計算機システムⅡ
命令レベル並列処理とアウトオブオーダ処理

和田俊和

講義計画

1. コンピュータの歴史1
2. コンピュータの歴史2
3. コンピュータの歴史3
4. 論理回路と記憶，計算：レジスタとALU
5. 主記憶装置とALU，レジスタの制御
6. 命令セットアーキテクチャ
7. 演習問題
8. パイプライン処理
9. メモリ階層：キャッシュと仮想記憶
10. 命令レベル並列処理（←本日）
11. 命令実行順序の変更
12. 入出力と周辺装置：DMA,割り込み処理
13. 演習問題
14. 現代的な計算機アーキテクチャの解説
15. 総括と試験

- 教科書：坂井修一著：電子情報通信学会レクチャーシリーズC−9，コンピュータアーキテクチャ，コロナ社

- 最終回の試験によって成績評価を行う．5回以上欠席で不合格とする．

本日の講義の範囲

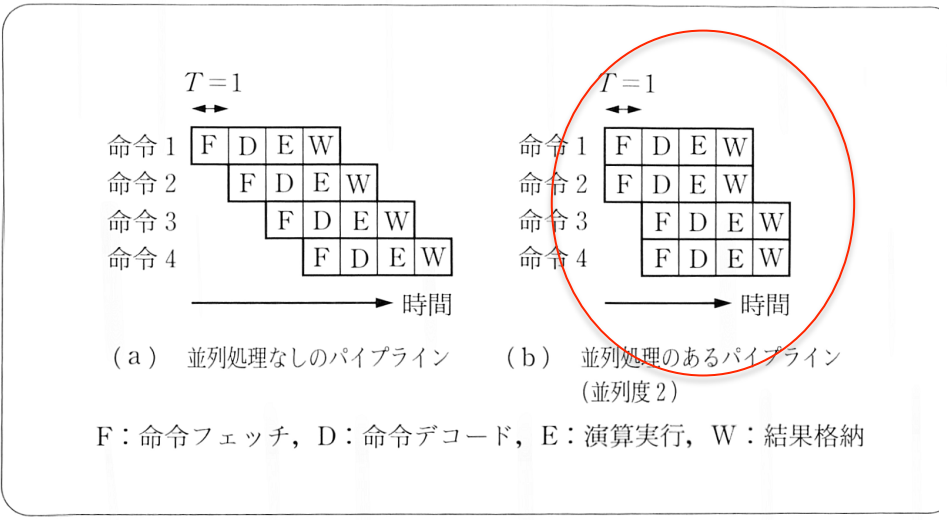


図 6.1 並列処理のある命令パイプライン

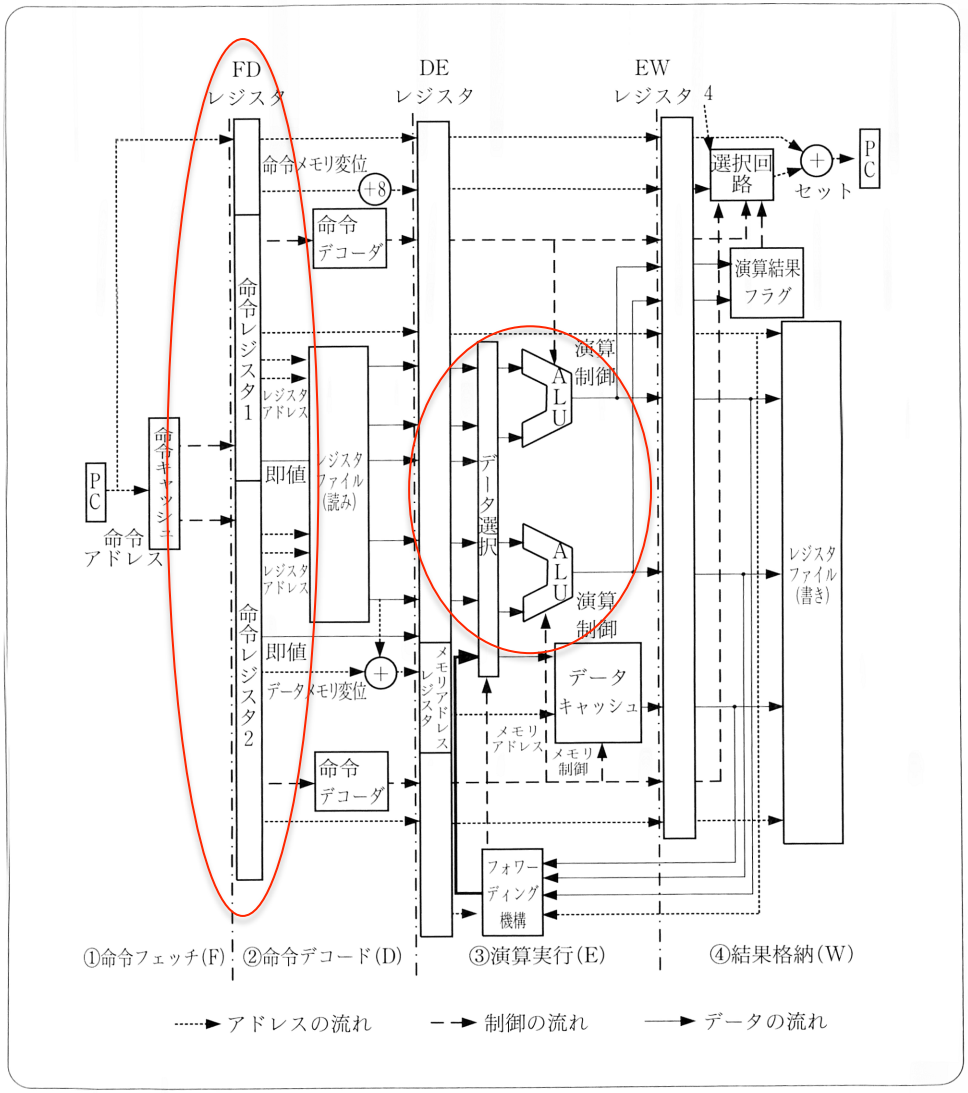
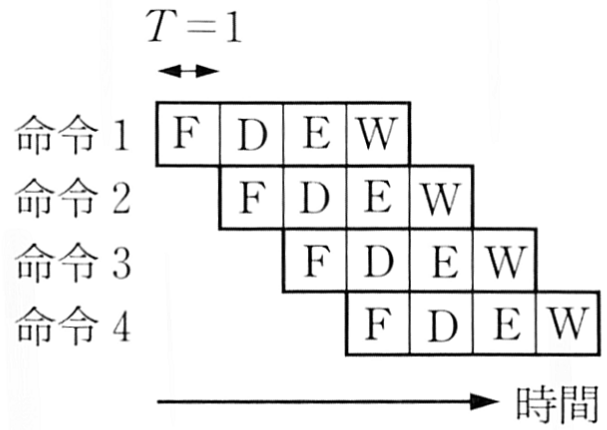


図 6.2 命令レベル並列処理の入ったパイプライン

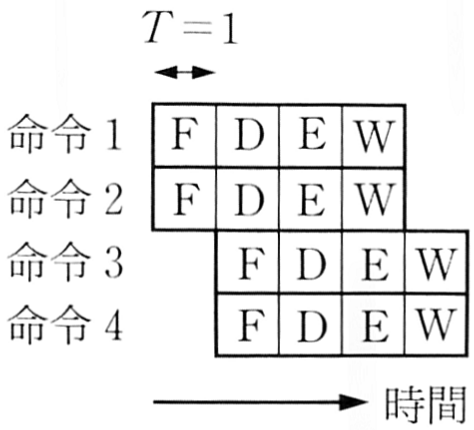
６．１　命令レベル並列処理

6.1.1 並列処理

- パイプラインレジスタの遅延は不可避．この状況でスループットを向上させるためには，並列処理しかない．



（a） 並列処理なしのパイプライン



（b） 並列処理のあるパイプライン
（並列度 2）

F：命令フェッチ，D：命令デコード，E：演算実行，W：結果格納

図 6.1 並列処理のある命令パイプライン

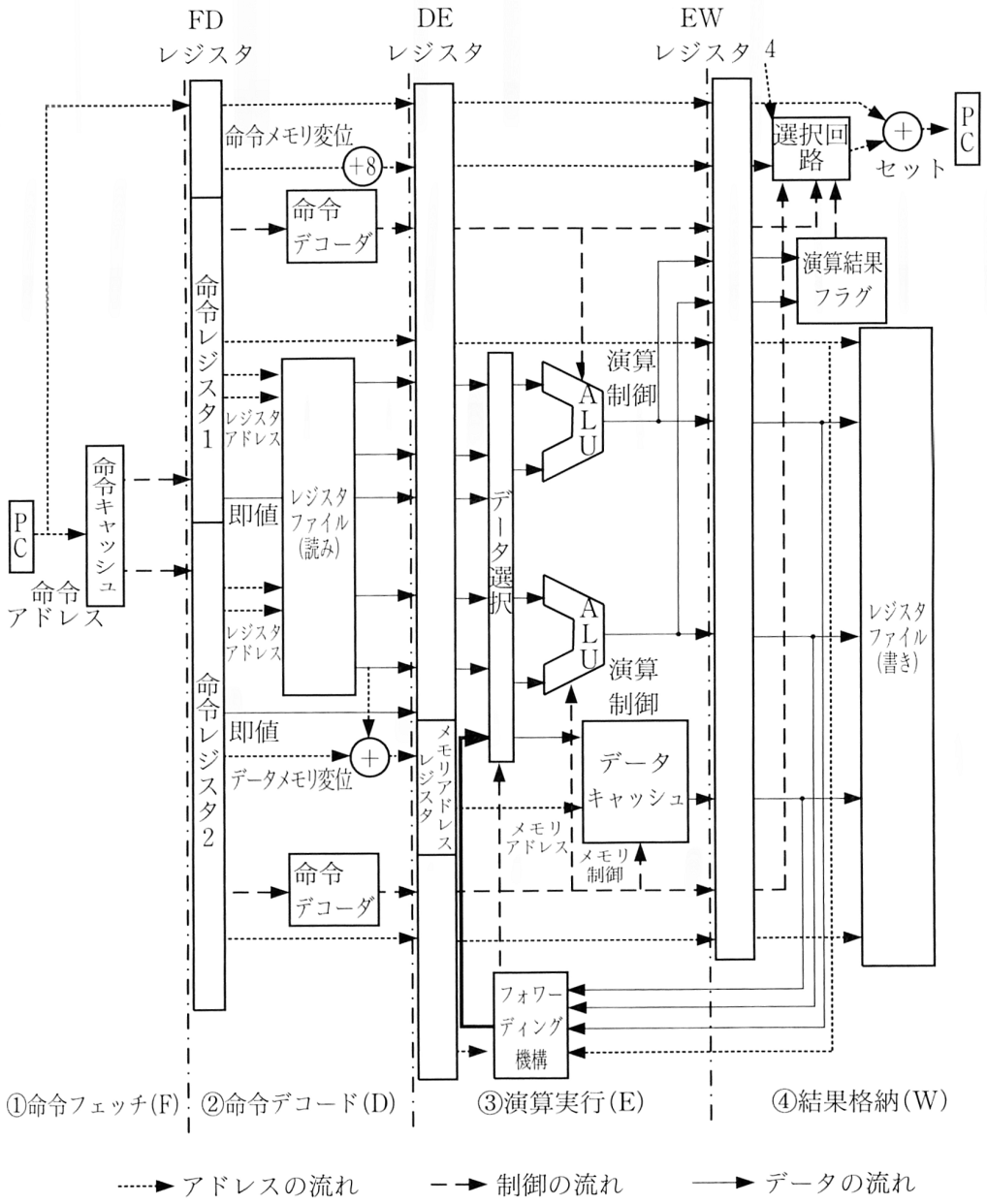


図 6.2　命令レベル並列処理の入ったパイプライン

6.1.2並列処理パイプライン

- 命令キャッシュからの読み取り，ALUなどを並列化
- ALUの演算とメモリアクセスは並列処理可能
- 並列度：P

命令レベル並列処理に必要な事項

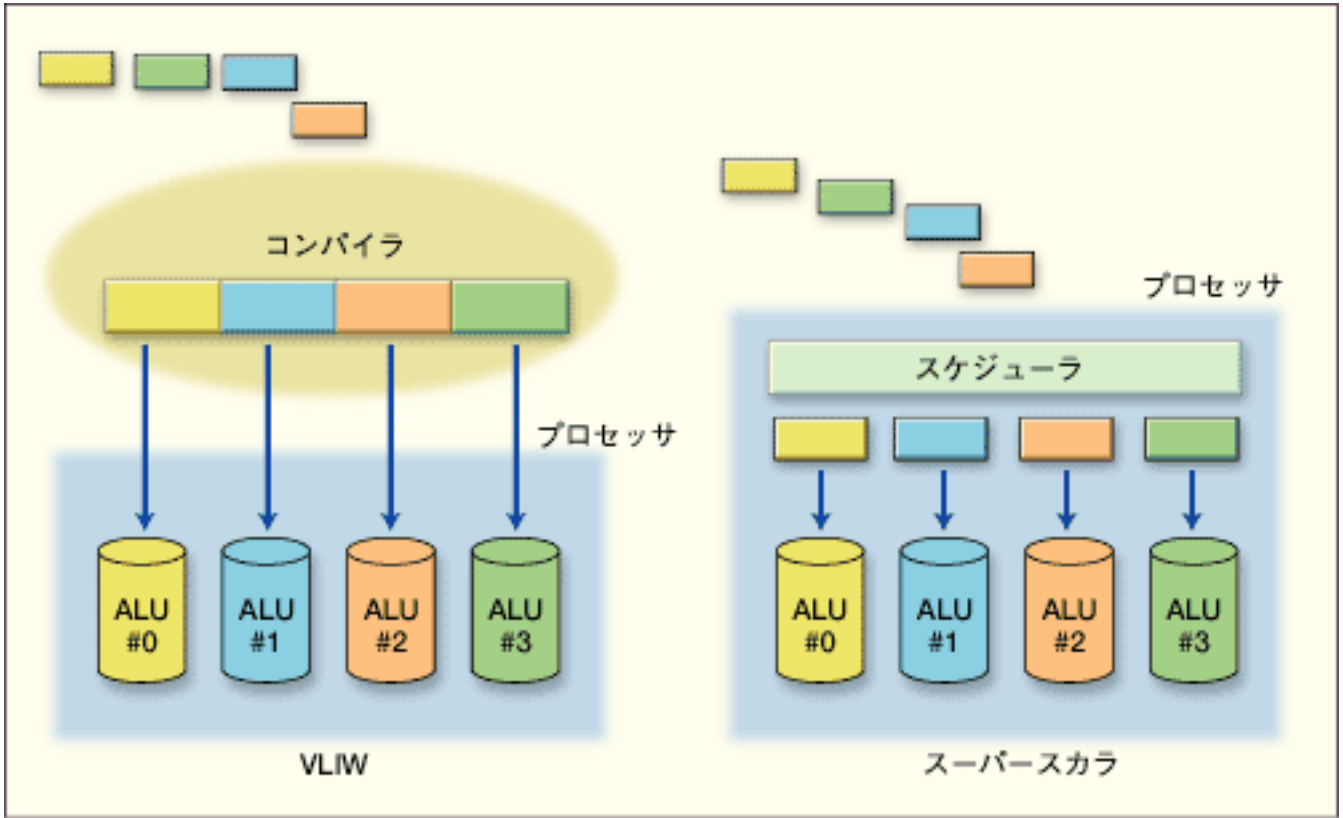
6.A 命令レベル並列処理に必要な事項		
①	ハードウエア資源の投入	命令キャッシュのバス幅，パイプラインレジスタ，デコーダ，演算フラグなどの必要個数がP倍になる.
②	レジスタファイルのポート数	読み出し用ポートが2P個，書き込み用がP個必要になる.
③	フォワーディング機構	前後の命令間のデータハザード解消のために用いられるフォワーディング機構を並列処理下で用いる場合，演算ユニットの出力が，全ての演算結果の入力にフォワードされなければならない．フォワードのデータ線とマルチプレクサのためにP^2に比例するハードウエアが必要になる.
④	並列実行の制御	同時に実行される命令間に依存関係がないことを保証しなければならない．依存関係がある場合には，並列実行を待たせなければならない．並列実行される命令間のデータハザードは，フォワーディングによっても解消されない.

Very Long Instruction Word

6.2 VLIW

6.2.1 VLIWプロセッサの構成と動作

- Very Long Instruction Wordとは，並列実行できる複数個の命令をあらかじめ一つの命令としてまとめておく方法.
  - ハードウエアがシンプルになる反面，ハザードを回避する高度なコンパイラが必要になる.



6.2.2 VLIWの特徴

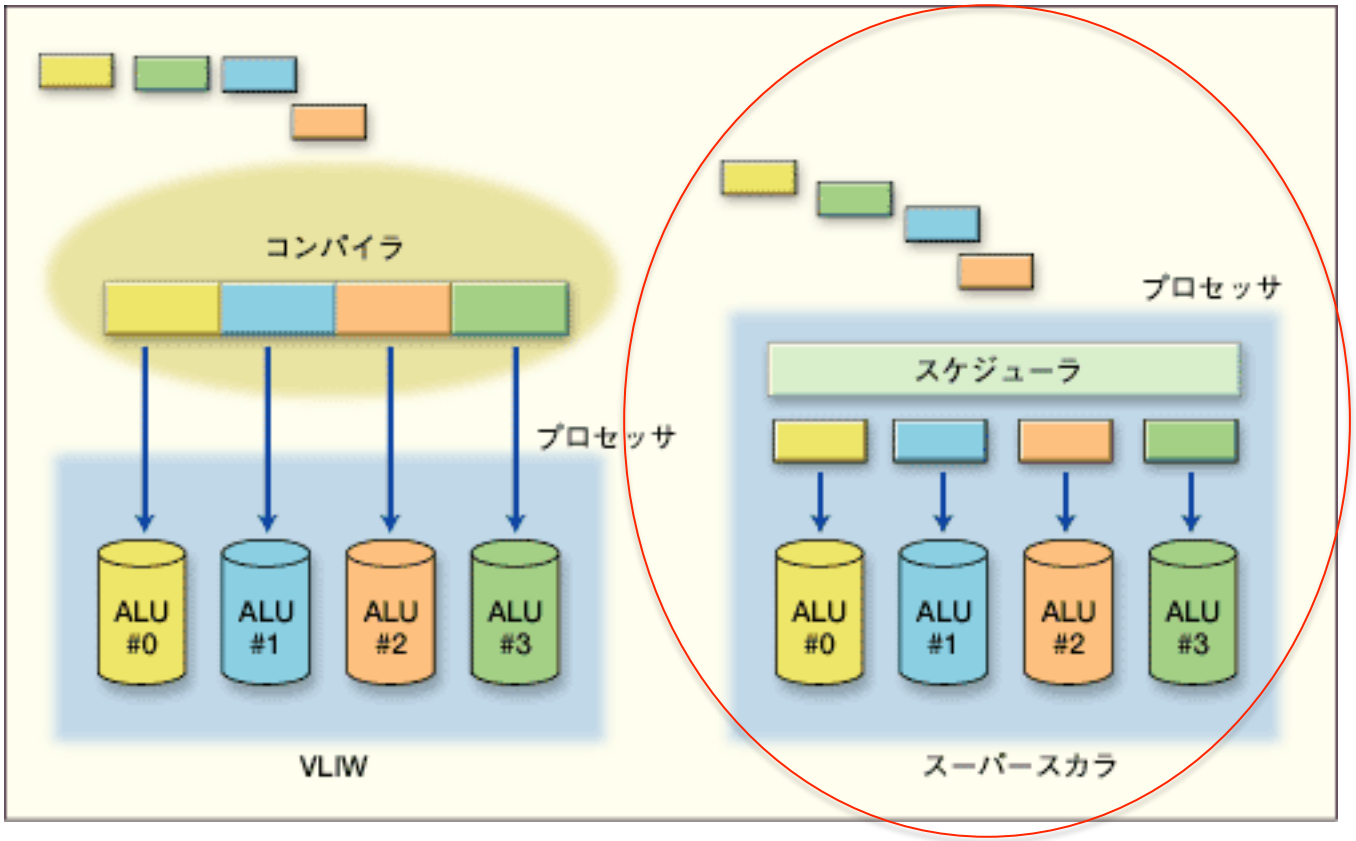
- ハザード検出にハードウエアを用いない

  メリット

  – 制御が簡単→回路もシンプルになる→クロックが上げられる.

  デメリット

  – 機械語プログラムがハードウエアによって規定され，透過性が損なわれる.
  – コンパイラがハザード検出をあらかじめ行うが条件分岐がある場合の扱いが難しい→トレーススケジューリング
  – 並列化が出来ない場合にはNOPで埋めるため，無駄


コンパイラ
プロセッサ
ALU #0
ALU #1
ALU #2
ALU #3
VLIW
プロセッサ
スケジューラ
ALU #0
ALU #1
ALU #2
ALU #3
スーパースカラ

6.3 スーパースカラ

6.3.1 スーパスカラプロセッサの構成と動作

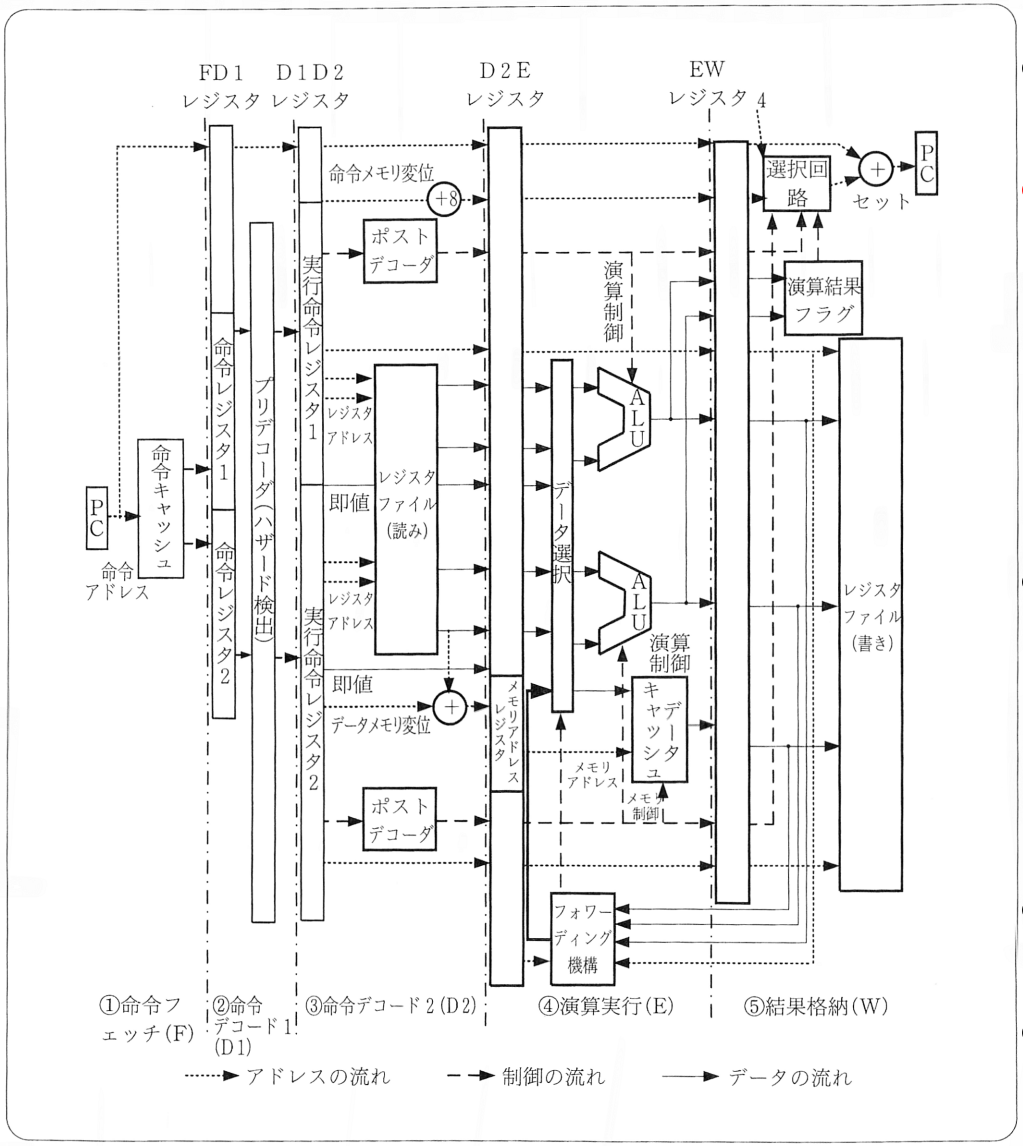


図 6.3　スーパスカラプロセッサのパイプライン基本構成

- 命令フェッチF
- 命令プリデコードD1
  - フェッチした命令と処理待ちの命令間の依存関係を調べ，，依存関係のない2命令を実行命令レジスタに入れる.
- 命令ポストデコードD2
  - 演算装置やメモリの制御信号を生成. レジスタファイルから演算対象の値を読み出す.
- 演算実行E
- 結果の格納W

6.3.2 並列処理とハザード

並列処理におけるハザード検出と対処法

- 構造ハザード
  - 図6.3ではデータキャッシュに対するアクセスは並列化されてないので，ロード・ストア命令を並列実行できない．このような場合は，ハードウエアで同時実行を避ける．
- データハザード
  - データ間の依存関係を発見した場合，後の命令は実行タイミングをずらす．
- 制御ハザード
  - 遅延分岐の間で実行する命令数が増える．分岐予測が外れた場合のペナルティも増すので，新たな対策が必要．

6.3.3 VLIWとスーパスカラの比較

表 6.1　VLIW とスーパスカラの比較

項　目	VLIW	スーパスカラ
透過性・互換性	×	○
ハザード検出・並列化	静的（コンパイラ）	動的（ハードウェア）
ハードウェア	簡　単	複　雑
制御の遅延	小	大
命令フィールドのむだ	有	無

６．４　静的最適化

6.4.1 機械語プログラムと命令間依存性

ハザードを減らすためにコンパイラが出来ること

1. 依存関係を解消あるいは緩和すること

2. 依存関係のある命令群をプログラム上で離れた位置に配置すること

6.4.2 ループアンローリング

- 小さなループを何周かまとめて一つのループにする. →分岐命令によるハザードを軽減する.

6.B 配列要素に定数を加えるプログラム

```
for (i=0; i<100; i++) a[i]=a[i]+5;
```

6.C 配列要素に定数を加えるプログラム(アセンブリ言語)

	addi r1, r0, 0	i=0	r1 : i
	addi r2, r0, 100	r2=100	
ForLoop:	lw r4, 0(r3)	r4=a[i]	(r3): a[0]
	addi r4, 5, r4	r4=a[i]+5	
	sw r4, 0(r3)	a[i]=r4	
	addi r1, r1, 1	i=i+1	
	addi r3, r3, 4	a[i] の位置を4byteずらす.	
	blt r1, r2, ForLoop	if (i<100) goto ForLoop	

ループアンローリング

- ALU,ロードストアユニットを2つ持つCPUを仮定
- 分岐予測を行う.
- r4のデータハザードのためストールが起きている.

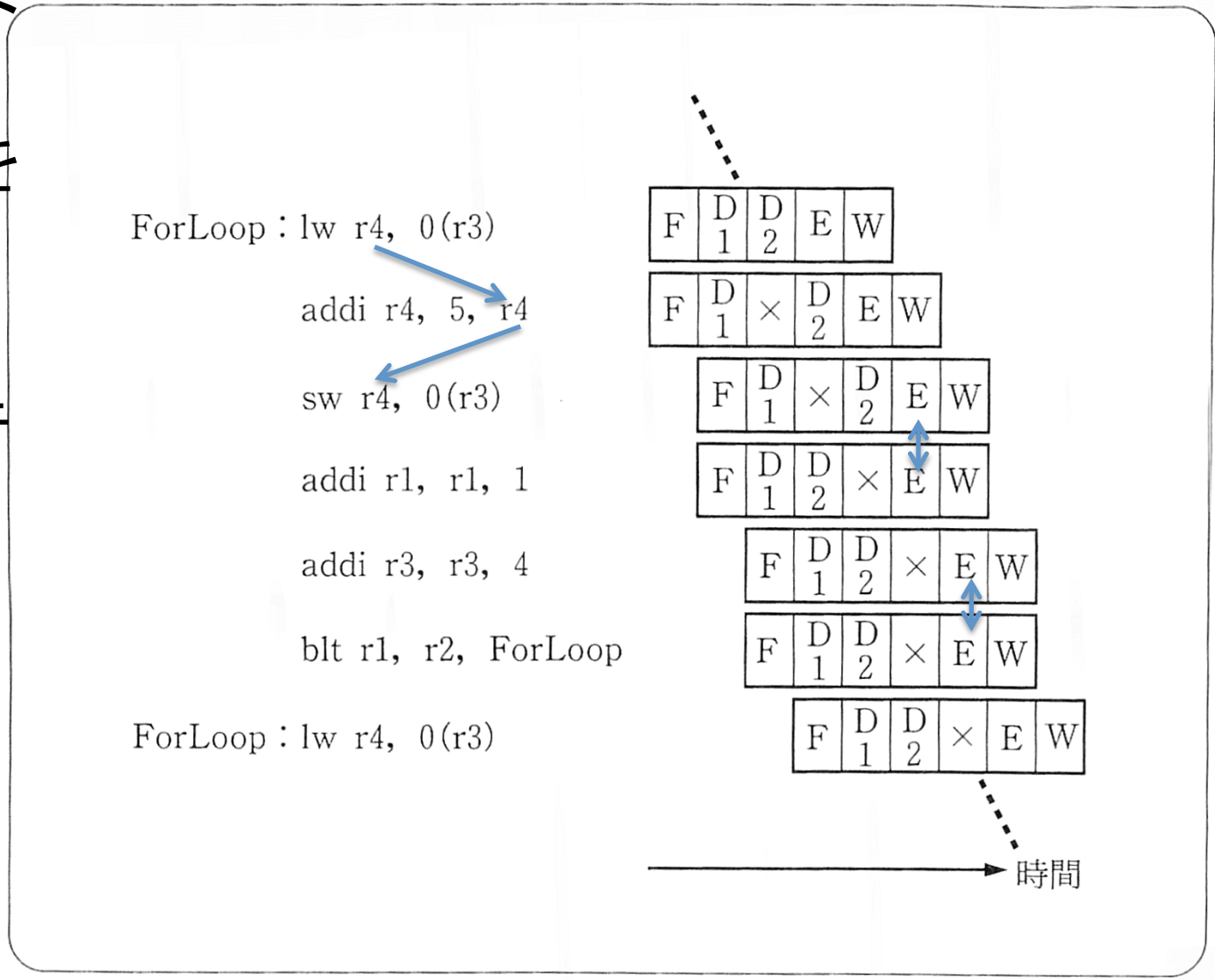


図 6.4　配列要素に定数を加えるプログラムのパイプライン進行

ループアンローリング

6.D ループアンローリングを施したプログラム(アセンブリ言語)

	addi r1, r0, 0	i=0　　r1 : i
	addi r2, r0, 100	r2=100
ForLoop:	lw r4, 0(r3)	r4=a[i]　　(r3): a[0]
	lw r5, 4(r3)	r5=a[i+1]
	lw r6, 8(r3)	r6=a[i+2]
	lw r7, 12(r3)	r7=a[i+3]
	addi r4, 5, r4	r4=r4+5
	addi r5, 5, r5	r5=r5+5
	addi r6, 5, r6	r6=r6+5
	addi r7, 5, r7	r7=r7+5
	sw r4, 0(r3)	a[i]=r4
	sw r5, 4(r3)	a[i+1]=r5
	sw r6, 8(r3)	a[i+2]=r6
	sw r7, 12(r3)	a[i+3]=r7
	addi r1, r1, 4	i=i+4
	addi r3, r3, 16	a[i] の位置を16byteずらす.
	blt r1, r2, ForLoop	if (i<100) goto ForLoop

ループアンローリングの効果

- r4に関する依存関係に起因するデータハザードが解消.
- bltの命令が1/4になるため，制御ハザードが起きにくい.

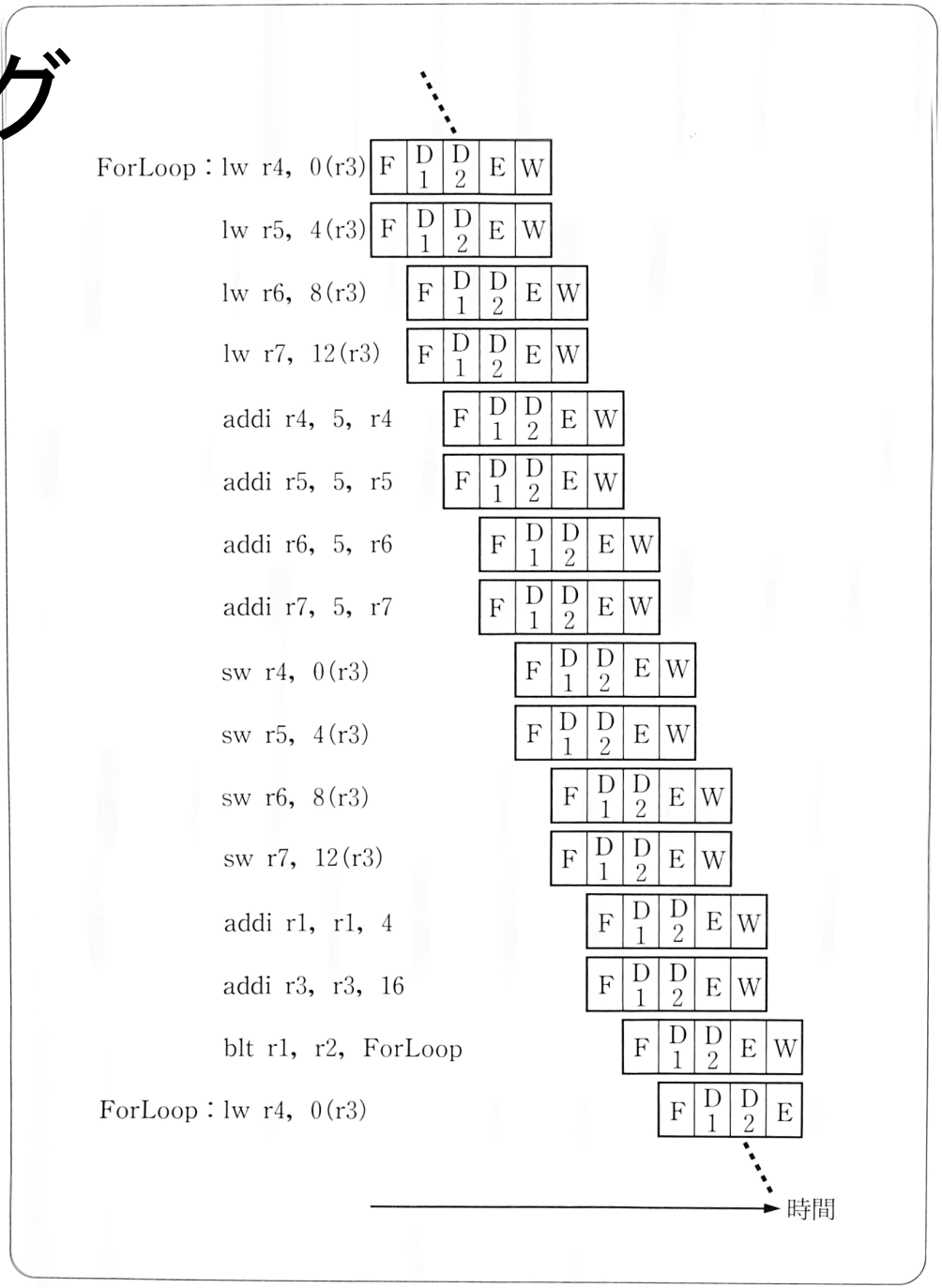


図 6.5　ループアンローリングを施したプログラムのパイプライン進行

6.4.3 ソフトウエアパイプライニング

- ループ間にまたがって命令を移動させ，依存関係のある命令同士を離す(教科書は誤り).

6.E ソフトウエアパイプライニングを施したプログラム(アセンブリ言語)

	addi r1, r0, 0	r1=0　　r1 : i
	addi r2, r0, 100	r2=100
	lw r4, 0(r3)	r4=a[0]　　(r3): a[0]
	addi r5, 5, r4	r5=r4+5
	lw r4, 4(r3)	r4=a[1]
ForLoop	sw r5, 0(r3)	a[i]=r5
	addi r5, r4, 5	r5=r4+5
	lw r4, 4(r3)	r4=a[i+1]
	addi r1, r1, 1	r1=r1+1
	addi r3, r3, 4	a[i] の位置を4byteずらす.
	blt r1, r2, ForLoop	if (i<100) goto ForLoop

ソフトウエアパイプライニング

- ループ間にまたがって命令を移動させ，依存関係のある命令同士を離す(教科書は誤り).

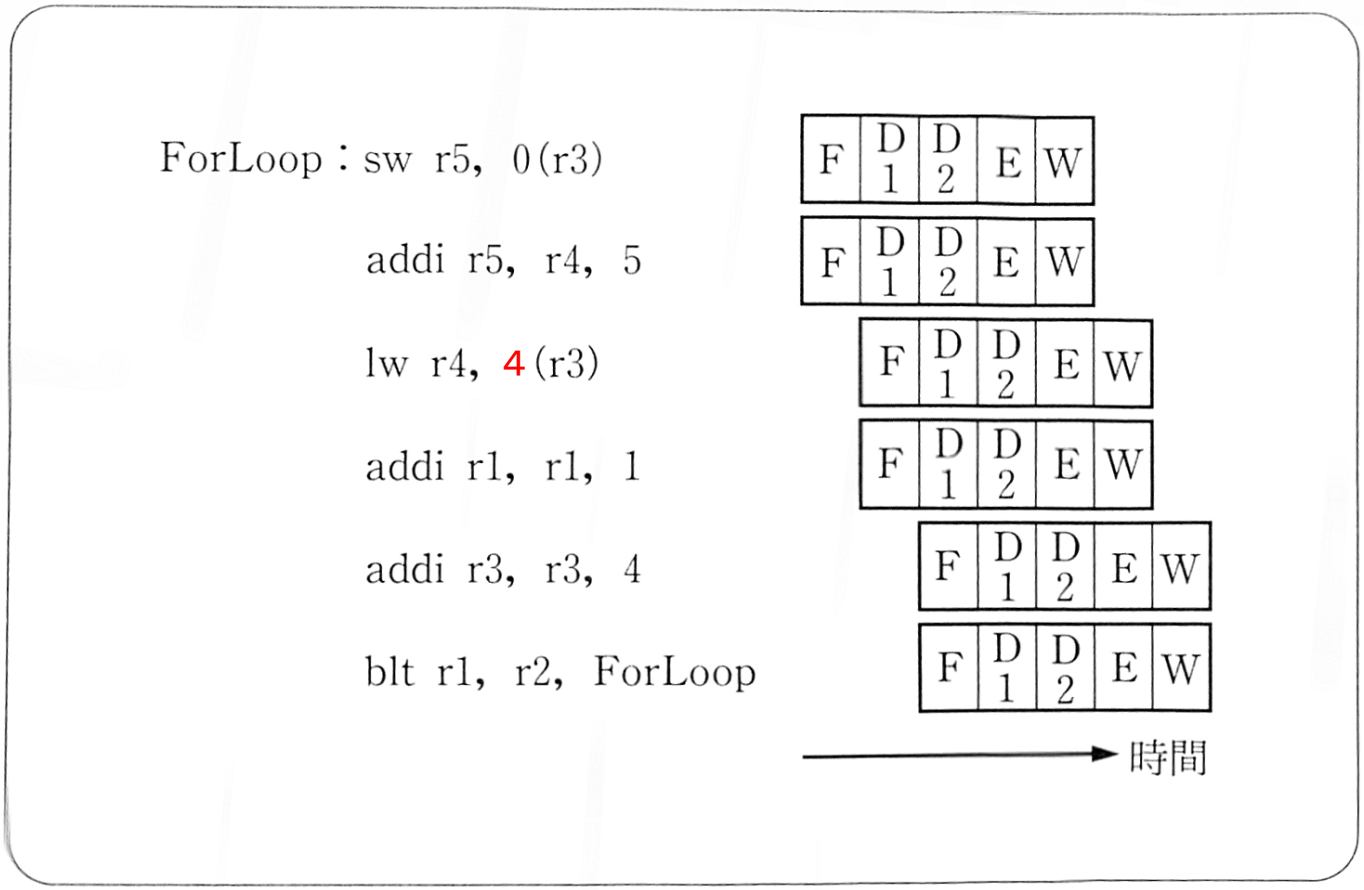
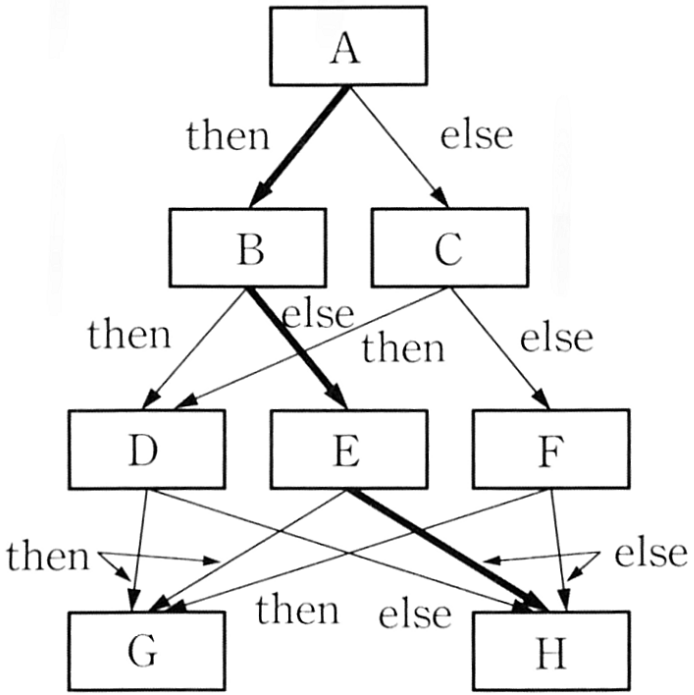


図 6.6　ソフトウェアパイプライニングを施したプログラムのパイプライン進行

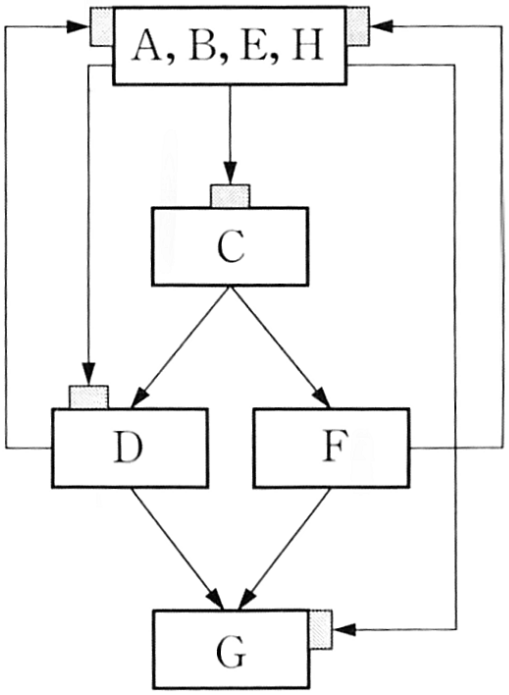
6.4.4 トレーススケジューリング

- 分岐予測をして複数の基本ブロックを統合し，制御依存を減らす手法.



太線が，最も確率の高い分岐パス

（a） 制御フローグラフ



ブックキーピング

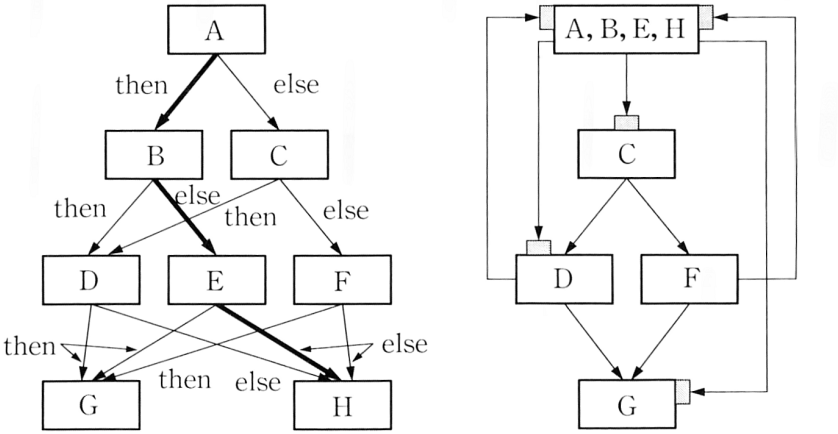
（b） 第一段階の制御フローグラフ

基本ブロック：ある分岐命令の直後から，次の分岐命令までの命令列.

制御フローグラフ：基本ブロック間の依存関係をグラフで表したもの.

トレーススケジューリング

6.F トレーススケジューリングの手順	
①	実行される確率の最も高い分岐パターンを調べる.（図中A,B,E,H）
②	①のパターンの上にある基本ブロックを統合する. これをトレースと呼ぶ.
③	トレースに命令スケジューリングを施すことで, トレース内の実行効率を高める. 分岐目入れを超えた命令移動も行う.
④	③によってトレース以外に分岐する場合に生じる不都合を防ぐため, 分岐先の入り口に補正用のコードを入れる. これをブックキーピングと呼ぶ.
⑤	トレースを除いた制御フローグラフにおいて, ①～④を繰り返す.



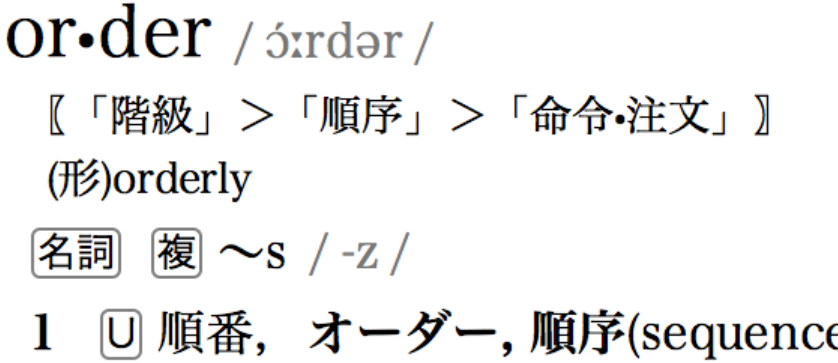
（a） 制御フローグラフ　　（b） 第一段階の制御フローグラフ

ここまでは，静的な最適化でしたが，ここからは命令の実行順序を動的に入れ替える処理の話です．

6.5 アウトオブオーダ処理

6.5.1 アウトオブオーダ処理とは何か

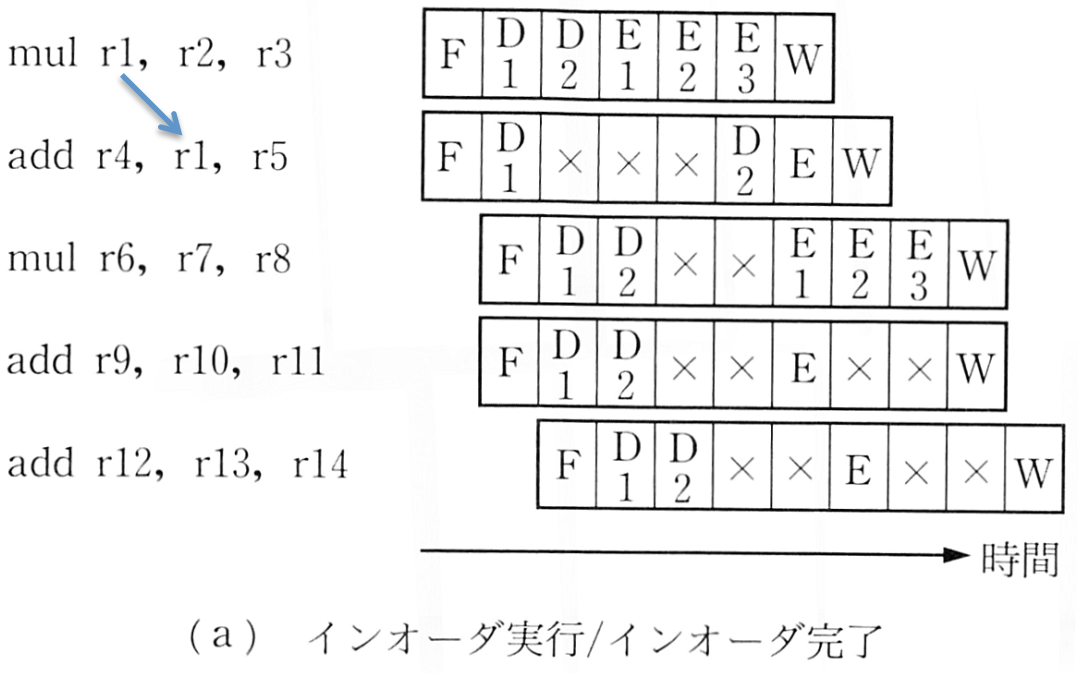
- 動的スケジューリングにおいて，命令の順序を入れ替えることを，out of order処理という．
- 入れ替えないことを in order処理という．



- out of order処理には，命令をEステージに入れる順番を入れ替えるout of order実行と，実行結果をレジスタに格納するWステージの順番を入れ替える out of order 完了とがある．

例6.1の解説−1

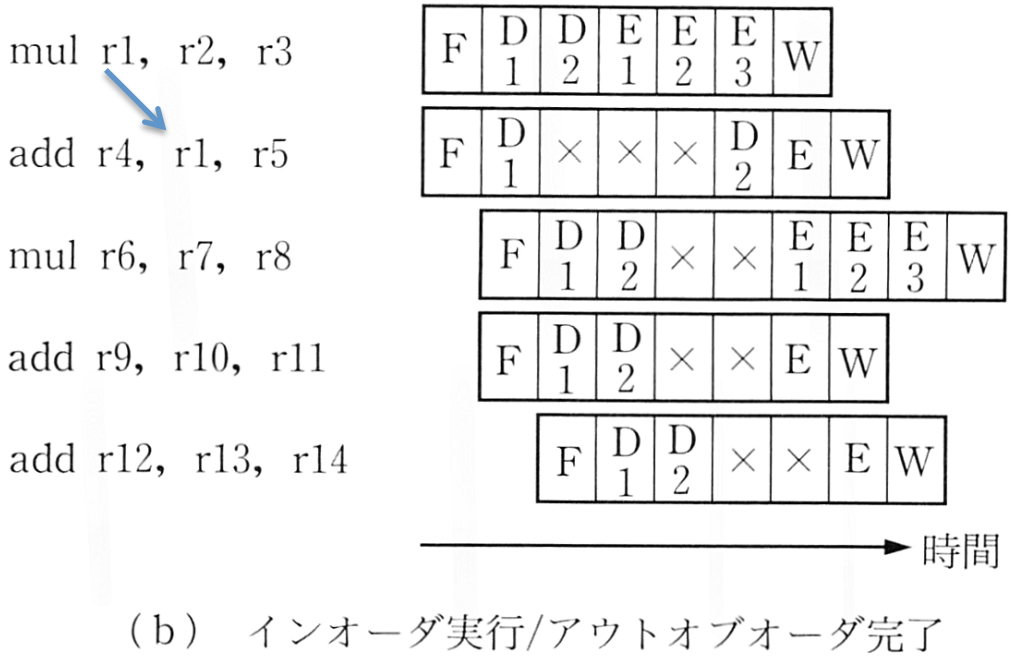
- 処理時間1のALU(整数加減算，論理演算器)を2つ，処理時間3の乗算器を2つ持つプロセッサがある．
- r1に関するデータハザードと，スーパースカラによる待ちが入り11クロック消費する．



（a） インオーダ実行/インオーダ完了

例6.1の解説-2

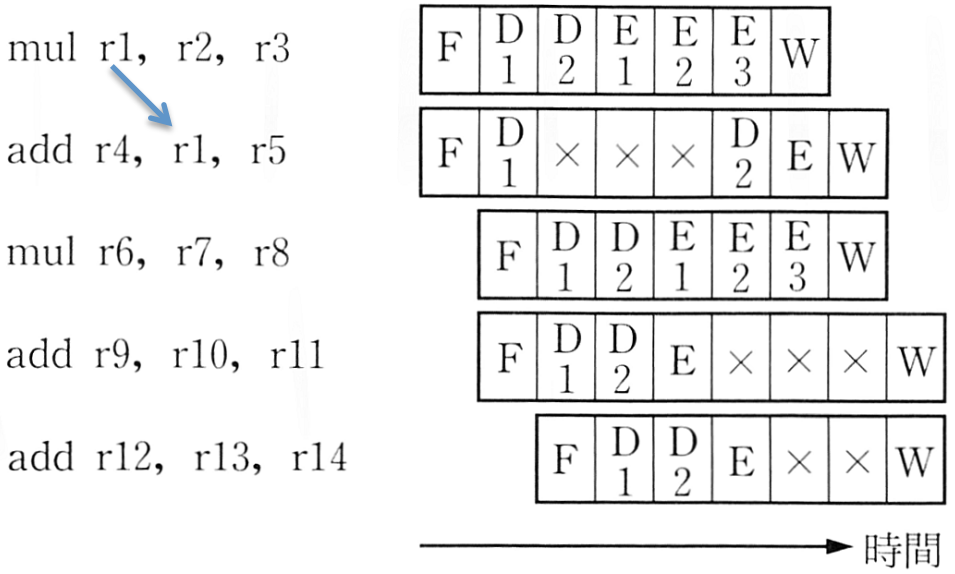
- インオーダ実行アウトオブオーダ完了
- この場合は，EとWの間のXが無くなる．
- この結果，10クロックで終了できる．



（b） インオーダ実行/アウトオブオーダ完了

例6.1の解説–3

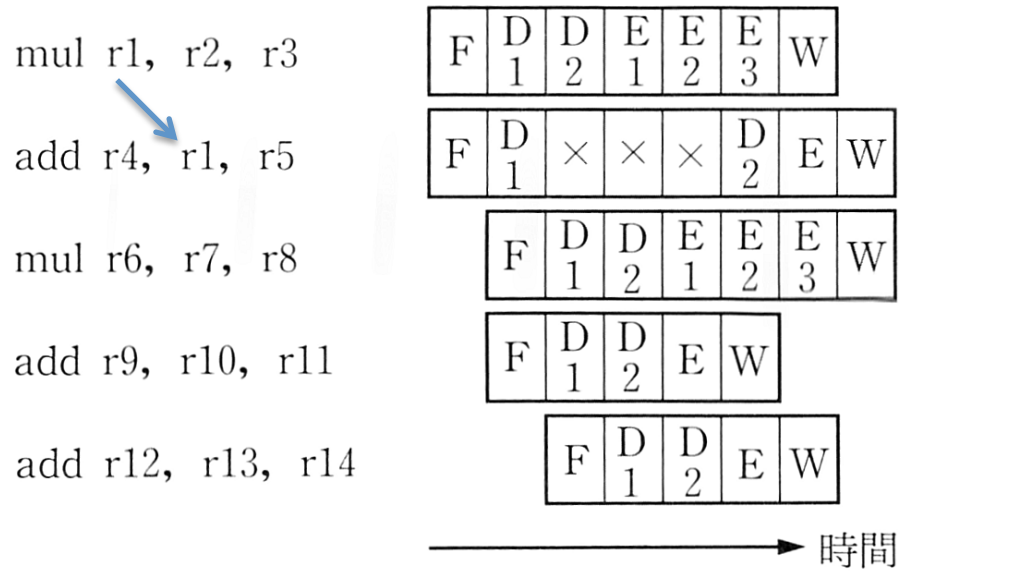
- アウトオブオーダ実行インオーダ完了
- この場合は，DとEの間のXが無くなる．
- この結果，9クロックで終了できる．



（c） アウトオブオーダ実行/インオーダ完了

例6.1の解説–4

- アウトオブオーダ実行アウトオブオーダ完了
- この場合は，DとEの間のXと，EとWの間のXが無くなる.
- この結果，8クロックで終了できる.

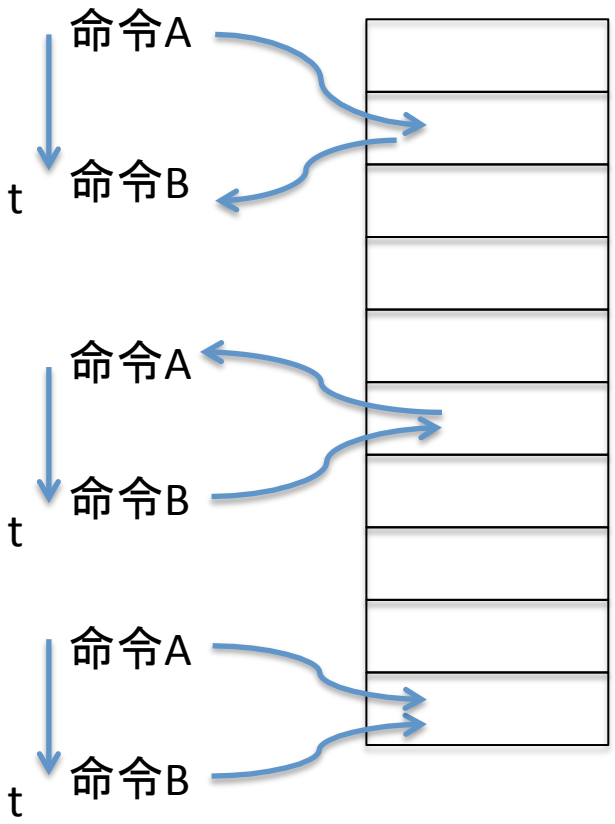


（d） アウトオブオーダ実行/アウトオブオーダ完了

6.5.2 データ依存再考

- アウトオブオーダ処理を許容すれば，これまで考えていなかったデータ依存関係を考えなければならなくなる．

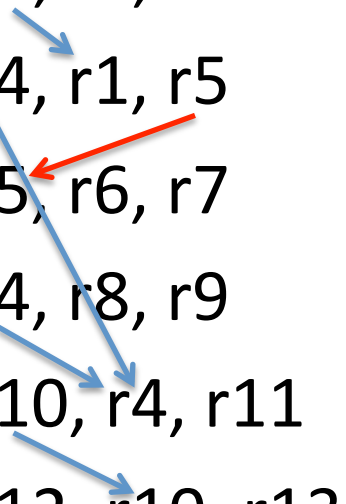
6.G データ依存の分類		
①	フロー依存	命令Aで書き込んだ値を後続の命令Bで読み出すことで起こるA⇒Bの依存関係．真の依存関係とも言う．
②	逆依存	命令Aで読み出したレジスタ（メモリ語）に後続の命令Bが書き込みを行うことで起こるA⇒Bの依存関係．
③	出力依存	命令Aで書き込んだレジスタ（メモリ語）に後続の命令Bが再度書き込みを行うことで起こるA⇒Bの依存関係．



例 6.3

3種類の依存

① mul r1, r2, r3
② add r4, r1, r5
③ add r5, r6, r7
④ add r4, r8, r9
⑤ add r10, r4, r11
⑥ add r12, r10, r13



- フロー依存
  - ①⇒②(r1)
  - ②⇒⑤(r4)
  - ④⇒⑤(r4)
  - ⑤⇒⑥(r10)
- 逆依存
  - ②⇒③(r5)
- 出力依存
  - ②⇒④(r4)

例 6.3 ハザードの種類

表 6.2　データ依存とデータハザード

データ依存	データハザード
フロー依存	RAW（read after write）ハザード
逆依存	WAR（write after read）ハザード
出力依存	WAW（write after write）ハザード

例6.3 ハザードによるストール

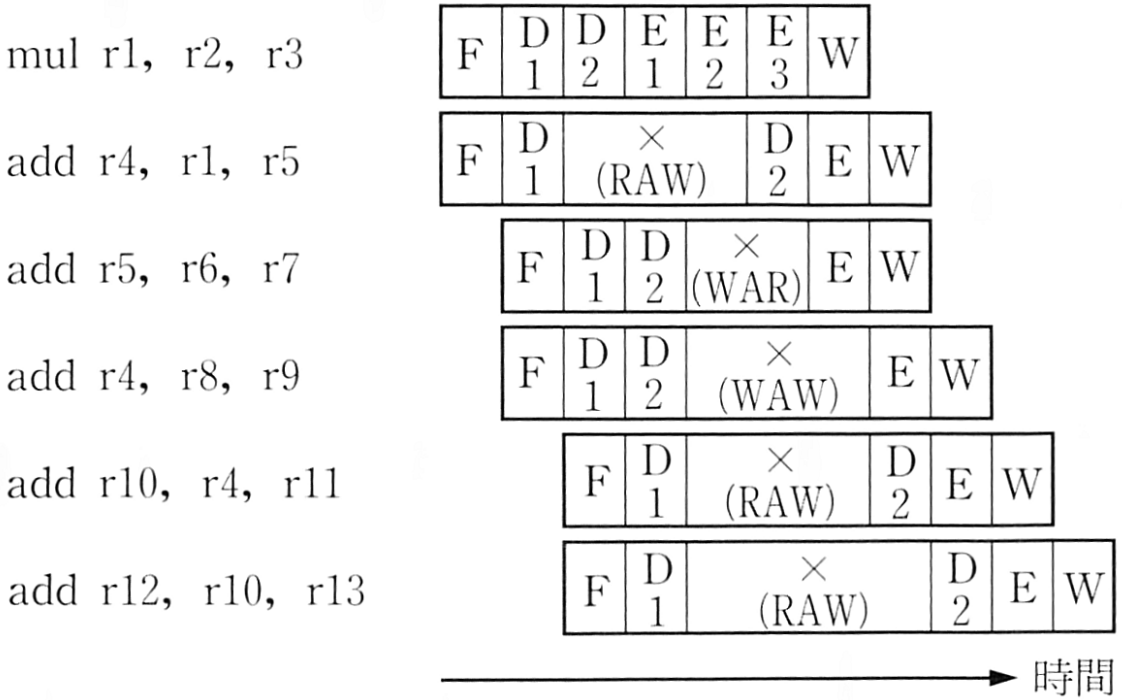


図 6.9　例 6.2 のプログラムのパイプライン実行

3種類の依存

①mul r1, r2, r3

②add r4, r1, r5

③add r5, r6, r7

④add r4, r8, r9

⑤add r10, r4, r11

⑥add r12, r10, r13

6.5.3 アウトオブオーダ処理の機構

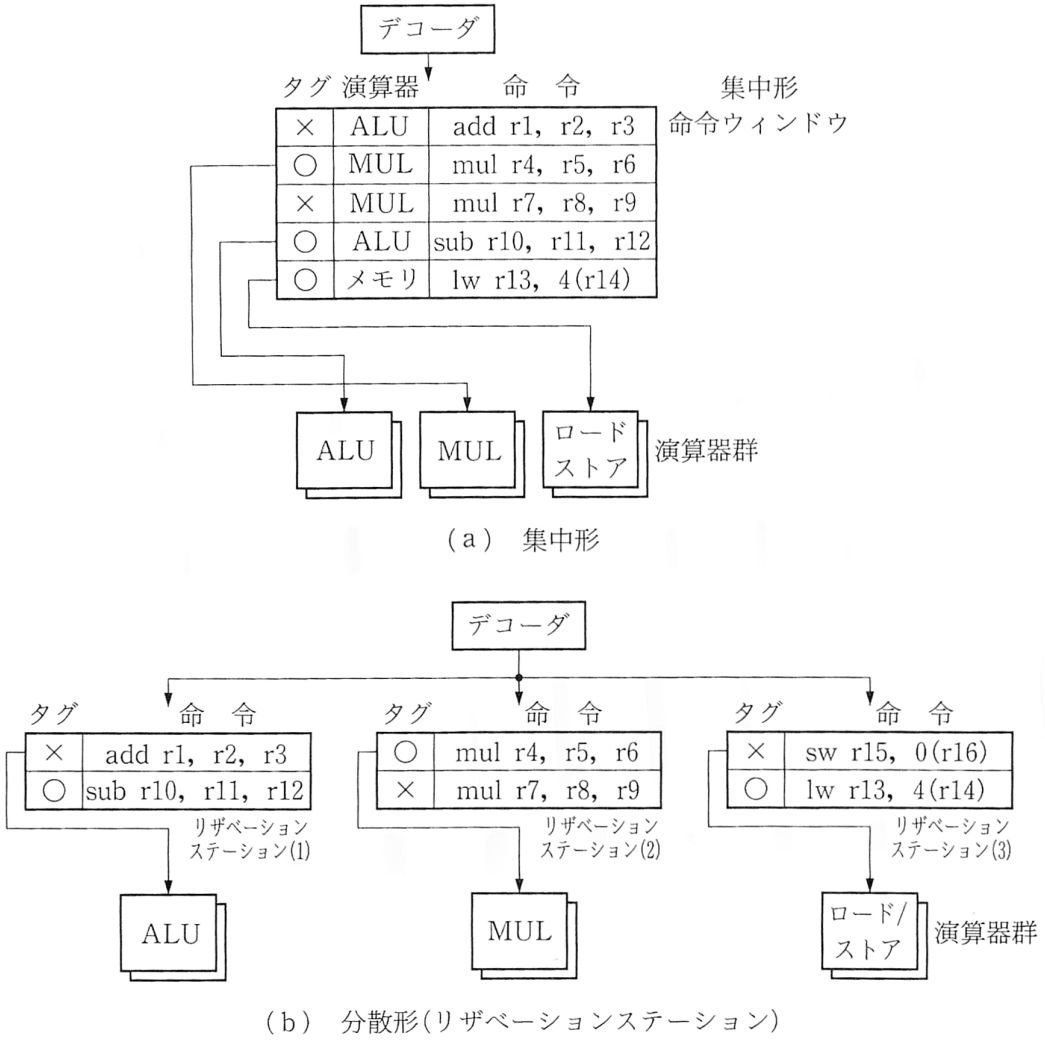


図 6.10 命令ウィンドウ

- 多数の命令の中で現在実行可能な命令を選び出す機構が「命令ウインドウ」
- デコードが終わった命令を格納しておく.
- タグは入力データが揃ったかどうか
- 演算器は使用するリソースを表す.
- 集中型と分散型がある

逆依存，出力依存を解決する方法

6.6 レジスタリネーミング

6.6.1 ソフトウエアによるレジスタリネーミング

例6.2

① mul r1, r2, r3
② add r4, r1, r5
③ add r5, r6, r7
④ add r4, r8, r9
⑤ add r10, r4, r11
⑥ add r12, r10, r13

例6.2’

① mul r1, r2, r3
② add r4, r1, r5
③ add r14, r6, r7
④ add r15, r8, r9
⑤ add r10, r4, r11
⑥ add r12, r10, r13

例6.2'の実行過程

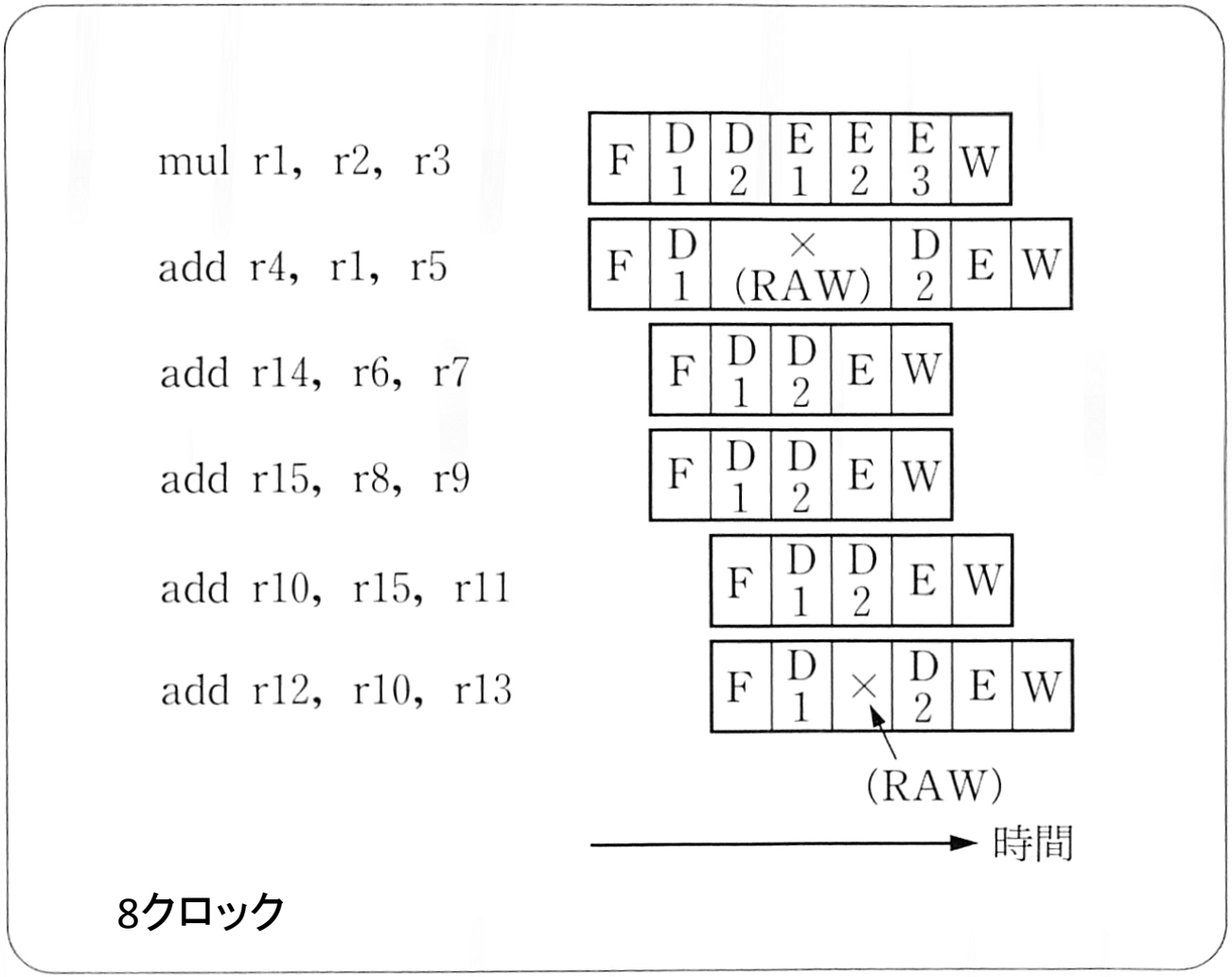
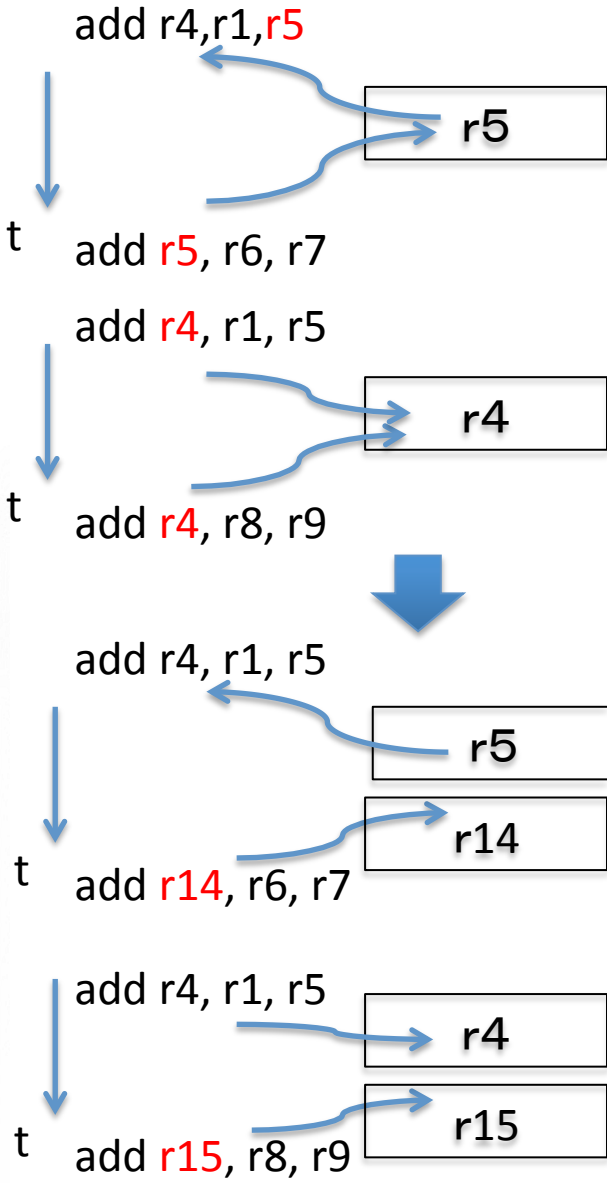


図 6.11　レジスタリネーミングによる並列性の向上

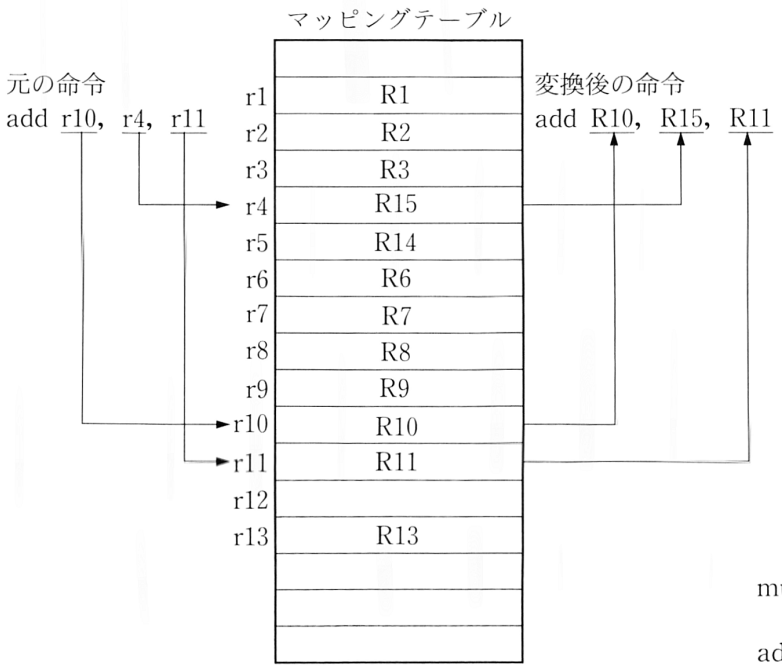


6.6.2 ハードウエアによるレジスタリネーミング(1) -マッピングテーブル-

6.H 静的なレジスタリネーミングの問題点	
①	機械語プログラムで指定できるレジスタ数には限界がある.
②	CPUアーキテクチャの細部（特に並列動作可能なユニット数）にプログラムが影響を受けるため，透過性・互換性が失われる
③	機械語プログラムの変換の手間がかかる.

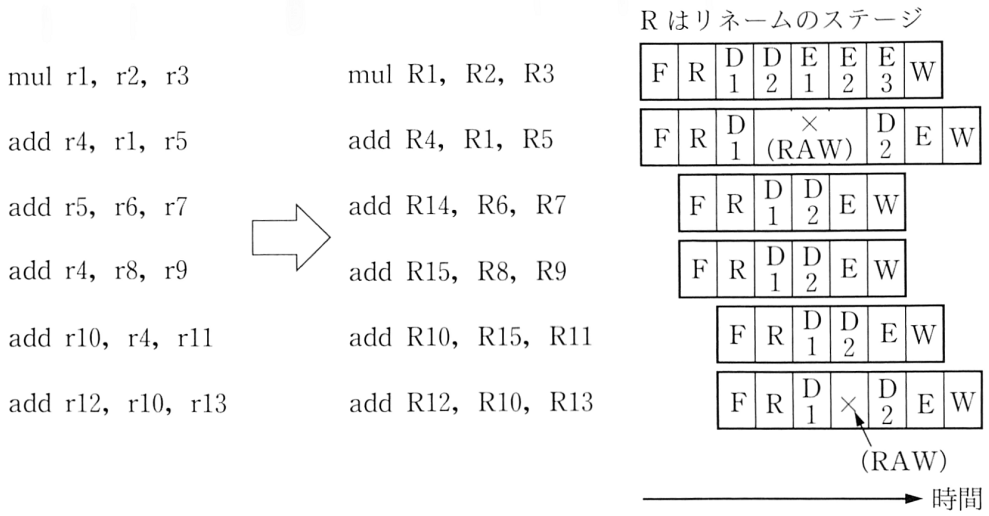
- ハードウエアによる動的なレジスタリネーミングが必要

マッピングテーブル



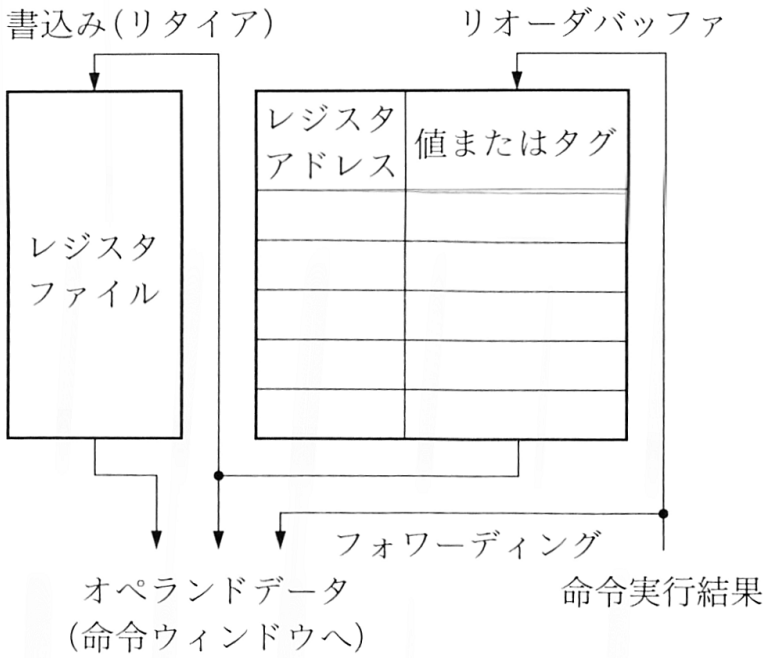
（a） マッピングテーブルによる命令の変換

- テーブルの作り方は省略
- Rステージが追加されている.（欠点）



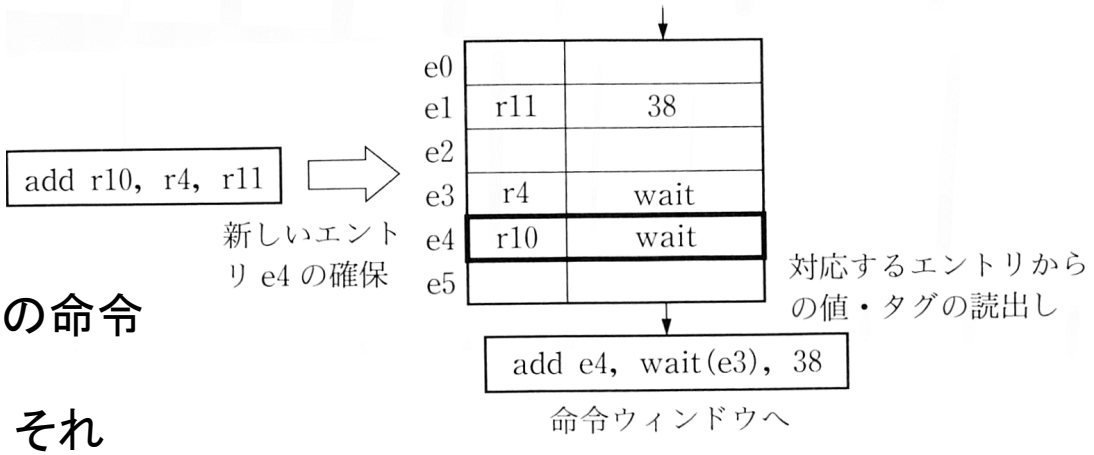
（b） マッピング機構を入れたパイプライン

6.6.3 ハードウエアによるレジスタリネーミング(1) –リオーダバッファ-



(a) リオーダバッファによるリネーミング

- レジスタのアドレスと値の組みから成るエントリ
- 命令をフェッチすると，その命令が書き出すレジスタに関するエントリを生成(初期の値はwait)
- 命令が読み出すレジスタの値のうち最新のものを検索する．(連想メモリ)
- 命令のレジスタをエントリー名とその値で置き換える．
- 値が確定した(waitではない)命令から実行ユニットに送る．



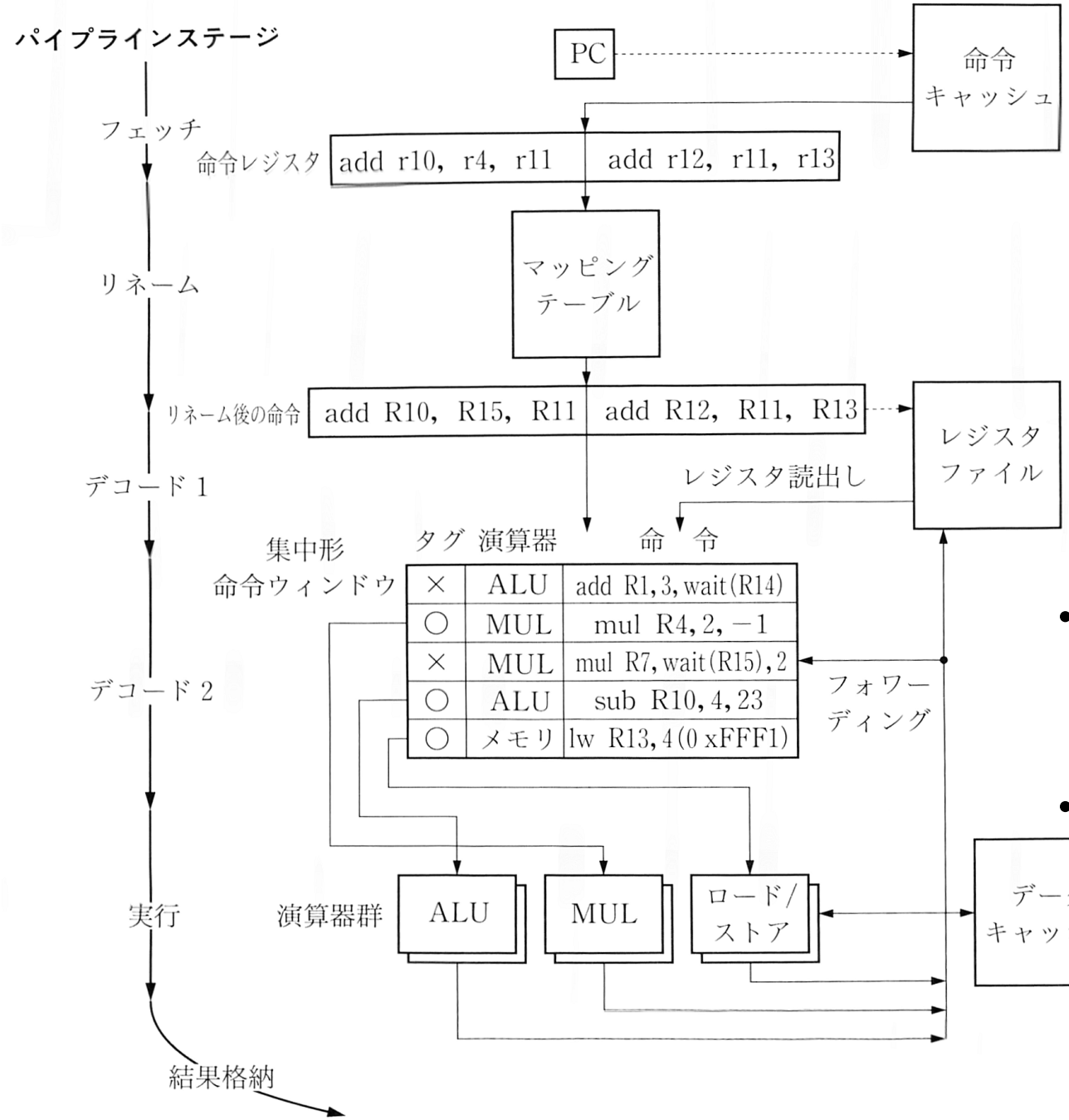
(b) リオーダバッファの動作

リタイアの2条件

① あるエントリを書き込んだ命令以前の命令が全て終了した．

② あるエントリのレジスタ値が確定し，それが書き込まれた．

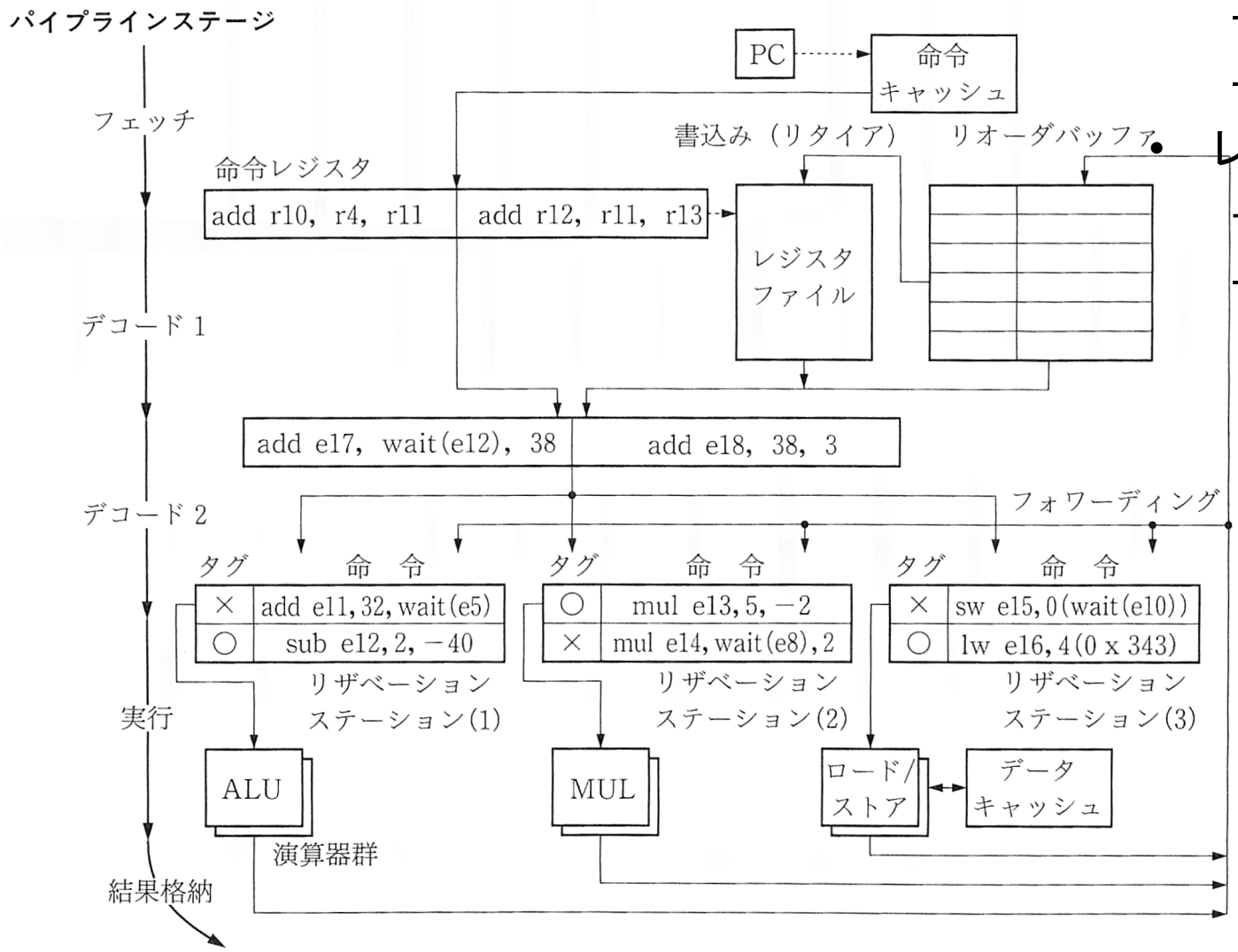
6.7 スーパスカラプロセッサの構成



6.7.1 アウトオブオーダ処理を行うプロセッサの構成

- 命令ウインドウ
  - 集中型
  - 分散型
- レジスタリネーミング
  - マッピングテーブル方式
  - リオーダバッファ方式

アウトオブオーダ処理を行うプロセッサの構成(2)



- 命令ウインドウ
  - 集中型
  - 分散型
- レジスタリネーミング
  - マッピングテーブル方式
  - リオーダバッファ方式

6.7.2 プロセッサの性能

- 概略性能

  プロセッサの性能 = クロック当たりの平均実行命令数 × クロック周波数
- クロック当たりの平均実行命令数は，フォワーディング，命令スケジューリング，分岐予測，キャッシュ，命令レベル並列処理，アウトオブオーダ処理等によって改善できる.
- クロック周波数は，パイプラインの各ステージの処理の複雑さによって上限が決まる.

本日の講義の範囲

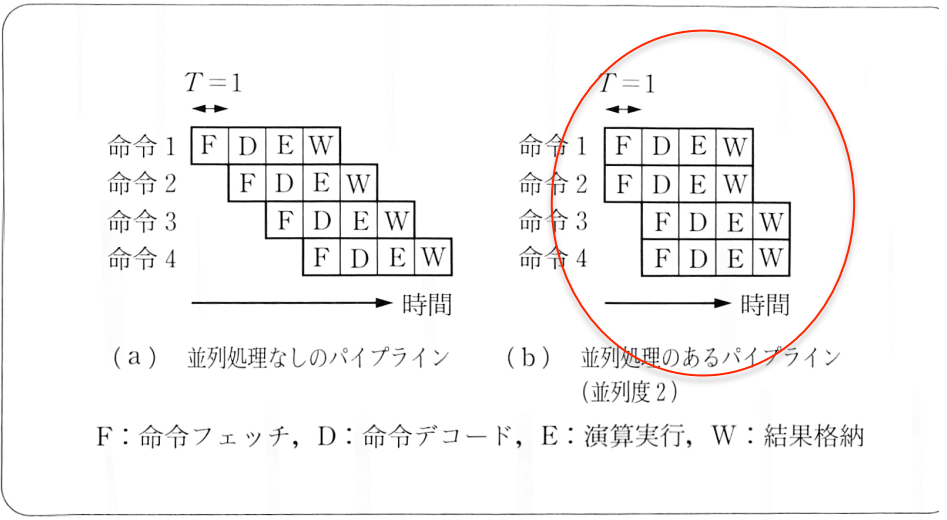


図 6.1 並列処理のある命令パイプライン

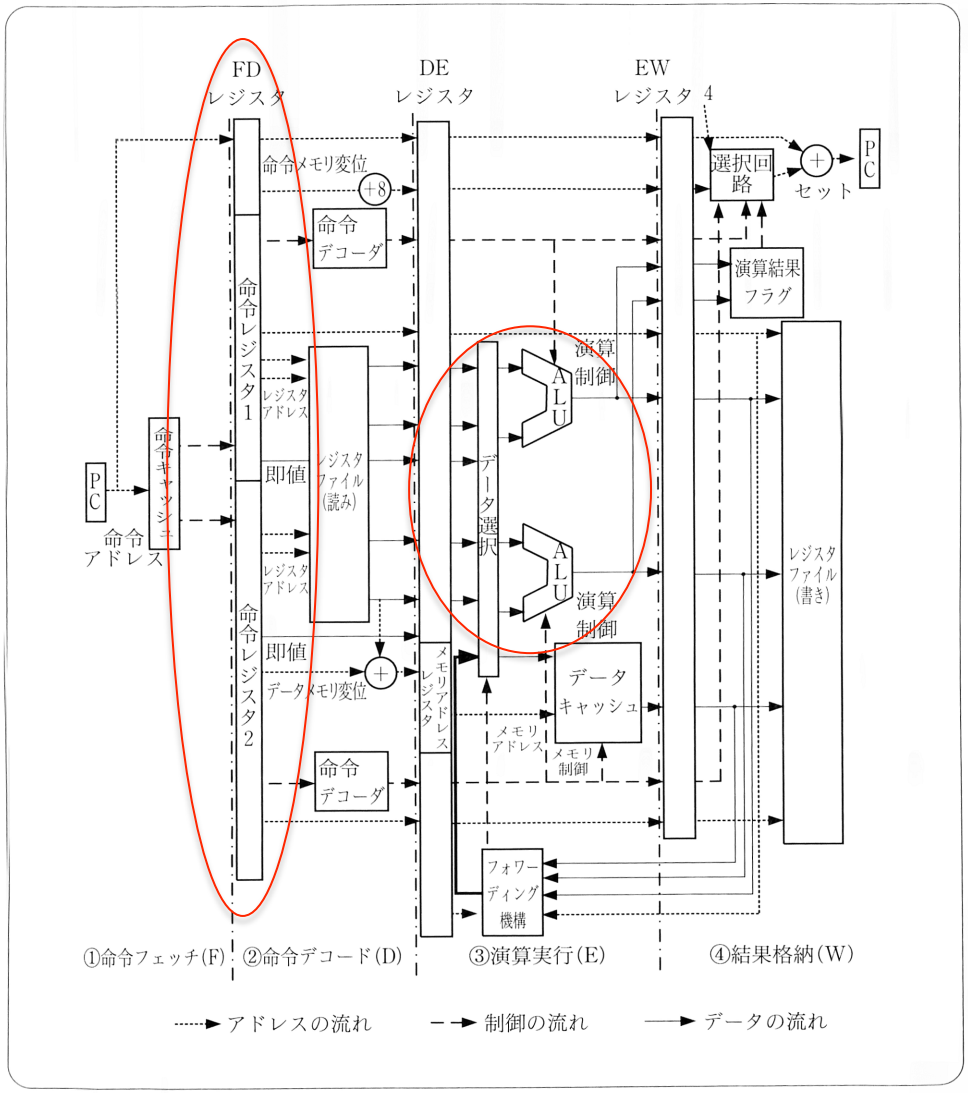


図 6.2 命令レベル並列処理の入ったパイプライン