ステートに含まれる命令をデキューする.
  - 例外時には，その命令以降のエントリをリオーダ・バッファから破棄する.

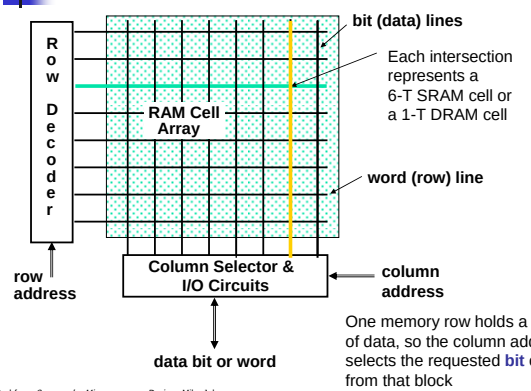


リオーダ・バッファ (reorder buffer, ROB)

- リオーダ・バッファには，優先度付きの連想検索の機能が必要



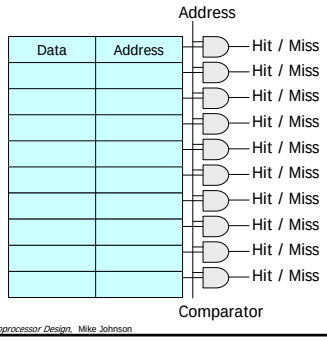
Classical RAM (random access memory) Organization



One memory row holds a block of data, so the column address selects the requested **bit** or **word** from that block

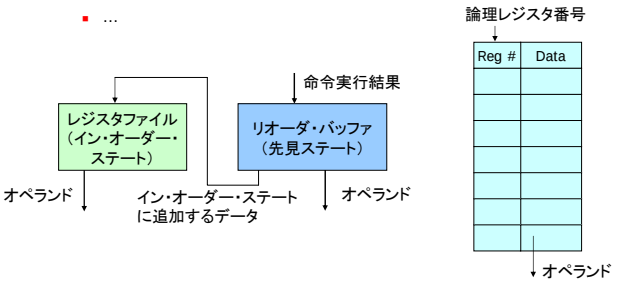
連想メモリ, CAM (content addressable memory)

- アドレスを入力として，連想メモリは全内容からそのアドレスと一致するもの検索して，対応するデータを出力する．



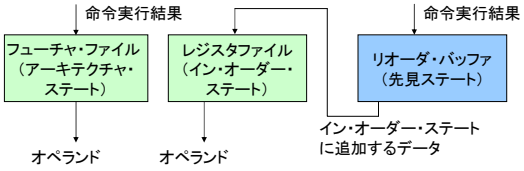
リオーダ・バッファ (reorder buffer, ROB)

- 短所
  - 連想検索の機能が必要となる．エントリ数の増大が困難．
- 長所
  - …



フューチャ・ファイル (future file)

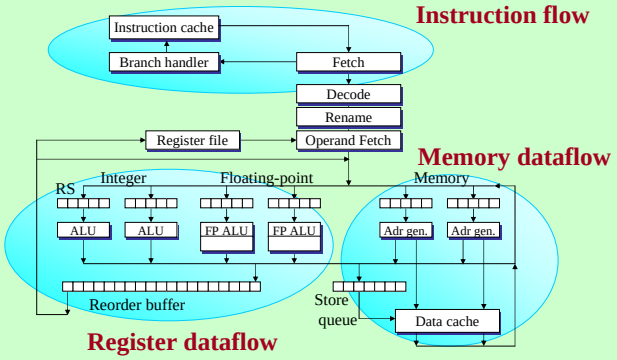
- リオーダバッファはオペランドを供給しない．
- レジスタファイルと同様の構成のフューチャ・ファイルを追加する．
  - フューチャ・ファイルにはアーキテクチャ・ステートを保存
- リオーダバッファの連想検索を排除できる．
- 例外の場合には，その時点までのインオーダステートを構築して，フューチャ・ファイルの内容を無効化する．



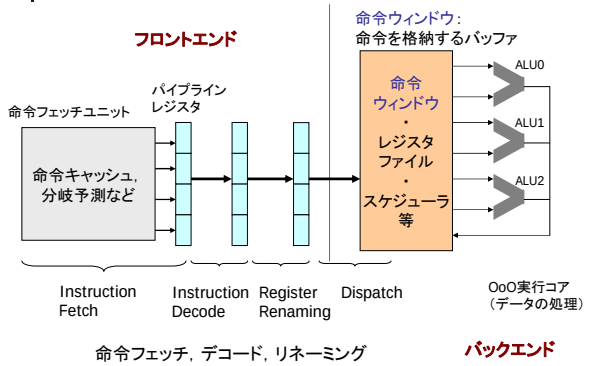
例外回復のためのバッファリング方法 まとめ

- チェックポイント回復
  - アーキテクチャ・ステートをバックアップ
- ヒストリ・バッファ
  - ヒストリバッファに先見ステートの履歴を格納
  - レジスタファイルにアーキテクチャ・ステートを格納
- リオーダ・バッファ
  - レジスタファイルにイン・オーダー・ステートを格納
  - リオーダ・バッファに先見ステートを格納
- フューチャ・ファイル
  - レジスタファイルにイン・オーダー・ステートを格納
  - リオーダ・バッファに先見ステートを格納
  - フューチャ・ファイルにアーキテクチャステートを格納

Out-of-orderスーパースカラ・プロセッサ

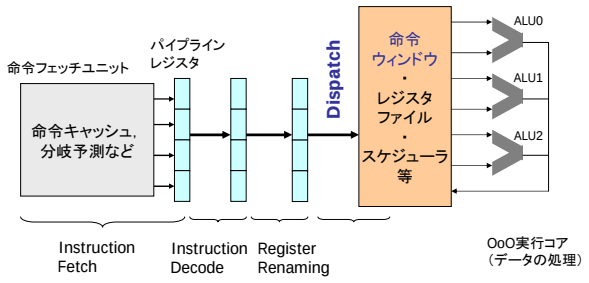


アウトオブオーダ実行プロセッサの構成



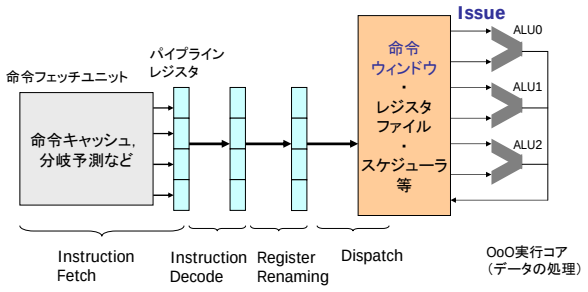
アウトオブオーダ実行プロセッサの構成

- ディスパッチ (dispatch)：命令ウィンドウに命令を格納する動作



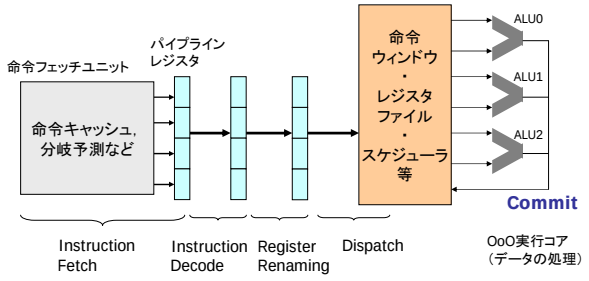
アウトオブオーダ実行プロセッサの構成

- 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．



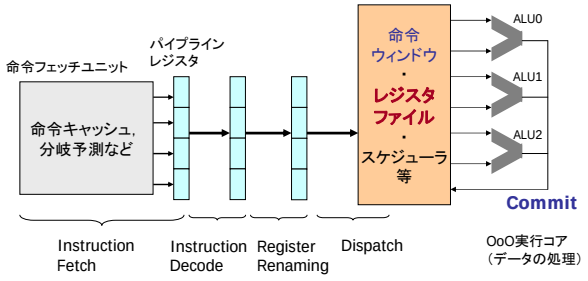
アウトオブオーダ実行プロセッサの構成

- コミット (commit)：命令の実行結果によってアーキテクチャ・ステートを更新する動作（リオーダバッファに値を書き込む）



アウトオブオーダ実行プロセッサの構成

- リタイア (retire)：命令の実行結果によってインオーダ・ステートを更新する動作
  - 割り当てた物理レジスタを回収



詳細は12月9日のスライド

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

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