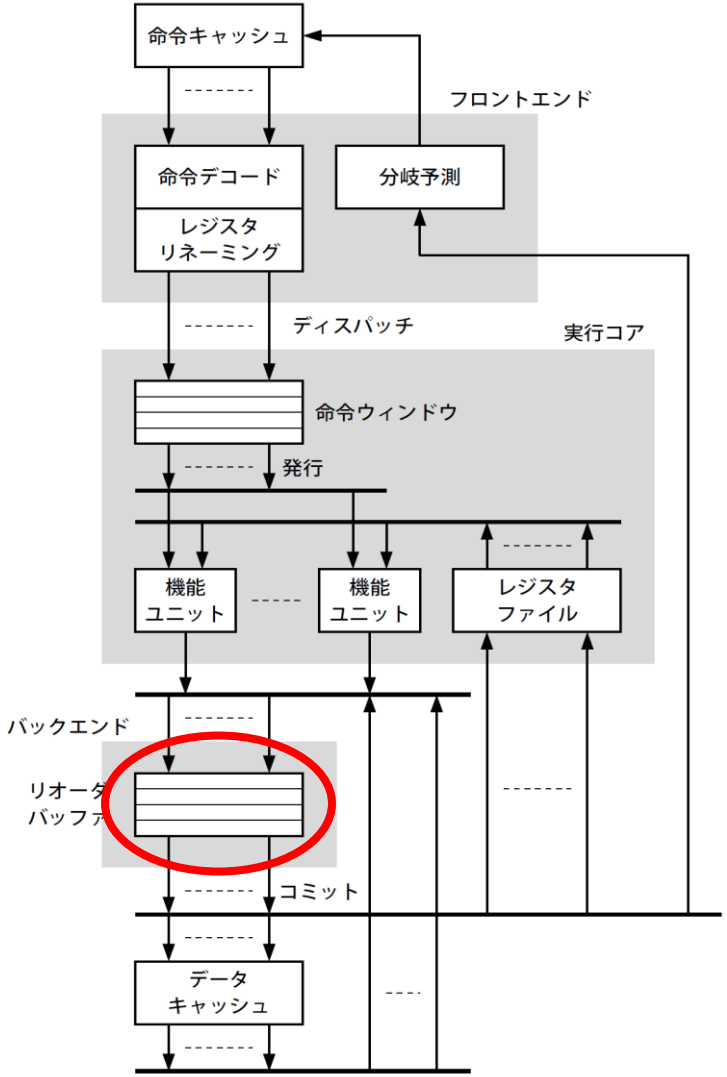
内容

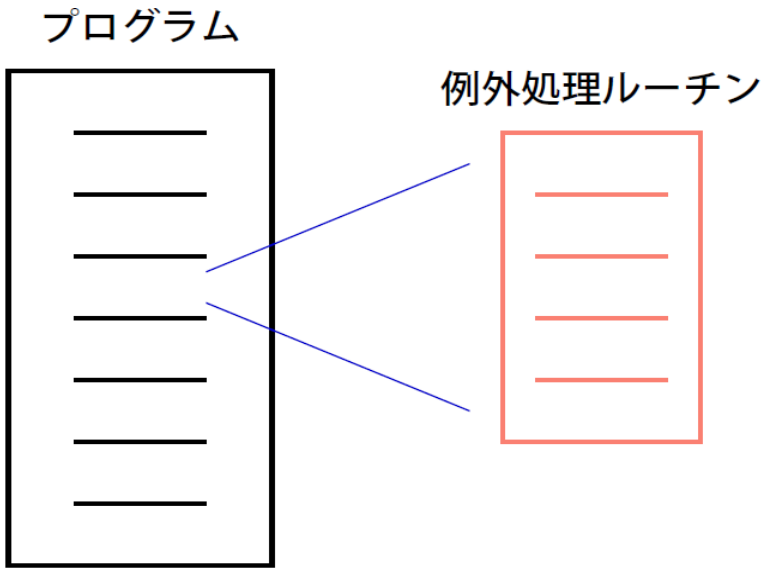
- リオーダ・バッファ

  [Smith 1988]
  - 例外とは
  - 正確な例外
  - 方式



例外

- プログラムにはない何らかのイベント
  - I/Oデバイス要求
  - 算術オーバーフロー
  - 未定義命令検出
  - ページ・フォールト
  - メモリ保護違反
  - OSの何らかのサービス
  - …
- 対処方法
  - 検出　by HW
  - 処理　by OS/HW
  - 停止または再開　by OS



例外処理手順

- 検出
  - プログラムの実行停止
  - 原因を記録（特別なレジスタ：e.g., MIPS cause register)
  - PCの退避（e.g., MIPS EPC←PC)

- 処理
  - 処理開始のルーチンにジャンプ
  - レジスタ保存
  - 原因を見て、さらに処理ルーチンへジャンプし、必要なら処理

例外処理手順(cont'd)

- 停止または再開
  - 停止：OSがプログラムを停止
  - 再開：
    - レジスタ回復
    - PC←EPC
    - プログラムの実行再開

正確な例外

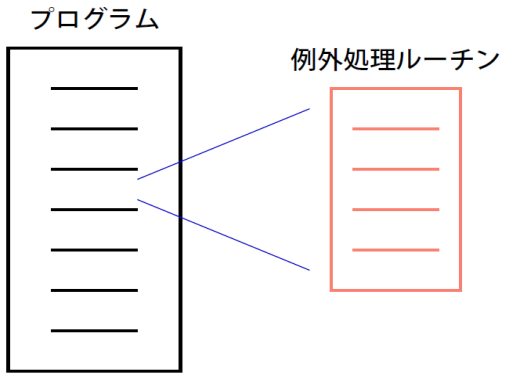
- 正確な例外

レ	i0:…
レ	i1:…
レ	i2:r1=load r1
E	i3:r2=load r3
	i4:…
	i5:r3=r4<<2
	i6:…

- 不正確な例外

レ	i0:…
	i1:…
レ	i2:r1=load r1
E	i3:r2=load r3
	i4:…
レ	i5:r3=r4<<2
	i6:…

正確な例外は必要

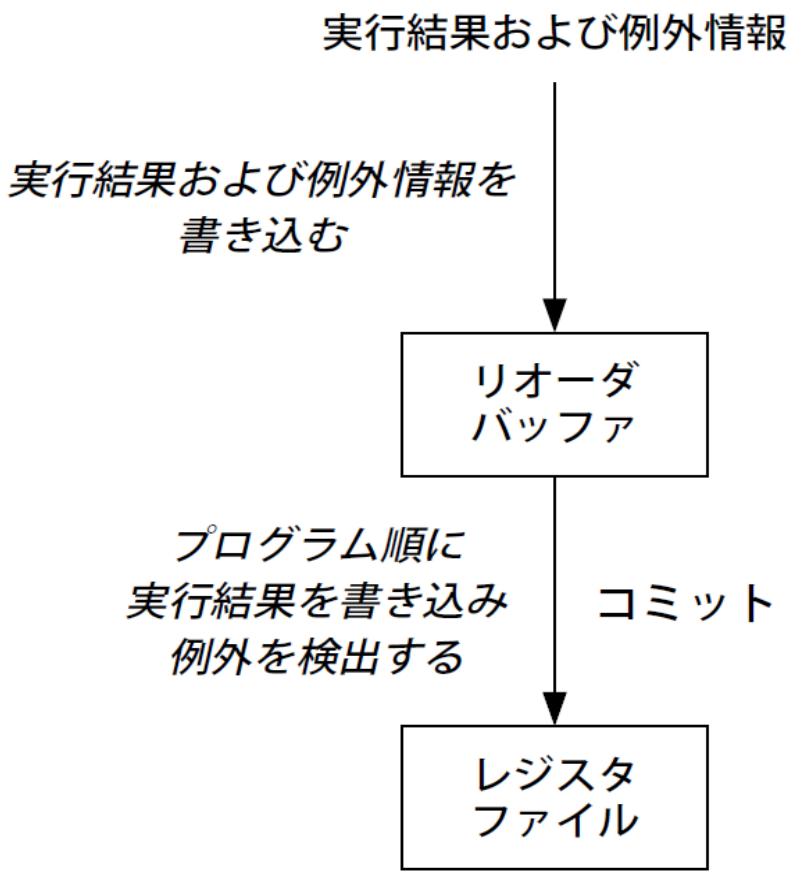


- 逐次実行モデル
  - プログラムが仮定している処理モデル
    - プロセッサの実行方式とは独立
  - 例外は、命令と命令の間に挿入された手続き
    - 例外命令を含め実行未終了の全ての命令を無効化
    - 例外処理
    - 例外命令から再開
  - 例外が生じても、逐次実行モデルは保たれなければならない

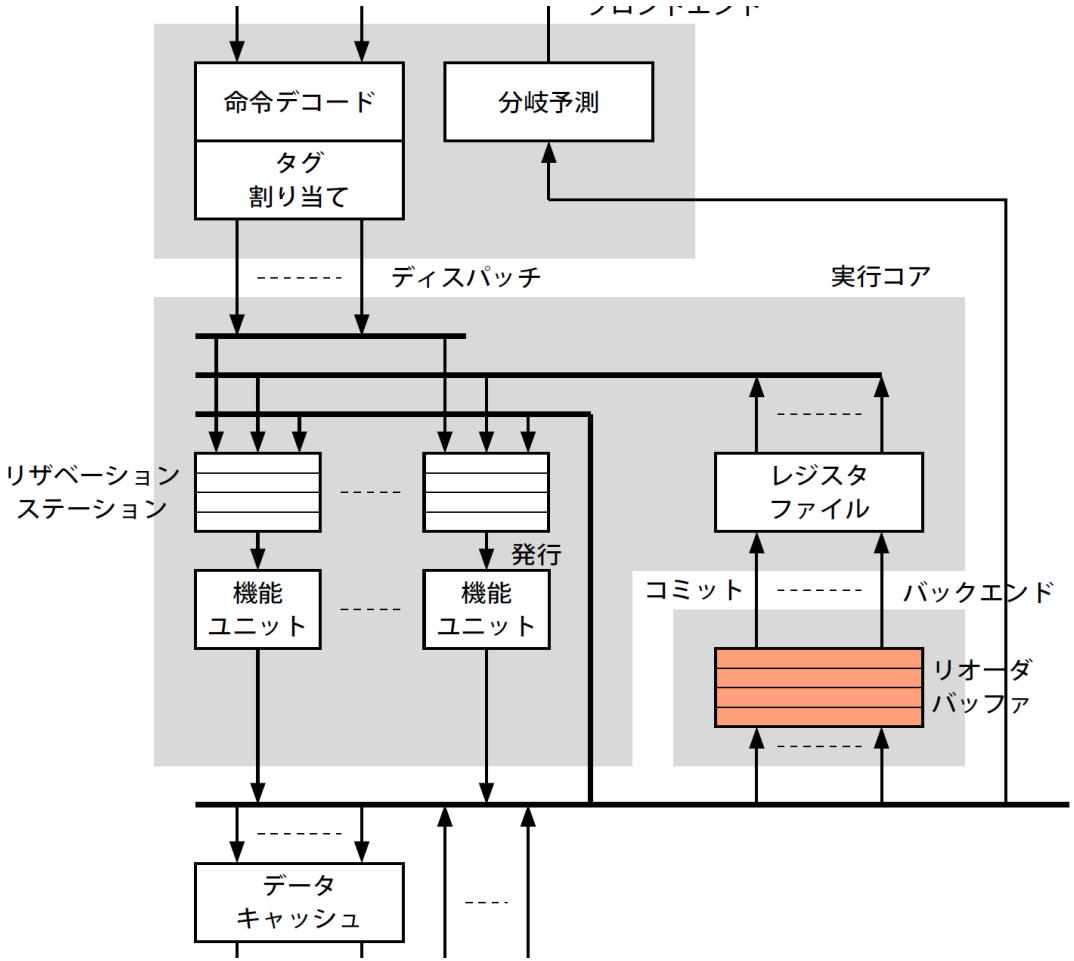
- アウト・オブ・オーダ実行
  - 不正確な例外
  - 未実行命令のみ実行は複雑
  - 逆依存
    - 希にしか起こらない例外に備えて、新たな制約が課せられて損

リオーダ・バッファ

- アウト・オブ・オーダ実行＋正確な例外
- プログラム順に状態を更新する
  - インオーダ・コミット
- 状態更新時に例外を処理
  - 正確な例外

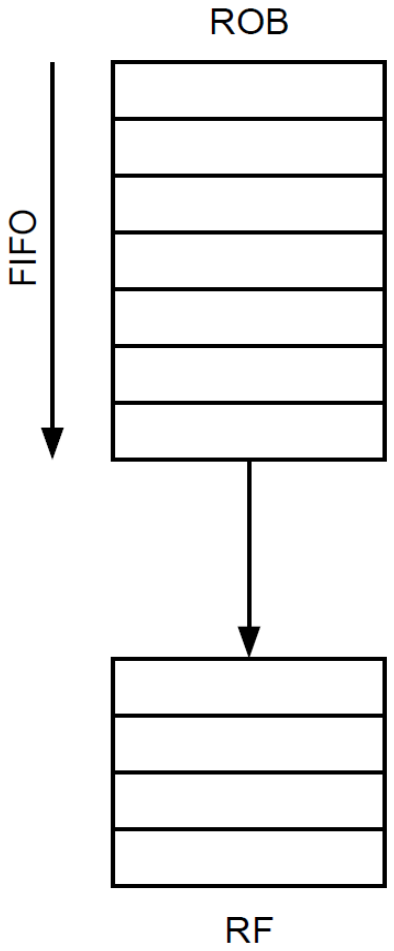


リオーダ・バッファの追加



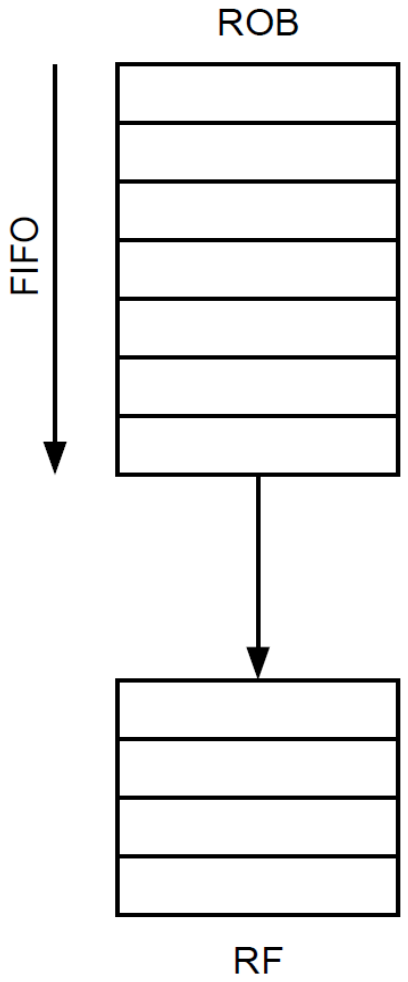
リオーダ・バッファの動作

- FIFO
  - プログラム順に命令にエントリを割り当て
    - フロントエンドにて
    - コミット順が容易にわかる
- 実行終了
  - 対応するエントリに書き込み
    - 実行結果
    - 例外：例外原因コード書き込み
  - アウト・オブ・オーダ



リオーダ・バッファの動作(cont'd)

- コミット(先頭のエントリから順に)
  - 結果あり && 例外なし
    - 正常な実行終了
    - 結果をレジスタに書き込み
  - 例外あり
    - パイプラインとリオーダ・バッファを無効化
    - 例外処理
    - (必要なら)実行再開
  - 結果なし (&& 例外なし)
    - 実行が未完了
    - 待つ



リオーダ・バッファの詳細

- エントリの内容
  - PC　：命令アドレス
  - R　：結果あり（レディ）
  - dreg　：デスティネーション・レジスタ番号
  - dtag　：デスティネーション・オペランド・タグ
  - result：結果
  - E　：例外原因コード

命令デコード時

- エントリ割り当て
  - 末尾エントリ: robtail
  - ROB[robtail].PC = inst.PC
  - ROB[robtail].R = 0
  - ROB[robtail].dreg = inst.dreg
  - ROB[robtail].dtag = inst.dtag

- 割り当てられたエントリ番号を持って、命令はディスパッチ
  - inst.robentry = robtail

実行終了時

PC	R	dreg	dtag	E	result

- 書き込み
- ROB[inst.robentry].R = 1
- ROB[inst.robentry].result = instの実行結果
- ROB[inst.robentry].E = instの例外コード（例外を起こしたら）または0（正常終了）

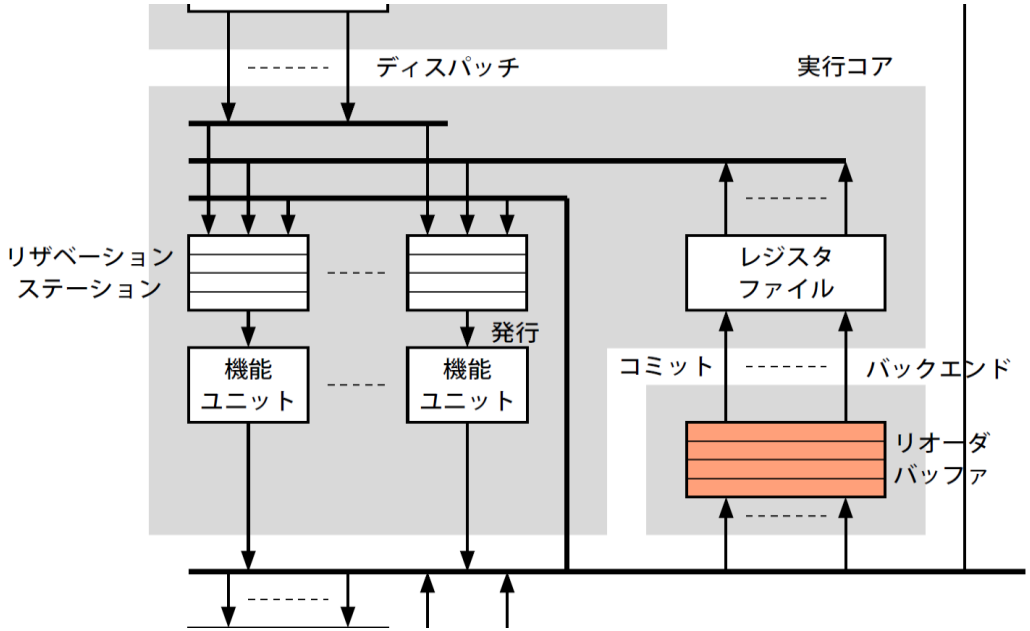
コミット

PC	R	dreg	dtag	E	result

- コミット
  - 先頭: robhead
  - ROB[robhead].R == 0 → 待つ
  - ROB[robhead].R && ROB[robhead].E==0
    → 結果をレジスタに書き込む
    - Reg[ROB[robhead].dreg] = ROB[robhead].result
  - ROB[robhead].R && ROB[robhead].E → 例外処理
    - EPC = ROB[robhead].PC
    - ROBの無効化 → ROB[*i*].R=0, foreach entry *i* of ROB
    - パイプライン無効化
    - OSに制御を渡し、例外処理（ROB[robhead].Eを参照）

ソース・オペランドのフェッチ

- ROBの状態はレジスタ・ファイルより新しい
  - オペランド値
  - オペランド・タグ
    - レジスタ・ファイルにタグを保存する必要はない



ROBからオペランド値・タグ読み出し

- 連想検索
  - sreg：命令のソース・レジスタ番号
  - foreach entry *i* of ROB (ROBの末尾から先頭に向かって){

```
if (ROB[ i ].dreg == sreg) {
  if (ROB[ i ].R == 1)
      ROB[ i ].resultを読み出す;
  else
      ROB[ i ].dtagを読み出す;
  break;
}
}
```

末尾

PC	R	dreg	dtag	E	result

先頭

オペランド値・タグ読み出し（優先順）

1. 同時デコード命令間
   - 依存チェック：ランダム論理
   - デスティネーション・オペランド・タグをソース・オペランド・タグとする

2. ROB連想検索
   - 値またはタグ

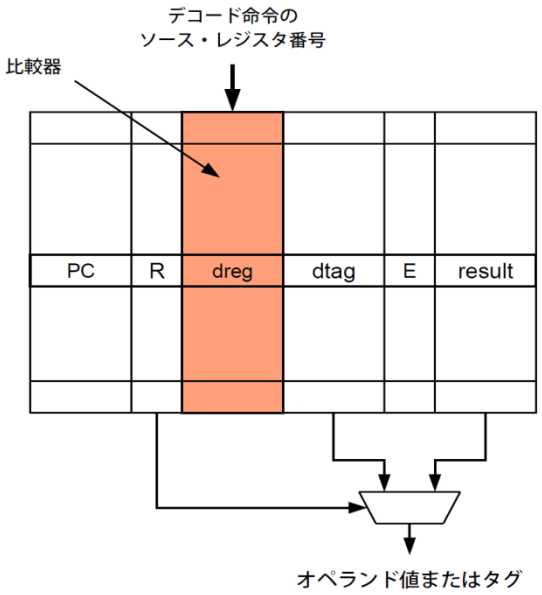
3. レジスタ・ファイル参照
   - 値

リオーダ・バッファ検索の複雑さ

- 複雑
  - 連想検索
  - 優先度付き

優先度付き検索

- 連想メモリ: CAM (contents addressable memory)



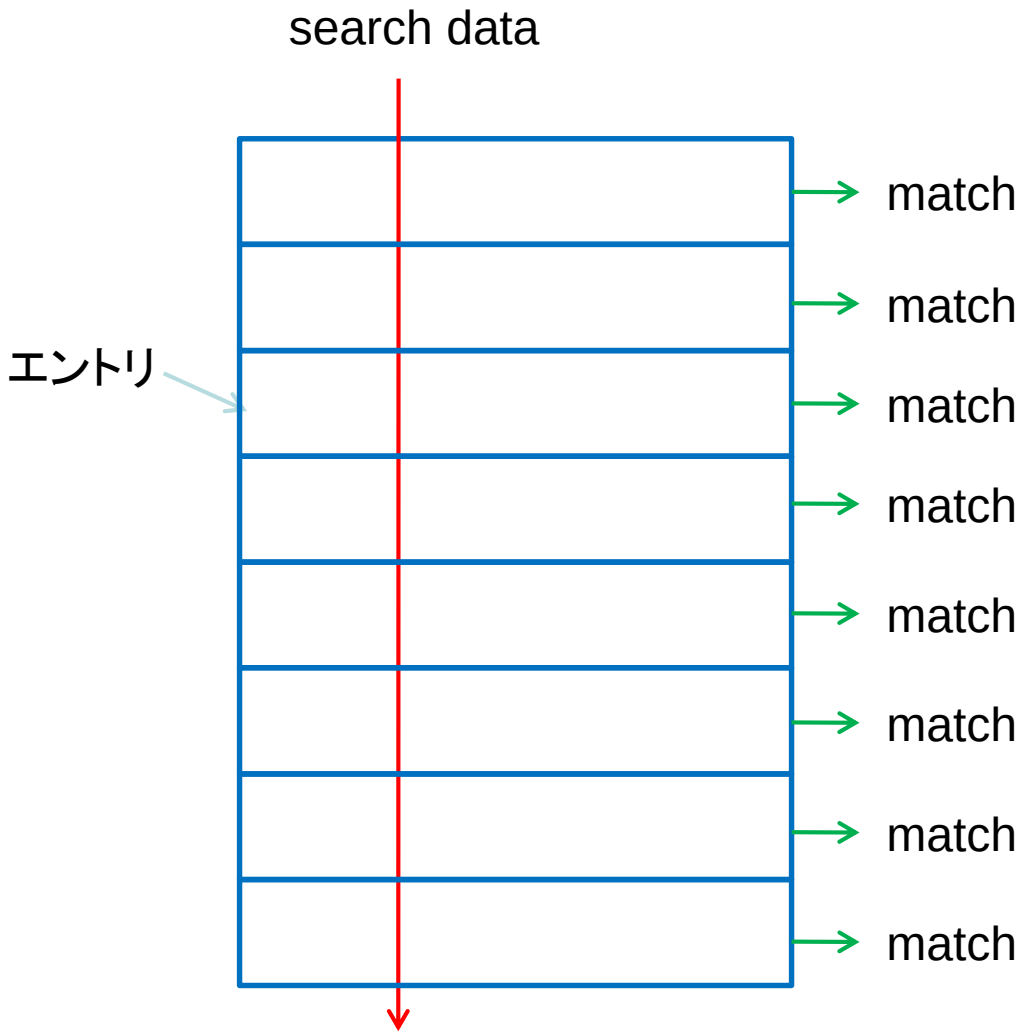
- 優先度: (*m*-1)入力NOR, *m*:ROBエントリ数
  - *m*: インフライト命令数（e.g., 200)

CAM

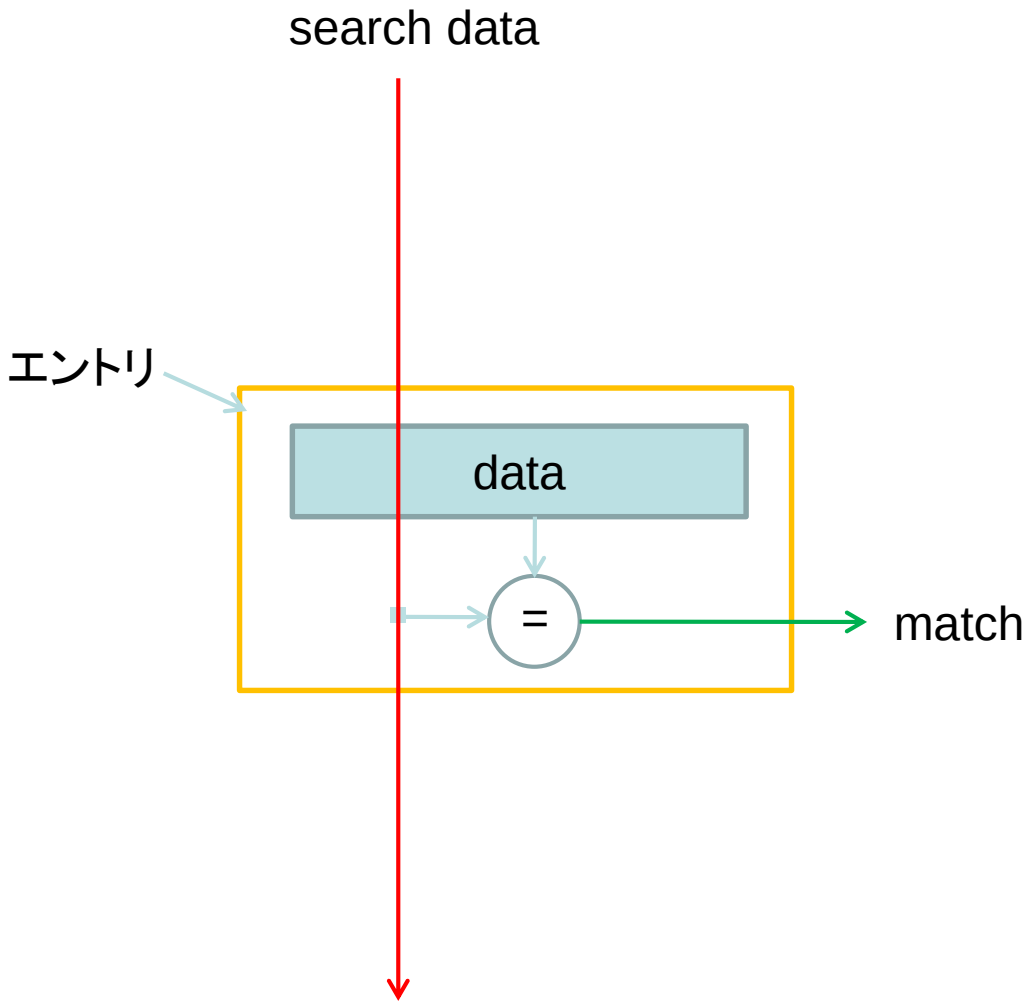
- 連想検索を実現するハードウェア

- CAM: contents addressable memory
  - データが一致するエントリを見つける
  - 各エントリに比較器が入っている
  - 大きく、遅く、大きな電力を消費するが、小さな容量なら許容範囲

CAMの論理（入出力）

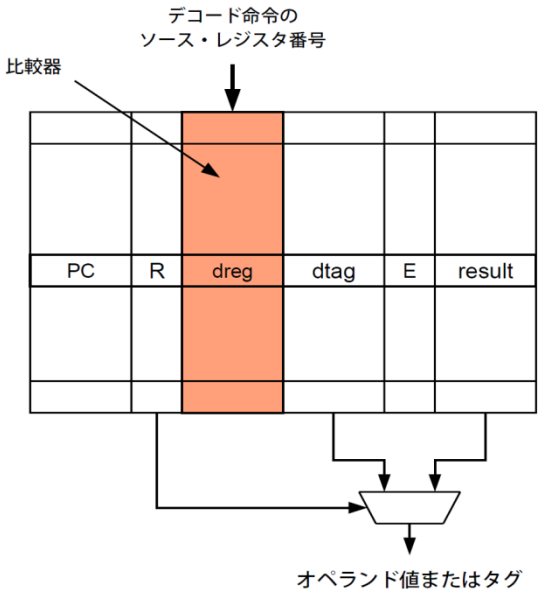

search data
エントリ
match
match
match
match
match
match
match
match

CAMの論理（エントリ）



優先度付き検索

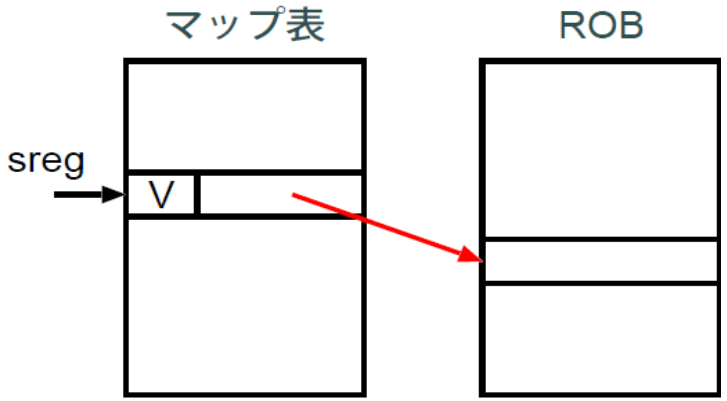
- 連想メモリ: CAM (contents addressable memory)



- 優先度: (*m*-1)入力NOR, *m*:ROBエントリ数
  - *m*: インフライト命令数（e.g., 200)

複雑さの緩和

- 「レジスタ→ROBエントリ」の最新の対応を保持する表
  - 連想検索を表参照に変換
    - 表に対応が記録されていれば、その対応
    - なければ、RFを読み出し
  - 表の維持
    - ROB割り当て時に、対応を登録
    - コミット時に、自分の対応が登録されていれば、表から削除



レジスタ・リネーミング

- 論理レジスタから物理レジスタへマッピング
  - 論理レジスタ
    - プログラマから見えるレジスタ
    - プログラムに書かれたレジスタ
  - 物理レジスタ
    - 実際に記憶しているレジスタ
    - レジスタ・ファイル＋ROB

まとめ

- リオーダ・バッファ
  - アウト・オブ・オーダ実行で正確な例外
    - プログラム順にエントリ割り当て
    - アウト・オブ・オーダで結果書き込み
    - プログラム順でコミット
  - オペランドまたはタグの供給
    - 連想検索
    - 複雑