第4章 プロセッサ・アーキテクチャ(2)

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

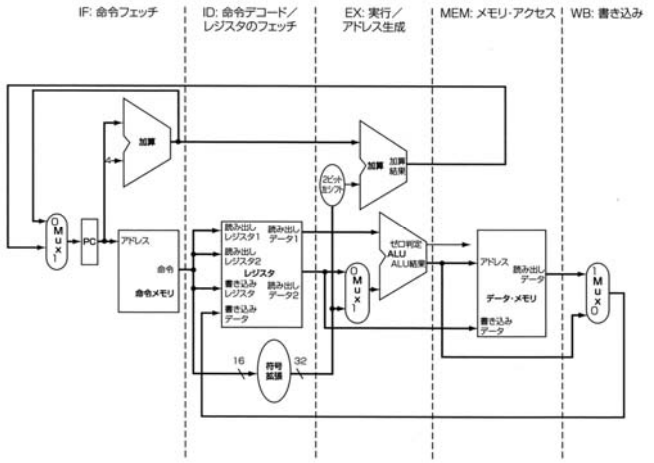
講義内容

- データパスのパイプライン化と制御
- データ・ハザード
- フォワーディング
- ストール
- 制御ハザード
- 分岐予測

パイプライン・ステージ

- IF:命令フェッチ(instruction fetch)
- ID:命令デコード(instruction decode)と
  レジスタ・フェッチ(register fetch)
- EXE:命令実行(execute)/
  アドレス生成(address generation)
- MEM:データ・メモリ・アクセス
  (memory access)
- WB:書き込み(write back)

図4.33 単一クロック・サイクルのデータパス



データパスでのデータの流れ

- 基本的に左（IF）から右（WB）
- 例外
  - WBステージで，結果がデータパスの中間部（ID）にあるレジスタ・ファイルに戻される
  - PCの次の値を設定する際，繰り上げられたPC値またはEXステージで生成された分岐先アドレスのどちらかが選択される

図4.34 単一クロック・サイクルのデータパス上で命令をパイプライン方式で実行する様子

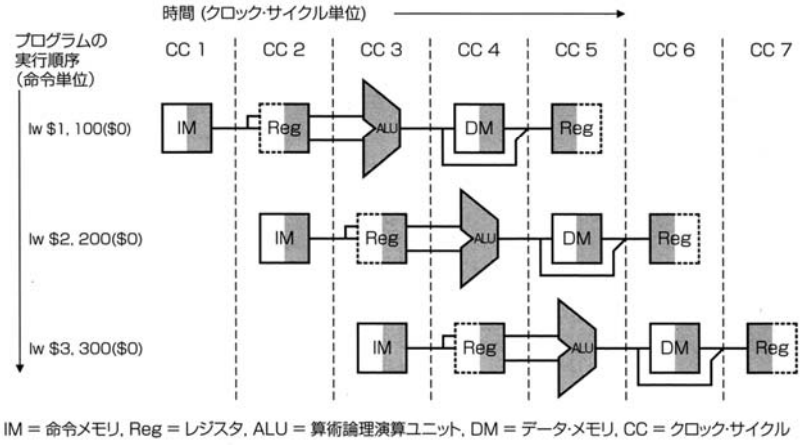
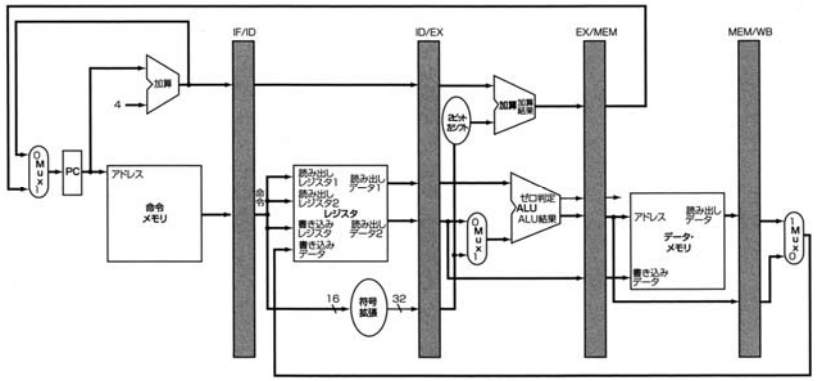


図4.35 図4.33のデータパスのパイプライン版



パイプライン・ステージ（1）

- IF：命令フェッチ（instruction fetch）
  - 命令メモリから命令を読み出す
  - PCの値をインクリメント（+4）
- ID: 命令デコード（instruction decode）
  - 命令のデコード
  - レジスタ・ファイルからの読み出し
- EX: 実行（execution）/アドレス生成
  - ALUを用いて演算を行う
  - 実効アドレス（effective address）を計算する

パイプライン・ステージ(2)

- MEM: メモリ・アクセス(memory access)
  - データ・メモリからのデータの読み出し
  - データ・メモリへのデータの書き込み
- WB: 書き込み(write back)
  - 演算結果またはメモリから読み出した値をレジスタファイルに書き込む

図4.36 (1) ロード命令のIFステージ

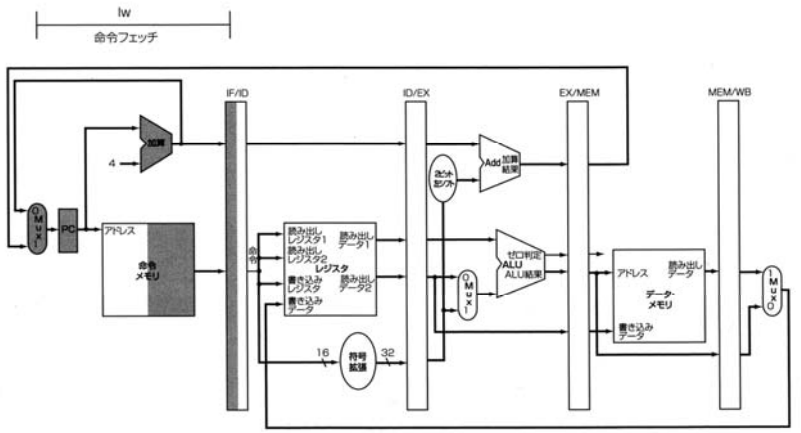


図4.36 (2) ロード命令のIDステージ

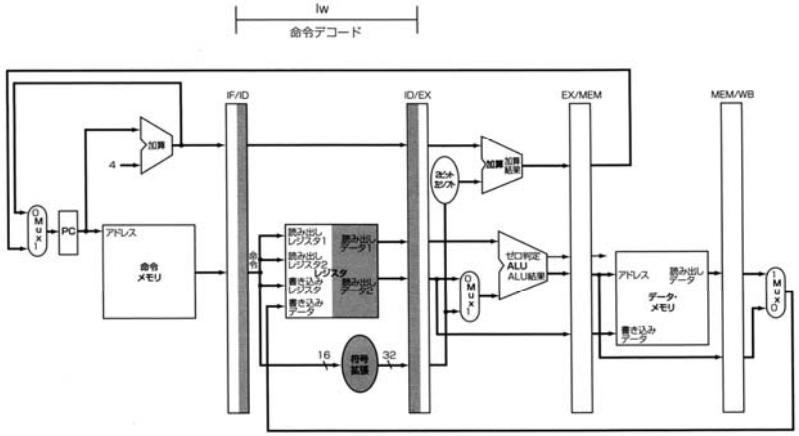


図4.37 ロード命令のEXステージ

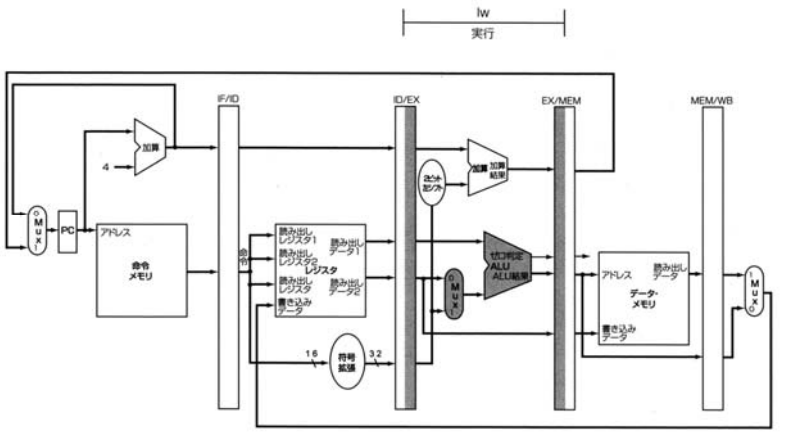


図4.38-a ロード命令のMEMステージ

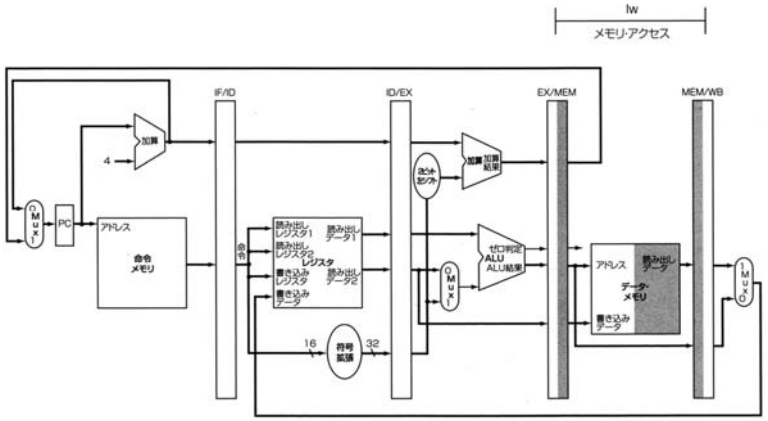


図4.38-b ロード命令のWBステージ

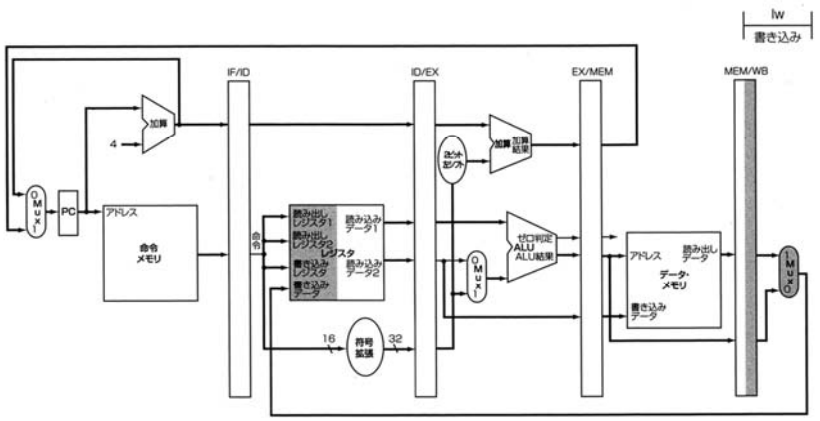


図4.39 ストア命令のEXステージ

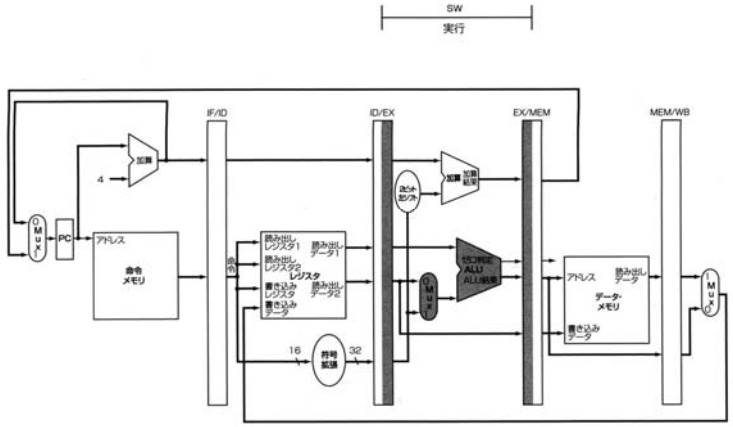


図4.40-a ストア命令のMEMステージ

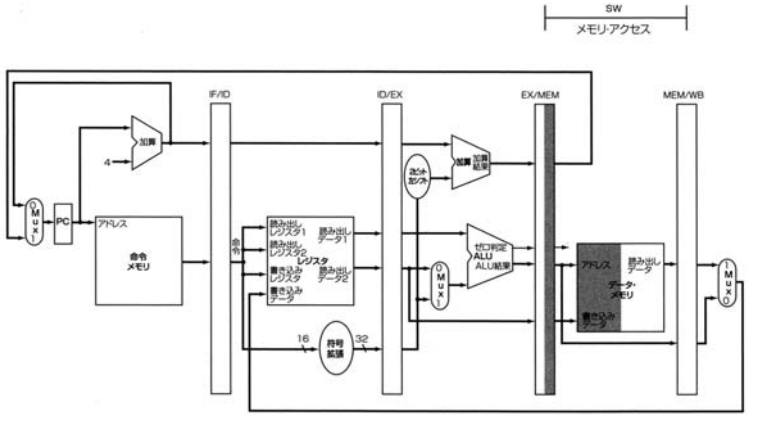


図4.40-b ストア命令のWBステージ

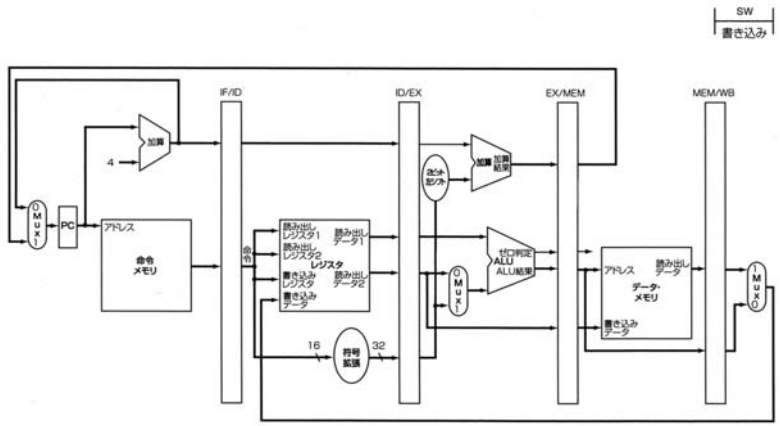


図4.41 ロード命令を正しく処理するために修正を加えたパイプライン方式のデータパス

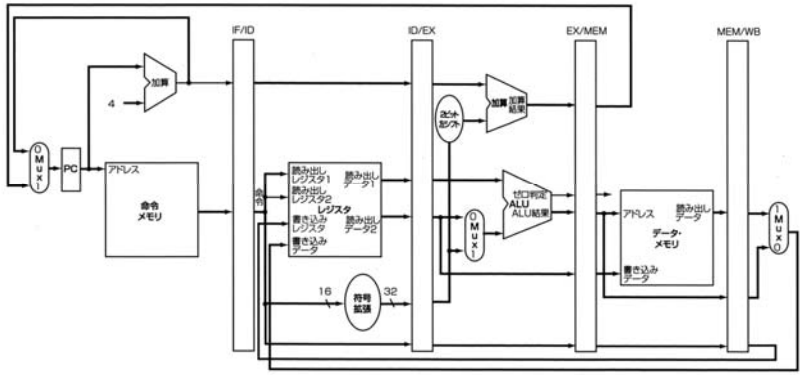
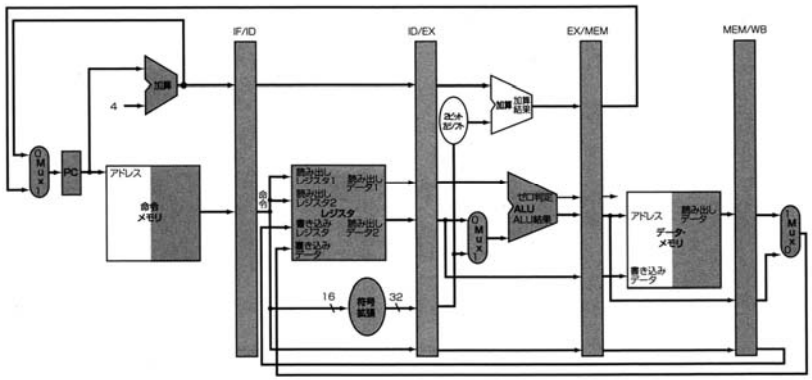


図4.42 図4.41のデータパスのうちで，ロード命令の5つのステージで使用される部分



パイプラインの模式図表現

- 命令列の例

```
lw   $10, 20($1)
sub  $11, $2, $3
add  $12, $3, $4
lw   $13, 24($1)
add  $14, $5, $6
```

図4.43 5つの命令に対する多重サイクル型のパイプライン図

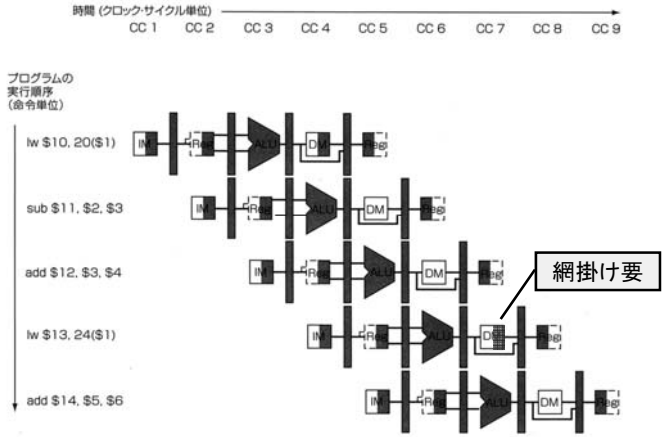


図4.44 図4.43の5つの命令に対する伝統的な多重サイクル型のパイプライン図

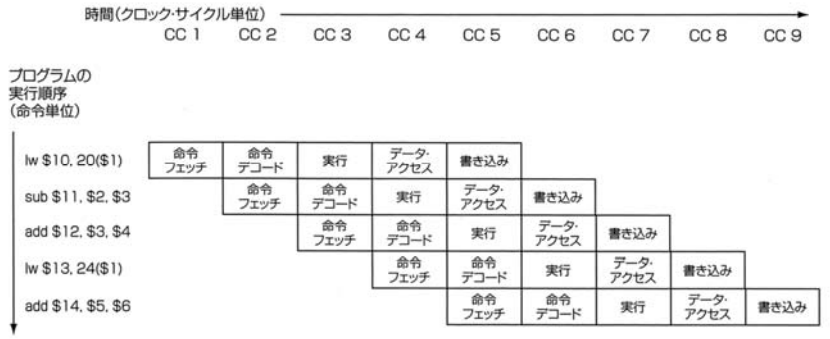
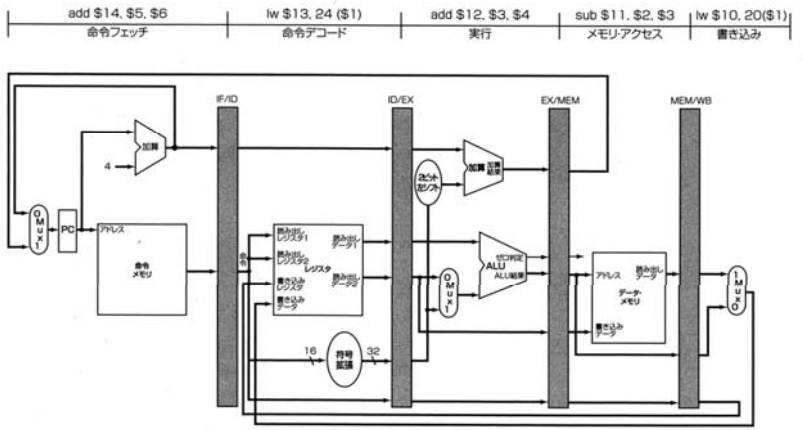


図4.45 図4.43と図4.44のクロック・サイクル5に相当する単一サイクル型の図



パイプラインの制御

- IF: 命令フェッチ
  - 命令メモリの読出し制御信号をアサート
  - PCの書込み制御信号をアサート
- ID: 命令デコードとレジスタファイルの読出し
  - 選択的制御信号をアサートする必要なし
- EX: 実行/アドレス生成
  - RegDst, ALUOp, ALUSrc を設定
- MEM: メモリ・アクセス
  - Branch, MemRead, MemWrite を設定
- RB: レジスタへの書込み
  - MemtoReg, RegWrite を設定

図4.46 図4.41のパイプライン化したデータパスに制御信号を付け加えたもの

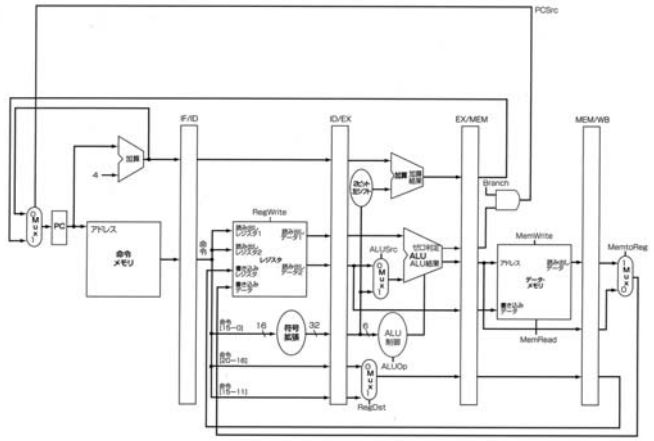


図4.47 ALU制御ビットの構成

命令操作コード	ALUOp（制御フィールド）	命令操作	機能コード（funct）	実行する演算	ALU制御コード
LW	00	Load word	XXXXXX	Add	0010
SW	00	Store word	XXXXXX	Add	0010
Branch equal	01	Branch equal	XXXXXX	Subtract	0110
R形式	10	Add	100000	Add	0010
R形式	10	Subtract	100010	Subtract	0110
R形式	10	AND	100100	And	0000
R形式	10	OR	100101	Or	0001
R形式	10	Set on less than	101010	Set on less than	0111

ALUOp: 命令中の制御フィールド（2ビット），主制御ユニットで使用

ネゲート：信号が論理的に低い，または偽であること
アサート：信号が論理的に高い，または真であること

図4.48 7つの制御信号の機能

信号名	ネゲートされた時の働き	アサートされた時の働き
RegDst	書き込みレジスタのデスティネーション・レジスタ番号がrtフィールドから得られる	書き込みレジスタのデスティネーション・レジスタ番号がrdフィールドから得られる
RegWrite	なし	書き込みレジスタ入力に指定されているレジスタに書き込みデータ入力の値が書き込まれる
ALUSrc	ALUの第2オペランドがレジスタ・ファイルの第2出力から得られる	ALUの第2オペランドが命令の下位16ビットを符号拡張したものになる
PCSrc	PC＋4を計算した加算器の出力によってPCが置き換えられる	分岐先を計算した加算器の出力によってPCが置き換えられる
MemRead	なし	読み出しアドレスによって指定されたデータ・メモリの内容が読み出しデータ出力上に流される
MemWrite	なし	書き込みアドレスによって指定されるアドレス上にあるデータ・メモリの内容が書き込みデータ入力の値によって書き換えられる
MemtoReg	レジスタの書き込みデータ入力へ渡される値がALUから得られる	レジスタの書き込みデータ入力へ渡される値がデータ・メモリから得られる

図4.49 命令の種類と制御信号の対応

命令	実行/アドレス生成ステージの制御信号				メモリ・アクセス・ステージの制御信号			書込みステージの制御信号	
	Reg Dst	ALU Op1	ALU Op0	ALU Src	Branch	Mem Read	Mem Write	Reg Write	Mem toReg
R形式	1	1	0	0	0	0	0	1	0
lw	0	0	0	1	0	1	0	1	1
sw	X	0	0	1	0	0	1	0	X
beq	X	0	1	0	1	0	0	0	X

図4.50 最後の3つのステージ用の制御線

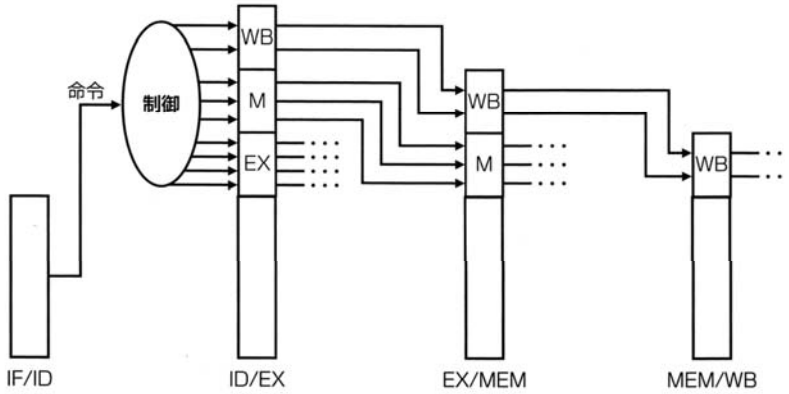
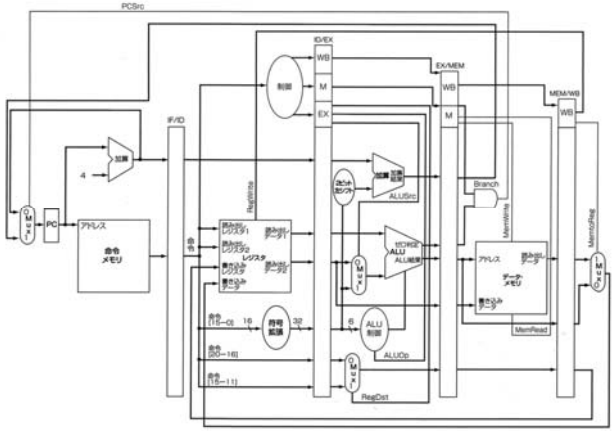


図4.51 パイプライン化したデータパスの全体像



講義内容

- データパスのパイプライン化と制御
- データ・ハザード
- フォワーディング
- ストール
- 制御ハザード
- 分岐予測

パイプライン・ハザード（pipeline hazard）

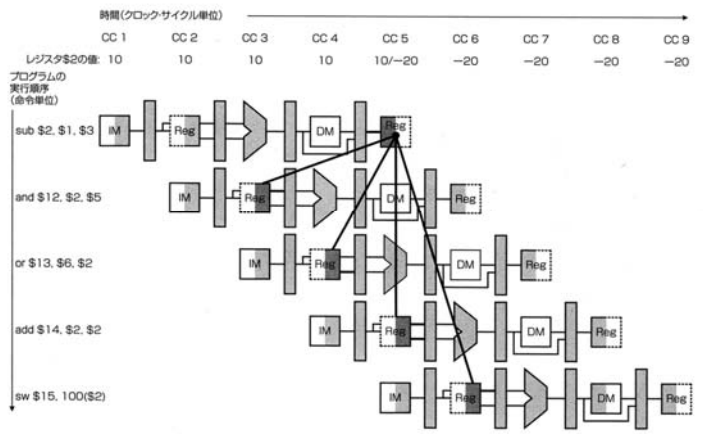
- 構造ハザード（structure hazard）
  - 同一リソースを複数のステージで使用
- データ・ハザード（data hazard）
  - 命令間の依存関係（dependency）
- 制御ハザード（control hazard）
  - 分岐命令の成立

データ・ハザードの例

□ 命令列

```
sub  $2,  $1,  $3  # subの結果を $2 に格納
and  $12, $2,  $5  # 第1オペランドが sub に依存
or   $13, $6,  $2  # 第2オペランドが sub に依存
add  $14, $2,  $2  # 第1，第2オペランドがsubに依存
sw   $15, 100($2)  # インデックスが sub に依存
```


図4.52 5つの命令をパイプライン方式で実行する際のデータの依存性



データ・ハザードが生じる条件

□ データ・ハザードが生じる条件

- 1a: EX/MEM.Register.Rd = ID/EX.Register.Rs
- 1b: EX/MEM.Register.Rd = ID/EX.Register.Rt
- 2a: MEM/WB.Register.Rd = ID/EX.Register.Rs
- 2b: MEM/WB.Register.Rd = ID/EX.Register.Rt

□ MIPSの場合（$zeroの扱い）

- 1a: EX/MEM.Register.Rd = ID/EX.Register.Rs and (EX/MEM.Register.Rd ≠ 0)
- 1b: EX/MEM.Register.Rd = ID/EX.Register.Rt and (EX/MEM.Register.Rd ≠ 0)

データ・ハザードの解析

□ 命令列

```
sub  $2,  $1,  $3  #
and  $12, $2,  $5  # タイプ1a
or   $13, $6,  $2  # タイプ2b
add  $14, $2,  $2  # ハザードではない
sw   $15, 100($2)  # ハザードではない
```

講義内容

- データパスのパイプライン化と制御
- データ・ハザード
- フォワーディング
- ストール
- 制御ハザード
- 分岐予測

データ・ハザードの解決策

- 演算結果，メモリアクセス結果のフォワーディング（forwarding）
- データハザードの検出を行うために，レジスタ番号をパイプライン・レジスタ経由でフォワーディング・ユニットに通知する必要がある

図4.53 フォワーディング後のパイプライン・レジスタ間の依存関係

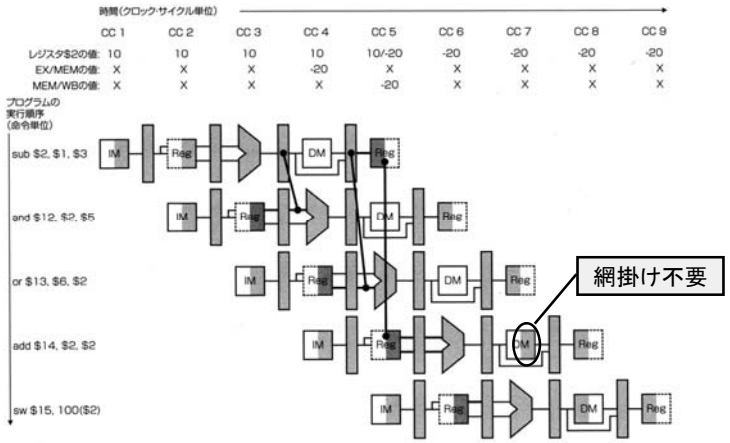


図4.54-a フォワーディング・ユニット追加前のALUとパイプライン・レジスタ

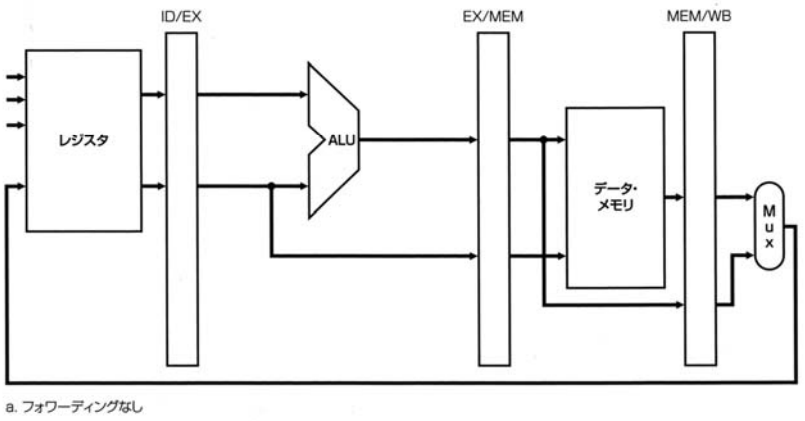
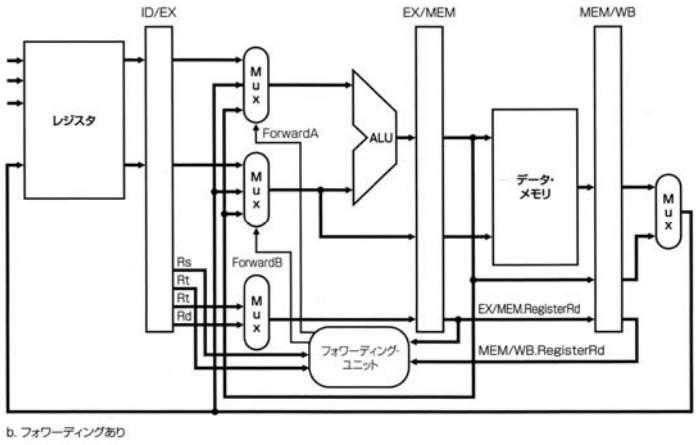


図4.54-b フォワーディング・ユニット追加後のALUとパイプライン・レジスタ



フォワーディング用マルチプレクサの制御

マルチプレクサ制御	ソース	ALUのオペランドのソース
ForwardA = 00	ID/EX	第1オペランドがレジスタ・ファイルから得られる
ForwardA = 10	EWX/MEM	第1オペランドが1つ前のALUの結果から先送りされる
ForwardA = 01	MEM/WB	第1オペランドがデータ・メモリまたは2つ前のALUの結果から先送りされる
ForwardB = 00	ID/EX	第2オペランドがレジスタ・ファイルから得られる
ForwardB = 10	EWX/MEM	第2オペランドが1つ前のALUの結果から先送りされる
ForwardB = 01	MEM/WB	第2オペランドがデータ・メモリまたは2つ前のALUの結果から先送りされる

EXハザードが生じる条件

- EX/MEM.RegWrite
  and (Ex/MEM.RegisterRd ≠ 0)
  and (Ex/MEM.RegisterRd = ID/EX.RegisterRs)
  が真の場合, ForwardA = 10

- EX/MEM.RegWrite
  and (Ex/MEM.RegisterRd ≠ 0)
  and (Ex/MEM.RegisterRd = ID/EX.RegisterRt)
  が真の場合, ForwardB = 10

MEMハザードが生じる条件(1)

- MEM/WB.RegWrite
  and (MEM/WB.RegisterRd ≠ 0)
  and not (EX/MEM.RegWrite
  and (EX/MEM.RegisterRd ≠ 0)
  and (EX/MEM.RegisterRd ≠ ID/EX.RegisterRs))
  and (MEM/WB. RegisterRd = ID/EX.RegisterRs)
  が真の場合, ForwardA = 01

MEMハザードが生じる条件(2)

- MEM/WB.RegWrite
  and (MEM/WB.RegisterRd ≠ 0)
  and not (EX/MEM.RegWrite
  and (EX/MEM.RegisterRd ≠ 0)
  and (EX/MEM.RegisterRd ≠ ID/EX.RegisterRt))
  and (MEM/WB. RegisterRd = ID/EX.RegisterRt)
  が真の場合, ForwardB = 01

図4.56 フォワーディングによってデータ・ハザードを解消するように修正を加えたデータパス

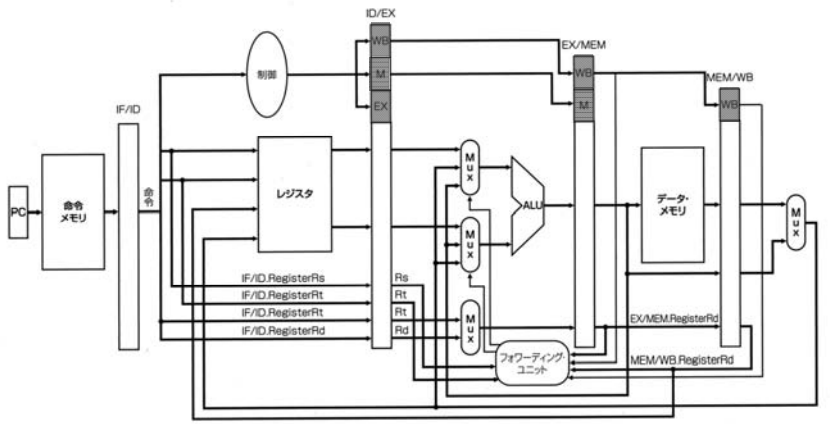
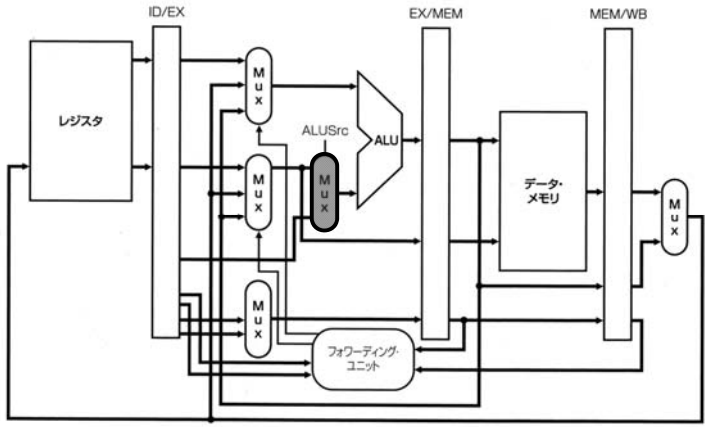


図4.57 図4.54のデータパスの改訂版



講義内容

- データパスのパイプライン化と制御
- データ・ハザード
- フォワーディング
- ストール
- 制御ハザード
- 分岐予測

データ・ハザードとストール

- フォワーディングで解消できないハザード
  - MIPSアーキテクチャでは，ロード命令が書込むレジスタの値を直後の命令が読み出そうとする場合
- ハザードの解消方法
  - ロード命令の後続命令をストール（stall）させる
  - ディレイド・ロード（delayed load）

ハザードの検出方法

- ハザード検出ユニット
  - IDステージで動作
- ハザードが生起する条件
  - ロード命令の次の命令がロードの結果を使用する
  - ID/EX.MemRead and
    ((ID/EX.RegisterRt = IF/ID.RegisterRs) or
    (ID/EX.RegisterRt = IF/ID.RegisterRt))

パイプラインのストール

- 後続命令のフェッチおよびデコードを遅延させる
  - PCおよびIF/IDパイプライン・レジスタの更新を止める
  - 見かけ上は後続命令のフェッチとデコードの繰り返し
- ロード命令の後にNOPを追加
  - EX，MEM，WBの各ステージで，結果の書き込みを禁止する（書き込み制御信号をすべて0にする）

図4.58 命令列のパイプラインでの実行状況（フォワーディングでは解消できない）

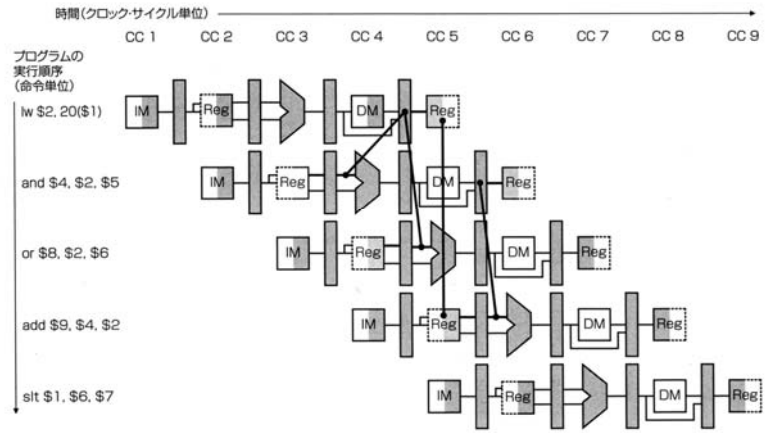


図4.59 パイプラインにストールを実際に挿入する様子

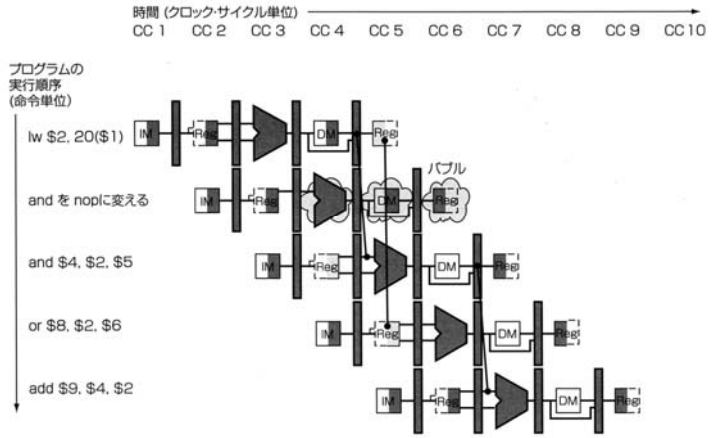
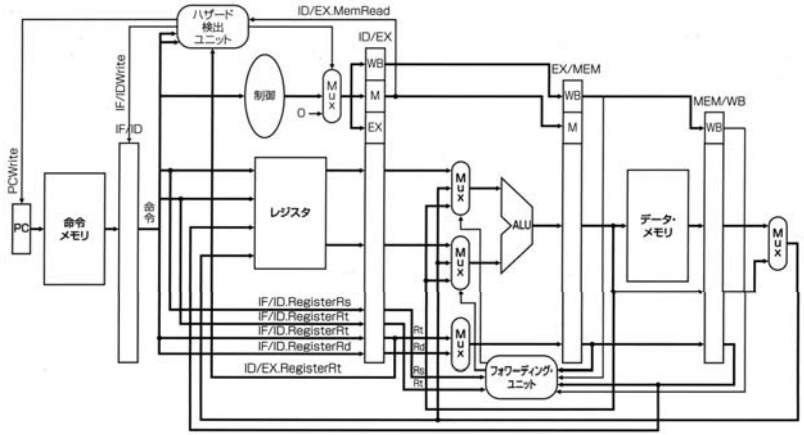


図4.60 パイプライン制御の概要図



講義内容

- データパスのパイプライン化と制御
- データ・ハザード
- フォワーディング
- ストール
- 制御ハザード
- 分岐予測

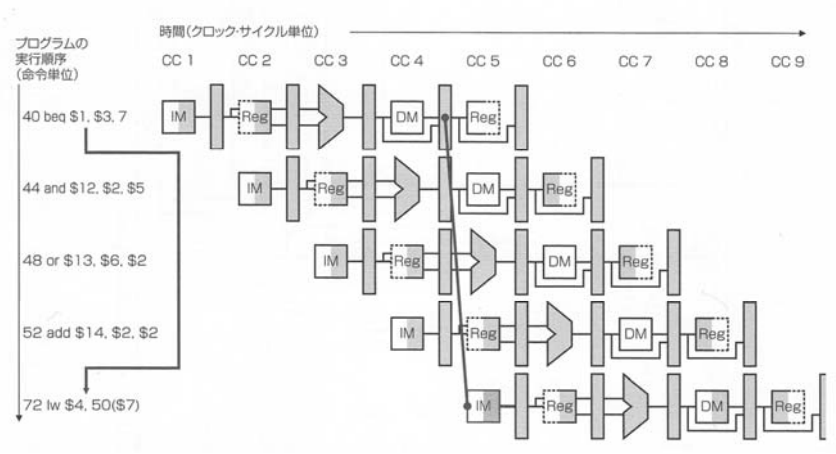
制御ハザード

- 分岐に起因するハザード
  分岐ハザード（branch hazard）とも呼ぶ
- 制御ハザードの解消策
  - 分岐が不成立と仮定して後続命令の実行を継続
  - 分岐が成立した場合には，フェッチおよびデコードを進めた命令を廃棄し，分岐先の命令から処理を続行する．
- 命令の廃棄
  - パイプラインのIF，ID，EXの各ステージの命令を一括消去（flush）

分岐が不成立と仮定

- 分岐が完了するまで後続命令をストールさせると性能が大幅に低下する
- 一般的な改善策は，分岐が不成立と仮定して，後続命令を継続する方法
- 分岐が成立した場合には，フェッチおよびデコードを進めた命令を放棄し，分岐先の命令から処理を続行

図4.61 パイプラインに対する分岐命令の影響



分岐による遅延の削減(1)

- 分岐の性能を改善する方法として，分岐の実行をMEMステージからパイプラインの早いステージに移動する方法が有効
- MIPSでは，単純な条件判定を用いる，高速な単一サイクルでの分岐をサポートしており，IDステージでの分岐条件の判定が可能
  - beq, bne

分岐による遅延の削減(2)

- 解決すべき問題
  - 分岐条件の判定に用いられるレジスタの値のフォワード
  - ハザードの検出
  - 分岐先アドレスの計算の前倒し

パイプラインにおける分岐の例

□ プログラム例

```
36  sub $10, $4, $8
40  beq $1,  $3, 7    # 40+4+7*4=72
44  and $12, $2, $5
48  or  $13, $2, $6
52  add $14, $4, $2
56  slt $15, $6, $7
    . . .
72  lw  $4, 50($7)
```


図4.62-a クロック・サイクル3のステージで分岐が成立すると判定される場合(1)

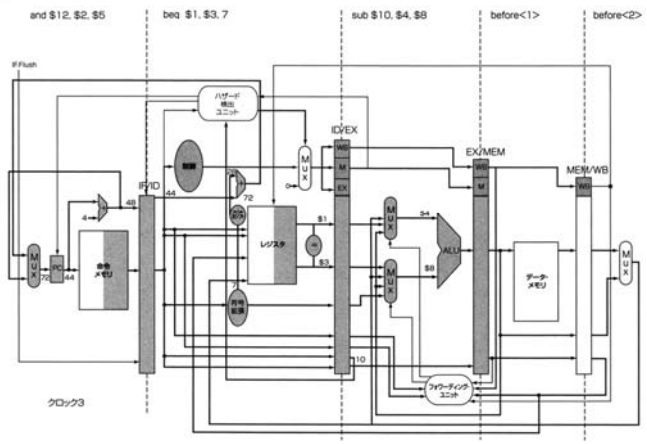
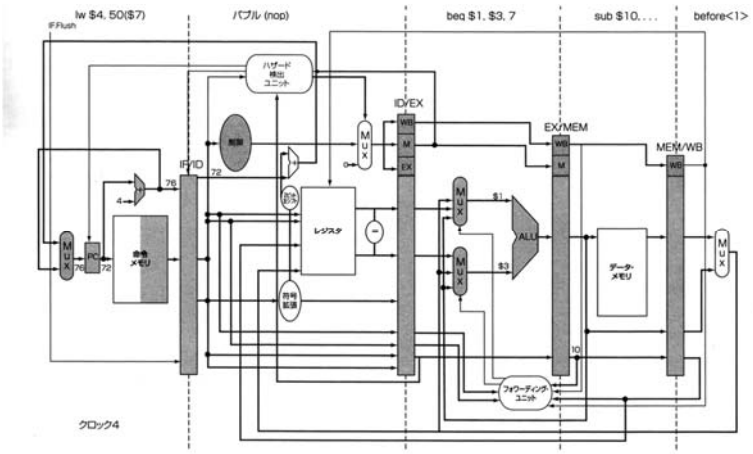


図4.62-b クロック・サイクル3のステージで分岐が成立すると判定される場合(2)



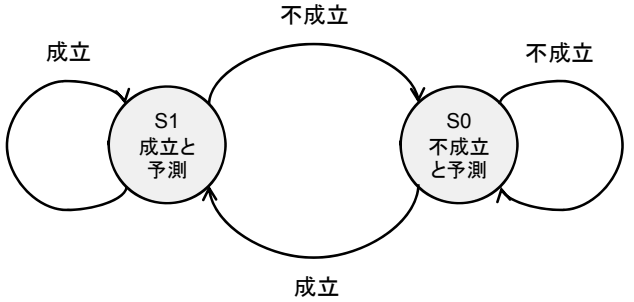
講義内容

- □ データパスのパイプライン化と制御
- □ データ・ハザード
- □ フォワーディング
- □ ストール
- □ 制御ハザード
- □ 分岐予測

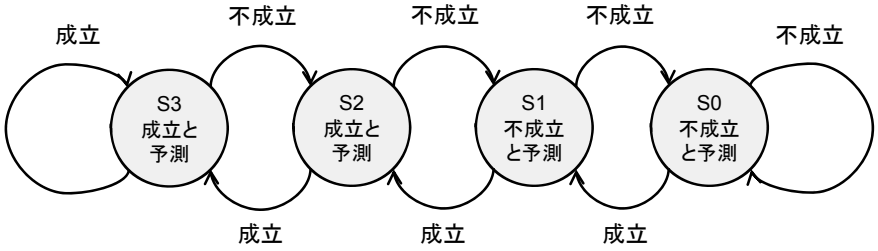
分岐の予測方法

- 静的分岐予測の例
  - 分岐は常に不成立と仮定する
  - 低位のアドレスに分岐は成立すると仮定する
- 動的分岐予測
  - 各分岐命令で分岐が成立したかどうかの履歴を記録
  - 近い過去の挙動に基づいて未来を予測
  - 分岐予測バッファ（branch prediction buffer）
  - 分岐予測テーブル（branch prediction table）

1ビット分岐予測方式



2ビット分岐予測方式



分岐遅延スロット（branch delay slot）のスケジューリング

- 分岐に影響を与えない命令を分岐スロットに移動する
  - 分岐命令より前にあって、分岐とは無関係な命令を分岐スロットに移動する（分岐に影響を与えない，分岐の影響を受けない）
  - 分岐が不成立でも差支えない分岐先の命令を分岐スロットに移動
    （他の分岐命令からも分岐する可能性があれば，分岐先の命令をコピーする）
  - 分岐命令より後ろにあって分岐が成立でも差支えない命令を分岐スロットに移動

分岐遅延スロットのスケジューリング(1)

- 先行命令の移動

```
add $s1, $s2, $s3
if $s2 = 0 then
```

遅延スロット

- スケジューリング後

```
if $s2 = 0 then
add $s1, $s2, $s3
```


分岐遅延スロットのスケジューリング(2)

- 分岐先命令の移動

```
sub $t4, $t5, $t6

add $s1, $s2, $s3
if $s1 = 0 then
```

遅延スロット

- スケジューリング後

```
sub $t4, $t5, $t6

add $s1, $s2, $s3
if $s1 = 0 then
sub $t4, $t5, $t6
```


分岐遅延スロットのスケジューリング(3)

- 後ろの命令の移動

```
add $s1, $s2, $s3
if $s1 = 0 then
```

遅延スロット

```
sub $t4, $t5, $t6
```

- スケジューリング後

```
add $s1, $s2, $s3
if $s1 = 0 then
sub $t4, $t5, $t6
```


その他の分岐予測方式

- 相関予測方式（correlating predictor）
  - 局所的な分岐の履歴情報と最近実行された大局的な挙動に関する情報を組み合わせることによって，分岐の予測精度を向上させる方法
- トーナメント分岐予測方式（tournament branch predictor）
  - 複数の予測情報を使用し，どれが最も良い結果を生むかを各分岐について追跡
  - 局所的な分岐の履歴情報と大局的な分岐の挙動を用いる

制御ハザードを軽減するその他の方法

- 条件付移送命令（conditional move）
  - 移送命令でのデスティネーションレジスタを条件によって変化させる方法
  - 条件が成立しない場合にはNOPとして機能する
  - MIPSでは、movn（move if not zero）命令およびmovz（move if zero）命令
- ARMの命令セットアーキテクチャ
  - 条件付命令（predicated instruction）