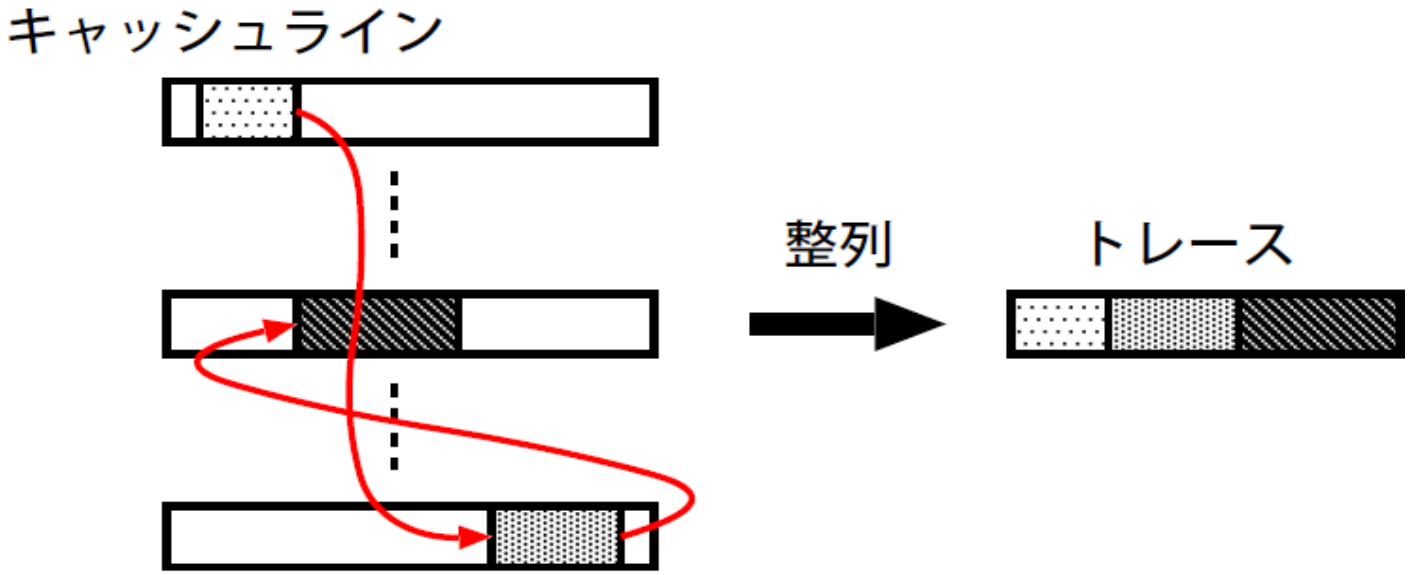
内容

- 高バンド幅命令フェッチ
  - トレース・キャッシュ
- 投機的実行
  - 定義と問題
  - リオーダ・バッファ支援
  - レジスタ・リネーミングとの関係
- Intel Pentium 4

命令フェッチ・バンド幅制約

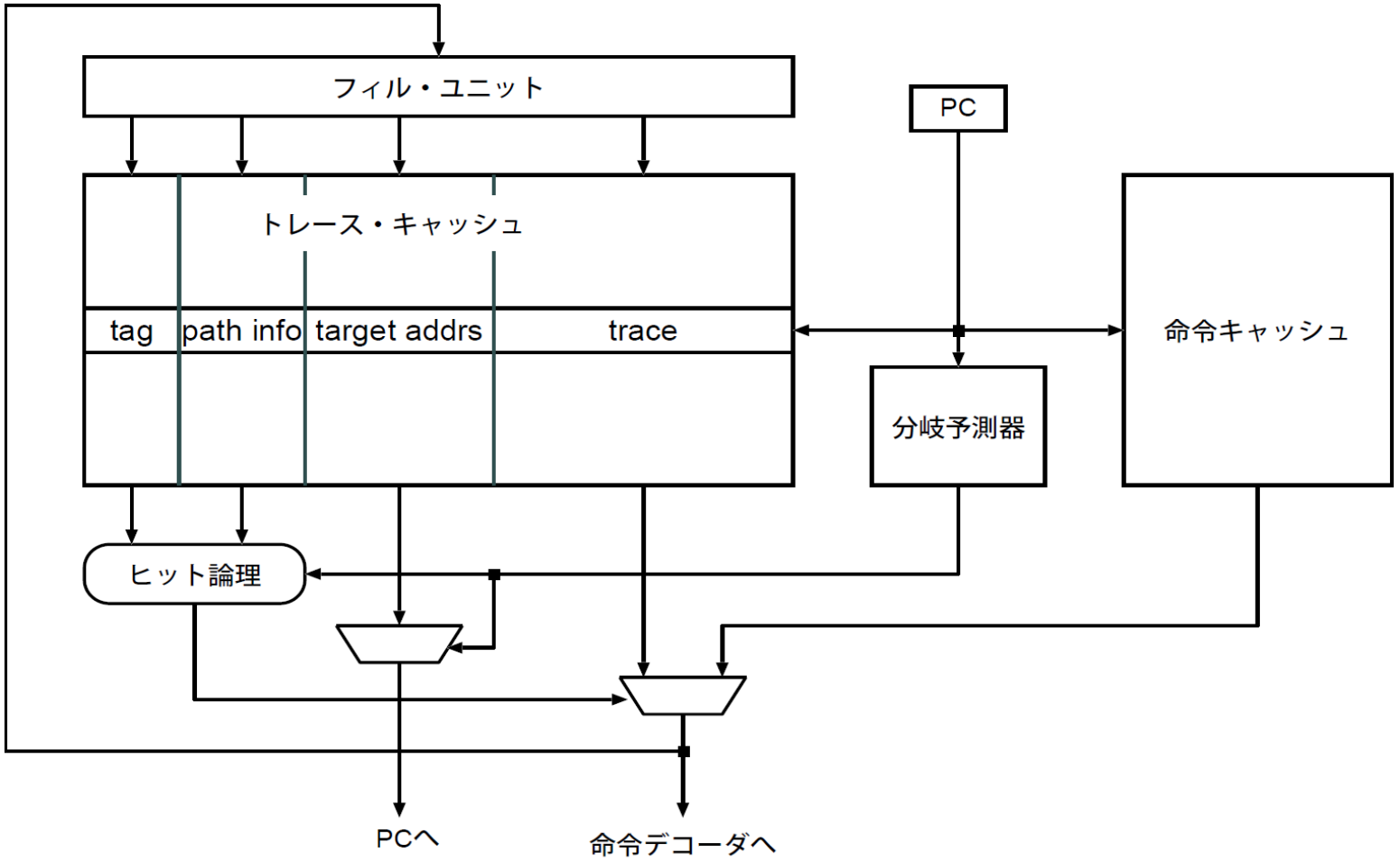
- 非整列データの命令キャッシュからのフェッチ
  - 命令キャッシュからは連続データしか読み出せない
    - ライン（ブロック）
  - 実行すべき命令は必ずしも連続に並んでいない
    - 分岐
  - とびとびのデータの読み出し方法？
    - 命令キャッシュのマルチポートと整列化論理→複雑
    - 整列化データを保持するキャッシュ：トレース・キャッシュ

非整列データの整理



- 整列化は困難
- あらかじめ整列させた命令例(トレース)を保持

トレース・キャッシュ


フィル・ユニット
PC
トレース・キャッシュ
tag
path info
target addrs
trace
命令キャッシュ
分岐予測器
ヒット論理
PCへ
命令デコーダへ

トレース・キャッシュ(1)

- フィールド
  - trace　　: トレース
  - tag　　: タグ（キャッシュのタグと同じ）
  - target addrs : トレースの次の2つのブロックのアドレス
  - path info　　: トレース内分岐の方向
- 長所
  - 比較的単純
- 短所
  - 高い面積コスト
  - 基本ブロックの多重記録による空間的な不効率性
    - 1つの基本ブロックに到達するパスは複数ある

トレース・キャッシュの動作

- トレース・キャッシュからトレースを読み、同時に分岐予測を行う
- トレース内分岐結果（path info）と一致していれば、フェッチしたトレースは有効
- 一致していなければ、
  - トレースは捨てる
  - 命令キャッシュを読む（または、TCと同時に読んでおいて選択する）
  - フィルユニットで結合し、トレースをつくり、TCに書き込む

トレース・キャッシュ：コメント

- TCにデコード済みの命令を入れておけば、命令デコードを省ける
  - TCではないが、以後のIntelのプロセッサで、採用されている
    - 多くのプロセッサでは、少しデコードしたコードを入れている
  - x86: μOPキャッシュ
  - 再利用されれば、複雑な命令のデコードが省略できる
    - 電力削減

- TC: Intel Pentium 4で採用された
  - 以後、採用されていない

フェッチ速度向上手法

- 分岐位置ヒント
  - デコードせずに分岐予測を行える⇒早期フェッチ切替可能
- フェッチの後方に比較的長いキュー

内容

- 高バンド幅命令フェッチ
  - トレース・キャッシュ
- 投機的実行
  - 定義と問題
  - リオーダ・バッファ支援
  - レジスタ・リネーミングとの関係
- Intel Pentium 4

投機的実行とは

- 定義
  - 実行すべきかどうかわかる前に行う実行
    - 何らかの先行制約を破る
    - 並列性増加
  - 一般には、予測に基づいて実行
    - 予測がはずれれば、その命令以降を無効化し、再実行

- 例
  - 分岐予測によりフェッチした命令を、分岐結果がわかる前に実行
    ⇒制御依存の緩和
  - 依存がないと予測したメモリ命令を、依存の有無がわかる前に実行
    - LSQでは、ロードは前方のストアのアドレスが判明していなければならなかった

    ⇒データ依存の緩和

投機的実行の問題(1)

- プログラムの意味の維持
  - 分岐結果がわかる前に、実行結果をプロセッサ状態（メモリやレジスタ）に反映する
  - 予測を誤ると、プログラムの意味が維持できない

```
i1:   branch LABEL if r1 == 1
i2:   r2 = load 0(r3)
   LABEL:
i3:   r4 = r2 + r3
```

← 投機的実行。r2が上書きされる

← 誤った実行結果となる

  - 生きている変数を投機的実行により上書きしてはならない

投機的実行の問題(2)

- 例外処理
  - 分岐結果がわかる前に、発生した例外（投機的例外）を処理する
  - 不正にプログラムを終了させるか、性能を大きく低下させる

```
i1:   branch LABEL if r1 == 1
i2:   r2 = load 0(r3)  ← 投機的実行（例外発生）
   LABEL:
i3:   r4 = r2 + r3
```

  - 投機的例外処理は必要かどうかわからない
    - ページ・フォールを不必要に処理すると、数百万サイクルを失う

投機的実行の問題への対処法

- 問題を起こすような投機的実行はしない
  - 明らかに大きな性能低下
  - 例：ロード/ストア命令の出現頻度は30%くらい

- コミット延期
  - 予測の正誤がわかり、投機的でなくなるまで、書き込みや例外処理を延期

```
i1:   branch LABEL if r1 == 1
i2:   r2 = load 0(r3)   ←投機的実行
   LABEL:
i3:   r4 = r2 + r3
```

リオーダ・バッファによる解決法

- コミット条件の修正
  - 例外処理と同様の方法
  - 投機命令は、予測が正しいことがわかったときのみコミット
  - 予測を誤れば、自身と後続命令を無効化

- 予測を行った命令に対応するROBエントリ
  - P: 予測に基づく実行かどうかのフラグ
  - C: 予測の正誤
  - result: 実行結果が得られるまでは、予測結果

動作

- 分岐のROB割り当て時
  - P = 1
  - result = 予測値
- 実行終了時
  - P = 0
  - 実行結果と予測値を比較
    - 正しければ、C = 1
    - 誤りならば、C = 0

動作(cont'd)

PC	P	C	R	dreg	dtag	E	result

- 分岐のコミット時
  - P = 1 : 待つ
  - P = 0 :
    - C = 1 : コミット
    - C = 0 : 後続命令を全て無効化
      依存していた全ての命令はコミットされない
- 最適化
  - 問題 : ROBでの待ち合わせによる分岐予測ミス・ペナルティの増大
  - 解決 :
    - 分岐命令に予測結果を持たせる
    - ミスが判明すれば、ROBの対応するエントリ以降を無効化
    - 複雑化

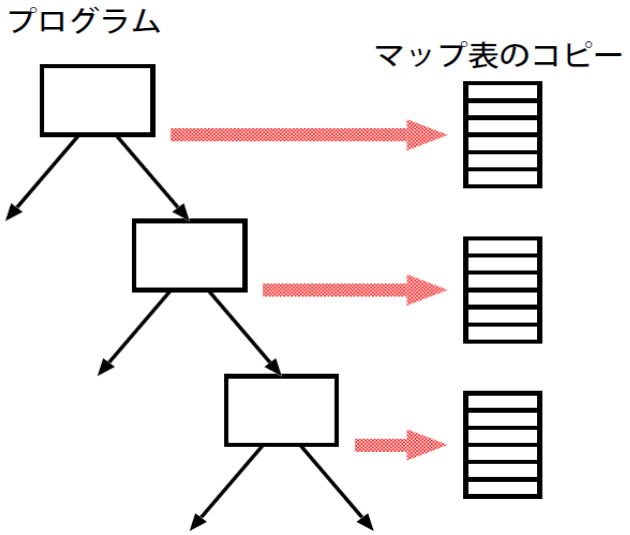
レジスタ・ファイルを使ったリネーミングの場合

- 巻き戻し方式
  - 予測ミス・ペナルティの著しい増加
    ⇒不可

- リタイアメント・マップ表
  - 分岐がコミットされる時に予測ミスと判明したら、
    - リタイアメント・マップをフロントエンド・マップにコピー
    - Intel Pentium 4
  - 分岐がROBの先頭に来るまで待ち合わせ
    ⇒予測ミス・ペナルティの増加

チェックポイント回復

- マップ表のコピー（MIPS R10000)
  - 分岐毎にコピー
  - 予測ミスしたら、その分岐に対応するコピーをマップ表に戻す
  - 予測ミス・ペナルティの最小化
  - マップ表に広バンド幅要求：
    - 回路的に解決



マップ表のチェックポイントの回復回路

- SRAMを複数並べる
  - SRAM間のデータ･コピーが必要
  - 遅い、電力を消費する

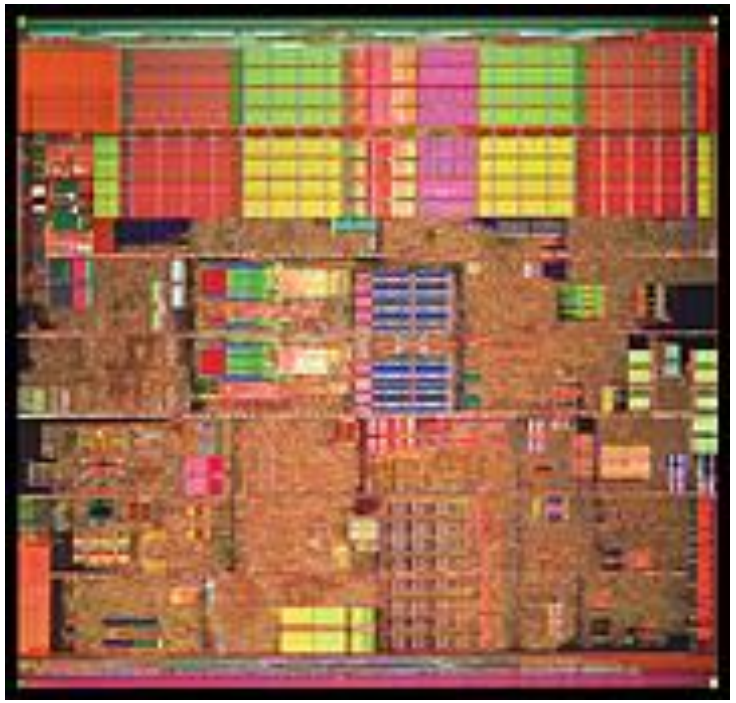
- 複数のSRAMセルを隣接して配置した特別なSRAM
  - 隣接するセル間でデータをコピー
  - 速い
  - マップ表の面積が大きくなり、
    - 遅い、電力を消費する

ここまでのまとめ

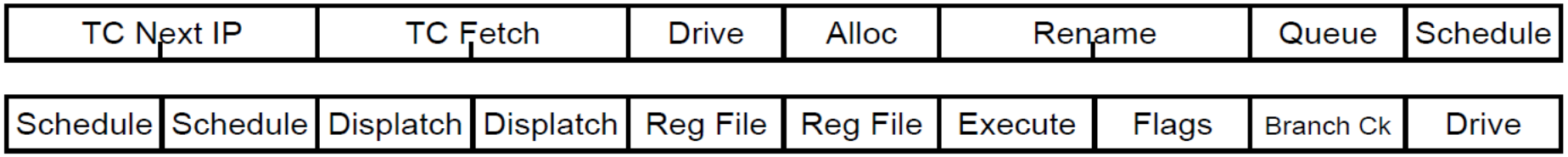
- 高バンド幅命令フェッチ
  - トレース・キャッシュ
- 投機的実行
  - 実行してよいかどうかわかる前に実行
    - 制約緩和による並列性の向上
  - 問題：プログラムの意味の維持と例外処理
  - 対処
    - リオーダ・バッファの修正
    - リタイアメント・マップ
    - チェックポイント回復

Intel Pentium 4

- 歴史
  - 2000年初出荷
  - 電力制約で製造中止
  - 非常に野心的
  - 後継機に大きな影響
- 第1世代チップ緒元
  - 0.18μm CMOS, 6層Al
  - 217mm^2
  - 42Mトランジスタ
  - 1.5GHz, 55W



パイプライン

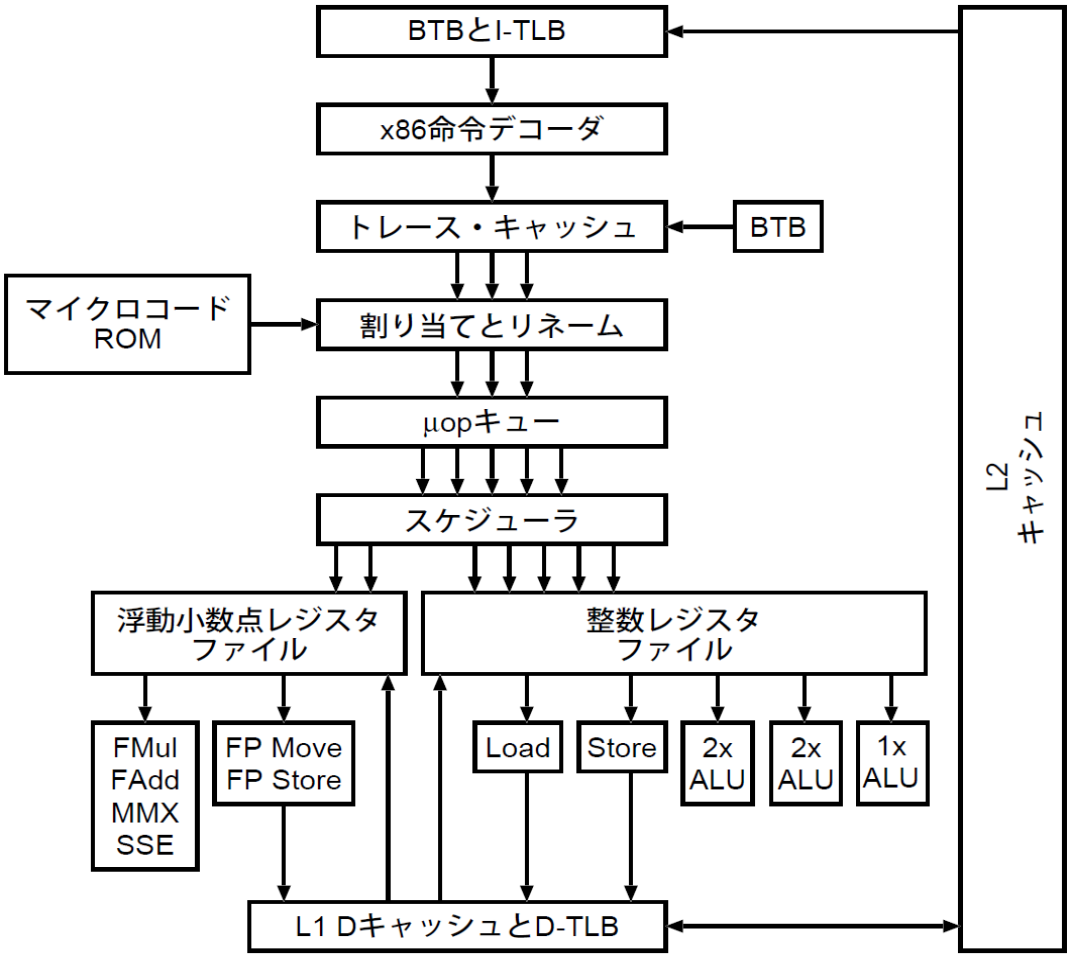


- 非常に深いパイプライン
  - 実行コアを抜けるまで20段
    - P6: 10段
    - P5: 5段
  - クロック速度重視

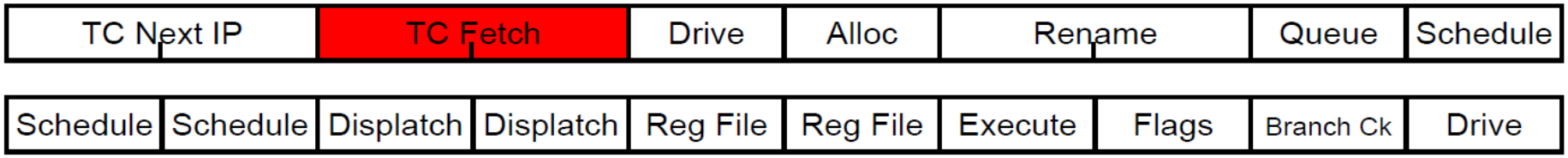
アーキテクチャの特徴

- x86命令をRISC命令μopに変換
- 126エントリのROB
- 128エントリのレジスタ・ファイル
- 6μopの命令発行幅
- 倍速動作ALUによる依存命令の同時実行
- 12Kμopのトレース・キャッシュ

アーキテクチャ

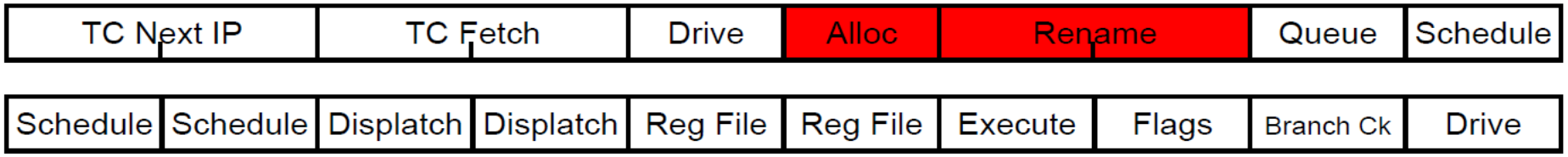

BTBとI-TLB
x86命令デコーダ
トレース・キャッシュ
BTB
マイクロコード
ROM
割り当てとリネーム
μopキュー
スケジューラ
浮動小数点レジスタ
ファイル
整数レジスタ
ファイル
FMul
FAdd
MMX
SSE
FP Move
FP Store
Load
Store
2x
ALU
2x
ALU
1x
ALU
L1 Dキャッシュ
とD-TLB
L2
キャッシュ

トレース・キャッシュ



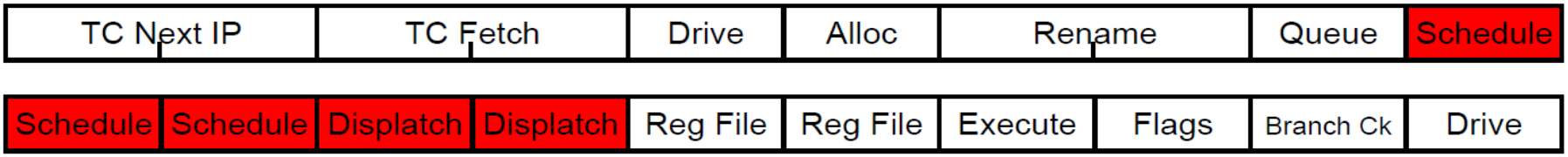
- L1キャッシュなし
- μopをキャッシュ
- TCミス時に、x86命令をμopにデコード
  - 4μop以下の命令のみ
  - それ以上になる命令はμROMを参照するようエンコード
  - x86デコードなし→分岐予測ミス・ペナルティ削減
- 命令フェッチ速度:
  - 6μop/2cycles
- TC用分岐予測器
  - 高精度(Intel says:ミス率1/3 of P6)←大きな分岐予測MP

割り当てとレジスタ・リネーミング

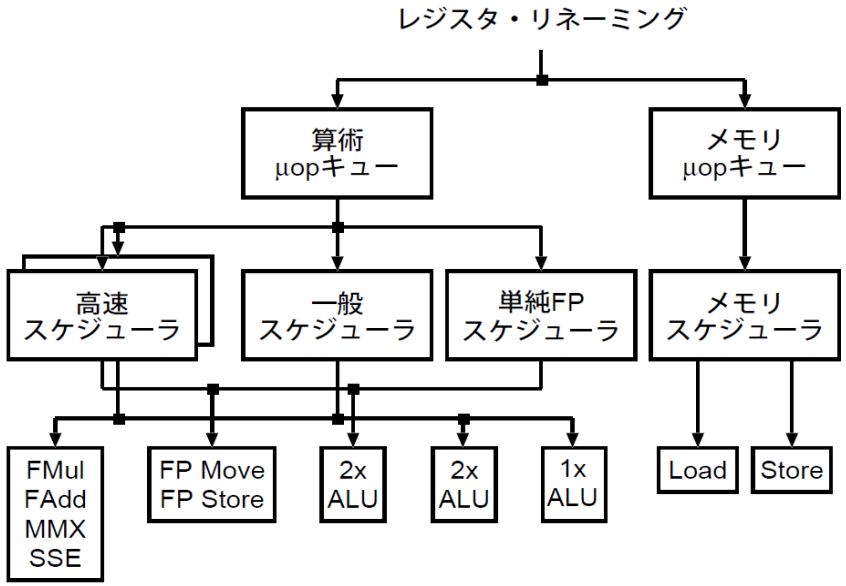


- ROB、物理レジスタ、ロード・バッファ、ストア・バッファ、μopキューの割り当て
- レジスタ・ファイルのみによるレジスタ・リネーミング
- フロントエンドRAT(register alias table)
  - リタイアメントRAT

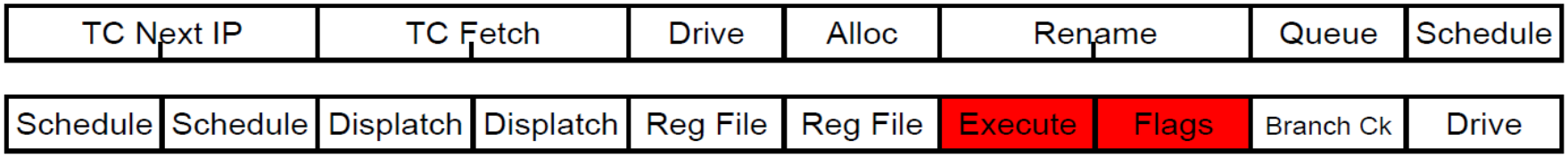
スケジューリングと発行



- 最大発行速度：6μop/cycle

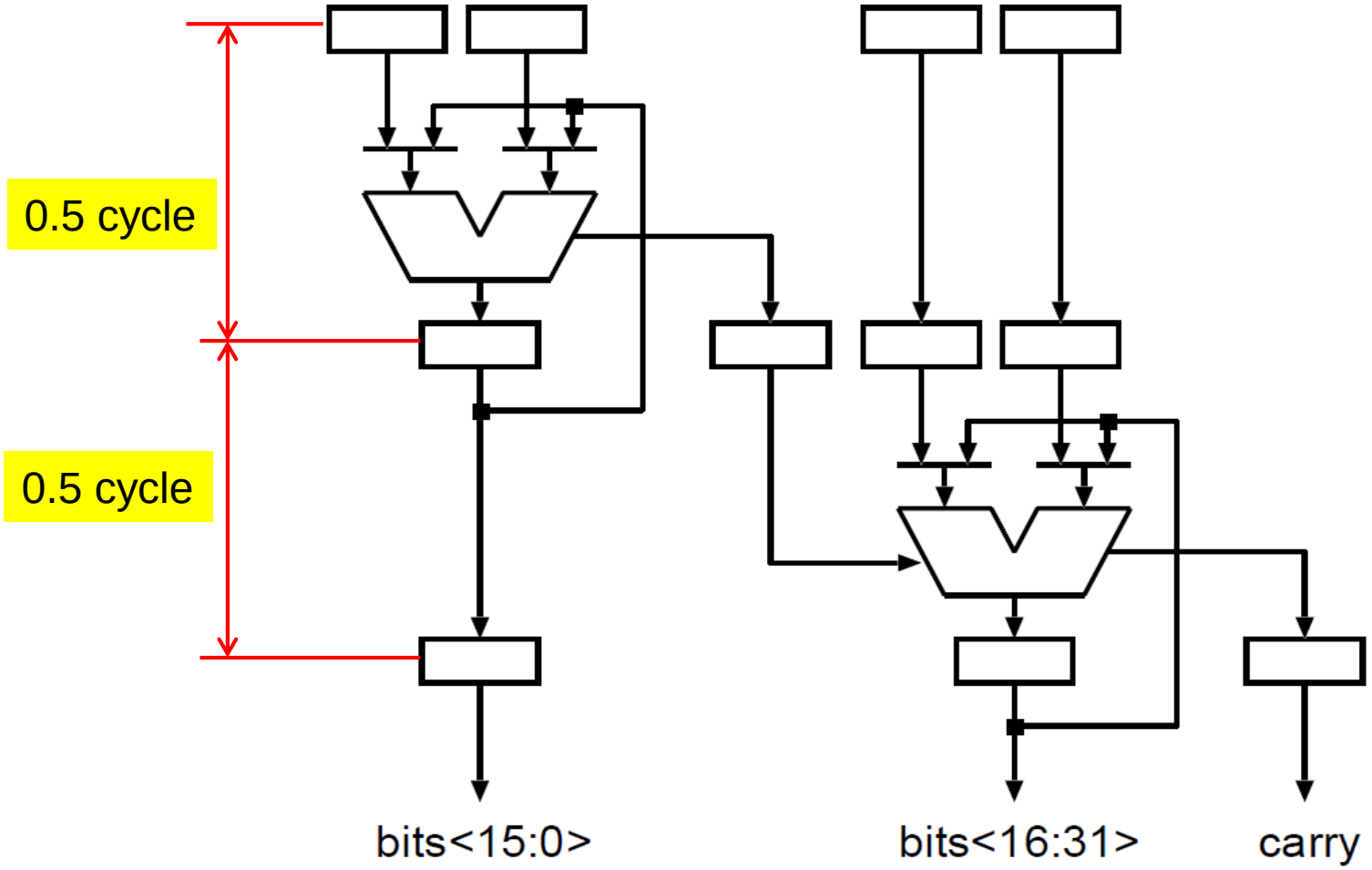


実行

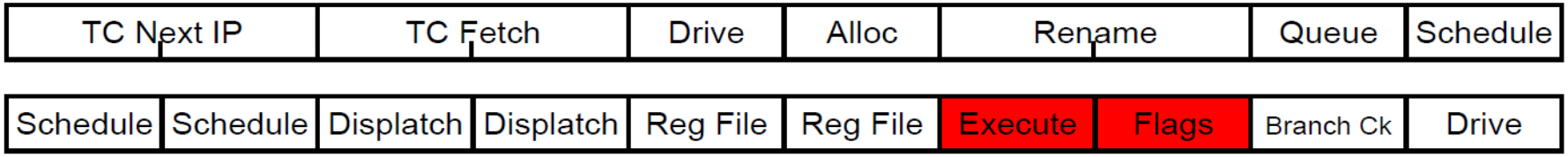


- 2つの倍速ALU
  - 加減算、論理演算
  - データ依存つぶし：依存のある2命令を1サイクルで実行
- SIMD(single-instruction stream, multiple-data stream)
  - 1ワードにパックされた複数のデータについて、同一の処理を並列に行う
  - MMX(multimedia extension)、SSE(streaming SIMD extension)
  - マルチメディア処理の高速化

倍速ALUによるデータ依存つぶし



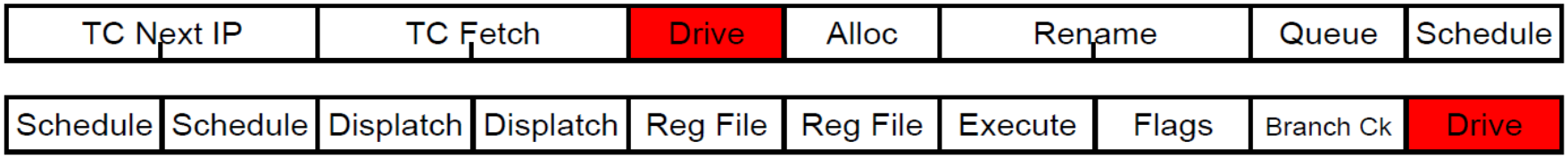
L1データ・キャッシュ



- 8KB、4ウェイ連想64Bライン
- ヒット・レイテンシ：2サイクル
- ロードの複雑なスケジューリング：
  - キャッシュ・ヒット/ミスでレイテンシが変わる
  - ロードに依存した命令の発行タイミングがわからない
  - ヒットと推定して依存命令をスケジュール
  - ミスしたら、依存命令（さらにその子供）だけを再発行(replay buffer)
    ⇒非常に複雑

    MIPS R10000, Alpha 21264: ロード以降の全ての発行を再発行

信号伝播のみのステージ



- ブロック間の長い配線
- 配線遅延
  - 古典的には、配線遅延はトランジスタのスイッチング遅延と共にスケーリングされていた
  - このころから配線遅延遅延はスケーリングされない
- チップ面積の増大
  - Pentium Ⅲ: 106mm^2→ Pentium 4: 217mm^2

Pentium 4後のチップ(1)

- 電力制約が厳しくなる
  - 空冷の限界に来ている
    - Dennard scalingの終わり
  - 回路をPentium 4ほど複雑化しない
  - パイプラインを浅く
    - Skylakeの分岐予測ミスペナルティ = 16サイクル
  - 単一スレッド性能を向上させるのは難しくなった
    - クロック速度はあげられない
    - 電力を多く消費するアーキテクチャ技術は入れられない
    - マルチコア

Pentium 4後のチップ(2)

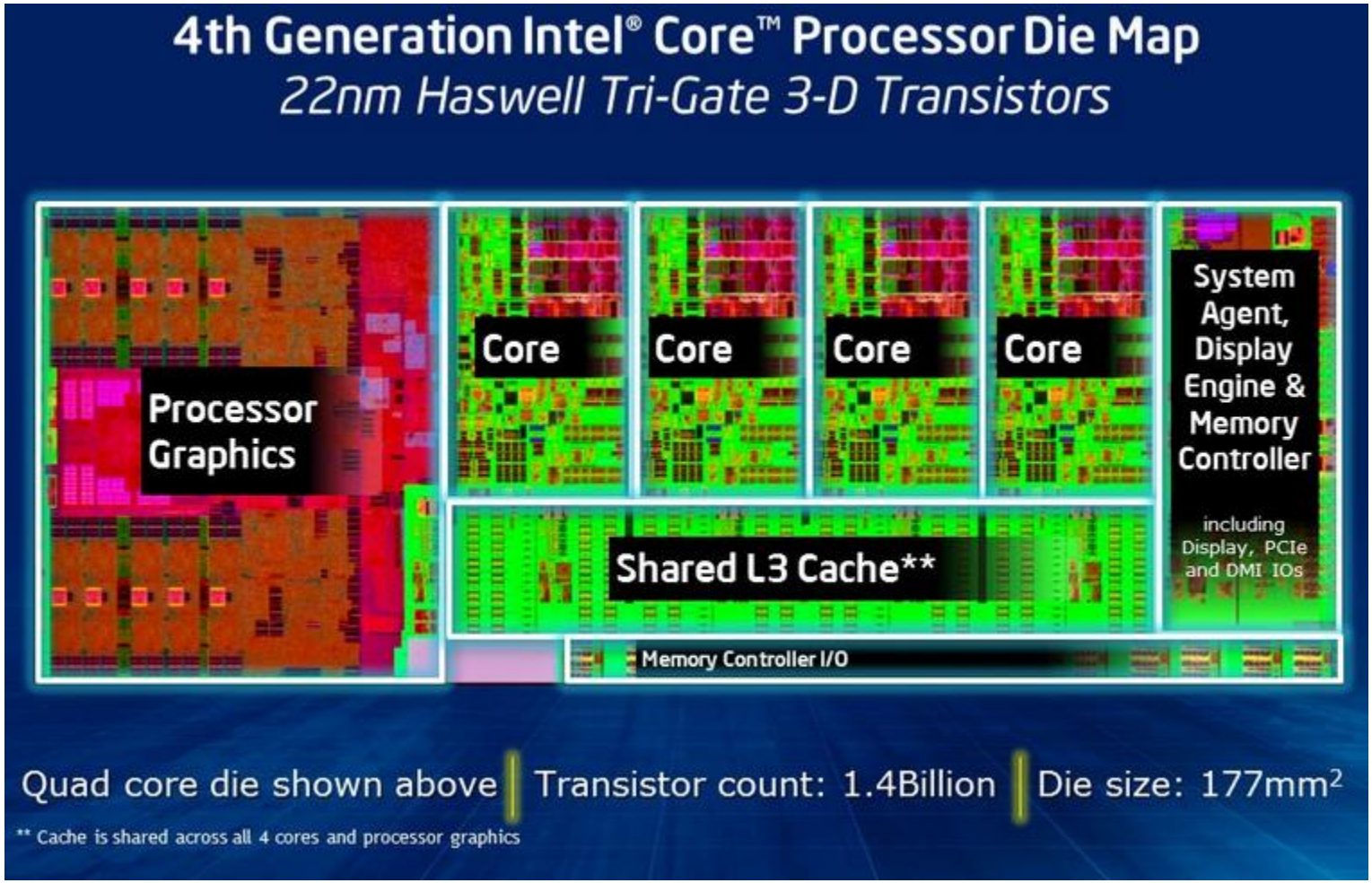
- 発行幅、ROB, 命令ウィンドウなどは徐々に大きくなっている
  - SkylakeのROBサイズ = 224エントリ
  - よりIPCをあげる
  - メモリ・レベル並列を利用する
  - SMT

- 同時マルチスレッド(SMT: simultaneous multithreading)

- マルチコア
  - ホモジニアス：同じコア
  - ヘテロジニアス：アウト・オブ・オーダとイン・オーダ

- SoC (System on a Chip)化
  - GPU、DRAMコントローラなど

Intel Haswell



Apple A10

L. Gwennap , Apple A10 Bruises Other CPUs, *Microprocessor Report*, *October 17, 2016.*