第4章 プロセッサ・アーキテクチャ(3)

大阪大学 大学院 情報科学研究科
今井 正治

arch-2014@vlsilab.ics.es.osaka-u.ac.jp

講義内容

- 例外
- 並列処理と高度な命令レベル並列性
- 実例: AMD Opteron X4(Barcelona)のパイプライン
- 誤信と落とし穴

例外と割込み

- 例外(exception)と割込み(interrupt)
  - 制御の流れの予期せぬ変更
  - 区別しないアーキテクチャも多い

事象のタイプ	発生源	MIPSの用語
入出力装置からのリクエスト	外部	割込み
ユーザー・プログラムからのOS起動	内部	例外
算術オーバーフロー	内部	例外
未定義命令の使用	内部	例外
ハードウェアの誤動作	内部または外部	例外または割込み

MIPSアーキテクチャにおける例外への対処法(1)

1. 問題を起こしているアドレスを例外プログラム・カウンタ(Exception program counter: EPC)に退避
2. OSの特定アドレスに制御を渡す
3. OSは適切な処理を行う

- 例
  - ユーザプログラムに何らかのサービスを提供
  - オーバーフローに対する処置
  - エラーメッセージの出力

MIPSアーキテクチャにおける例外への対処法(2)

1. 問題を起こしているアドレスを例外プログラム・カウンタ（Exception program counter: EPC）に退避
2. ベクタ割込みを実行
   - 例外の原因に基づいて制御を移す先を指定

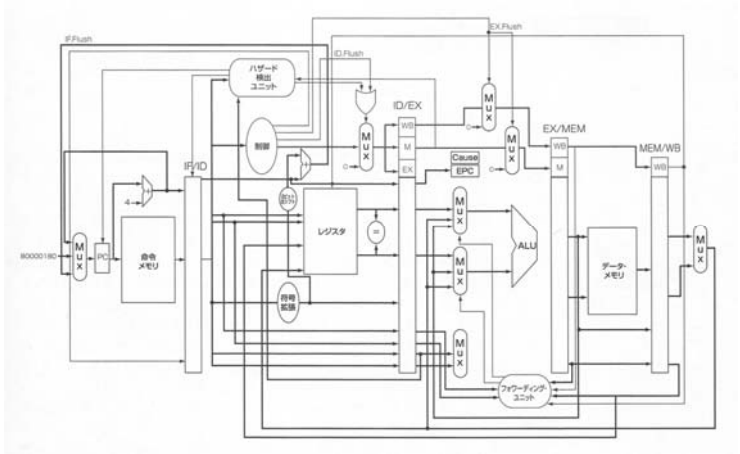
例：

例外のタイプ	例外時の飛び先アドレス
未定義命令	8000 0000
算術オーバーフロー	8000 0180

パイプライン方式における例外

- 制御ハザードのバリエーションとして扱う
- 例：加算命令でオーバーフローが発生した場合
  - 加算命令（add）の後続命令をフラッシュし，新しいアドレスから命令をフェッチ
- 実行中のパイプラインの各ステージをフラッシュする必要がある
  - 例外が起きた原因を調べる必要があるので、レジスタに演算結果を書き戻してはいけない

図4-66 例外処理を制御するデータパス



パイプライン化されたコンピュータにおける例外の例

- 命令列

```
40₁₆     sub  $11, $2, $4
44₁₆     and  $12, $2, $5
48₁₆     or   $13, $2, $6
4C₁₆     add  $1,  $2, $1
50₁₆     slt  $15, $6, $7
54₁₆     lw   $16, 50($7)
```

- 例外が発生したときに呼出されるルーチンの冒頭

```
80000180₁₆    sw   $25, 1000($zero)
80000184₁₆    sw   $26, 1004($zero)
```

図4.67-a add命令で算術オーバーフローによる例外が発生した場合に起こる現象(1)

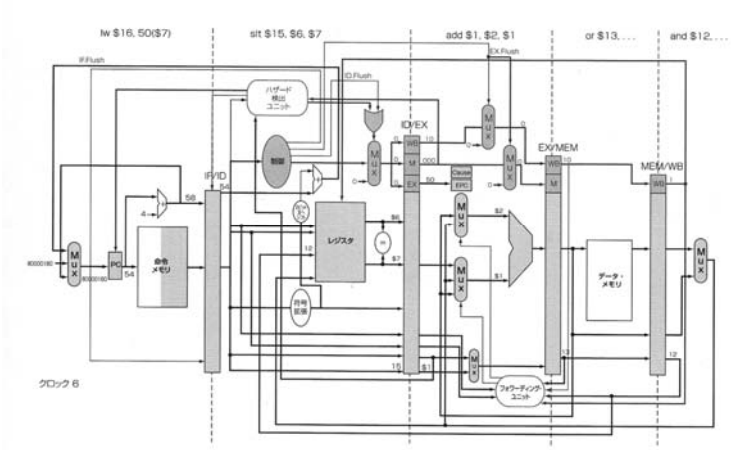
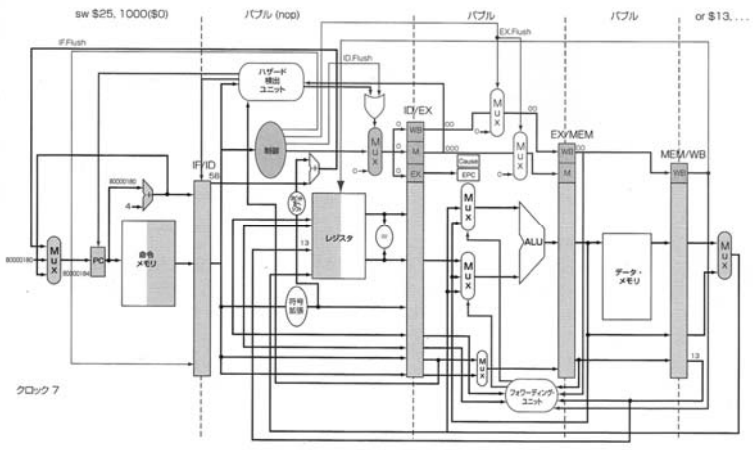


図4.67-b add命令で算術オーバーフローによる例外が発生した場合に起こる現象(2)



講義内容

- 例外
- 並列処理と高度な命令レベル並列性
- 実例: AMD Opteron X4(Barcelona)のパイプライン
- 誤信と落とし穴

並列処理と高度な命令レベル並列性

- 命令レベル並列性(Instruction level parallelism: ILP)を利用して高性能化
  - パイプラインの段数を増やす
  - 複数の命令を同時に発行する
- 複数命令発行(multiple issue)
  - 静的(static)に複数命令発行
    - コンパイル時にコンパイラが同時に発行される命令を決定
    - 超長形式命令(very long instruction word: VLIW)
  - 動的(dynamic)に複数命令発行
    - 実行時にプロセッサが同時に発行される命令を決定
    - スーパースカラ(Superscalar)

複数命令を発行するパイプラインでの命令の処理

- 命令を発行スロット(issue slot)に埋める
  - 静的命令発行の場合，コンパイラが対応
  - 動的命令発行の場合，実行時にプロセッサが対応
- データ・ハザードおよび制御ハザードに対処する
  - 静的命令発行の場合，コンパイラが対応
  - 動的命令発行の場合，実行時にプロセッサが対応

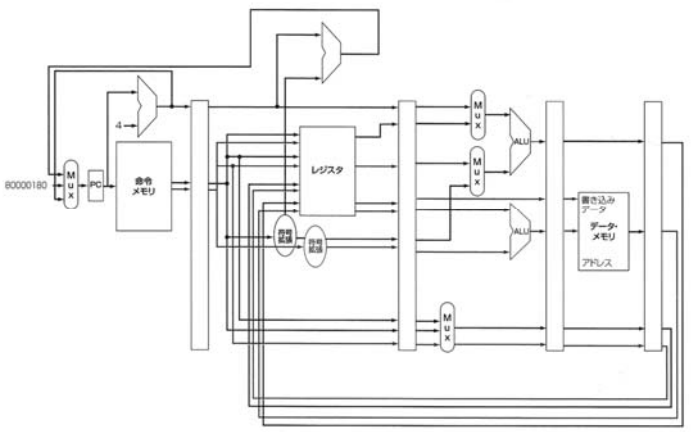
投機的実行(speculation)

- コンパイラまたはプロセッサが，命令のこれから先の挙動を「見込んで」，対象になっている命令の完了を待つ必要がある他の命令の実行をいち早く開始可能にする技法
- 例:
  - 分岐命令の後で実行される命令
  - ストア命令に続く，ストアの結果を利用しないロード命令
- 「見込み」が外れた場合の対策が必要
  - 見込みが正しかったかどうかのチェック
  - 見込みに従っていち早く実行した命令の効果の復元

図4.68 静的2命令実行パイプライン中の命令の進行

命令タイプ	パイプライン・ステージ							
算術論理演算または分岐命令	IF	ID	EX	MEM	WB			
ロードまたはストア命令	IF	ID	EX	MEM	WB			
算術論理演算または分岐命令		IF	ID	EX	MEM	WB		
ロードまたはストア命令		IF	ID	EX	MEM	WB		
算術論理演算または分岐命令			IF	ID	EX	MEM	WB	
ロードまたはストア命令			IF	ID	EX	MEM	WB	
算術論理演算または分岐命令				IF	ID	EX	MEM	WB
ロードまたはストア命令				IF	ID	EX	MEM	WB

図4.69 静的2命令発行データパス



単純な複数命令発行パイプラインにおけるコードのスケジューリング

- 例：MIPSの静的2命令発行パイプライン

```
Loop: lw   $t0, 0($s1)       # $t0に配列要素を代入
      addu $t0, $t0, $s2     # $s2中の定数を加算
      sw   $t0, 0($s1)       # 結果をストア
      addi $s1, $s1, -4      # ポインタを繰り下げ
      bne  $s1, $zero, Loop # $s1がゼロでなければ分岐
```

- パイプライン・ストール数をできるだけ減らすように命令の順序を変更する
  - 分岐は予測される
  - 制御ハザードはハードウェアで対処

図4.70 2命令発行MIPSパイプライン上でスケジューリングされた命令

	算術論理演算または分岐命令	データ転送命令	クロック・サイクル
Loop:		lw $t0, 0($s1)	1
	addi $s1, $s1, -4		2
	addu $t0, $t0, $s2		3
	bne $s1, $zero, Loop	sw $t0, 0($s1)	4

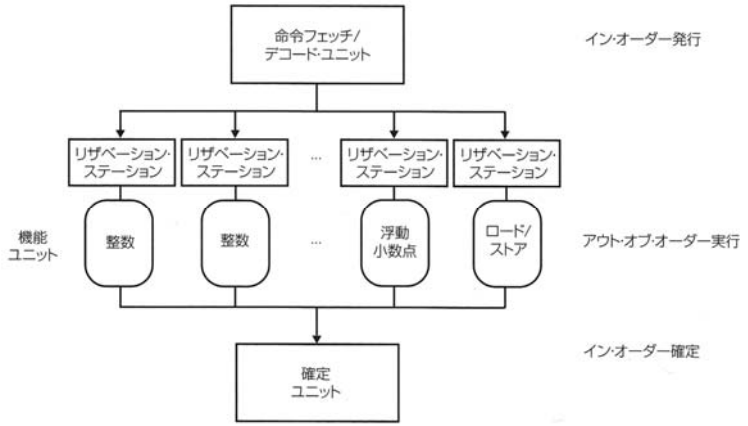
図4.71 静的2命令発行MIPSパイプライン上でスケジューリングした様子

	算術論理演算または分岐命令	データ転送命令	クロック・サイクル
Loop:	addi $s1, $s1, -16	lw $t0, 0($s1)	1
		lw $t1, 12($s1)	2
	addu $t0, $t0, $s2	lw $t2, 8($s1)	3
	addu $t1, $t1, $s2	lw $t3, 4($s1)	4
	addu $t2, $t2, $s2	sw $t0, 16($s1)	5
	addu $t3, $t3, $s2	sw $t1, 12($s1)	6
		sw $t2, 8($s1)	7
	bne $s1, $zero, Loop	sw $t3, 4($s1)	8

動的な複数命令発行プロセッサ

- スーパースカラ(superscalar)
  - 動的なパイプライン・スケジューリングを採用
  - 命令フェッチ/デコード・ユニット
  - レザベーション・ステーション
  - アウト・オブ・オーダ実行（out-of-order execution）
  - イン・オーダー確定（in-order commit）
    - リオーダ・バッファ（reorder buffer）

図4.72 動的スケジューリングを行うパイプラインの3つの主要ユニット



処理手順(1)

- 命令は発行された時点で，該当する機能ユニットのリザベーション・ステーションにコピーされる．
- レジスタ・ファイルまたはリオーダ・バッファ内で利用可能なオペランドも，ただちにリザベーション・ステーションにコピーされる．
- 発行された命令は，すべてのオペランドと機能ユニットが利用可能になるまで，リザベーション・ステーション内に保持される．
- レジスタへの書き込みが発生した場合には，該当するオペランドに上書きされる．

処理手順(2)

- あるオペランドがレジスタ・ファイルまたはリオーダ・バッファの中に無い場合には，機能ユニットがそれを生成するまで待つ．
- 結果を生成する機能ユニットはトレースされ，該当する機能ユニットから結果が生成されると，レジスタ・ファイルをバイパスして，待機しているリザベーション・ステーションに直接コピーされる．

投機実行(speculation)

- コンパイラまたはプロセッサが，命令のこれから先の挙動を「見込んで」，対象になっている命令の完了を待つ必要のある他の命令に実行を開始可能にする技法
- アーキテクチャはより複雑になる
  - 「見込み」が正しかったかどうかを判定する機構
  - 「見込み」が間違っていた場合に，命令の実行の効果を復元する機構

図4.73 パイプラインの複雑性，コア数，消費電力から見たIntelとSunのマイクロプロセッサの比較

プロセッサ	年	クロック周波数(MHz)	パイプライン段数	発行命令数	アウト・オブ・オーダー/投機実行	コア数/チップ	電力(W)
Intel 486	1989	25	5	1	No	1	5
Intel Pentium	1993	66	5	2	No	1	10
Intel Pentium Pro	1997	200	10	3	Yes	1	29
Intel Pentium 4 Willamette	2001	2000	22	3	Yes	1	75
Intel Pentium 4 Prescott	2004	3600	31	3	Yes	1	103
Intel Core	2006	2930	14	4	Yes	2	75
Sun UltraSPARC III	2003	1950	14	4	No	1	90
Sun UltraSPARC T1 (Niagara)	2005	1200	6	1	No	2	70

講義内容

- 例外
- 並列処理と高度な命令レベル並列性
- 実例：AMD Opteron X4（Barcelona）のパイプライン
- 誤信と落とし穴

図4.74 AMD Opteron X4のアーキテクチャ

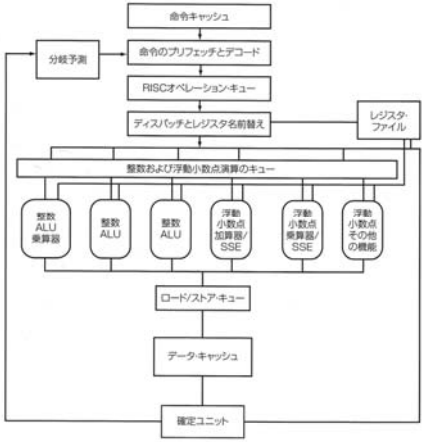
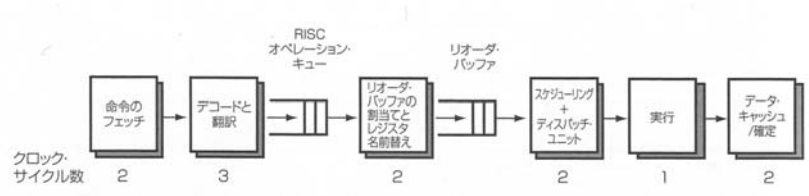


図4.75 Opteron X4のパイプライン・フロー



講義内容

- 例外
- 並列処理と高度な命令レベル並列性
- 実例: AMD Opteron X4（Barcelona）のパイプライン
- 誤信と落とし穴

誤信と落とし穴

- 誤信: パイプライン処理は容易である
- 誤信: パイプラインの設計思想は製造テクノロジと独立に実現できる
- 落とし穴: 命令セットの設計がパイプラインに負の効果を与える場合があることを検討し忘れる