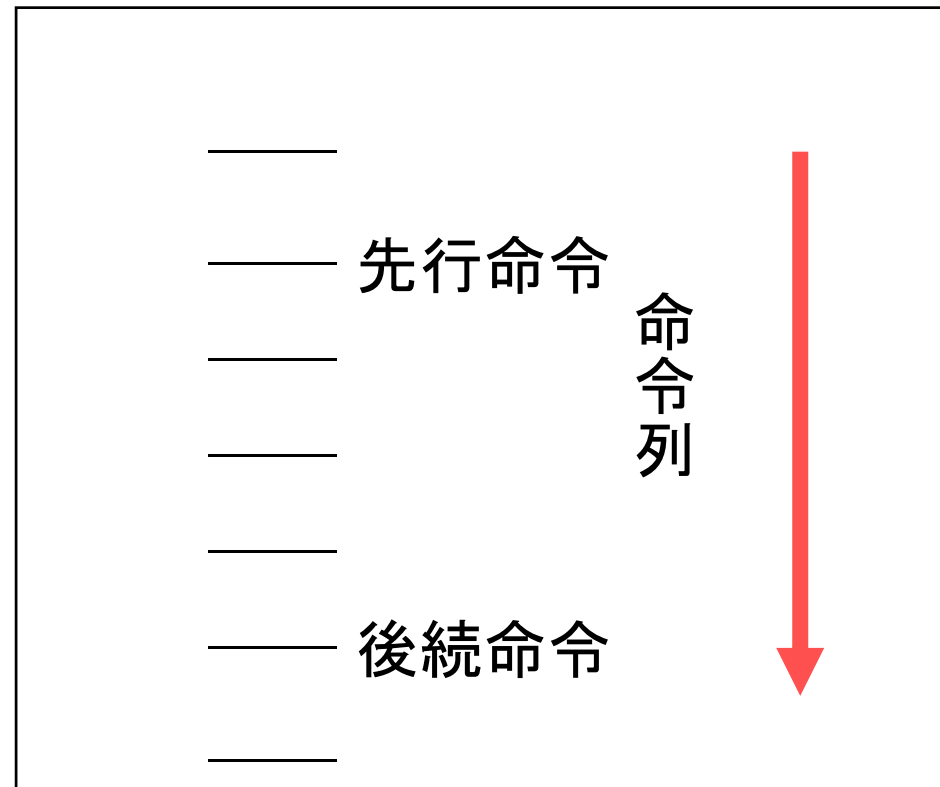


第6章 機械命令レベルでの並列処理

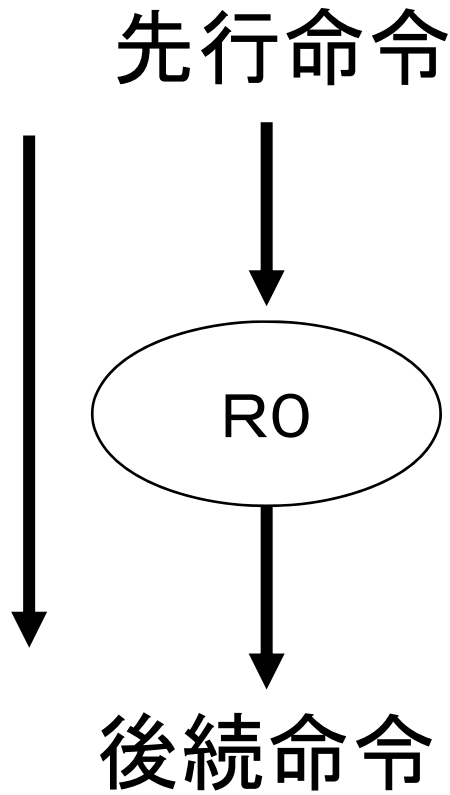
6.1 機械命令の依存関係

先行／後続命令の並列実行を
阻むもの

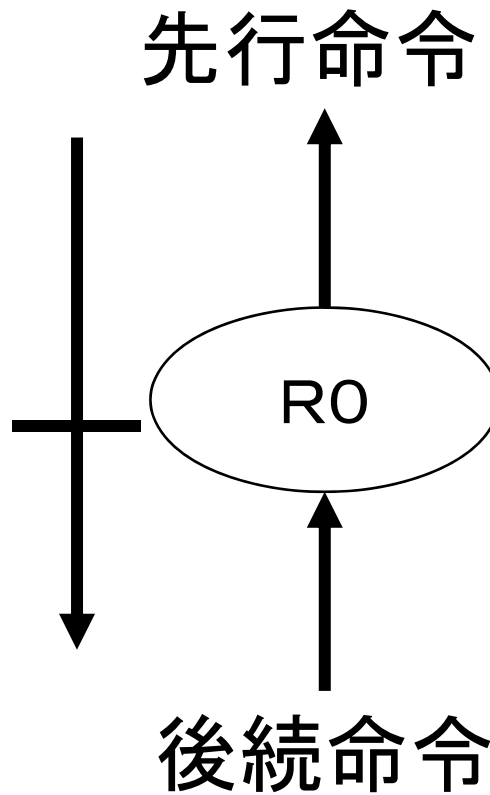
データ依存
制御依存
資源競合



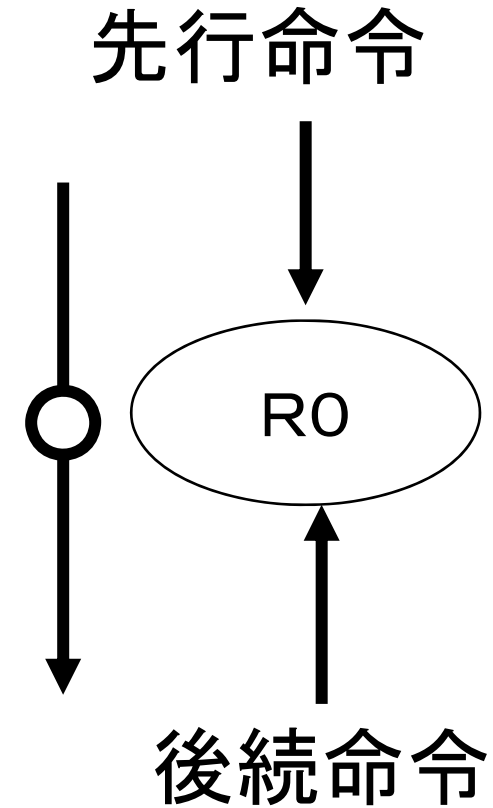
フロー依存



逆依存

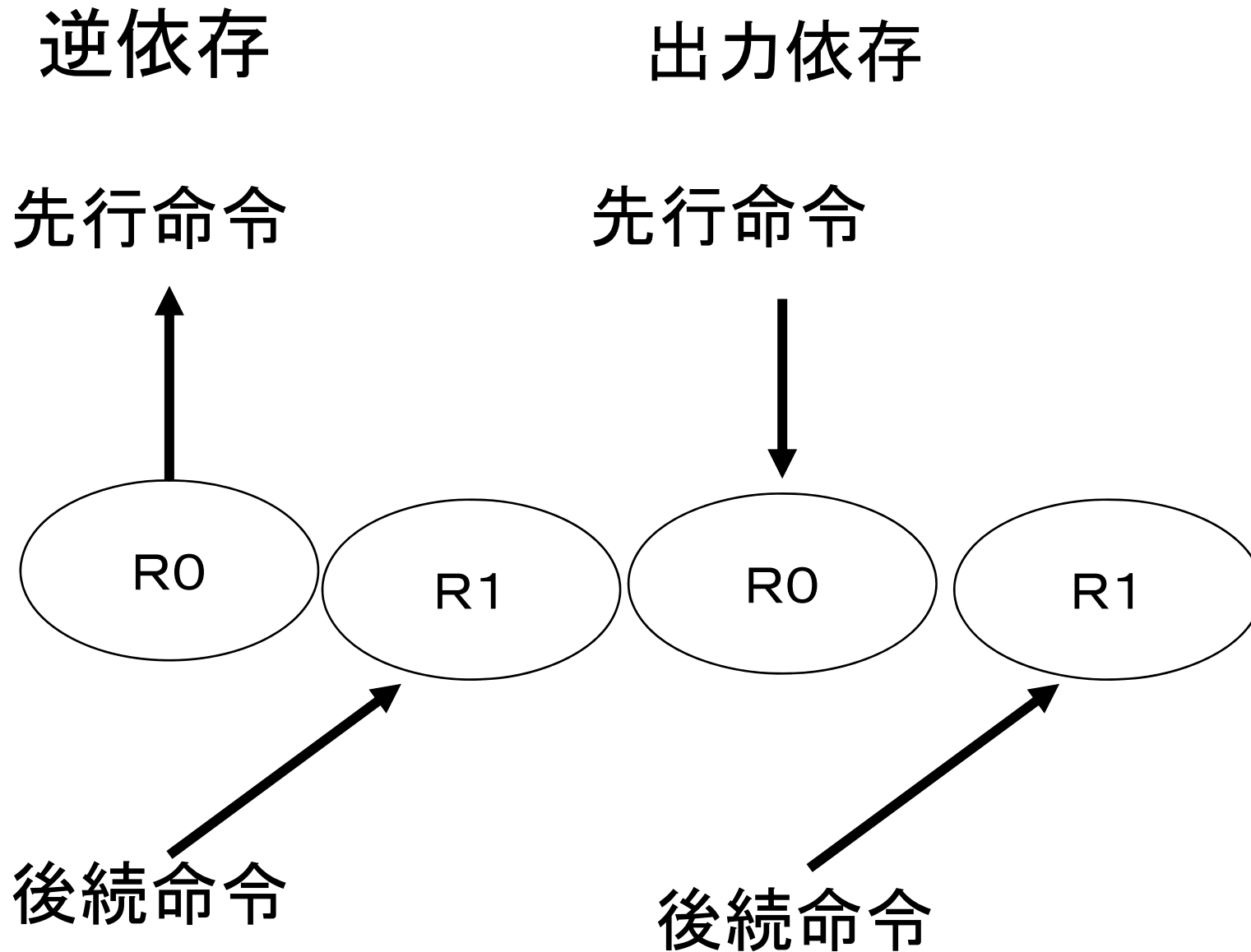


出力依存



データ依存

リネーミング



リネーミング

ADDF R1 R2 R3

SUBF R4 R1 R3

ADDF R1 R5 R6

MULF R7 R1 R2

(1)

ADDF R1 R2 R3

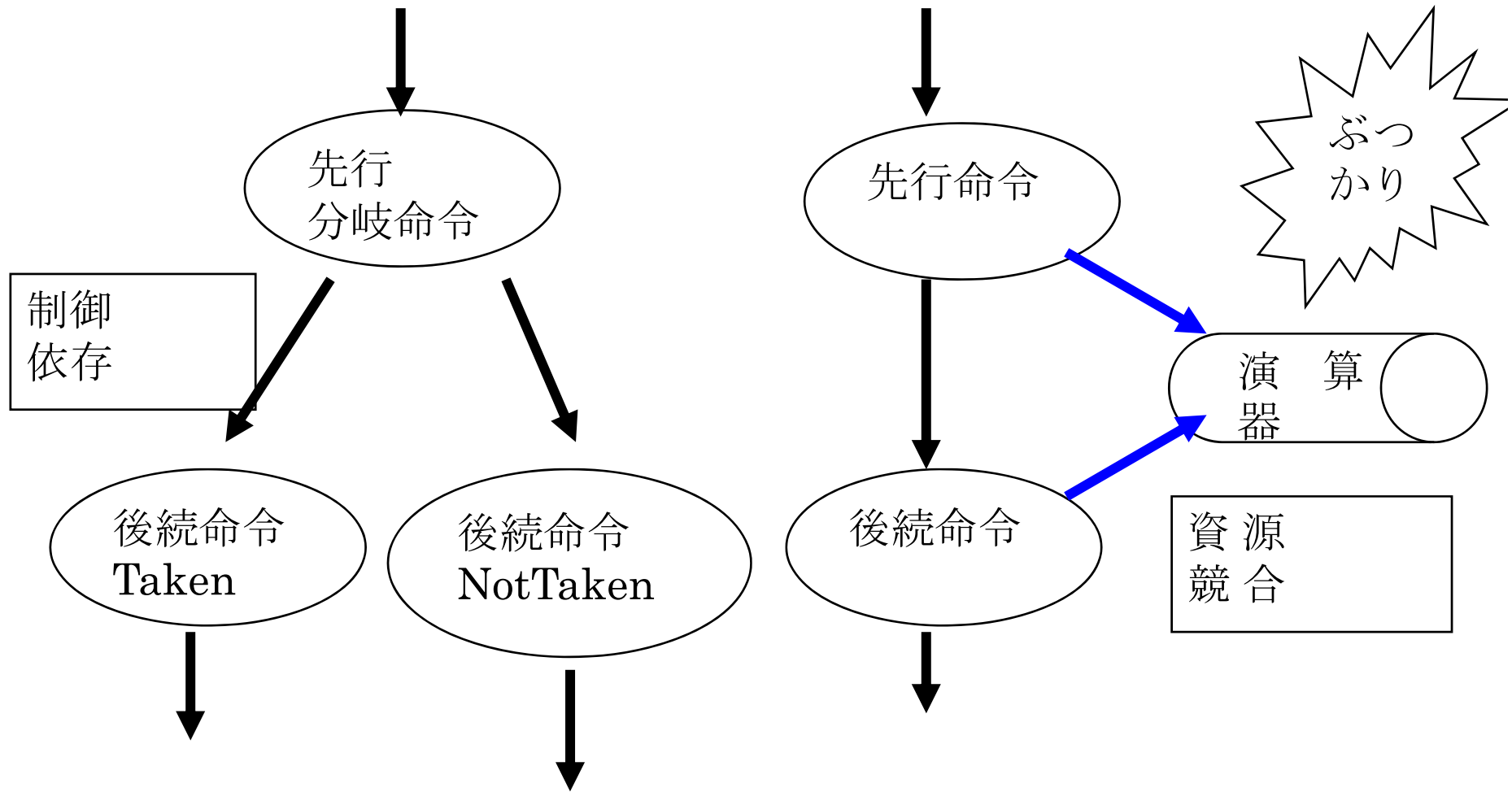
SUBF R4 R1 R3

ADDF R8 R5 R6

MULF R7 R8 R2

(2)

R1をR8にリネーミング



制御依存と資源競合

6.1.2 命令レベルでの並列処理の分類

命令パイプラインのモデル

- ① I F (命令フェッチ)
- ② D (デコード) : 命令デコード、レジスタ読出し
- ③ E X (実行) :
 - 整数演算は 1 サイクル
 - 浮動小数点加算、乗算は、3 ステージのパイプ
 - 演算器バイパス機構が存在
 - ロードストア命令 2 サイクル
 - 分岐命令の分岐先アドレス : $M(R+D)$
 - E X ステージで確定し、直ちに I F ステージへ
- ④ S (結果の格納)

命令レベルでの並列処理の分類項目

(1) 静的／動的命令スケジュール

- ①ハードウェアに命令の依存関係を検出する機構がないとき：静的

データ依存

ADDF R1 R2 R3

SUBF R4 R1 R3

ADDF IF D EX EX EX S

NOP IF D EX S

NOP IF D EX S

SUBF IF D EX EX EX S

コンパイラが命令を探す

挿入された命令 (3)

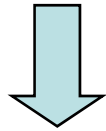
挿入された命令

遅延分岐

遅延スロットは2である。

ADDF R1 R2 R3

BRCZ M(R10)



ADDF IF D EX EX EX S

NOP IF D EX S

挿入命令

NOP IF D EX S

BRCZ IF D EX S

分岐命令 (4)

NOP IF D EX S

遅延スロット

NOP IF D EX S

遅延スロット

IF D EX EX EX S

分岐先命令

静的スケジューリングの特徴

- ・ ハードウェア：単純、高速化。
- ・ NOPが多用、プログラムサイズ増大、キャッシュメモリとの親和性悪化
- ・ 遅延スロット数の後継コンピュータでの変更：互換性喪失

②ハードウェアに命令の依存解析をする能力があるとき：動的
データ依存

ADDF IF D EX EX EX S

SUBF IF D - - EX EX EX S (5)

ハードウェアによる
データ待ち合わせ

分岐

- ①分岐条件が確定するまで分岐先命令のフェッチを開始しない方式では、

ADDF IF D EX EX EX S

BRCZ IF D EX - S

分岐命令 (6)

IF D EX S 分岐先命令

- ②分岐予測を採用しているときには、

ADDF IF D EX EX EX S

BRCZ IF D EX - S

分岐命令

IF D EX S分岐先命令 (予測側) (7)

IF D EX S (予測失敗のとき)

(2) 命令の発行順序と終了順序

命令の発行：命令パイプラインのDステージからEXステージに命令実行ステージを移すこと

① 順発行

先行命令と依存関係にある後続命令がくると、その命令およびそれ以降の命令の発行を停止する。後続命令は先行命令がEXステージに入ってから始めて発行できる。

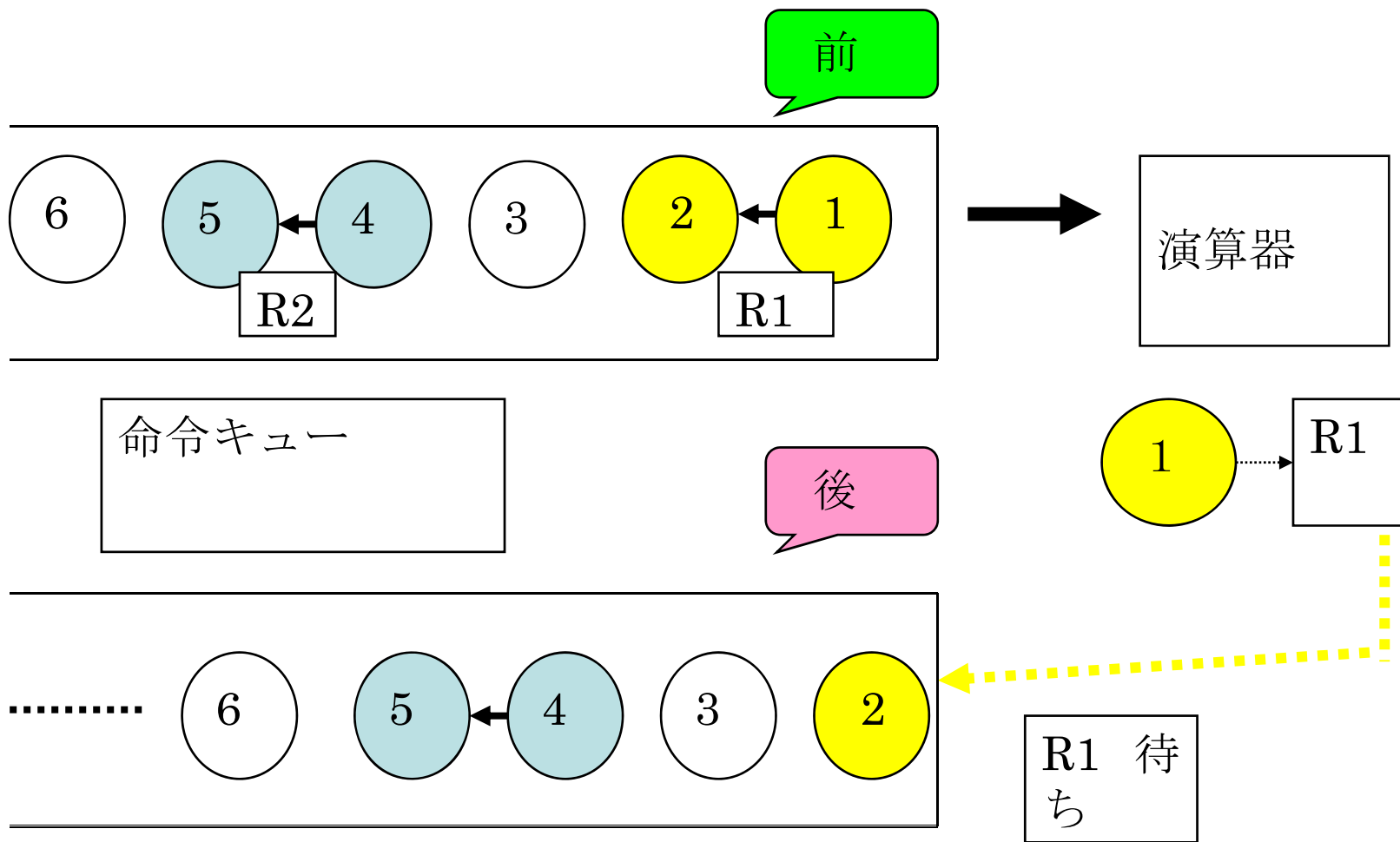
② 乱発行

依存関係にある命令はリザーベーションステーションに格納され、その命令の後続命令が発行される。したがって、プログラムに並んだ順番とは異なった順番で命令が発行される。

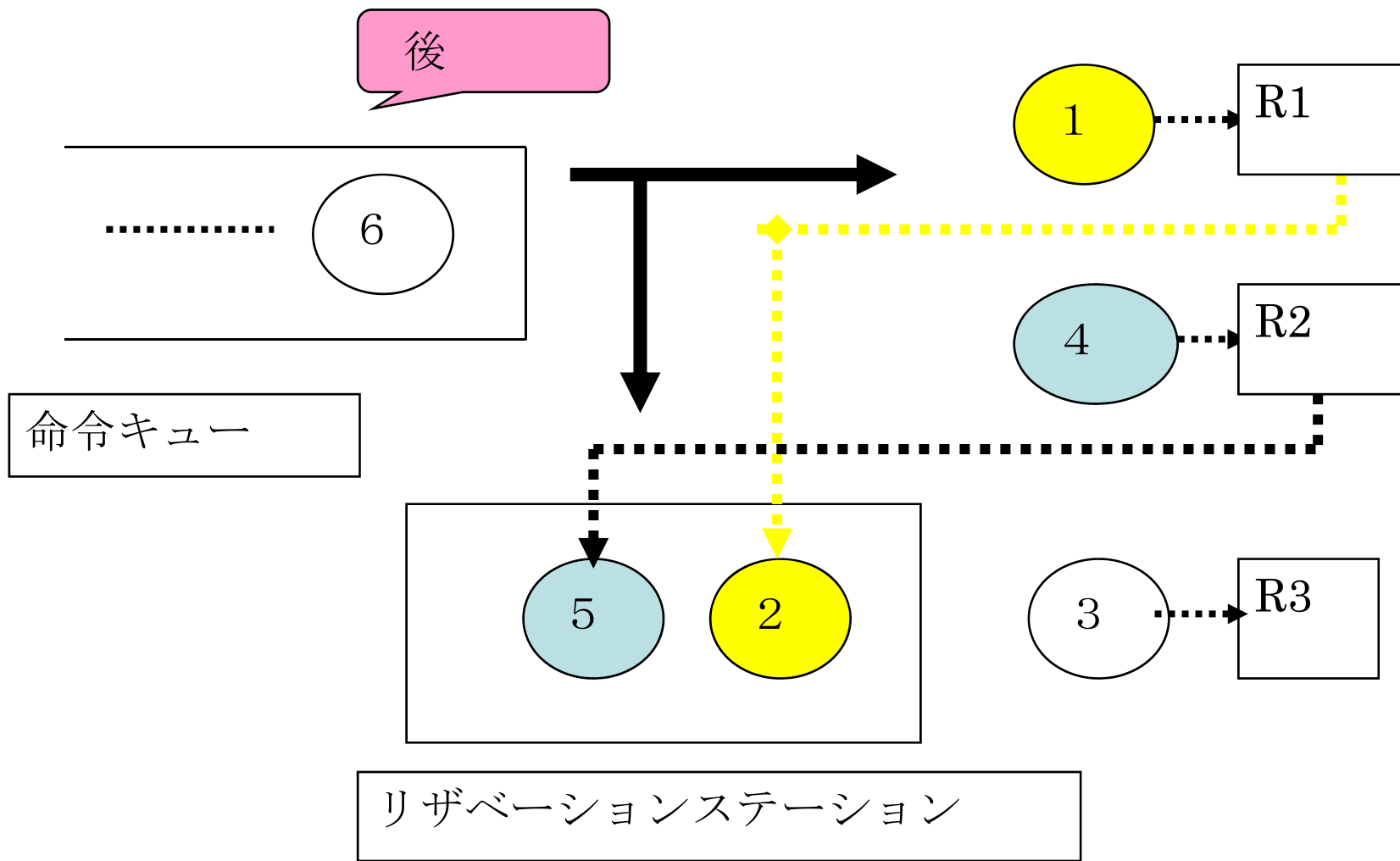
命令の終了とは、Sステージの実行終了を指す。

- ① 順終了 ② 乱終了

順発行



乱発行



連想メモリ

①順発行順終了

ADDF R1 R2 R3	IF D	EX	EX	EX	S																
SUBF R4 R1 R3	IF D	-	-	EX	EX	EX	S														
MULF R1 R5 R6	IF D	-	-	EX	EX	EX	S	(8)													
ADDF R7 R1 R2	IF D	-	-	-	-	EX	EX	EX	S												
ADDI R10 R11 R13	IF D	-	-	-	-	EX	-	-	S												
SUBI R14 R10 R11	IF D	-	-	-	-	EX	-	-	S												

②順発行乱終了

ADDF R1 R2 R3	IF D	EX	EX	EX	S				
SUBF R4 R1 R3	IF D	-	-	EX	EX	EX	S		
MULF R1 R5 R6	IF D	-	-	EX	EX	EX	S	(9)	
ADDF R7 R1 R2	IF D	-	-	-	-	EX	EX	EX	S
ADDI R10 R11 R13	IF D	-	-	-	-	EX	S		
SUBI R14 R10 R11	IF D	-	-	-	-	EX	S		

③乱発行順終了

ADDF R1 R2 R3	IF D	EX	EX	EX	S				
SUBF R4 R1 R3	IF D	-	-	EX	EX	EX	S		
MULF R8 R5 R6	IF D	EX	EX	EX	-	-	S	(10)	
ADDF R7 R8 R2	IF D	-	-	EX	EX	EX	S		
ADDI R10 R11 R13	IF D	EX	-	-	-	-	S		
SUBI R14 R10 R11	IF D	EX	-	-	-	-	S		

④乱発行乱終了

リネーミングあり

ADDF R1 R2 R3	IF D	EX EX EX S	
SUBF R4 R1 R3	IF D	- - EX EX EX S	
MULF R8 R5 R6	IF D	EX EX EX S	(11)
ADDF R7 R8 R2	IF D	- - EX EX EX S	
ADDI R10 R11 R13	IF D	EX S	
SUBI R14 R10 R11	IF D	EX S	

ADDF R1 R2 R3	IF D	EX EX EX S	
SUBF R4 R1 R3	IF D	- - EX EX EX S	
MULF R8 R5 R6	IF D	EX EX EX - S	
ADDF R7 R8 R2	IF D	- - EX EX EX S	
ADDI R10 R11 R13	IF D	EX - - - S	
SUBI R14 R10 R11	IF D	EX - - S	

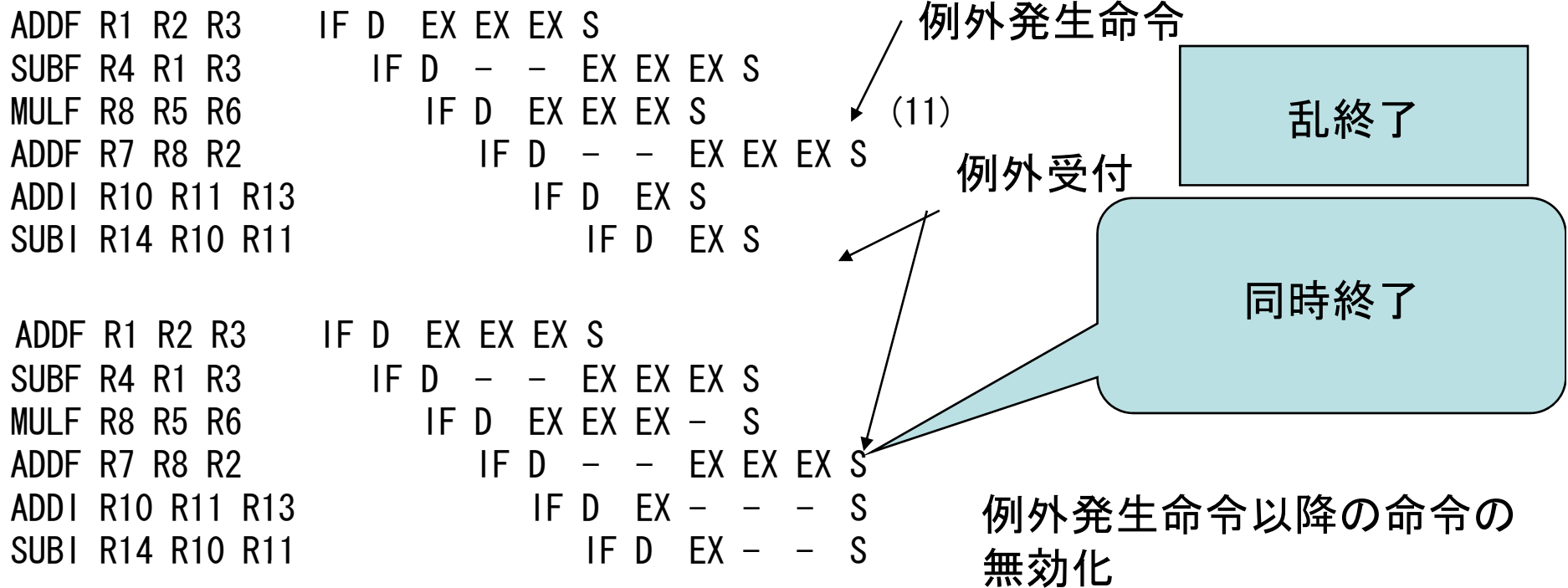
同時終了でも乱終了
と同一性能

正確な割り込み

旧コンピュータ：PCの指した命令のSステージ終了時に割り込み
(例外) 発生。状態保存し、再実行可能

不正確な割り込み

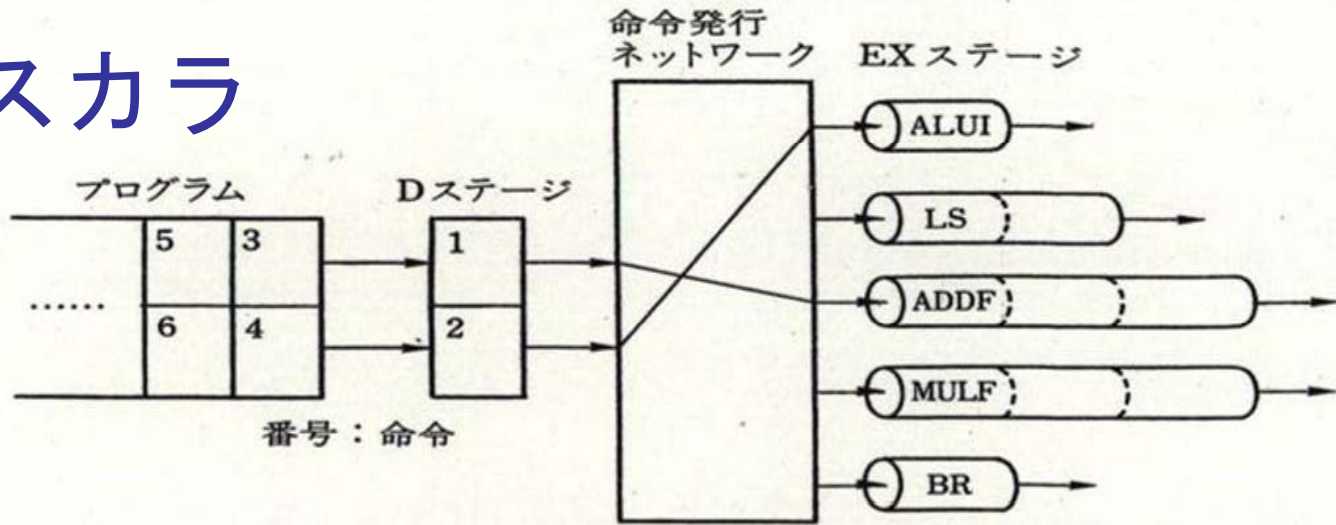
乱終了：例外を起こした命令よりずっと先の命令をPCは指している。命令実行終了のもの未終了のもの混在。状態保存困難、再実行困難。例外発生命令のPC ≠ 例外を受け付けた命令のPC



(3) 命令フェッチ方法

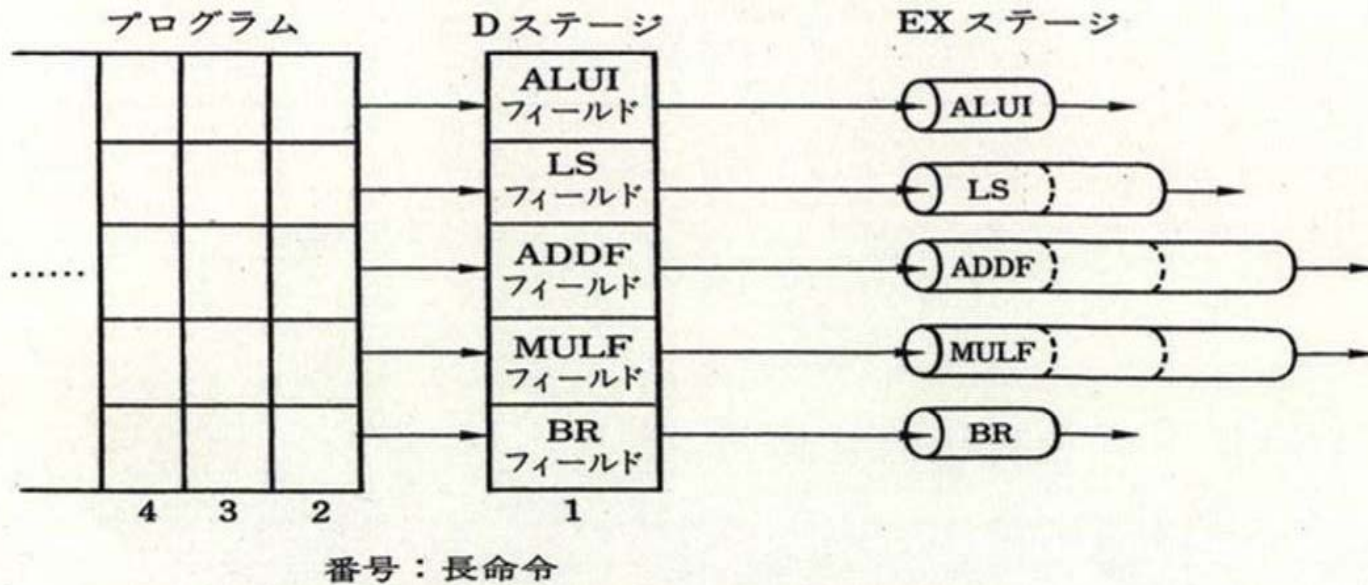
- ① スカラ方式
- ② スーパスカラ方式
- ③ VLIW方式

スーパスカラ



(a) スーパスカラ方式での命令と演算装置の対応

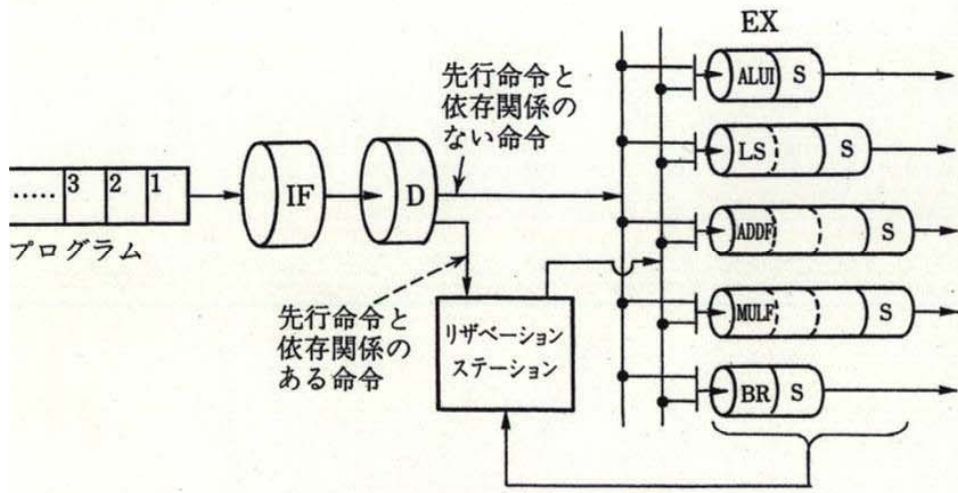
VLIW



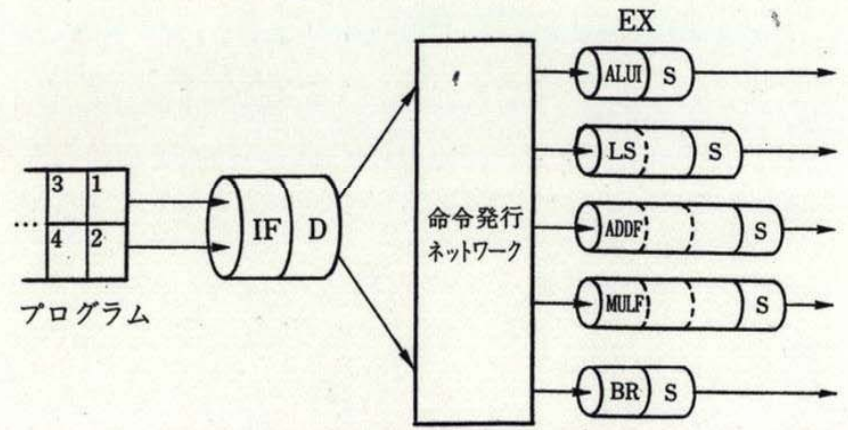
(b) VLIW 方式での命令と演算装置の対応

ALUI：整数，LS：ロードストア，ADDF：浮動小数点加算，
MULF：浮動小数点乗算，BR：分岐

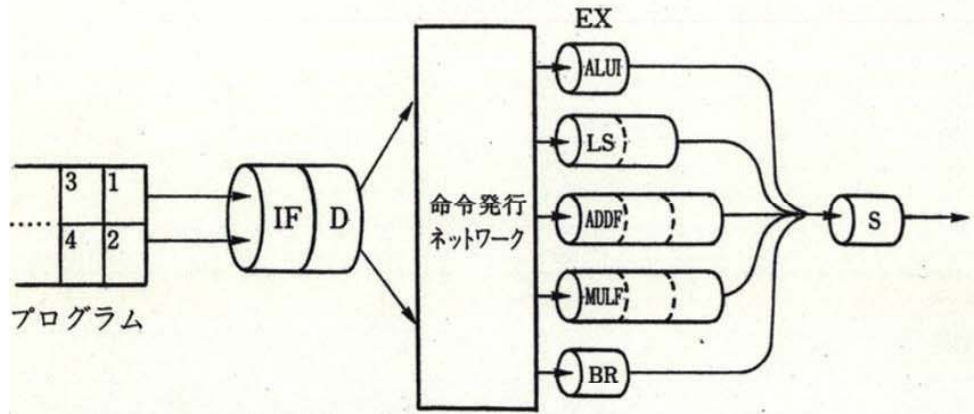
図 6.2 スーパスカラ/VLIW 方式での命令と演算装置の対応



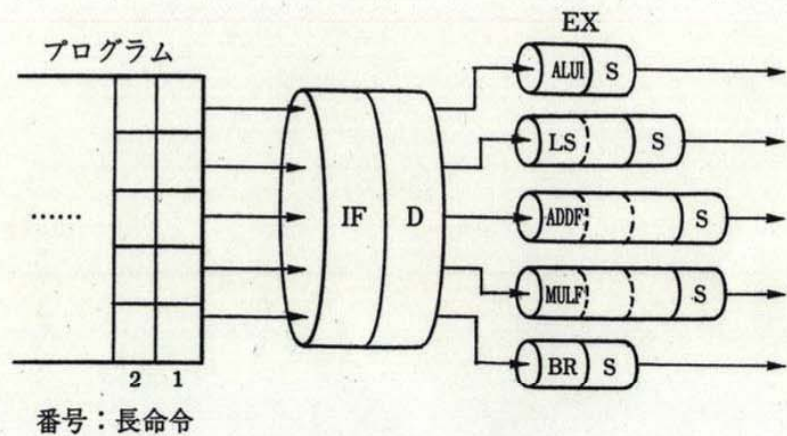
(a) 動的乱発行乱終了スカラ方式



(c) 動的順発行乱終了スーパースカラ方式



(b) 動的順発行順終了スーパースカラ方式



(d) VLIW 方式

図 6.3 種々のパイプライン方式

6. 2動的乱終了スカラ方式

6. 2. 1動的順発行乱終了スカラ方式

①レジスタへの同時書込み

マルチポートのレジスタ必要

②EX、Sステージでの制御複雑

③分岐予測を採用する際

予測された側の命令系列を命令パイプラインに投入
分岐命令のSステージが終了後、予測された側の命令系列を終了させる

6.2.2 動的命令乱発行乱終了スカラ方式

(1) IBM360/91 : 1967年稼働

(2) IBM360/91の動作

リザーベーションステーション

レジスタリネーミング

共有バス

①逆依存

②フロー依存

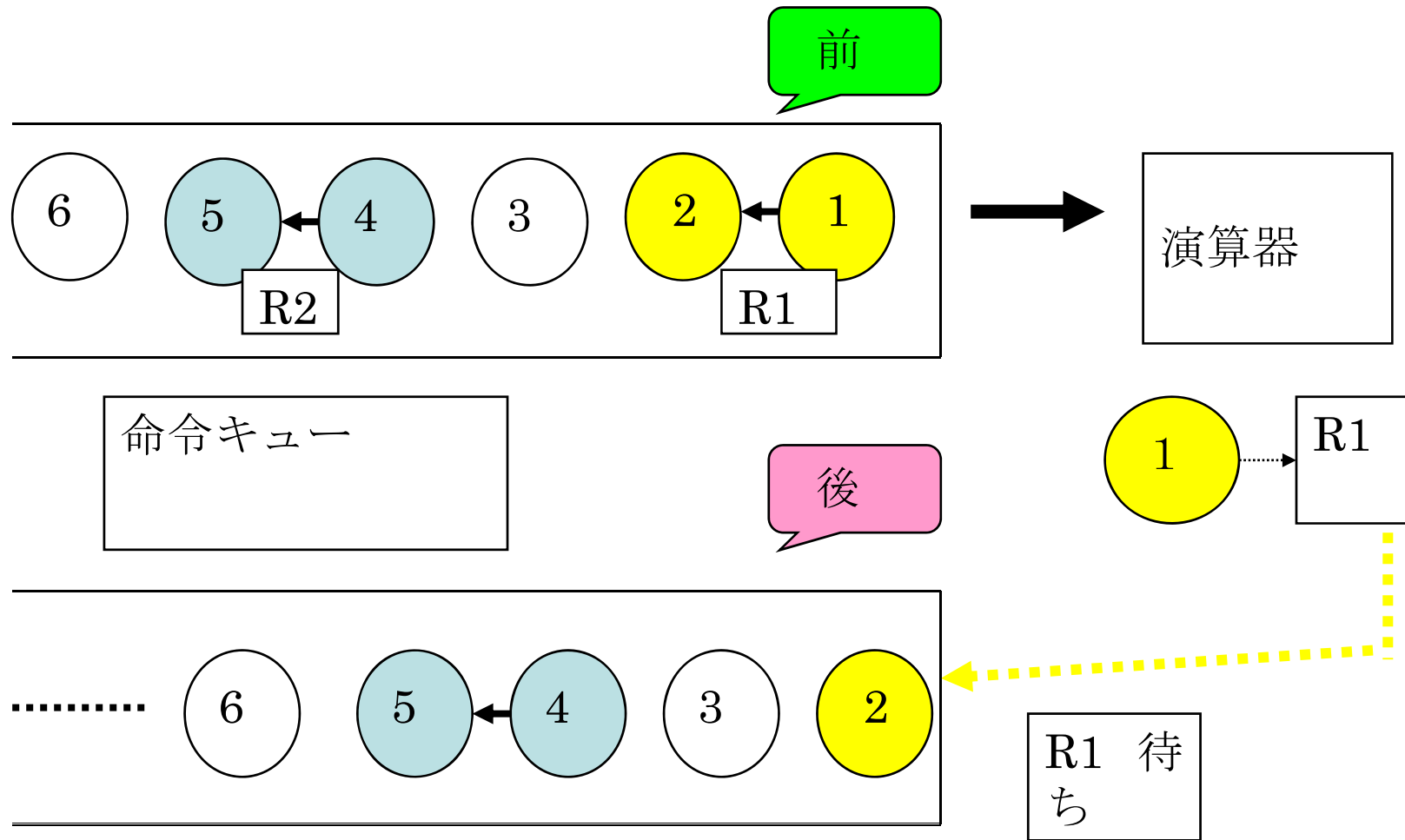
③出力依存

(3) CDC6600のスコアボード方式

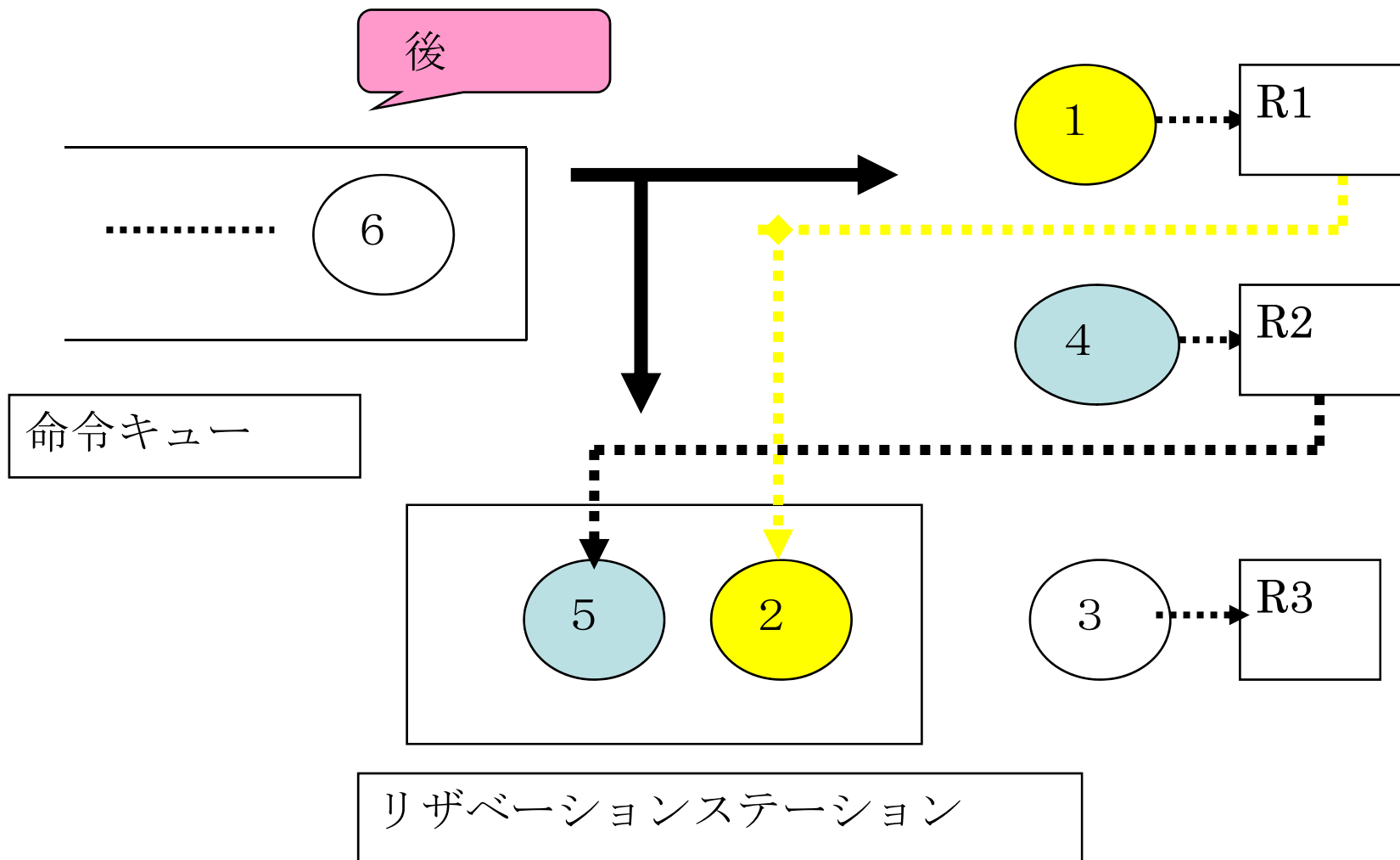
出力依存のときインターロック

6.2.3 不正確な割込み

順発行

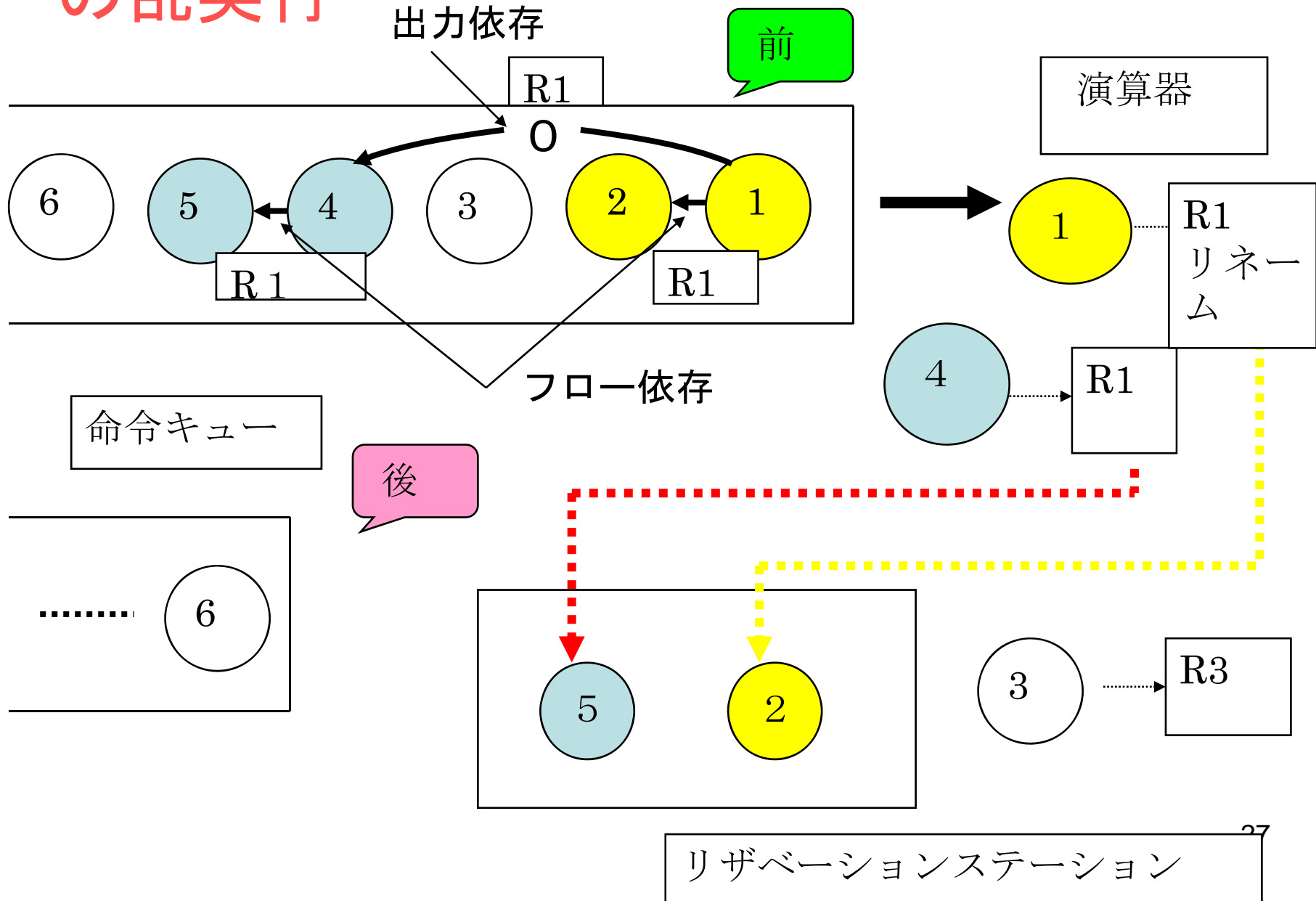


乱発行



連想メモリ

Tomasulo方式 の乱実行



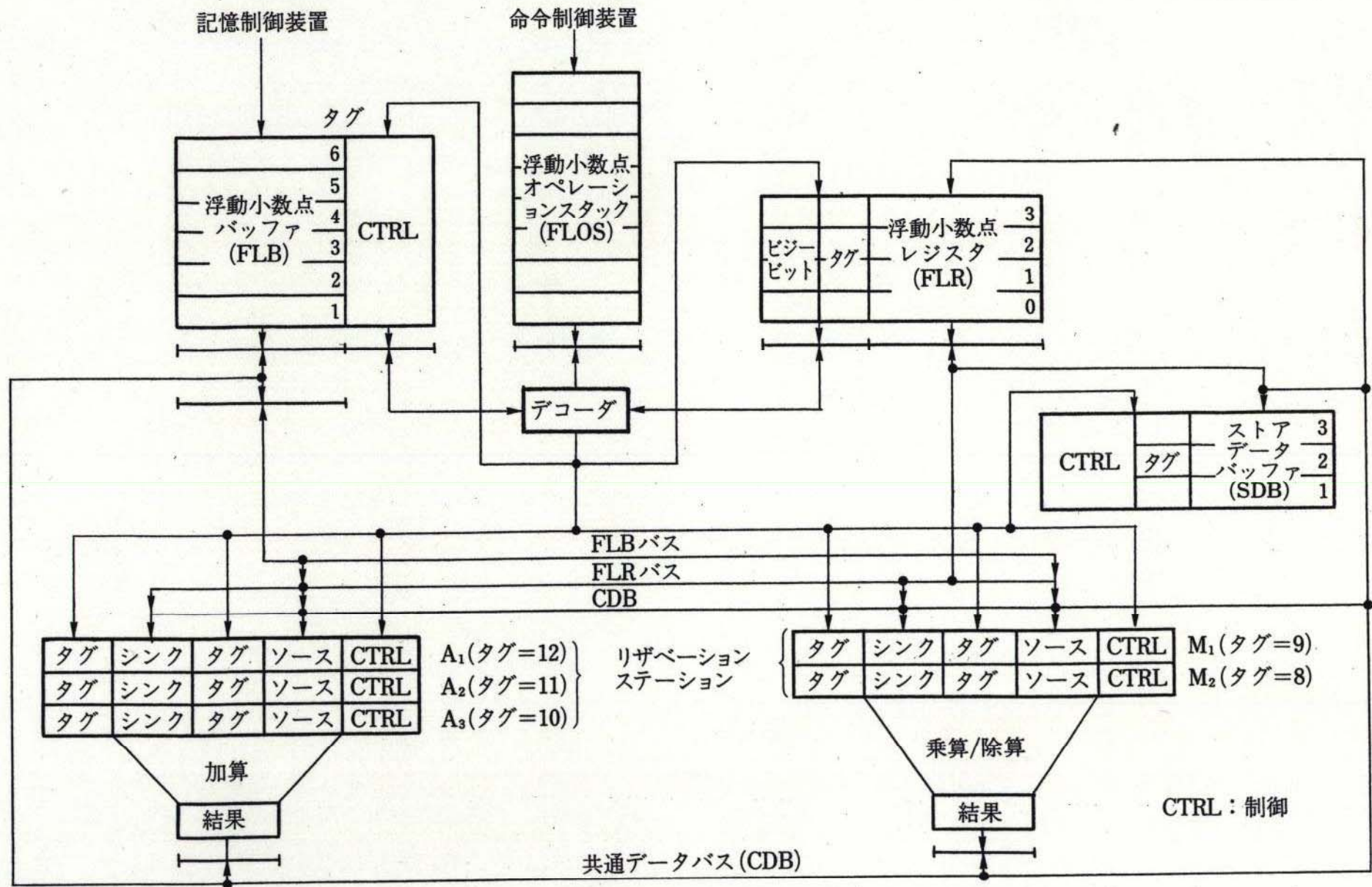


図 6.4 IBM 360/91 の演算装置の構成 [R. M. Tomasulo: An Efficient Algorithm for Exploiting Multiple Arithmetic Units, *IBM Journal*, pp. 25-33 (1967)]

ビジー ビット	タグ	FLR
		3
		2
0		1
1	12	0

タグシンク	タグソース
A1 12	0 R0 0 R1
A2 11	
A3 10	

タグ

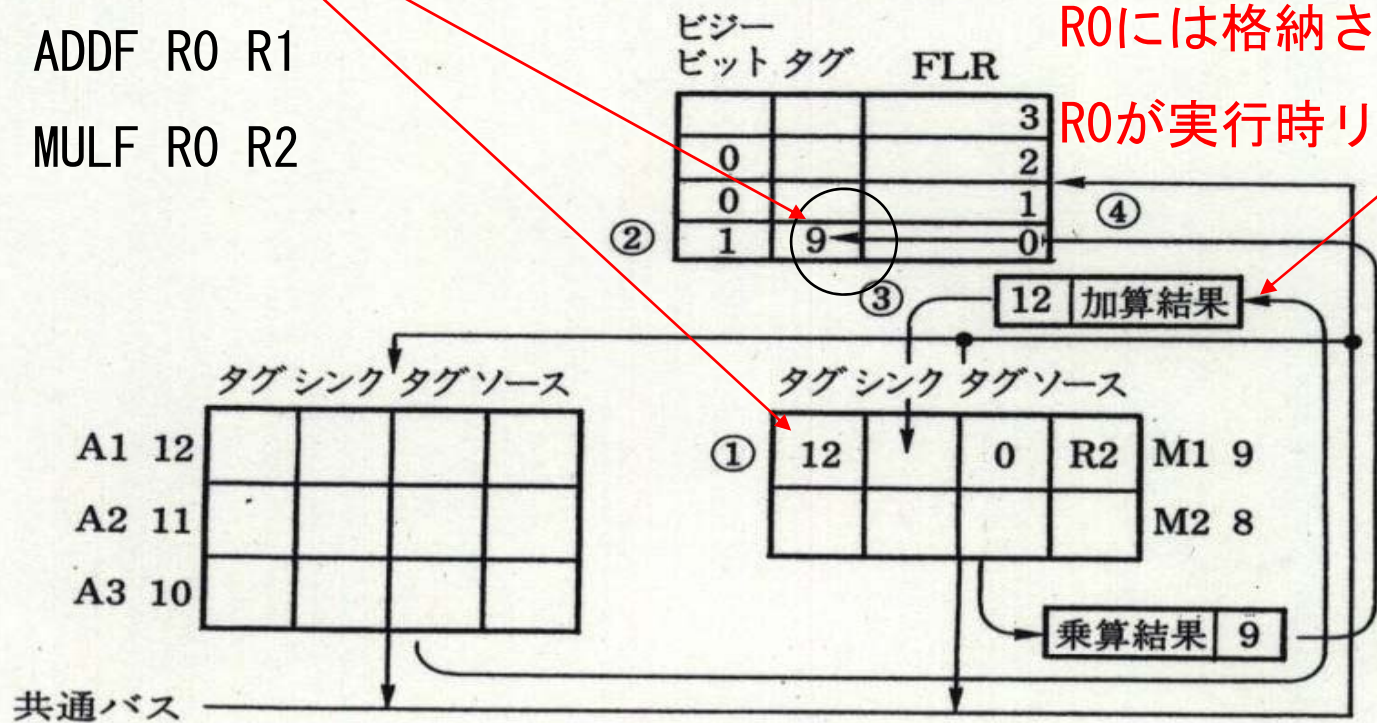
				M1 9
				M2 8

タグ

(1) ADDF R0 R1 起動時

ADDF R0 R1
MULF R0 R2

ADDFの結果は
R0には格納されない
R0が実行時リネーム

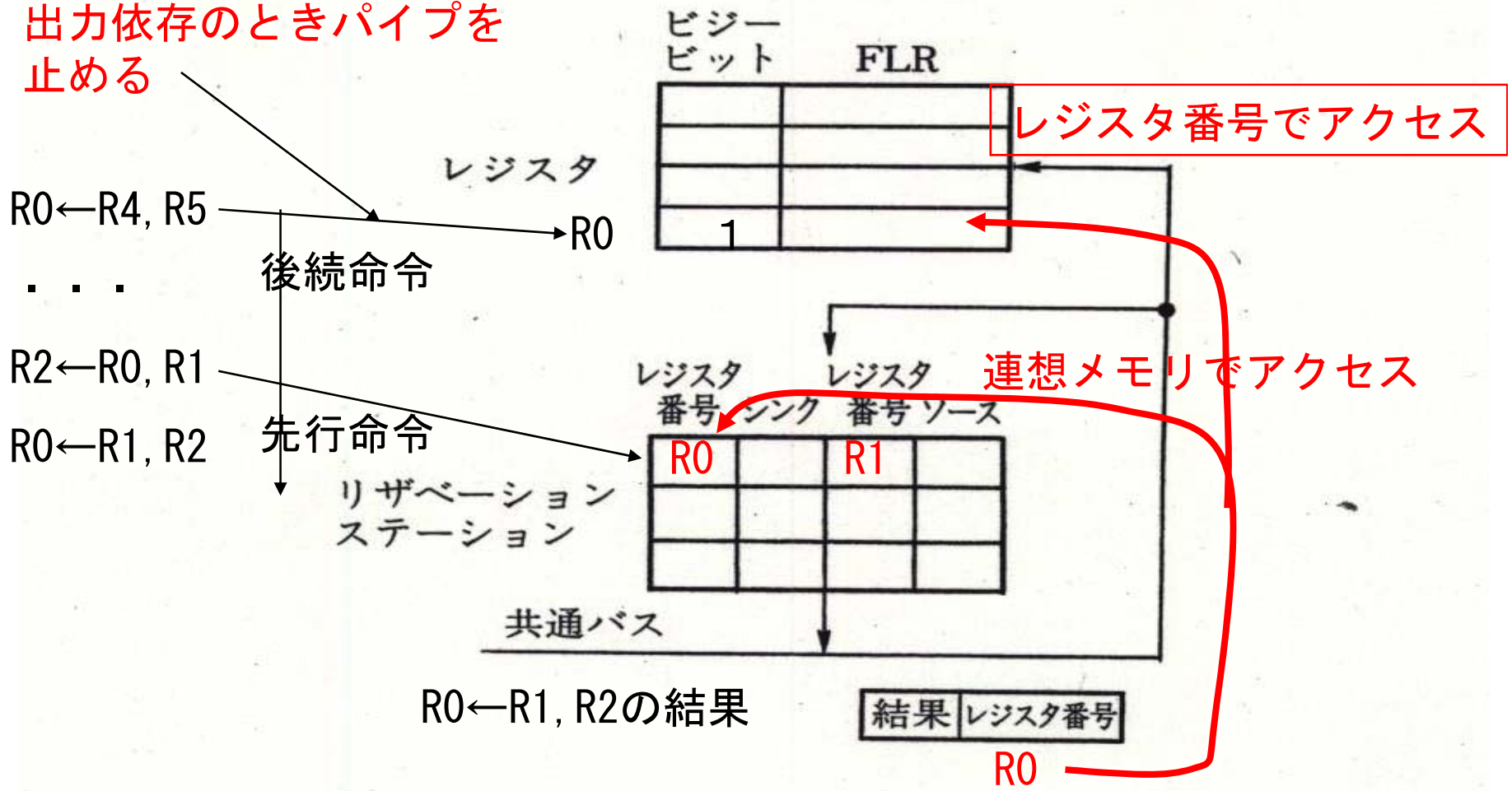


(2) MULF R0 R2 起動後

(a) IBM360/91 での動作例

CDC6600 : 1964年稼働
 S. Cray
 RISCの原型
 3-オペランド

出力依存のときパイプを止める



(b) CDC6600での動作例

6.3 投機的命令実行方式

6.3.1 基本方式

分岐命令を越えて投機的実行を行う利点

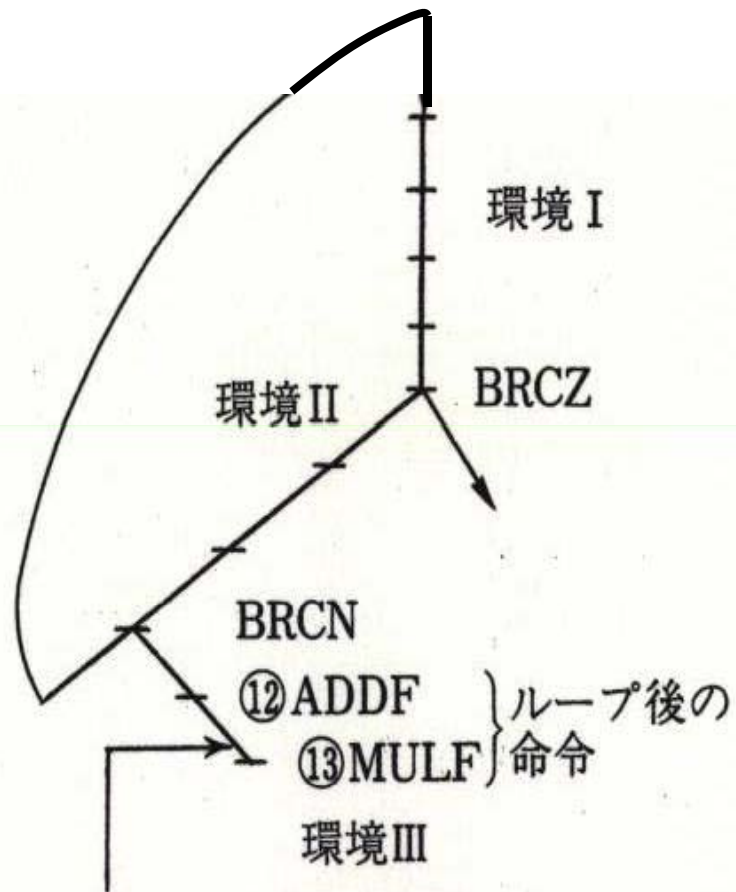
```
DO 10 I=1, N
  Z(I)=X(I)/Y(I)
  IF (Z(I).EQ.0) GOTO 20   (12)
10 CONTINUE
  A=A+B
  D=A*D
20 ---
```

```

L F①LOAD R1 M(R10+D1)
→F②LOAD R2 M(R10+D2)
U③DIVF R1 R2
U④STORE M(R10+D3) R1
..U⑤BRCZ M(J)
F⑥ADDI R10 #I1
F⑦ADDI R11 #I1
F⑧BRCN M(L)
F⑫ADDF R3 R4
→U⑬MULF R5 R3
⋮
J

```

命令の実行終了/未終了状態



この時点でBRCZ⑤の
分岐予測ミス発生

(b) 分岐予測の誤り

図 6.6 レジスタの状態

```

L   LOAD   R1 M(R10+D1)
    LOAD   R2 M(R10+D2)
    DIVF   R1 R2
    STORE  M(R10+D3) R1
    BRCZ   M(PC+10)      Jに分岐
    ADDI   R10 #11       (13)
    ADDI   R11 #11
    BRCN   M(PC-16)      Lに分岐
    ADDF   R3 R4
    MULF   R5 R3

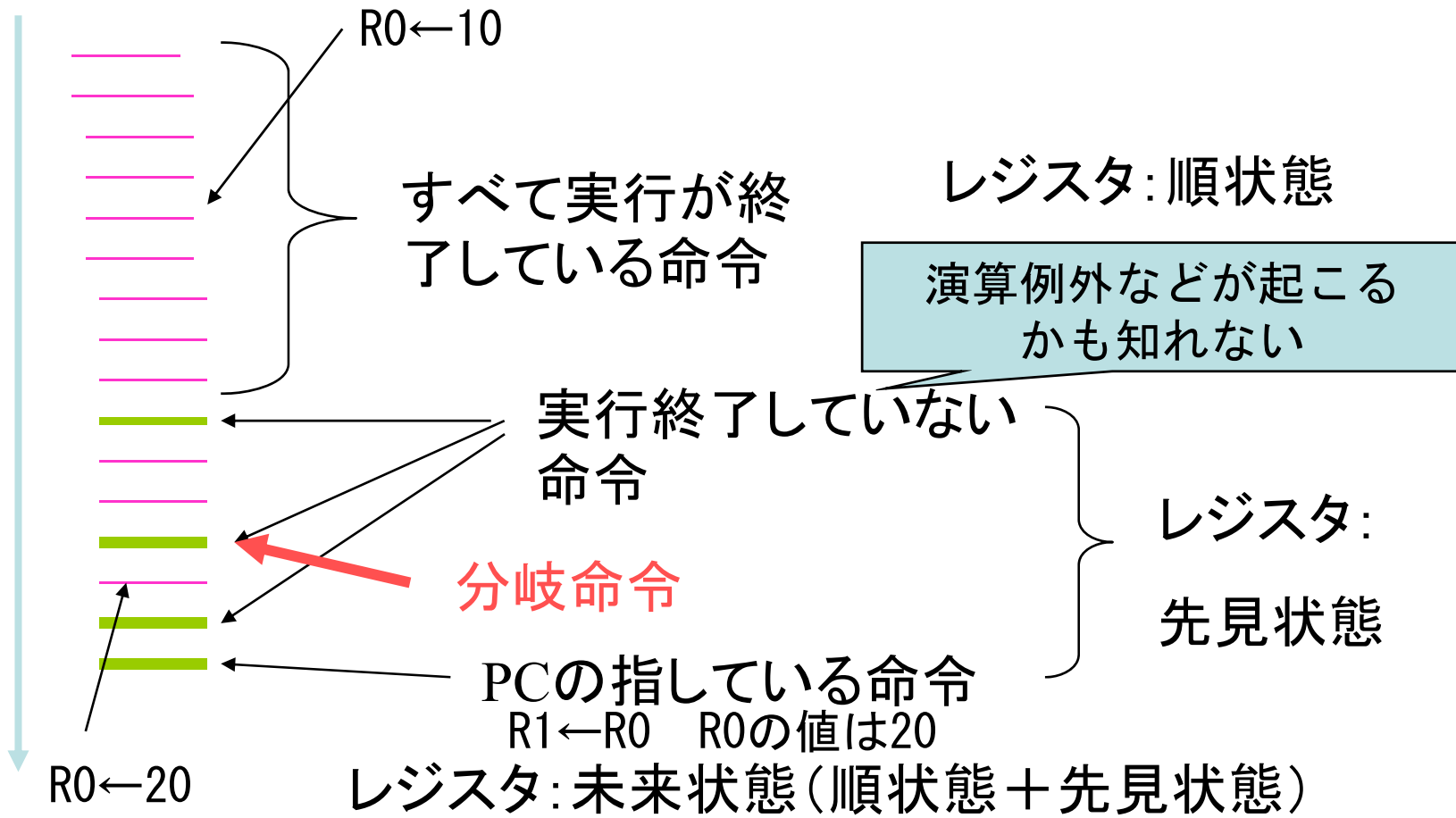
```

J ---

除算に 1 2 マシンサイクル必要

レジスタの状態

先行命令



- ① LOAD IF D EX EX S
- ② LOAD IF D EX EX S
- ③ DIVF IF D - EX EX EX EX EX EX EX EX EX EX EX S
- ④ STORE IF D - - - - - - - - - - EX EX S
- ⑤ BRCZ IF D EX - - - - - - - - - S
- ⑥ ADDI IF D EX S
- ⑦ ADDI IF D EX S
- ⑧ BRCN IF D EX S (14)

- ⑨ LOAD IF D EX EX S 2回目の反復
- ⑩ LOAD IF D EX EX S
- ⑪ DIV IF D - - - EX EX
EX ...

- ⑫ ADDF IF D EX EX EX S ループ終了
- ⑬ ADDF IF D - - EX EX EX S

6.3.2レジスタの復旧機構

①チェックポイントリペア (Check point repair)

②ヒストリバッファ (History buffer)

③リオーダバッファ (Reorder buffer)

投機実行

正確な割込みも保証

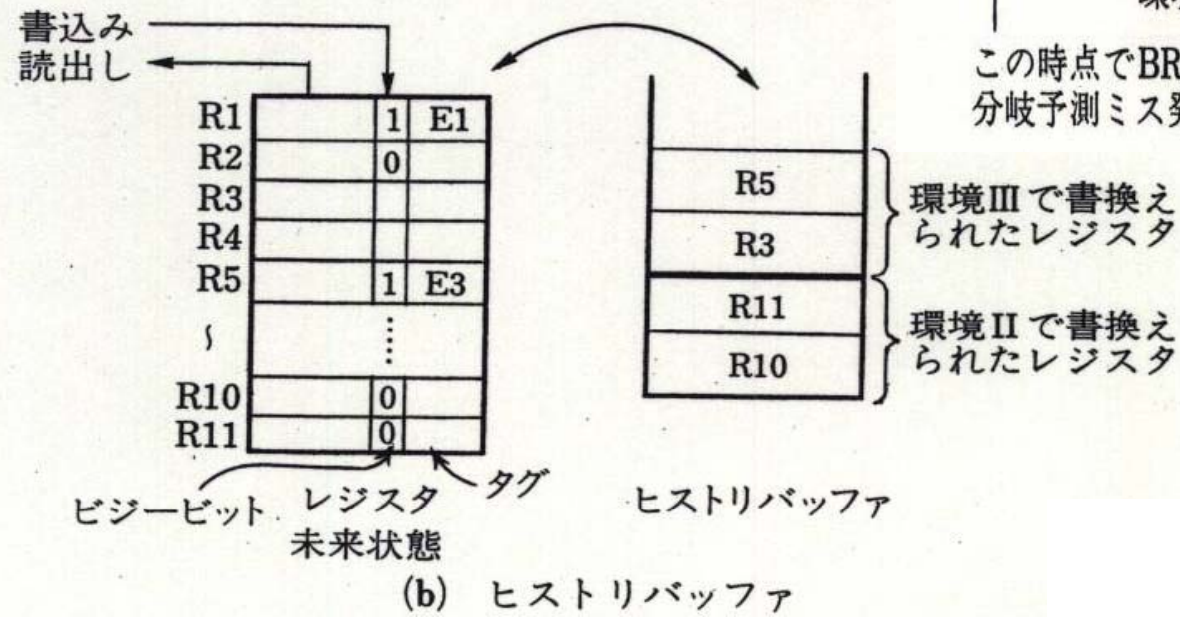
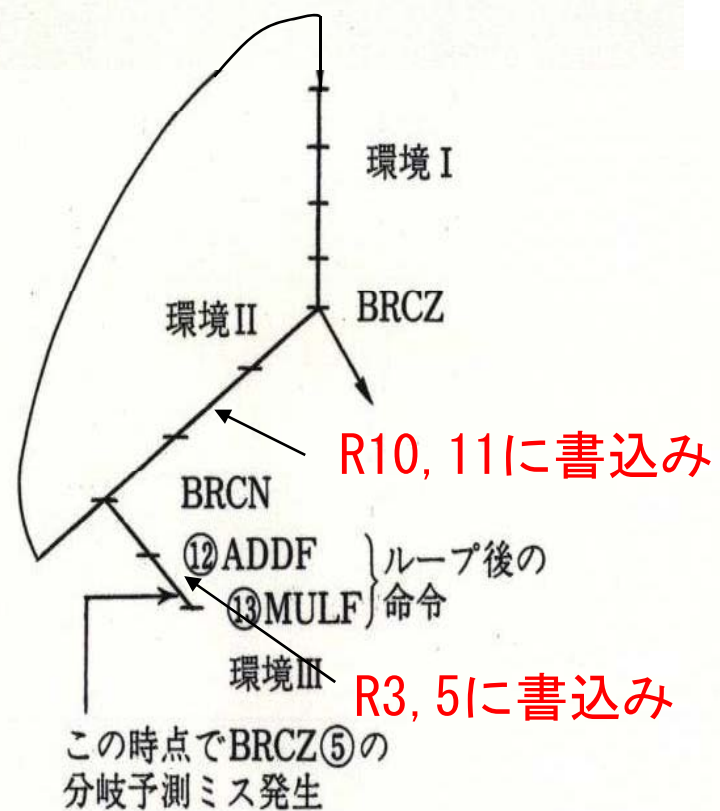
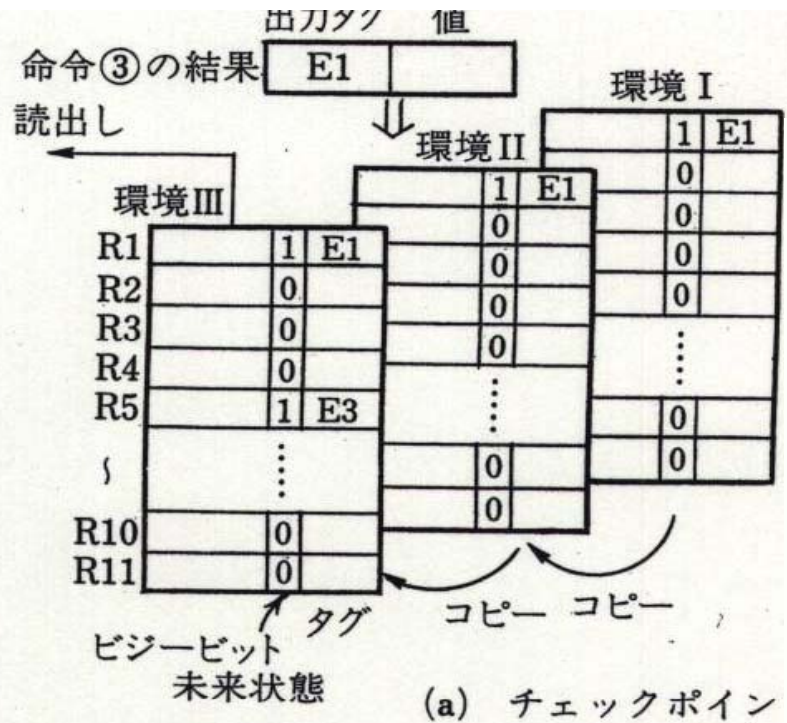
データ呼出し：連想記憶

④フューチャファイル (Future file)

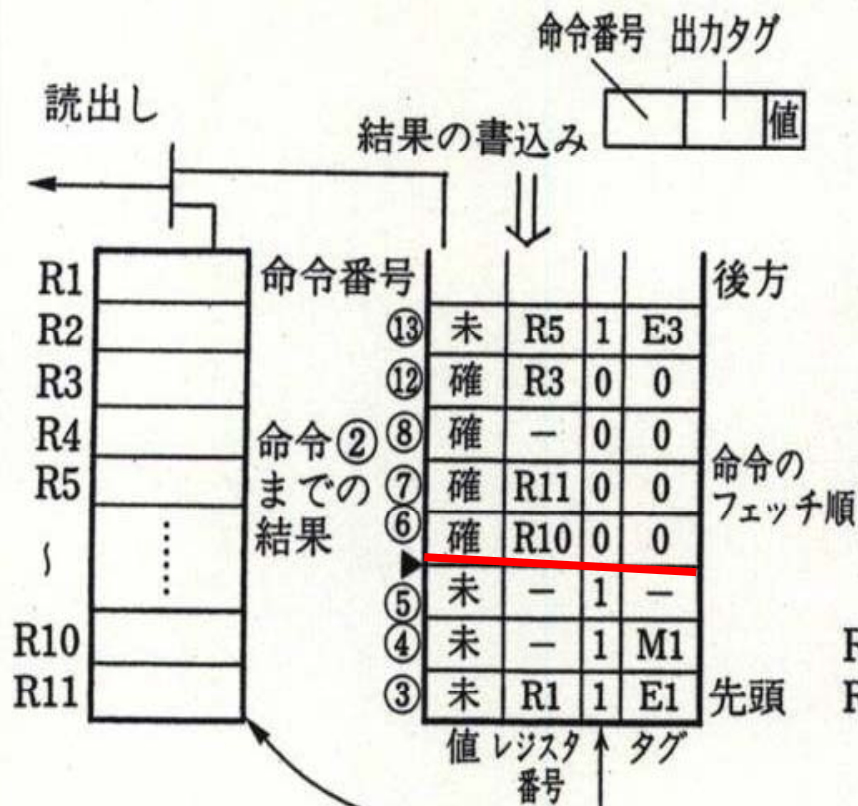
リオーダバッファ

フューチャファイル：未来状態を保持

⑤リネーミングレジスタ



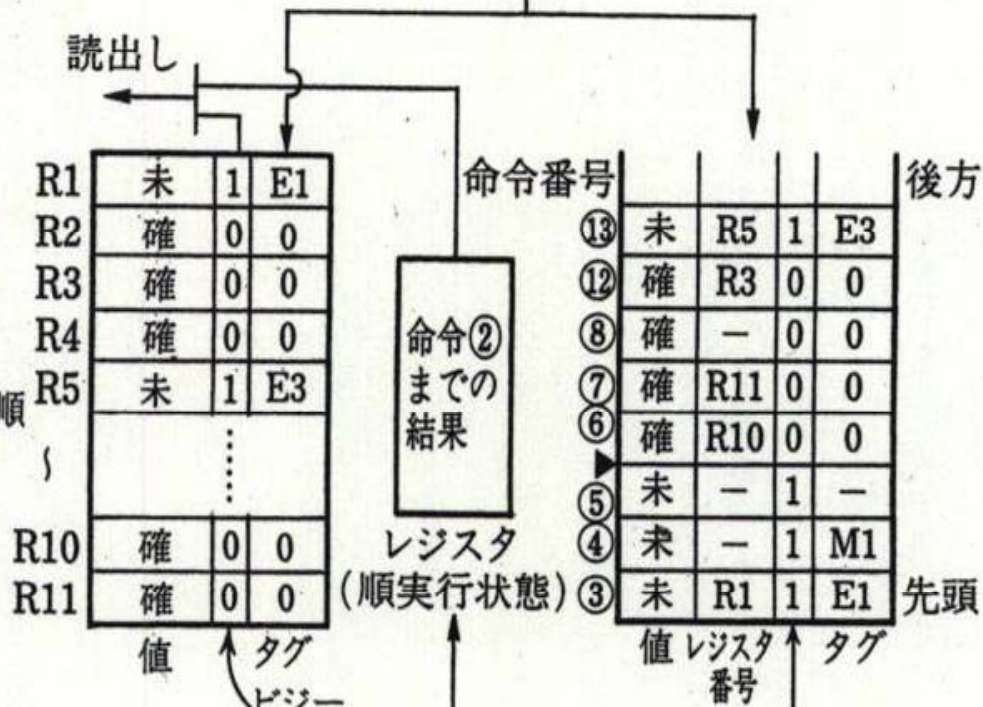
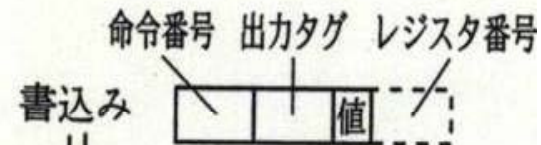
(b) ヒストリバッファ



レジスタ
(順実行状態)

リオーダバッファ
(先見状態)

(c) リオーダバッファ



フューチャ
ファイル
(未来状態)

リオーダバッファ
(先見状態)

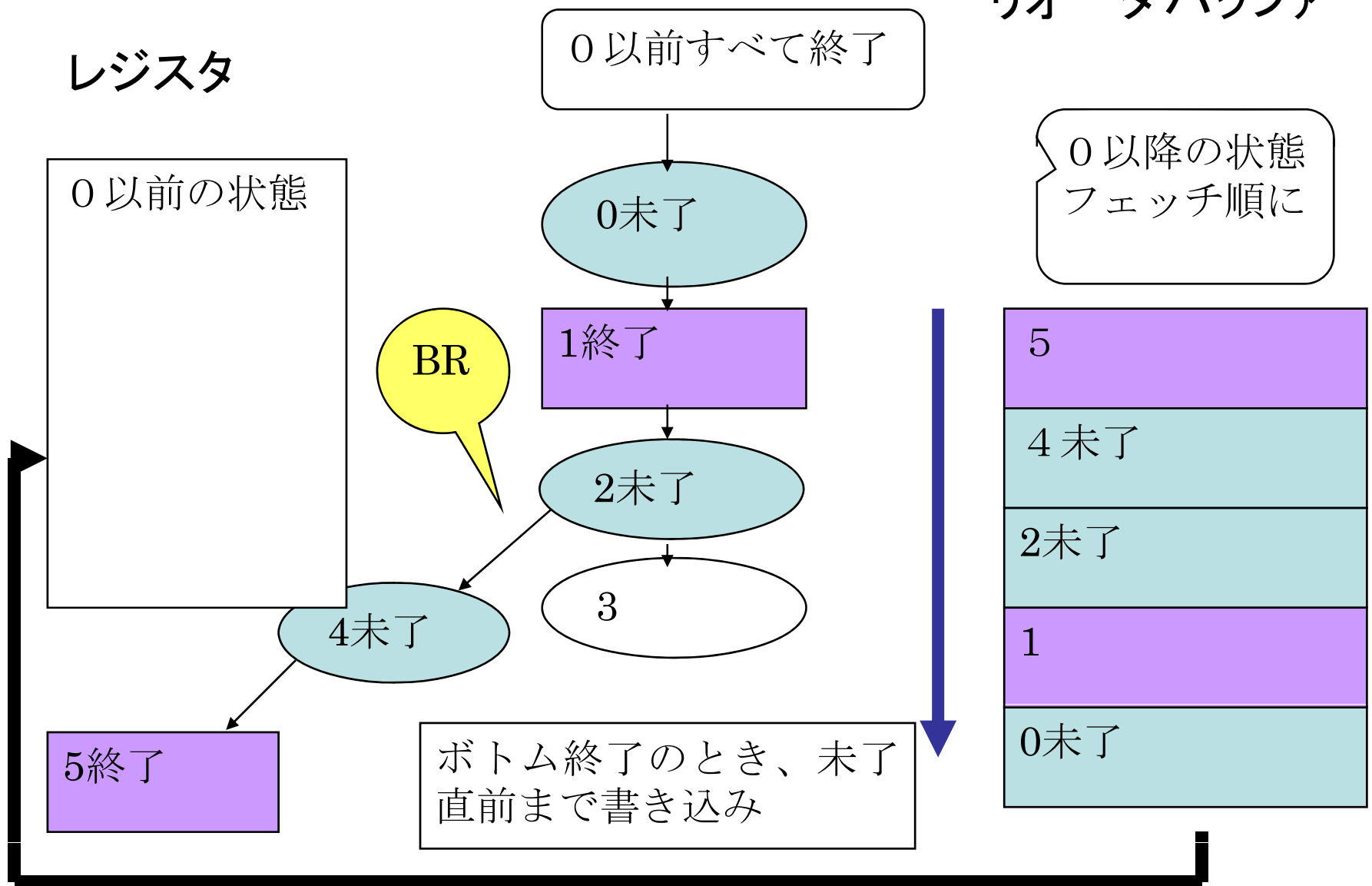
(d) フューチャファイル

投機実行のメカニズム

レジスタリネーミング法

- ①リオーダーバッファを利用する方法
- ②大きな物理レジスタを利用する方法
(狭義のレジスタリネーミング法)

①リオーダーバッファ法



レジスタ

0 以前すべて終了

リオーダーバッファ

0 以前の状態
+
0 と 1 の結果

BR

0 終了

1 終了

2 未了

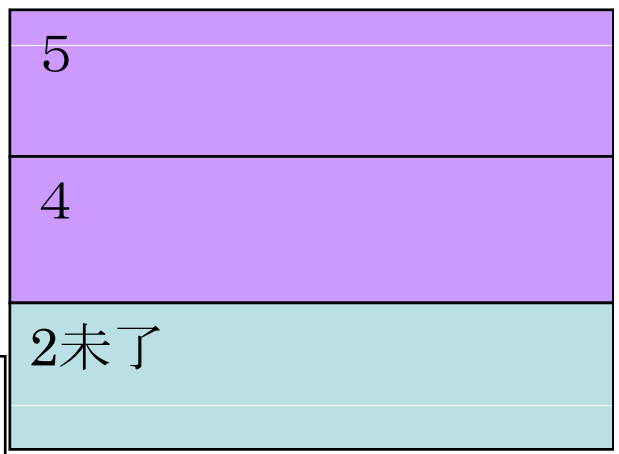
4 終了

3

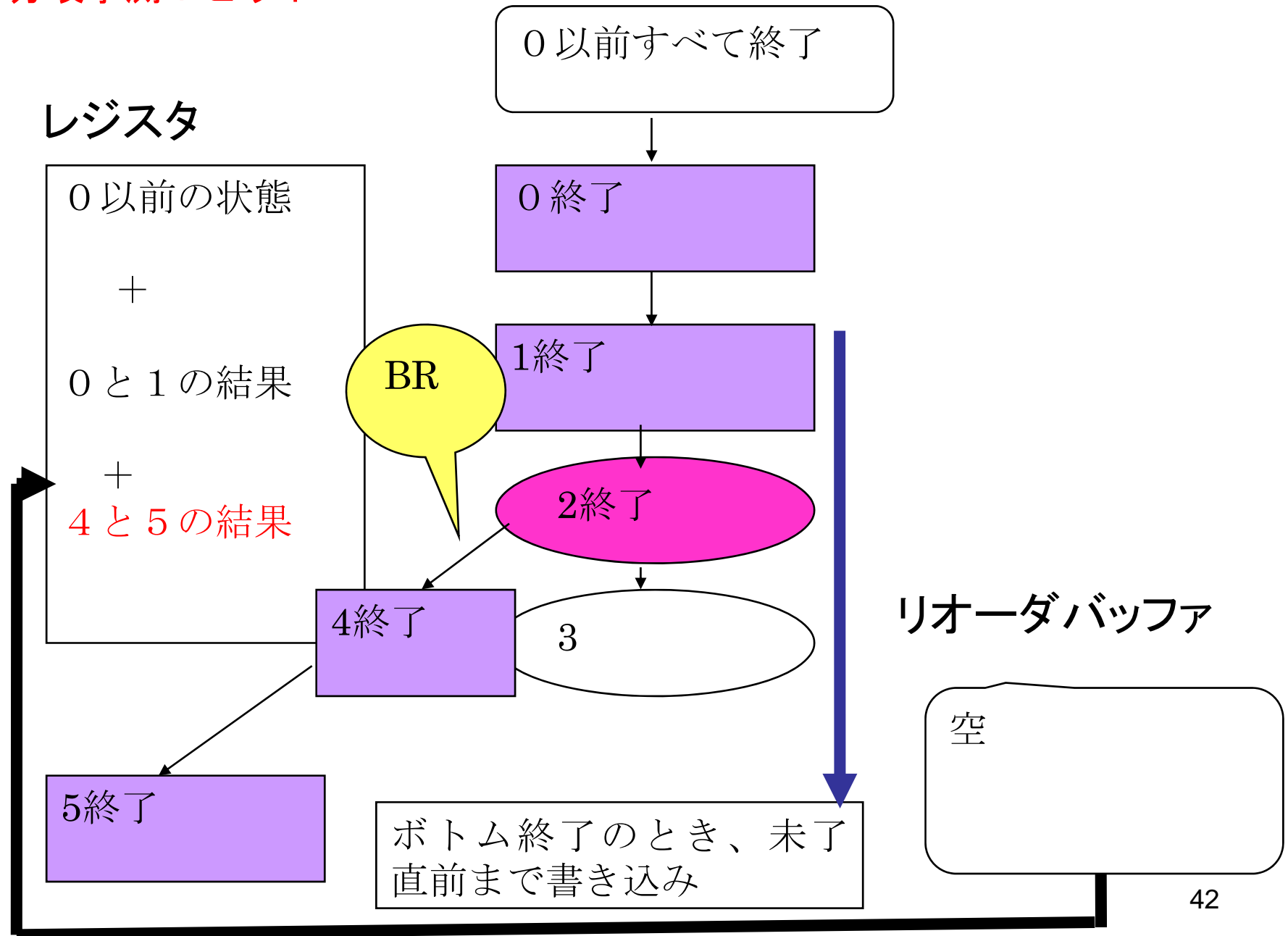
5 終了

ボトム終了のとき、未了
直前まで書き込み

2 以降の状態
フェッチ順に



分岐予測：ヒット



分岐予測：ミス

レジスタ

0 以前の状態
+
0 と 1 の結果

B
R

4 無効化

5 無効化

0 以前すべて
終了

0 終了

1 終了

2 終了

3 未了

ボトム終了のとき、未了直前まで書き込み

リオーダーバッファ

3 以降の状態
フェッチ順に

2 終了
予測ミス

3 未了

リオーダーバッファの構成

後続命令

R12 ← R7, R8

⑦

R12

R7 R

R8 R R

R5 ← R11, R2

⑥

R5

R11④ W

R2 R

R9 ← R3, R5

⑤

R9

R3 R

R5③ W

R11 ← R0, R6

④

R11

R0① W

R6② W

R5 ← R3, R4

③

R5

R3 R

R4 R R

R6 ← R0, R1

②

R6

R0① W

R1 R

R0 ← R1, R2

①

R0

R1 R

R2 R R

先行命令

命令
番号

デスティネーション

ソース 1

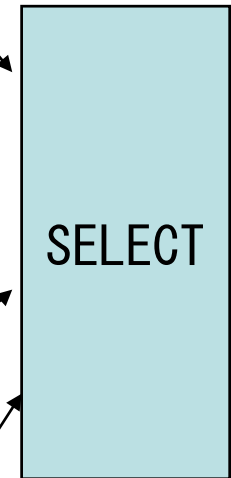
ソース 2

赤 :
Ready

R: Ready

W: Wait

命令 1 を
選択し、
実行中



オペランドリードの方法

- ①連想メモリの方法
- ②レジスタと命令番号の対応表をとる方法
- ③フーチャファイル

R10←R5, R1⑧のオペランドリード

後続命令

R10←R5, R1 ⑧ R10 R5⑥ W R1 R

R12←R7, R8 ⑦ R12 R7 R R8 R R

R5←R11, R2 ⑥ R5 R11④ W R2 R

R9←R3, R5 ⑤ R9 R3 R R5③ W

R11←R0, R6 ④ R11 R0① W R6② W

R5←R3, R4 ③ R5 R3 R R4 R R

R6←R0, R1 ② R6 R0① W R1 R

R0←R1, R2 ① R0 R1 R R2 R R

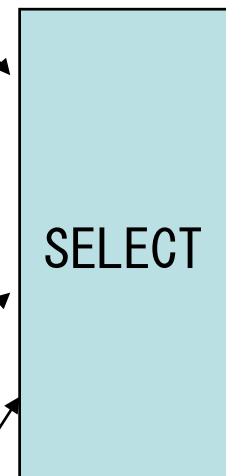
先行命令

命令
番号

デスチネー
ション

ソース 1

ソース 2



① 連想メモリによる方式 R:Ready

R5の検索：命令⑥、③ヒット。⑥選択 W:Wait

R1の検索：なし。レジスタから

命令 1 を
選択し、
実行中

R10←R5, R1⑧のオペランドリード

後続命令

R10←R5, R1⑧

R12←R7, R8⑦

R5←R11, R2⑥

R9←R3, R5⑤

R11←R0, R6④

R5←R3, R4③

R6←R0, R1②

R0←R1, R2①

先行命令

命令
番号

デスチネー
ション

ソース 1

ソース 2

R: Ready

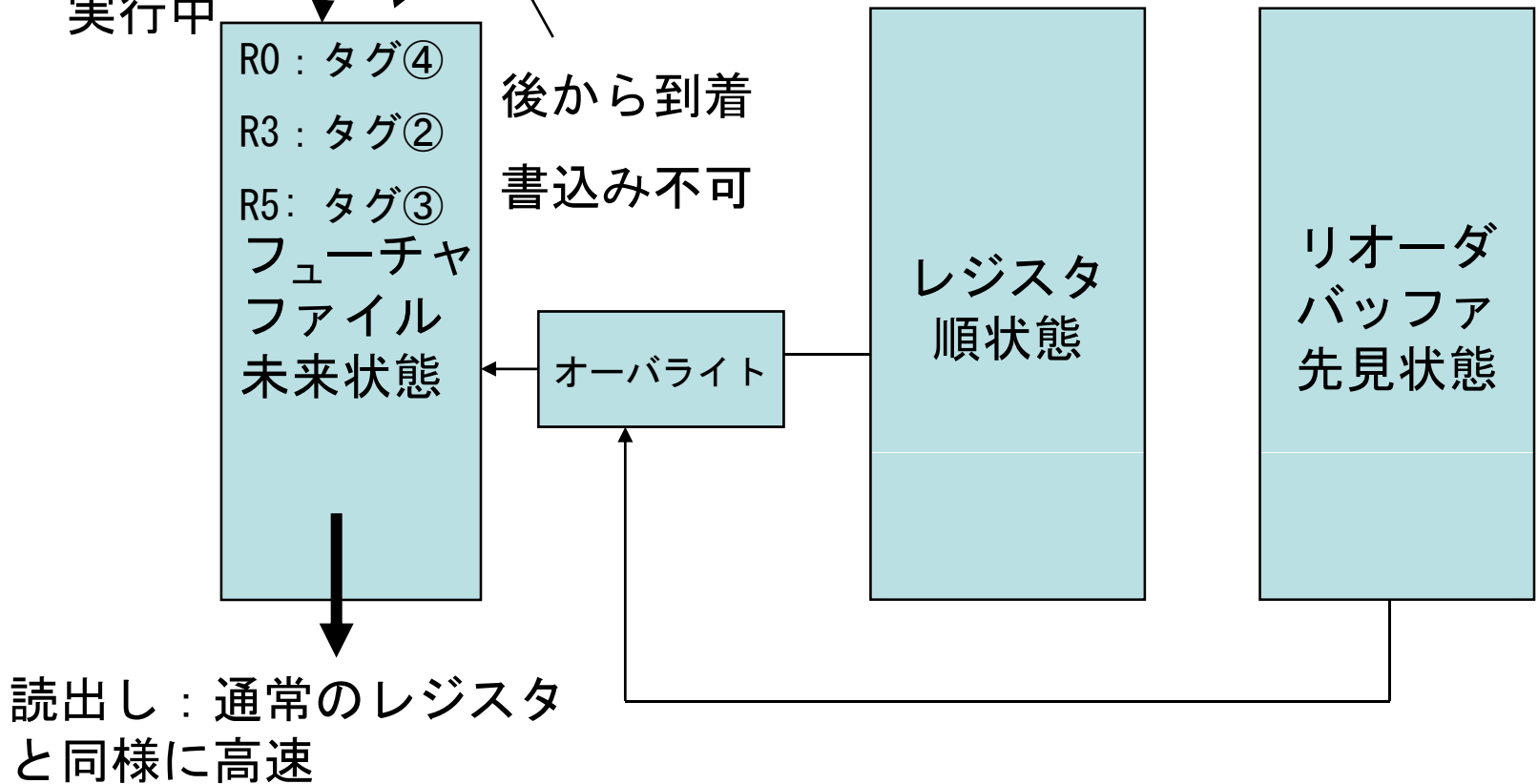
W: Wait

R12	⑦
R11	④
R10	⑧
R9	⑤
R5	⑥
R2	R2
R1	R1
R0	①

②レジスタと命令番号の
対応表をとる方式

③ フューチャファイル 命令番号：タグ

- ④ R0 ← R1, R2 R0: タグ④
 - ③ R5 ← R0, R3 R5: タグ③。タグ①、タグ②待ち
 - ② R3 ← R4, R5 R3: タグ②
 - ① R0 ← R1, R2 R0: タグ① 演算長く遅れ
- 書込み可
連想メモリ
- ④命令の
実行中



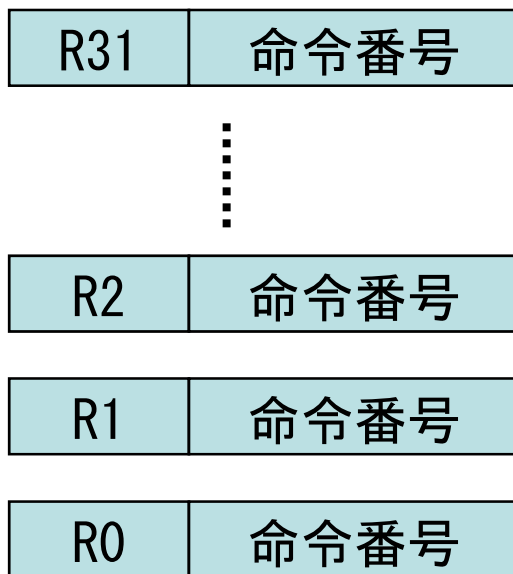
連想メモリの簡略化

リオーダバッファでの
エントリ番号

結果

レジスタ番号	命令番号	データ
--------	------	-----

レジスタ番号で直接アクセスし、命令番号比較
一致：書込み、不一致：破棄



レジスタ-命令
番号変換表

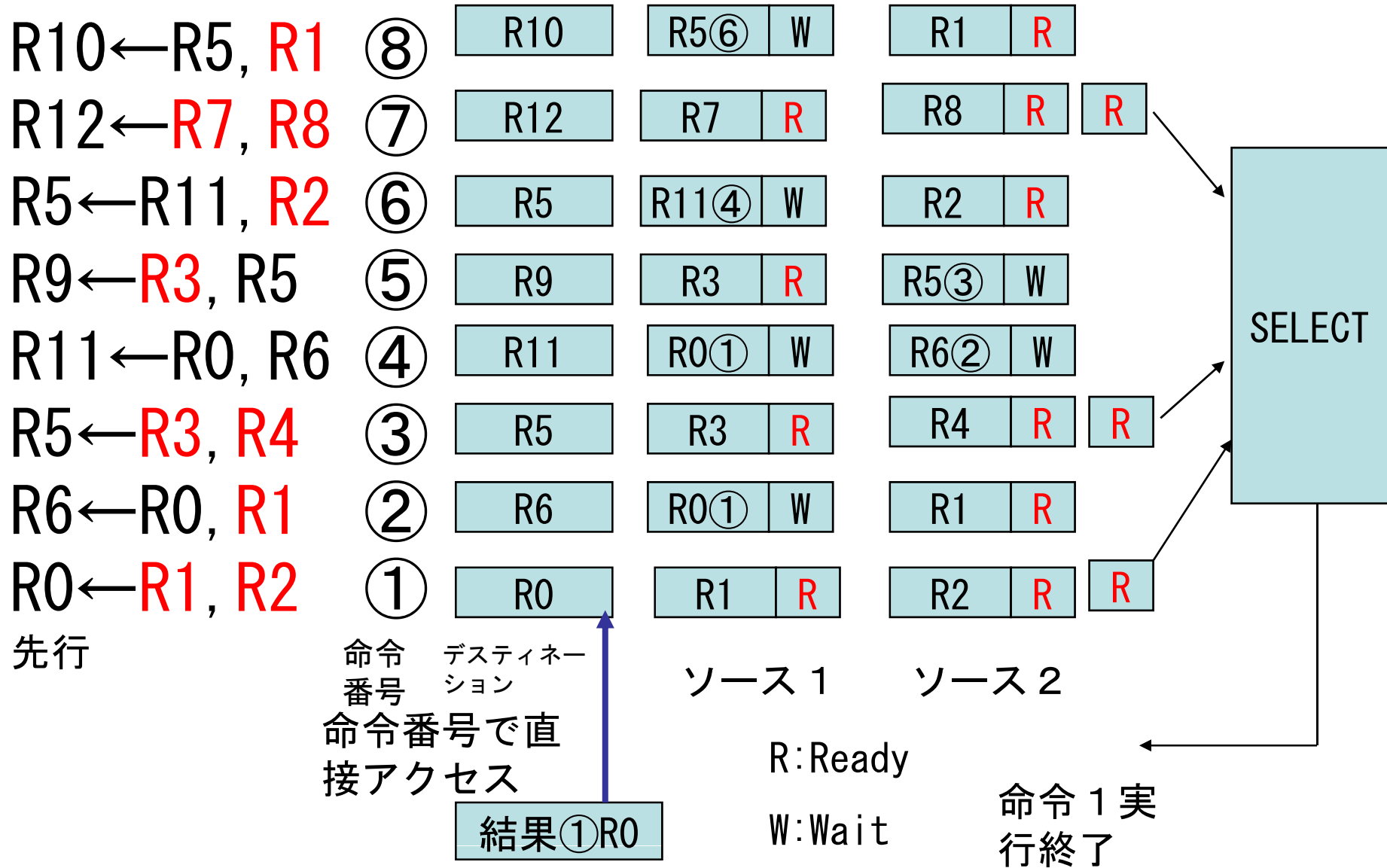
レジスタ

後続命令のWake-Up、リザベーションステーション：困難

結果のライトと次命令Wake-Up方法

- ①連想メモリを使用する方法
- ②依存行列テーブル法

R0←R1, R2①の結果のライト

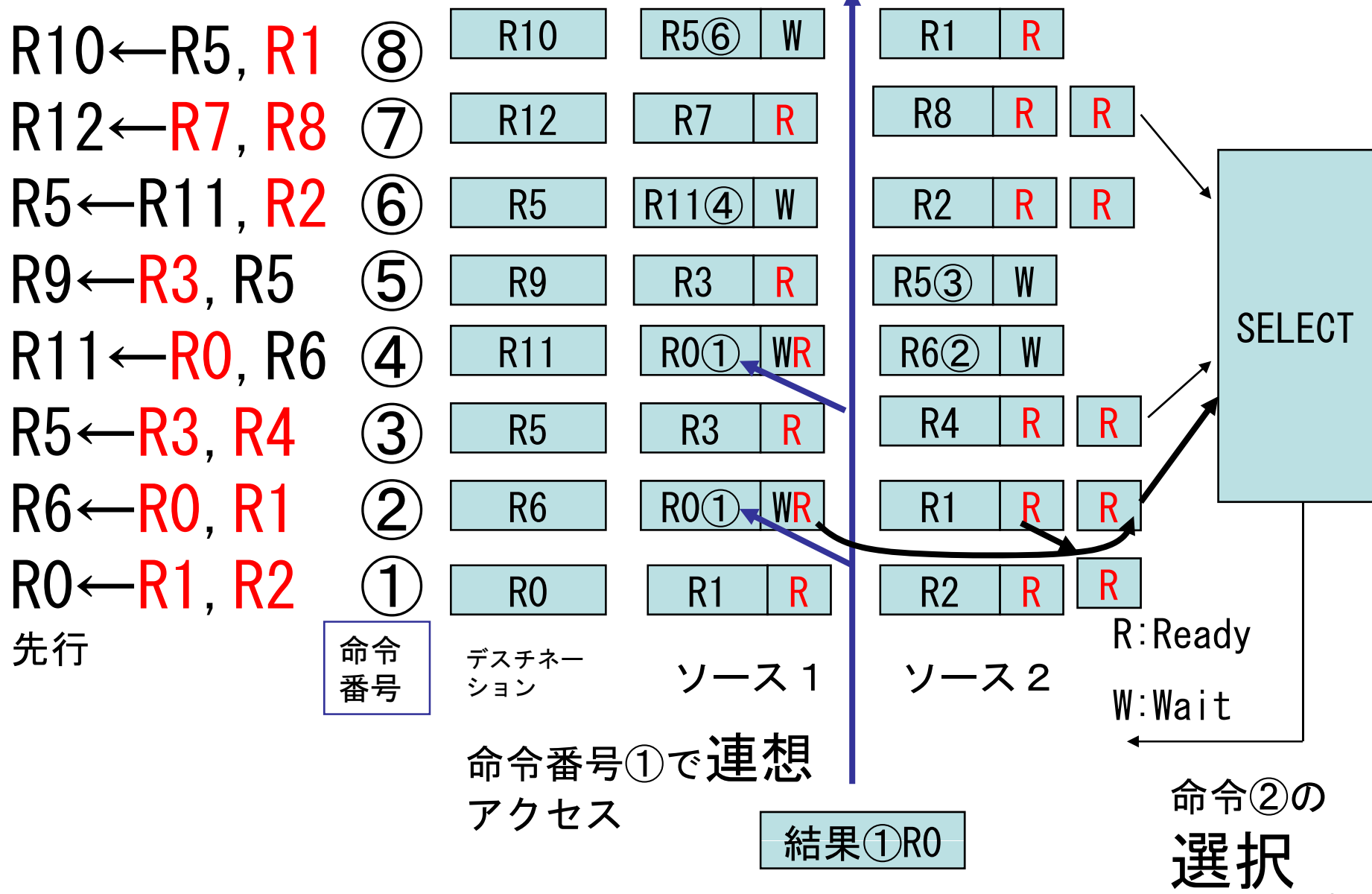


R0←R1, R2の結果による後続命令のWake-UP



①連想メモリによる方法

R0←R1, R2の結果による後続命令のSelect



レジスタ-命令番号
変換表

R11	命令番号④
R9	命令番号⑤
...	...
R6	命令番号②
R5	命令番号③
...	...
R0	命令番号①

②依存行列テーブル (DMT) 法



命令 1, 3 から 1 が選択



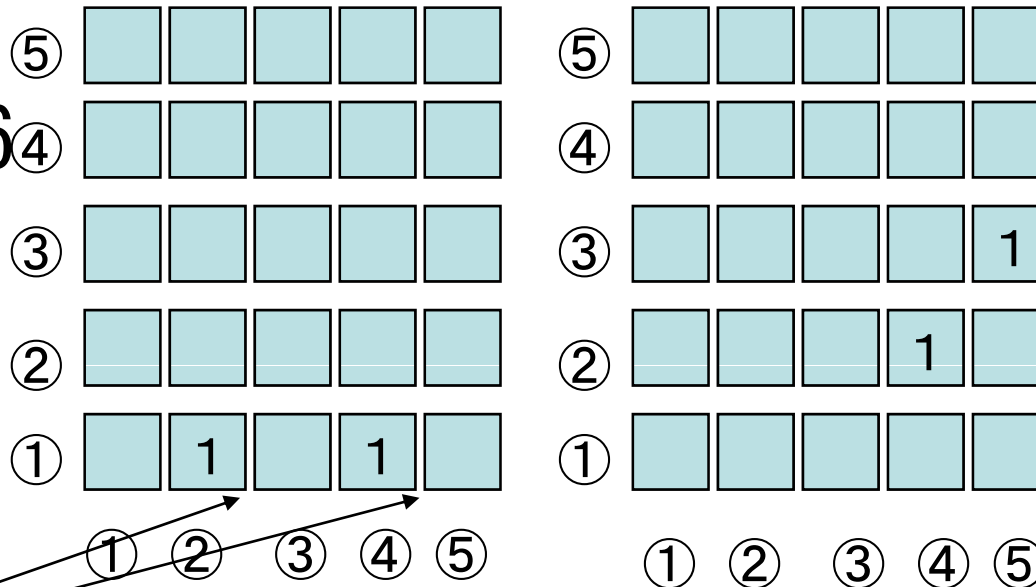
後続⑤ R9 ← R3, R5

④ R10 ← R0, R6

③ R5 ← R3, R4

② R6 ← R0, R1

先行① R0 ← R1, R2



オペランド 1

オペランド 2

命令 1 の結果 R0 を
オペランド 1 で使
用する命令番号

①

命令2、3から選択

SELECT

後続⑤R9←R3, R5

④R10←R0, R6

③R5←R3, R4

②R6←R0, R1

先行①R0←R1, R2

命令1が実行、
R0が確定

Wake Up



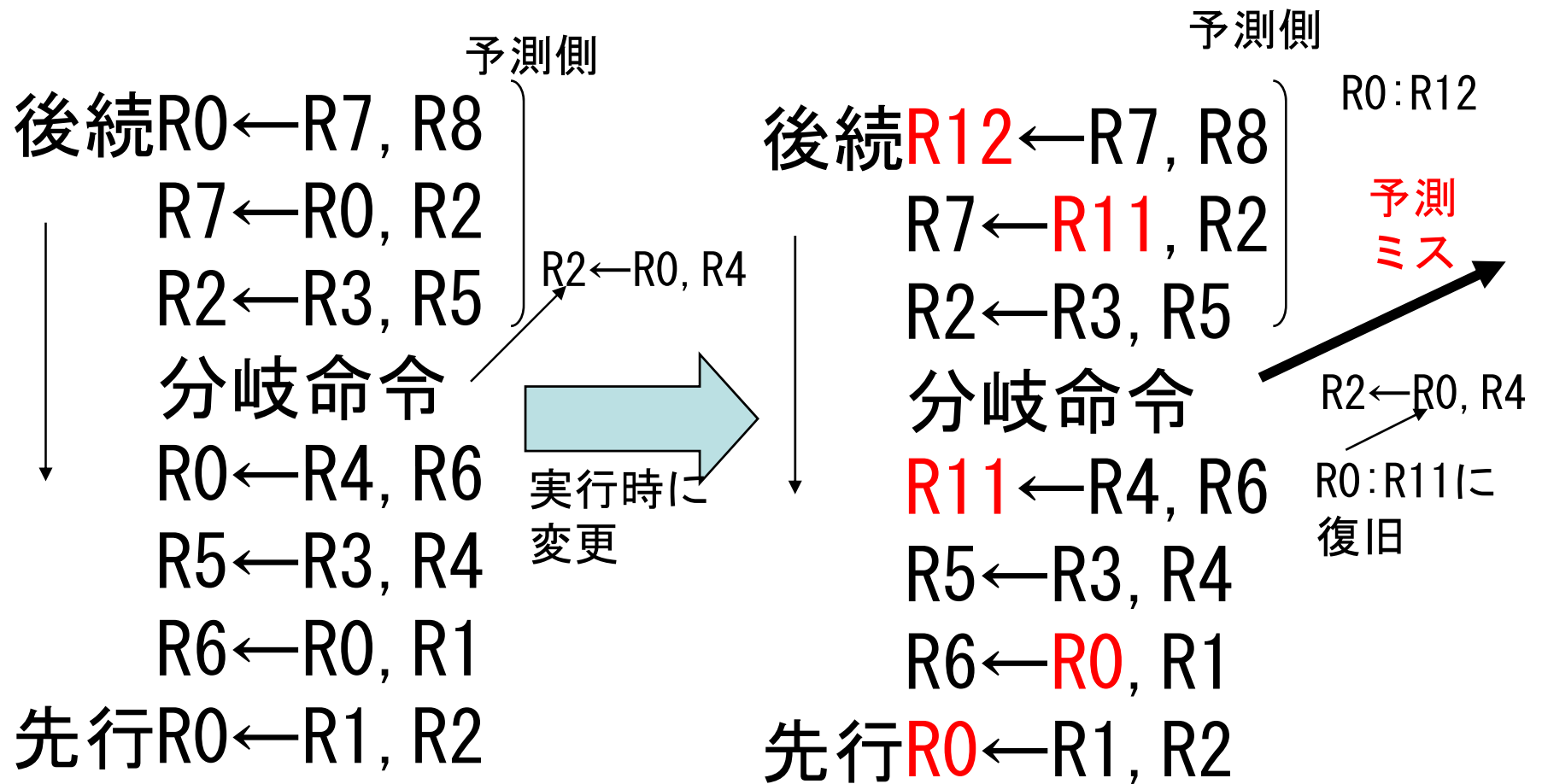
① ② ③ ④ ⑤

① ② ③ ④ ⑤

オペランド1

オペランド2

②大きな物理レジスタによる方式 (狭義レジスタリネーミング方式)



論理-物理レジスタ対応表

論理レジスタ番号	物理レジスタ番号
0	10
1	0→3
2	4
.....	
31	

大きな物理レジスタ使用

連想メモリなし

巻き戻し: 時間かかる

物理レジスタ

0
1
2
3
4
10
63

⑤
値
書込み

① 空き物理レジスタリスト

3
5
6

命令4
のデコード

アクティブリスト
リオーダーバッファ

④ 物理レジスタ
20, 15の登録

物理レジスタ番号
3から0に返す

⑦

後

命令フェッチ順

命令番号	完了ビット	論理レジスタ番号	旧物理レジスタ番号
4	未	1	0
3	未	2	13
2	完	1	15
1	未	0	20

予測ミス
分岐命令 ⑥

命令番号	完了ビット	論理レジスタ番号	旧物理レジスタ番号
4	未	1	0
3	未	2	13
2	完	1	15
1	未	0	20

③
R1: 物理 3 を利用

R1 ← R7, R8

後続

R1 ← R5, R6

R0 ← R3, R4

⋮

R0 ← R3, R4

先行

R1 ← R5, R6

②
R0: 物理 1 0 を利用
R1: 物理 0 を利用

①
R0: 物理 2 0 を利用
R1: 物理 1 5 を利用

②のR0が順状態となれば①のR0, R1は必要なくなる

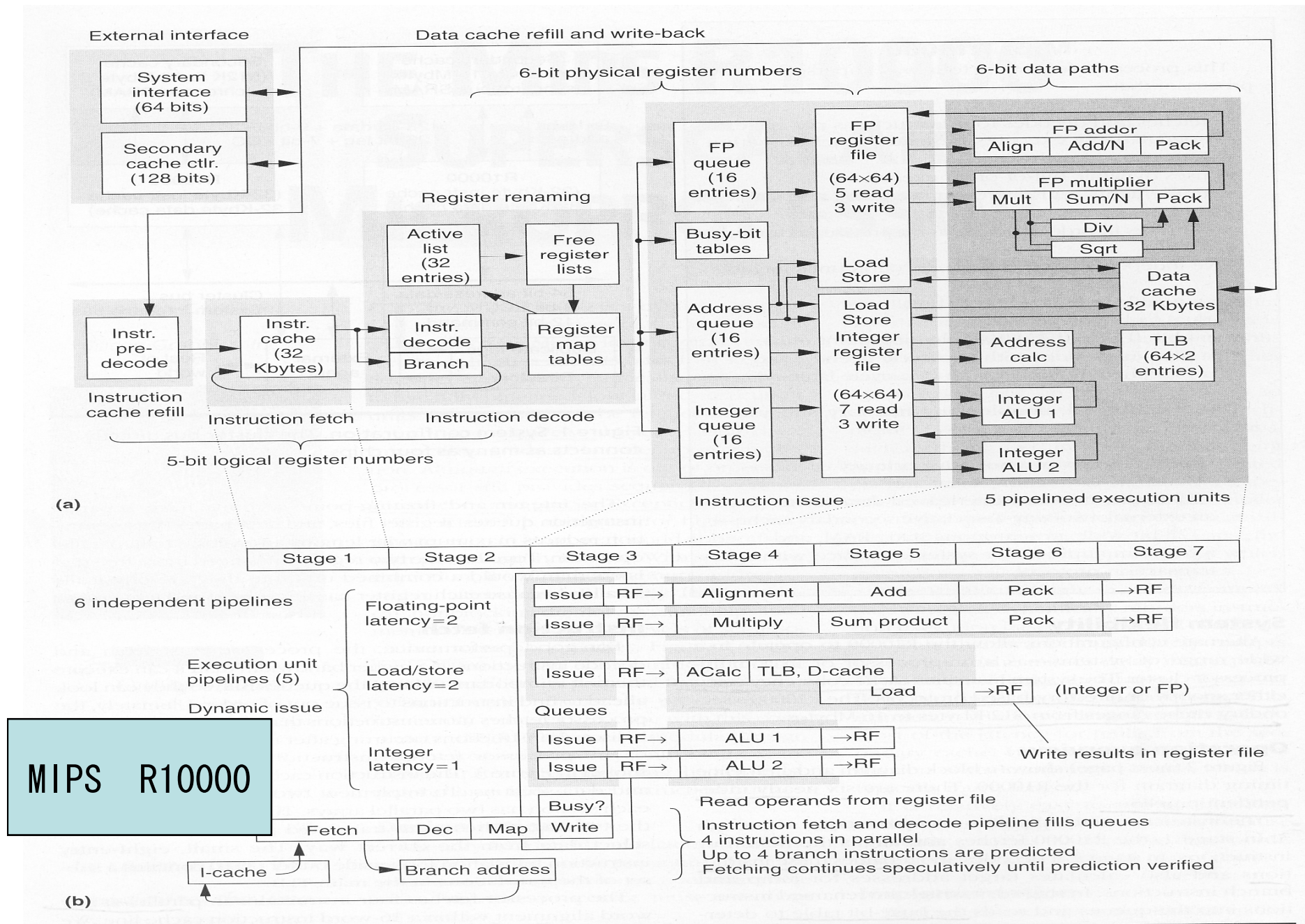
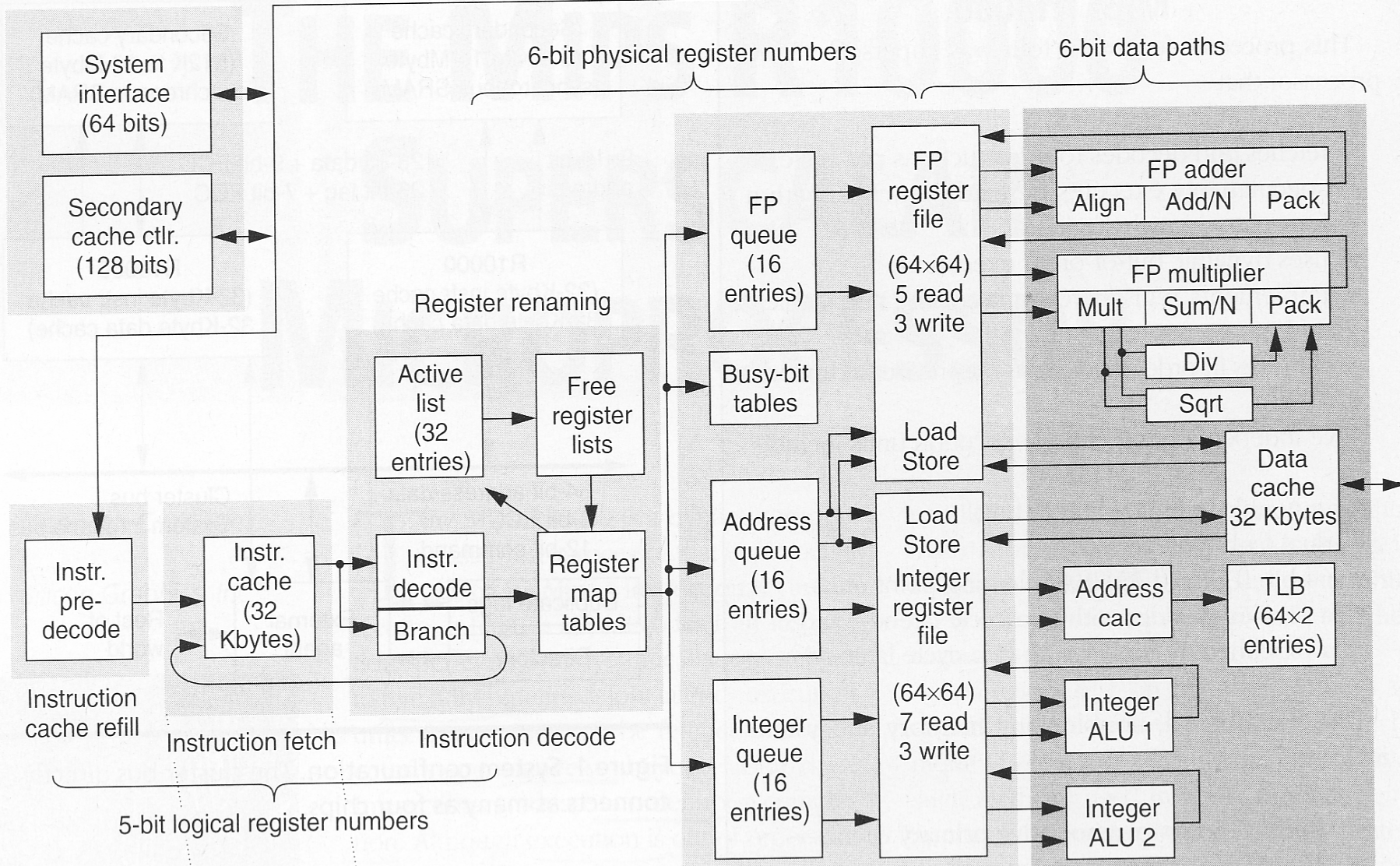


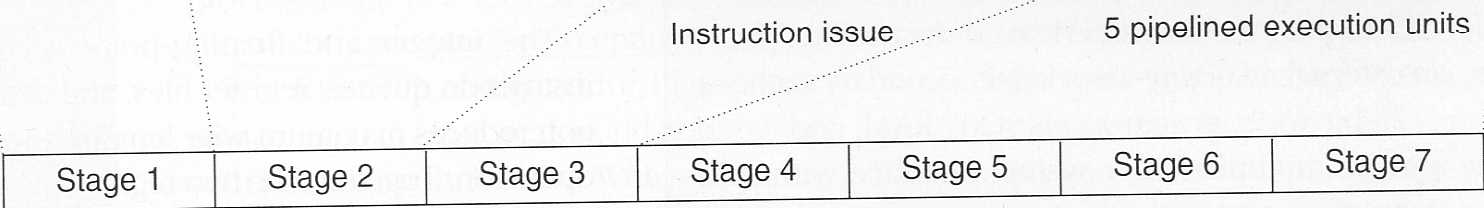
Figure 2. R10000 block diagram (a) and pipeline timing diagram (b). The block diagram shows pipeline stages left to right to correspond to pipeline timing.

External interface

Data cache refill and write-back



(a)



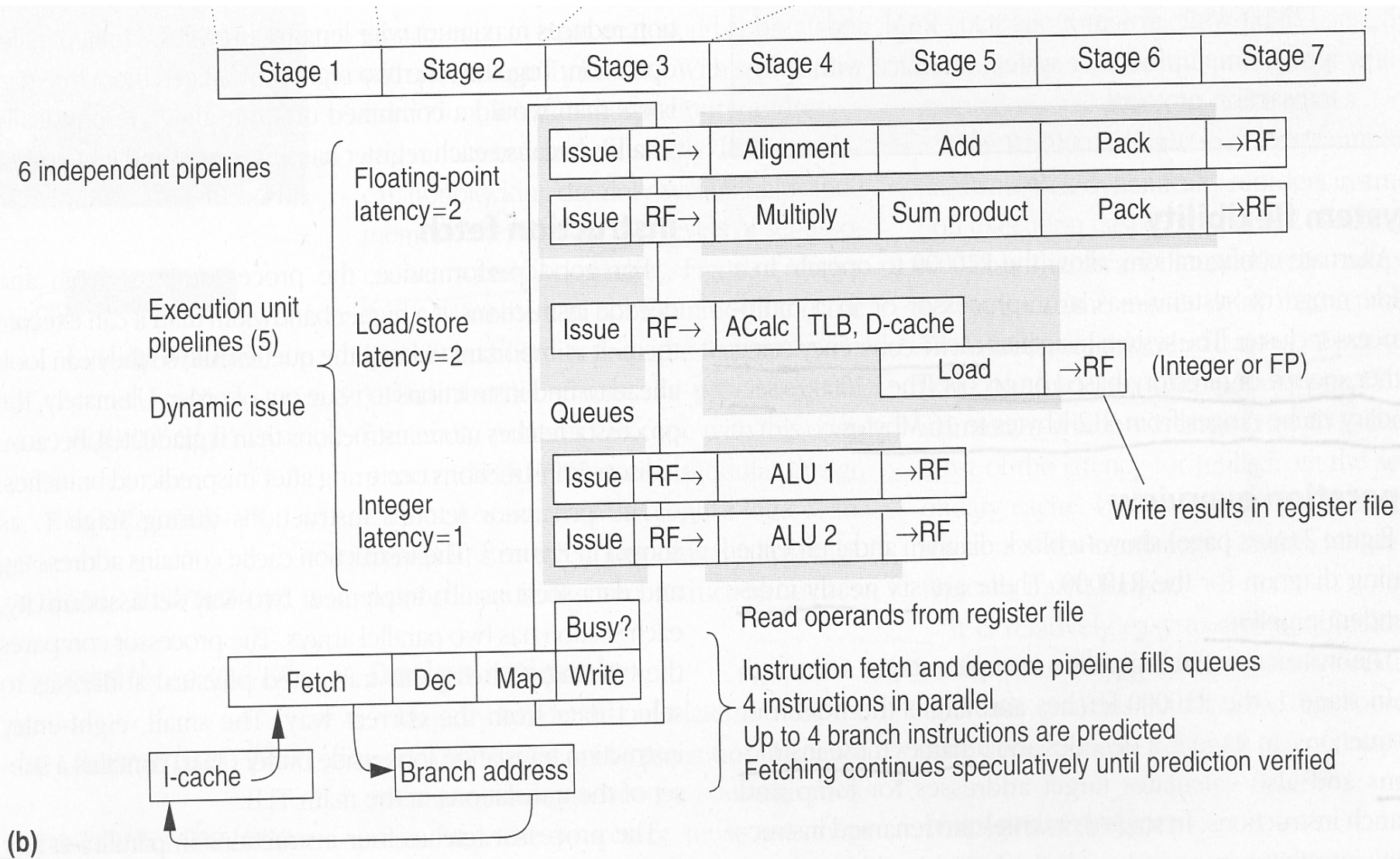


Figure 2. R10000 block diagram (a) and pipeline timing diagram (b). The block diagram shows pipeline stages left to right to correspond to pipeline timing.

6.4 スーパスカラ方式

6.4.1 基本方式

① 実行時での並列性検出

② 互換性

③ 機械命令のビット使用率

順発行乱終了スーパスカラ方式のハードウェアの増加

① 命令発行ネットワーク

② 同時に読出し書込みのできるマルチポートレジスタ

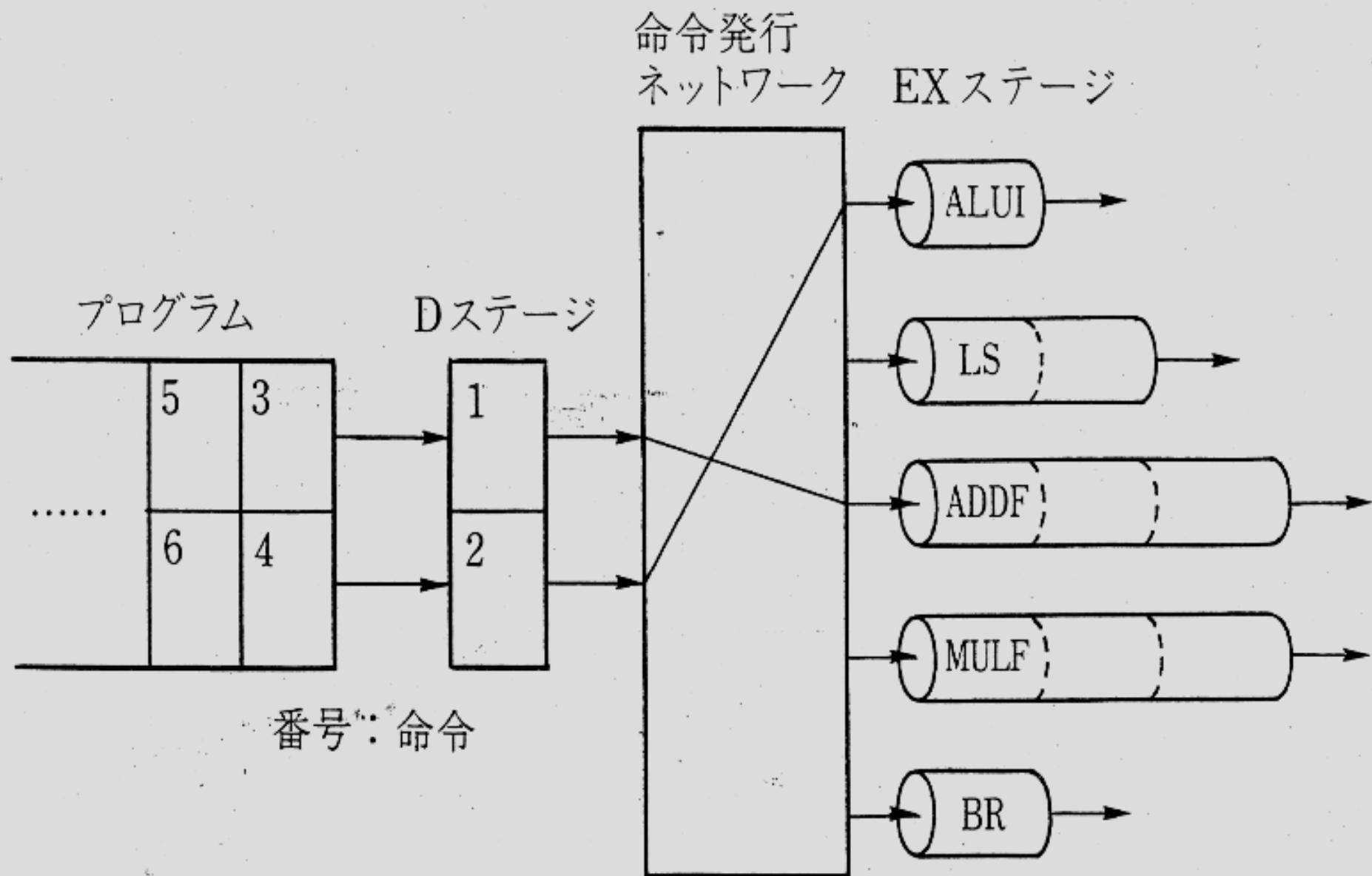
③ 演算装置の多重化

④ ロードストア装置の多重化とキャッシュメモリ強化

乱発行順終了（リオーダバッファ）

① リザーベーションステーションでの待合わせ

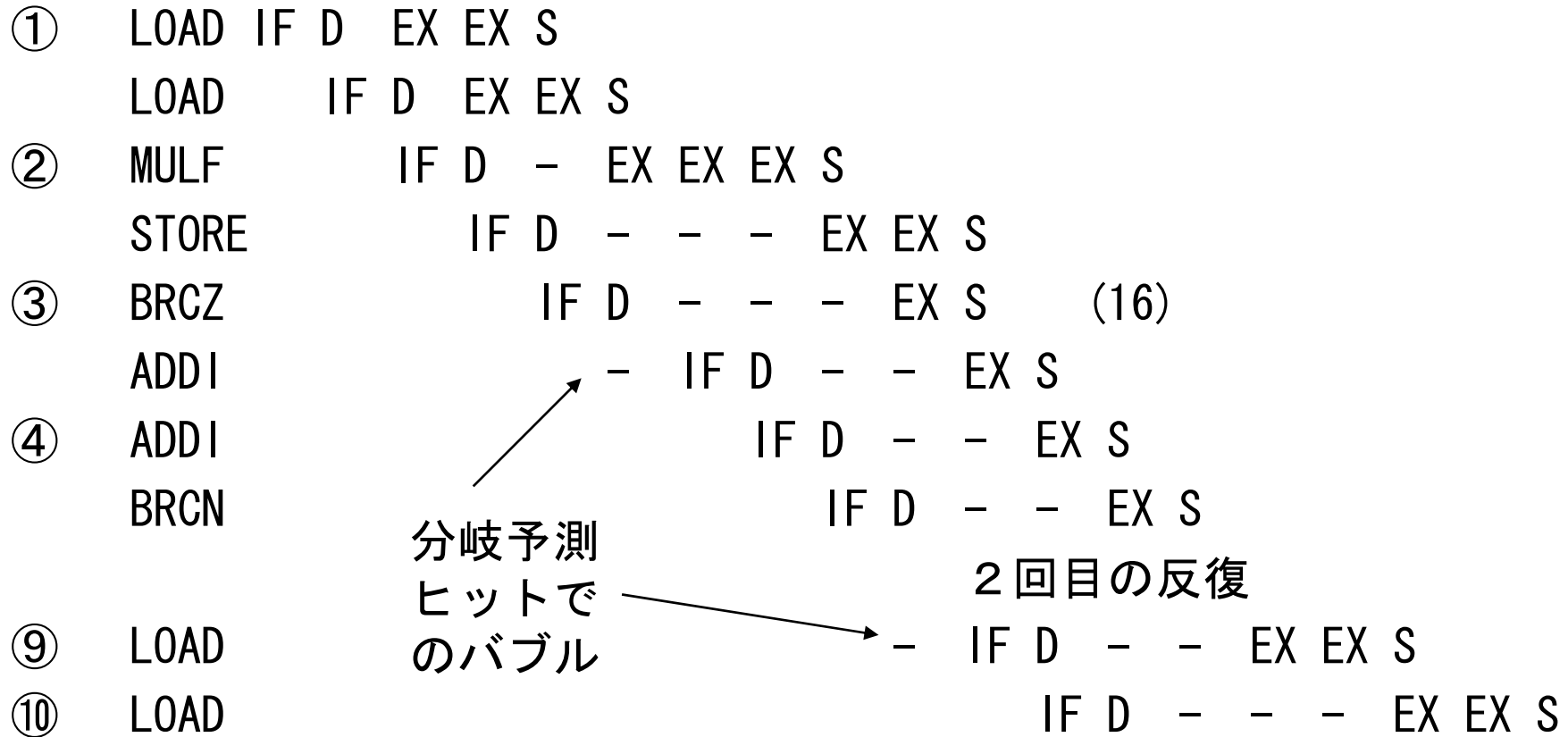
② 投機実行，正確な割込みのためのリオーダバッファ



(a) スーパスカラ方式での命令と演算装置の対応

```
DO 10 I=1, N
  Z(I)=X(I)*Y(I)
  IF (Z(I).EQ.0) GOTO 20    (12)
10 CONTINUE
  A=A+B
  D=A*D
20 ----
```

順発行乱終了スカラ



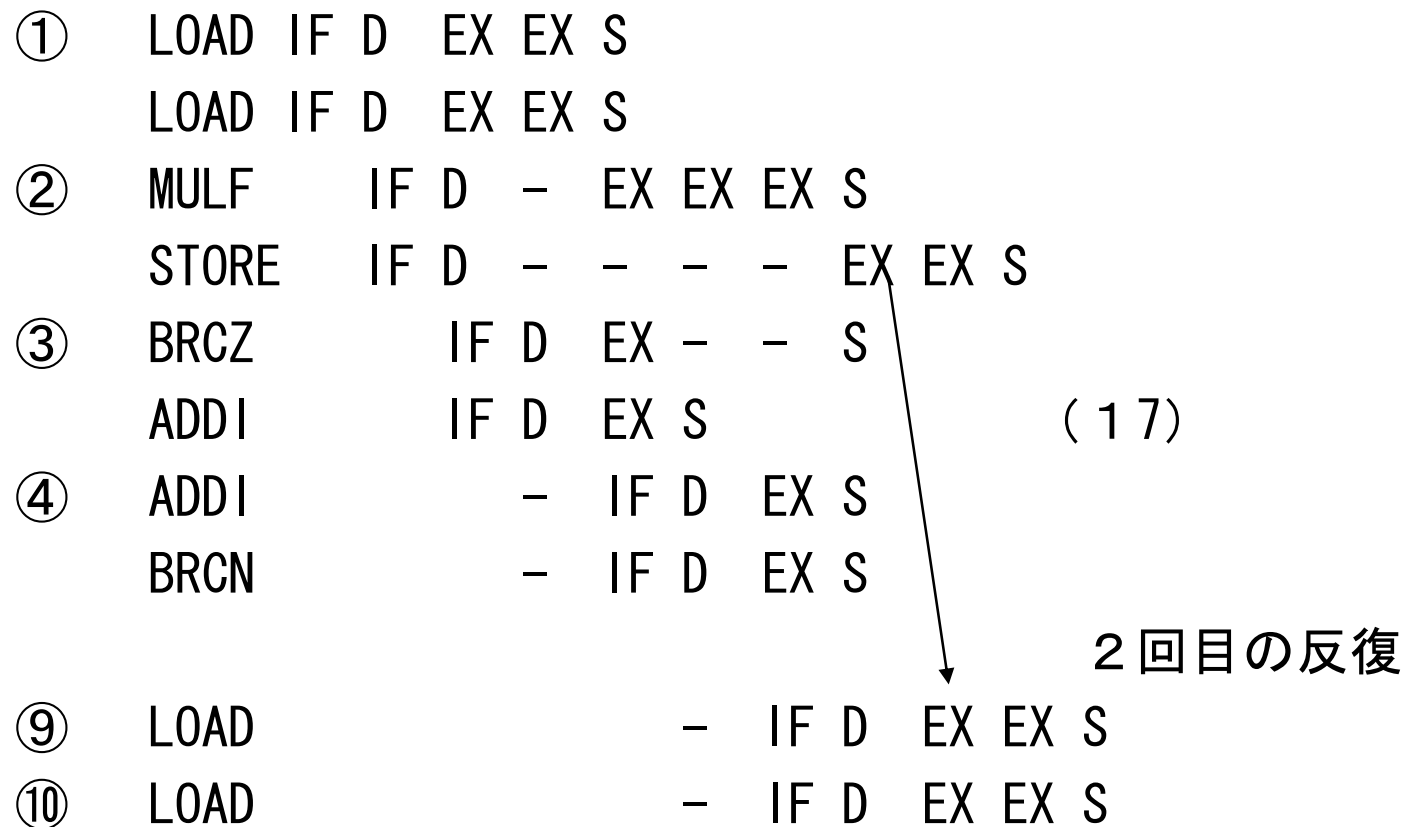
1反復当たり12サイクル

順発行乱終了スーパースカラ

①	LOAD	IF	D	EX	EX	S		
	LOAD	IF	D	EX	EX	S		
②	MULF	IF	D	-	EX	EX	EX	S
	STORE	IF	D	-	-	-	-	EX EX S
③	BRCZ	IF	D	-	-	-	-	EX S (15)
	ADDI	IF	D	-	-	-	-	EX S
④	ADDI	-	IF	D	-	-	-	EX S
	BRCN	-	IF	D	-	-	-	EX S
								2回目の反復
⑨	LOAD	-	IF	D	-	-	-	EX EX S
⑩	LOAD	-	IF	D	-	-	-	EX EX S

1反復当たり8サイクル

乱発行乱終了スーパースカラ



1 反復当たり 6 サイクル

6.4.3 初期のスーパースカラ方式

表6.1

6.4.4 最新のマイクロプロセッサ

表6.2 : Pentium4、Power4、UltraSparcIII

表 6.1 初期のスーパースカラ方式

	<u>DECα 21064</u>	<u>HP-PA 7100</u>	<u>VR 4000</u>	<u>Pentium</u>	<u>Power PC 601</u>
トランジスタ数	168 万	85 万	130 万	310 万	280 万
ピン数	431	504	447	273	304
電力	30 W	23 W	10 W	16 W	9.1 W
マシンサイクル	200 MHz	100 MHz	50 MHz	66 MHz	80 MHz
命令長	32 ビット	32 ビット	32 ビット	多様	32 ビット
パイプライン			(内部 100 MHz)		
方式	スーパースカラ	スーパースカラ	スーパー	スーパースカラ	スーパースカラ
多重度	2 多重	2 多重	パイプライン	2 多重	3 多重
ステージ数					
整数	7	5	8	5	4
浮動小数点	10	8	?	8	6
内蔵キャッシュ(ファーストキャッシュ)					
命令	8KB	なし	8KB	8KB	統合
データ	8KB	なし	8KB	8KB	32KB
方式	ダイレクト マッピング		ダイレクト マッピング	2ウェイセット アソシアティブ	8ウェイセット アソシアティブ
	ストアスルー		ストアイン	ストアイン	ストアイン
	実アドレス		仮想・実	?	実アドレス
			アドレス混合		
TLB エントリ数	40	136	48	96	280
分岐	分岐予測	遅延分岐	遅延分岐	分岐予測	分岐予測

深／浅パイプライン

深：ステージ機能低い，周波数高

浅：ステージ機能高い，周波数低

Pentium III : 10段, 1GHz

Pentium 4 : 20段, 1.5GHz

性能：1.2倍

12 高性能マイクロプロセッサの現状

表6. 2 最新のマイクロプロセッサ

モデル名	21264	Pentium4	UltraSPARCIII	Power4
発表年	1998	2000	2001	2001
デザインルール	0.35 μm	0.18 μm	0.18 μm	0.18 μm
Tr 数	1520万	4200万	2900万	17000万
ピン数	587	478	1368	2200
配線層	6	6	6	7
周波数	600MHz	1.4GHz	900MHz	1.3GHz
電力	60W	55W	70W	50W
プロセッサ数	1	1	1	2
パイプライン				
命令フェッチ	4多重	3多重	4多重	8多重
基本ステージ数	7	20	14	15
演算器個数				
整数	4	4	2	2
浮動	2	1	2	2
ロードストア	2	2	1	2
命令発行	リネーミング レジスタ	リネーミング レジスタ	フューチャ ファイル (スコアボード)	リネーミング レジスタ
	乱発行同時終了	乱発行同時終了	乱発行同時終了	乱発行同時終了
リオーダバッファ	80命令	126命令	?	200命令
リネーミング	整数：80個	合計128個	?	80個
レジスタ	浮動：72個			72個

GHz

分岐予測

方式	ハイブリッド	ハイブリッド?	大域	ハイブリッド
予測テーブル容量	2.9Kビット	3.2Kビット	3.2Kビット	4.8Kエン트리
	局所2レベル適応		2レベル適応	局所: 1.6Kエン트리
	大域2ビット		(Gshare)	大域: 1.6Kエン트리
	カウンタ		1.6Kエン트리	上記選択: 1.6Kエン트리
			2ビットカウンタ	

1次キャッシュ

命令	6.4KB	トレースキャッシュ 1.2K μ 命令	3.2KB	6.4KB
データ	2ウェイ 6.4KB	8ウェイ 8KB	4ウェイ 6.4KB	ダイレクトマップ 3.2KB
	2ウェイ ストアイン	4ウェイ ストアスルー	4ウェイ ストアスルー	2ウェイ ストアスルー
2次キャッシュ	1.6MB オフチップ	2.56KB オンチップ	8MB オフチップ	1.5MB オンチップ
	統合, ダイレクト マップ	統合, 8ウェイ ストアイン	統合, ダイレクト マップ	統合, 8ウェイ, ストアイン, スヌープコントローラ (2台のプロセッサで共有)

性能

測定時周波数	1GHz	2GHz	900MHz	1.3GHz
SPECint2000	621	640	438	790
SPECfp2000	776	704	427	1098

CHART WATCH: WORKSTATION PROCESSORS

Processor	Alpha 21264C	AMD Athlon XP	HP PA-8700	IBM Power4	Intel Itanium	Intel Itanium 2	Intel Xeon	MIPS R14000	Sun Ultra-III
Clock Rate	1,001MHz	1,800MHz	750MHz	1,300MHz	800MHz	1,000MHz	2,530MHz	600MHz	1,050MHz
Cache (L1/D/L2/L3)	64K/64K	64K/64K/256K	750K/1.5M	64K/32K/1.5MB	16K/16K/96K	16K/16K/256K/3M	12K/8K/512K	32K/32K	32K/64K
Issue Rate	4 issue	3 x86 instr	4 issue	8 issue	6 issue	8 issue	3 ROPs	4 issue	4 issue
Pipeline Stages	7/9 stages	9/11 stages	7/9 stages	12/17 stages	10 stages	8 stages	22/24 stages	6 stages	14/15 stages
Out of Order	80 instr	72ROPs	56 instr	200 instr	None	None	126 ROPs	48 instr	None
Rename Regs	48/41	36/36	56 total	48/40	328 total	328 total	128 total	32/32	None
BHT Entries	4K x 9-bit	4K x 2-bit	2K x 2-bit	3 x 16K x 1-bit	512 x 2-bit	512 x 2-bit	4K x 2-bit	2K x 2-bit	16K x 2-bit
TLB Entries	128/128	280/288	240 unified	1,024 unified	64I/32L1D/96L2D	64I/32L1D/96L2D	128I/64D	64 unified	128I/512D
Memory B/W	8GB/s	2.1GB/s	1.54GB/s	12.8GB/s	2.1GB/s	6.4GB/s	3.2GB/s	539 MB/s	4.8GB/s
Package	CPGA-588	PGA-462	LGA-544	MCM	PAC-418	mPGA-700	PGA-423	CPGA-527	FC-LGA 1368
IC Process	0.18µ 6M	0.13µ 6M	0.18µ 7M	0.18µ 7m	0.18µ 6M	0.18µ 6M	0.13µ 6M	0.25µ 4M	0.15µ 7M
Die Size	115mm ²	80mm ²	304mm ²	400mm ² **	300mm ² *	400mm ² *	131mm ²	204mm ²	210mm ²
Transistors	15.4 million	37.5 million	130 million	174 million**	25 million	221 million	55 million	7.2 million	29 million
Est Die Cost	\$28*	\$16*	\$96*	\$144**	\$116*	\$139*	\$23*	\$76*	\$72*
Power (Max)	95W*	67W(MTP)	75W*	135W**	130W	130W	59W(TDP)	30W*	75W*
Availability	3Q01	2Q02	4Q01	4Q01	2Q01	3Q02	2Q02	1Q02	1Q02

Source: vendors, except *MDR estimates. Estimated manufacturing costs does not include external cache chips. ** Contains two processors on one die.

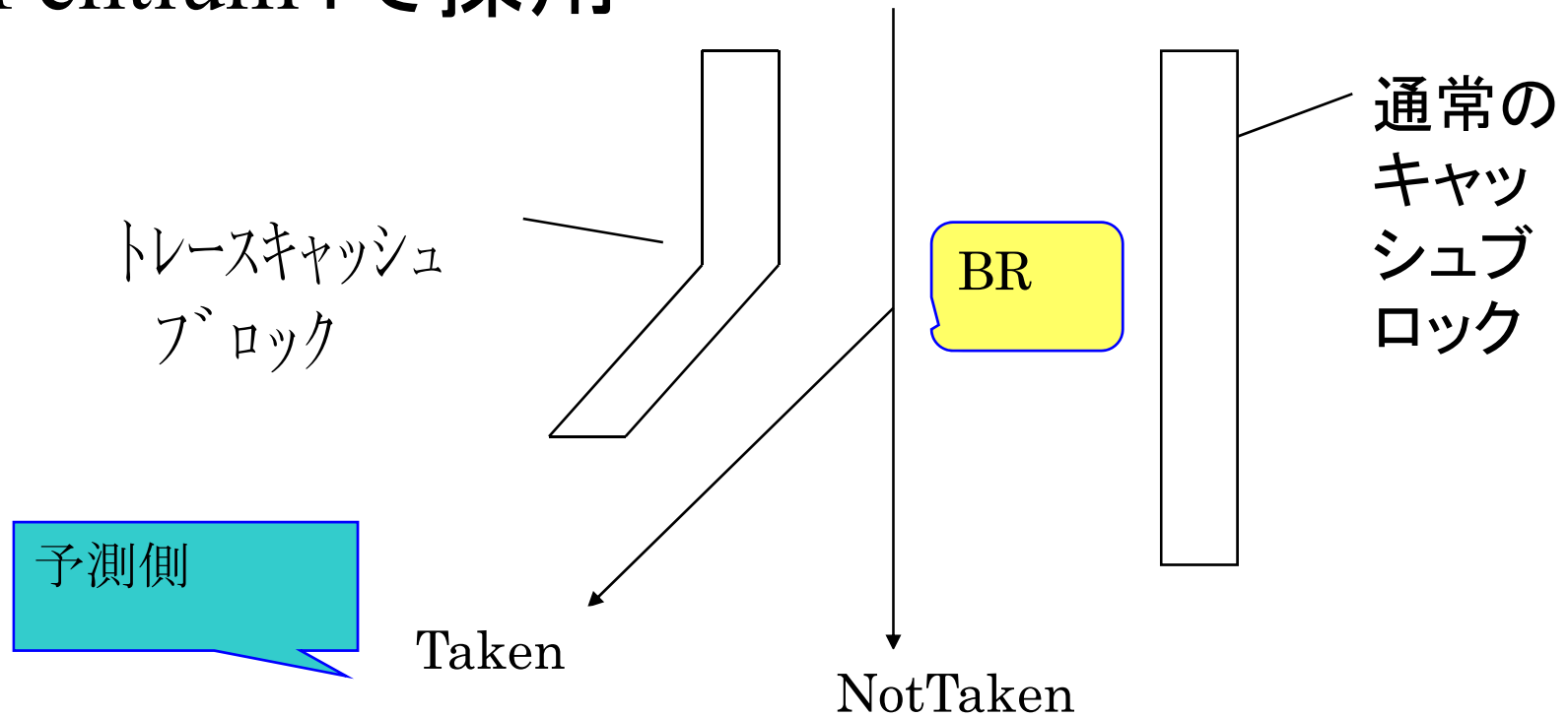
The table above gives the vital statistics for the key high-end processors available. The table below provides the best reported SPEC CPU2000 (base) results for each shipping processor.

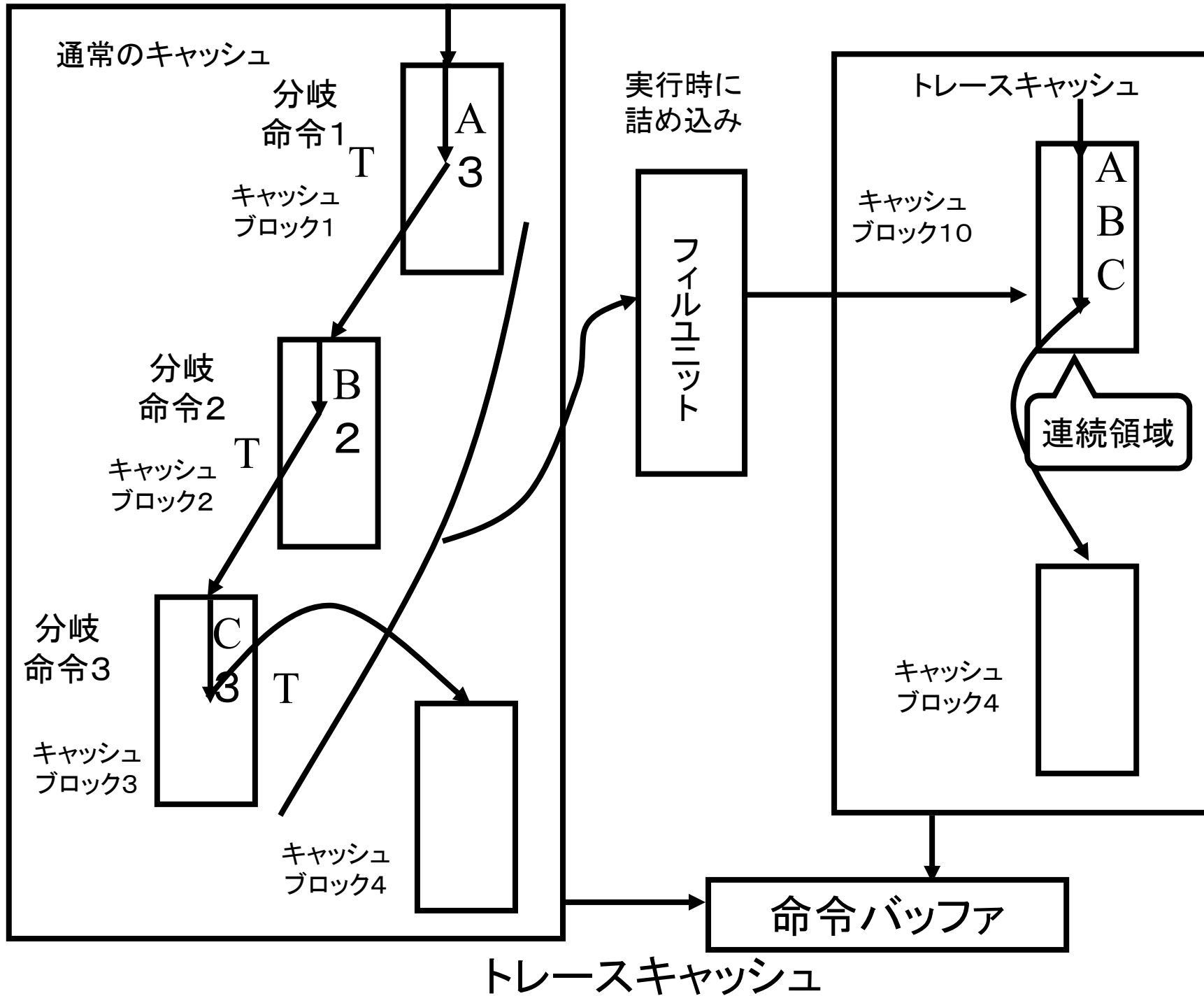
Processor	Alpha 21264C	AMD Athlon XP	HP PA-8700	IBM Power 4	Intel Itanium	Intel Itanium 2	Intel Xeon	MIPS R14000	Sun UltraSPARC III
System or Motherboard	Alpha E540 Model 6	EpoX BKHA+ rp7400	HP9000 rp7400	pSeries 690 Turbo	HP I2000	HP RX2600	Dell Prec. 340	SGI 3200	Sun Blade 2050
Clock Rate	1,001MHz	1.8GHz	750MHz	1,300MHz	800MHz	1,000MHz	2,530MHz	600MHz	1,050MHz
External Cache	8MB	None	None	128MB	4MB	None	None	8MB	8MB
164.gzlp	463	903	495	563	332	583	911	322	433
175.vpr	534	448	550	718	376	704	511	572	460
176.gcc	693	504	710	788	407	1,014	1,058	445	577
181.mcf	550	346	396	1,087	402	834	625	783	659
186.crafty	816	1,059	642	673	356	781	860	502	558
197.parser	418	674	427	445	296	660	883	409	488
252.eon	799	1,452	508	985	370	1,004	1,162	507	527
253.perlbmk	635	1,094	510	694	340	875	1,171	367	540
254.gap	454	844	242	746	256	680	1,130	308	372
255.vortex	844	1,157	930	1,246	459	1,193	1,338	679	738
256.bzip2	656	593	411	868	334	759	709	493	629
300.twolf	795	568	781	1,027	449	880	750	645	570
SPECint_base2000	621	738	520	790	358	810	893	483	537
168.wupside	660	928	333	1,570	591*	1,003	1,234	434	659
171.swim	1502	840	790	1,493	1,369*	3,205	1,425	529	980
172.mgrid	517	551	469	726	749*	1,720	845	379	487
173.applu	741	575	591	960	1,022*	2,033	939	381	310
177.mesa	800	968	570	665	329*	642	1,043	425	543
178.galgel	1638	578	1,374	2,637	1,019*	2,505	1,133	1,398	1,713
179.art	1782	431	500	1,703	2,369*	4,226	586	1,436	9,389
183.earthquake	332	574	239	1,696	834*	1,871	903	347	645
187.facerec	1024	757	334	1,310	637*	1,152	1,134	647	958
188.ammp	592	477	472	751	511*	788	563	573	509
189.lucas	891	666	300	1,124	837*	1,206	1,155	442	371
191.fma3d	703	711	273	895	323*	747	806	306	400
200.sixtrack	401	456	407	505	575*	894	459	298	366
301.aspl	641	504	591	940	350*	678	684	406	471
SPECfp_base2000	776	624	464	1,098	703*	1,356	878	499	701

*Dell PowerEdge 7150 with 4MB L3 cache

トレースキャッシュ

Pentium4で採用





6.5 VLIWプロセッサ

(1) VLIW方式の特徴

- ① コンパイル時の並列性検出
- ② シンプルなハードウェア構成
- ③ 機械命令のビット使用率の低下
- ④ 目的コードレベルでの非互換性

(2) プログラムの実行例

プログラムの実行例

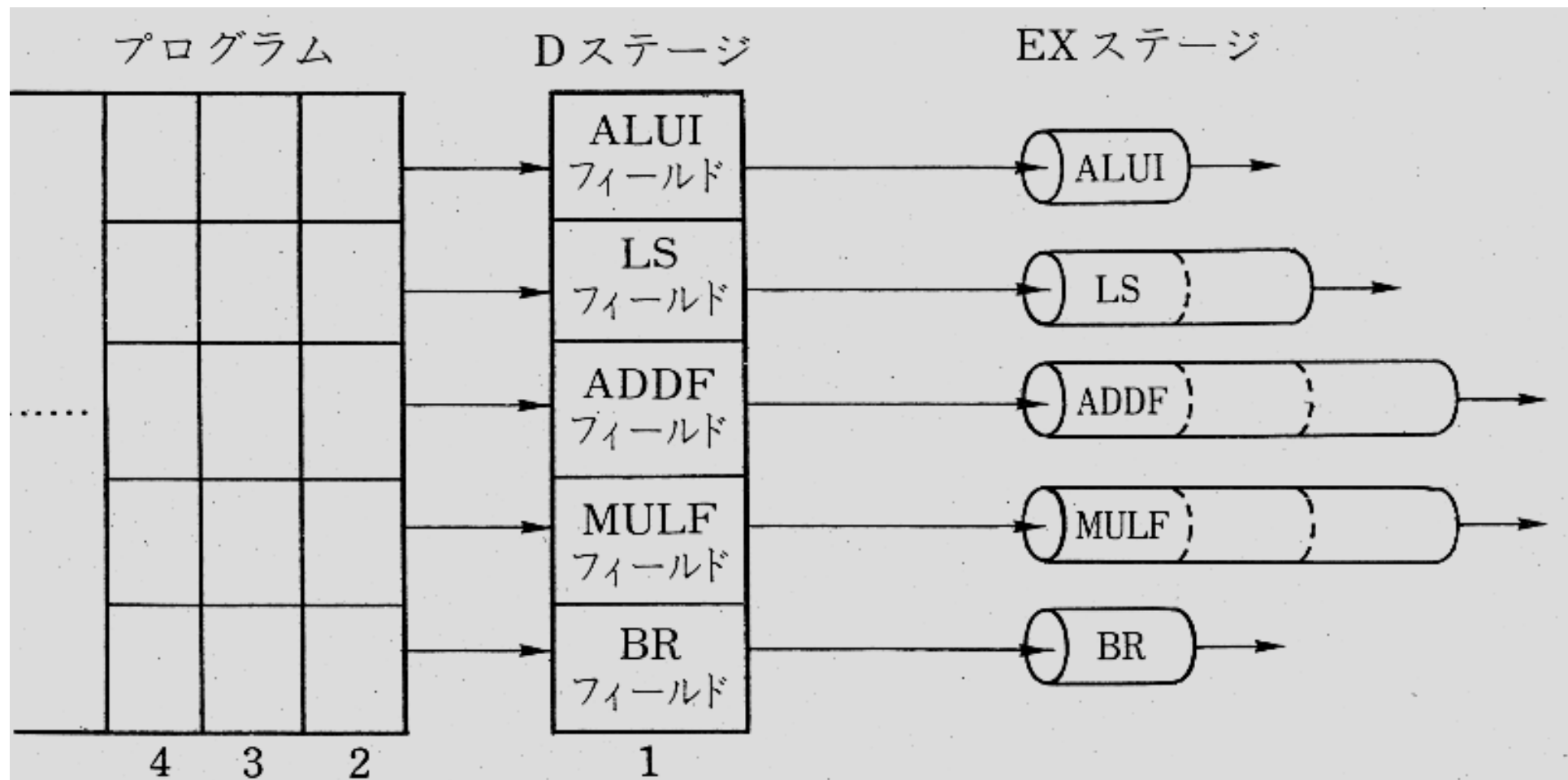
ロードストア装置 2、整数演算装置 2、浮動
小数点加算装置 1、

浮動小数点乗算装置 1、分岐装置 1 とし、遅
延分岐を採用

1970 / 1980年代のVLIW方式
VLIWの命名者 J. Fisher、1983年

1976年 : AP-120B、QA-1/2

1987年 : ELI-512、Trace、Cydra

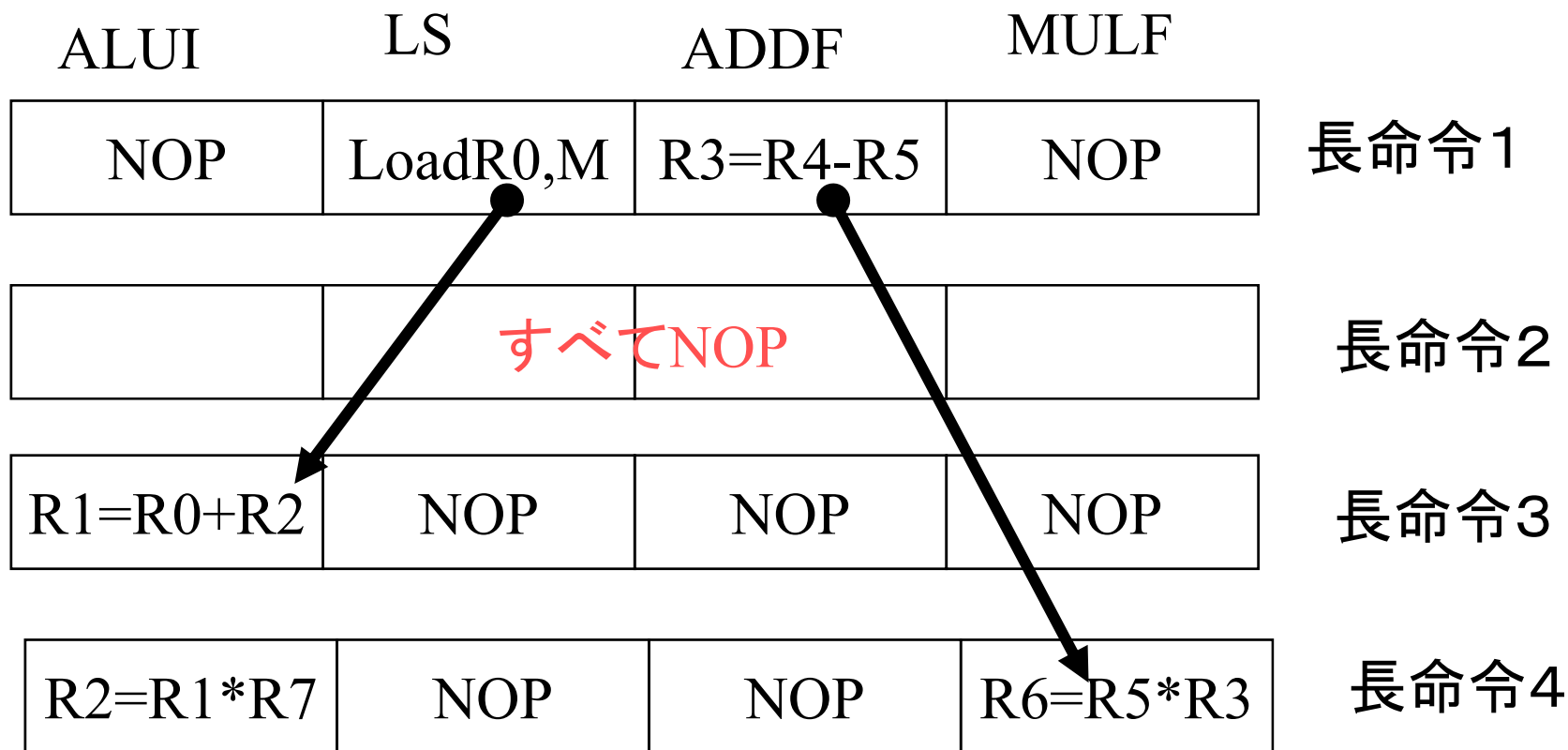


番号：長命令

(b) VLIW 方式での命令と演算装置の対応

ALUI：整数，LS：ロードストア，ADDF：浮動小数点加算，
MULF：浮動小数点乗算，BR：分岐

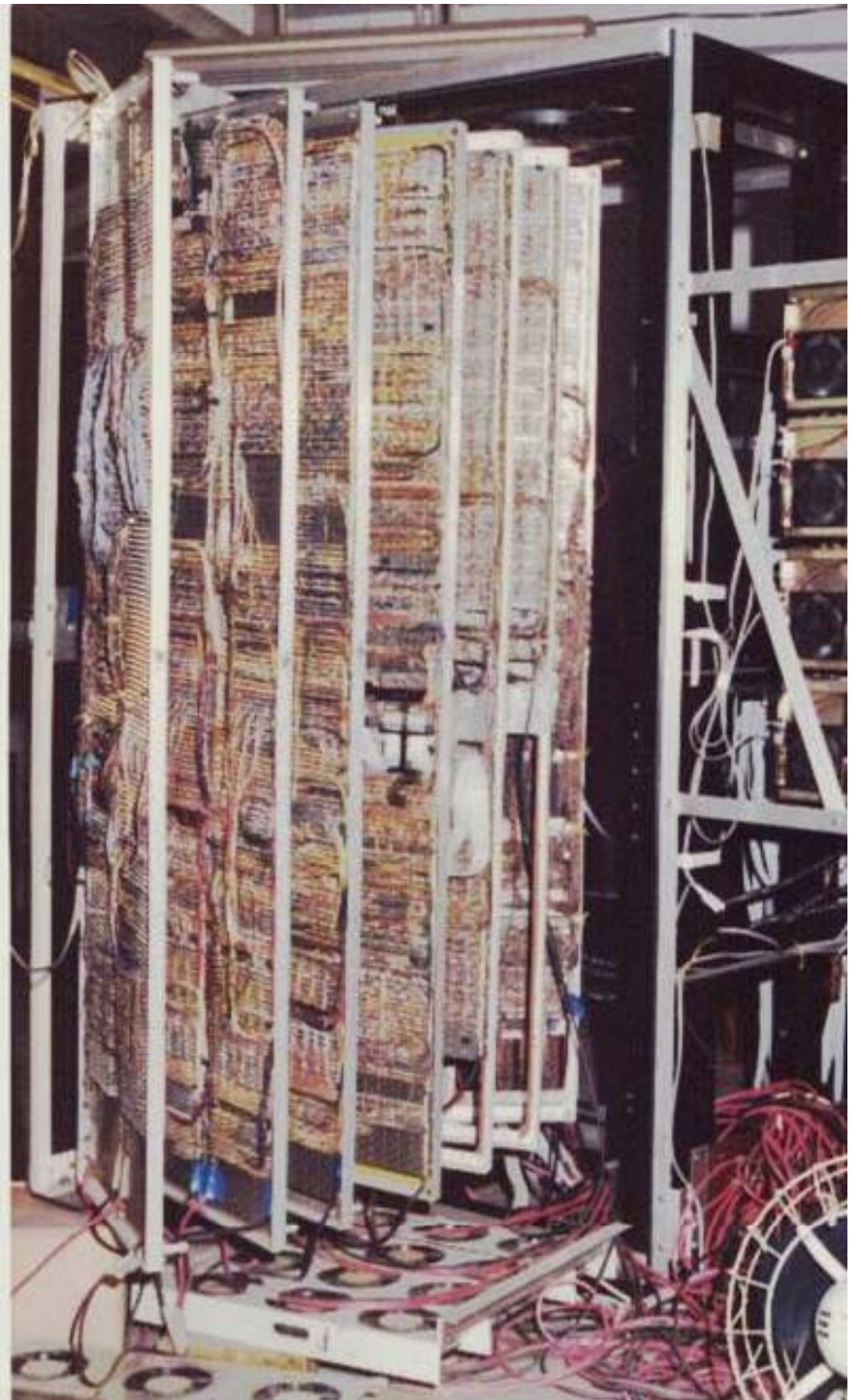
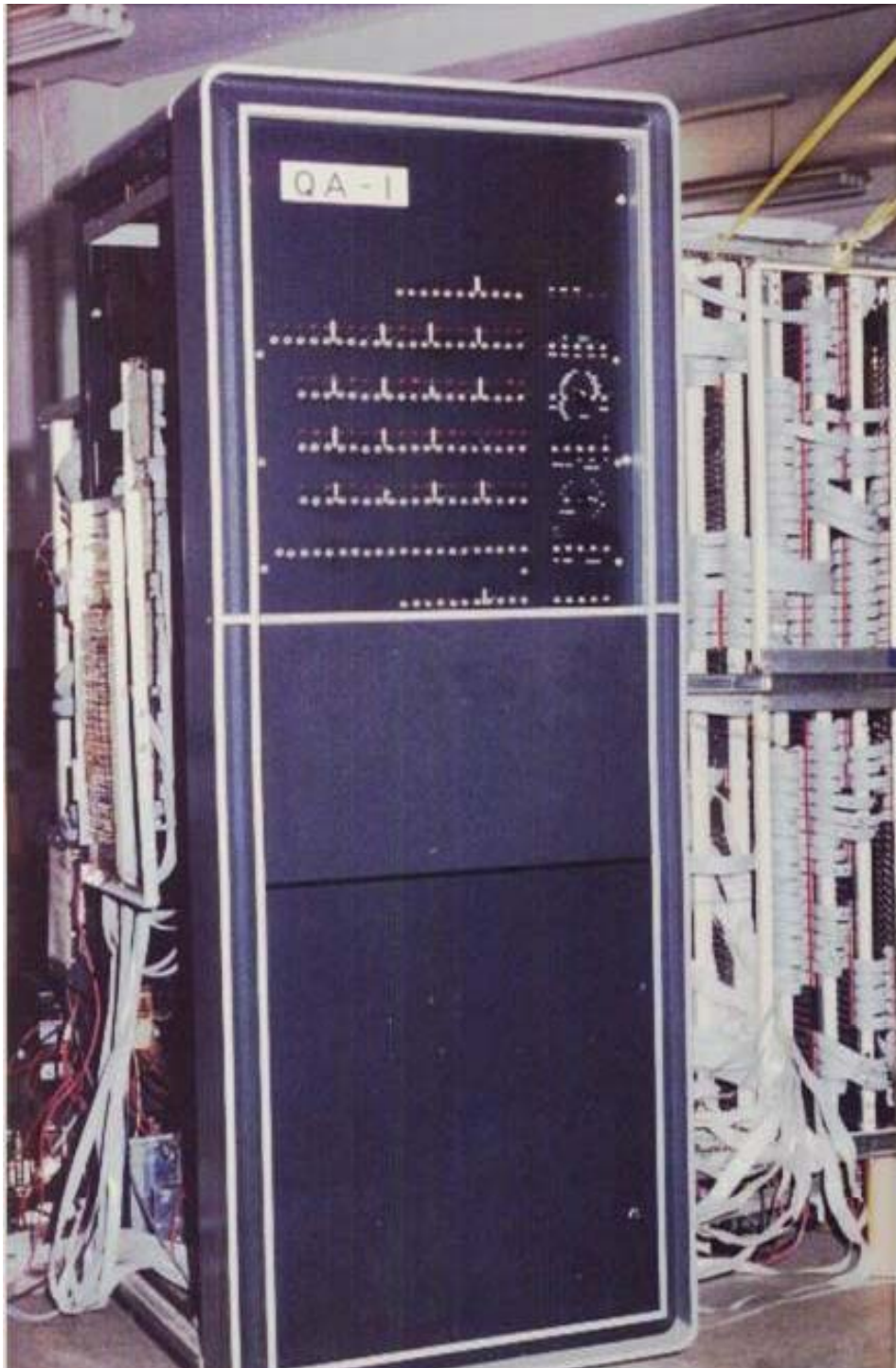
スーパースカラ/VLIW での命令と演算装置の対応

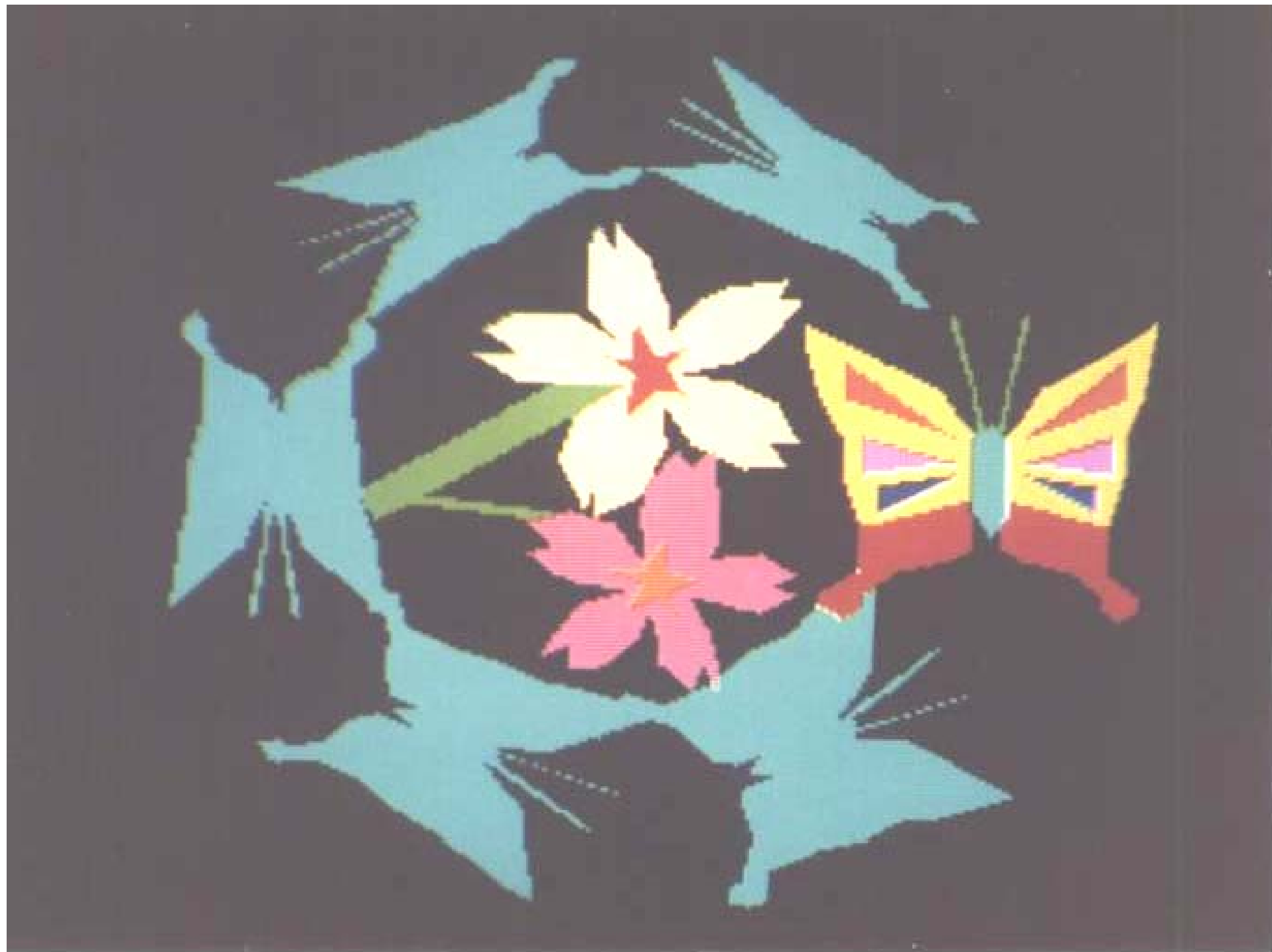


LS: 2サイクル

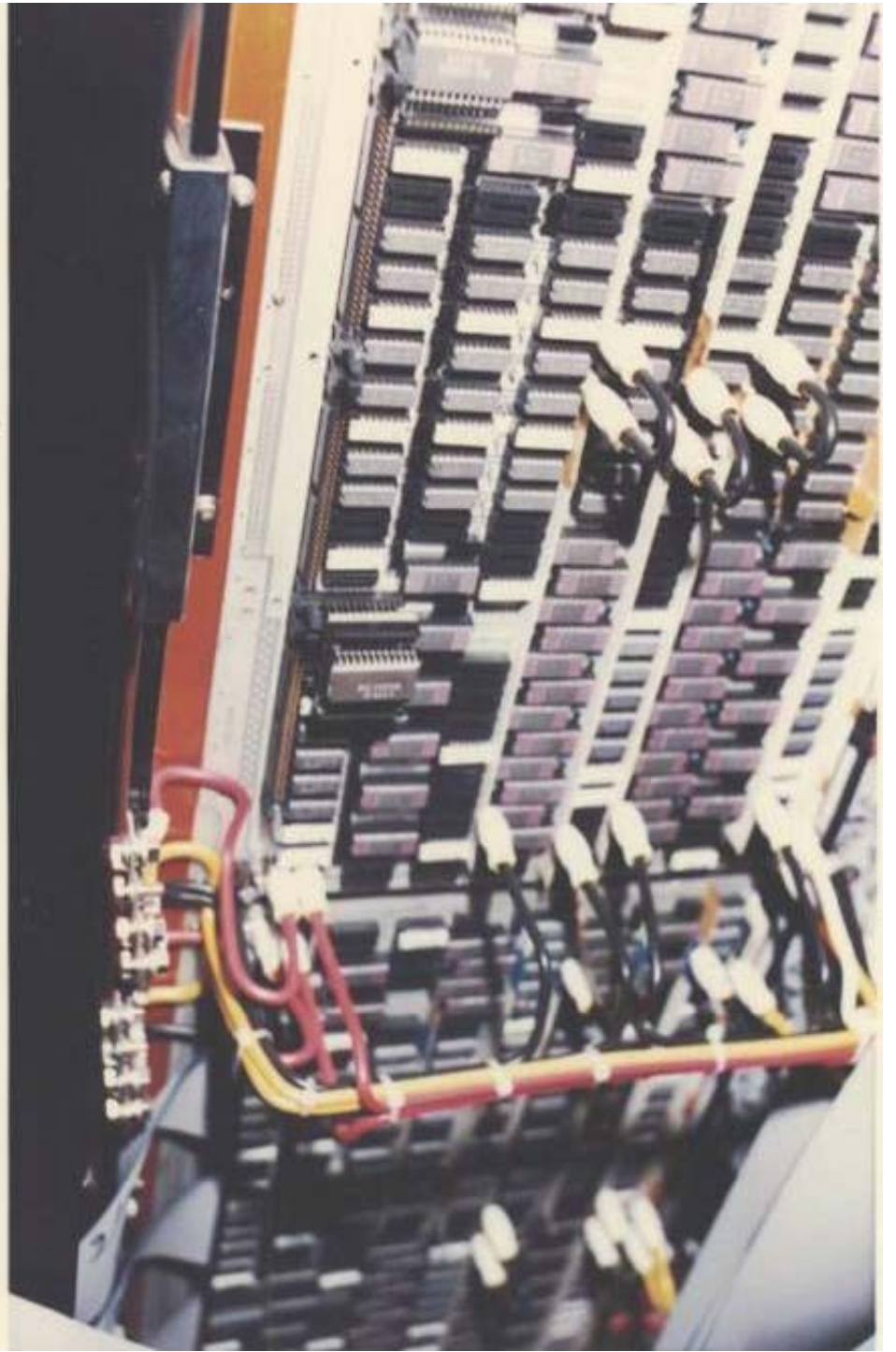
ADDF: 3サイクル

長命令間および内でのNOP









ソースプログラム

```
DO 10 I=1, N
```

```
Z(I)=X(I)+Y(I)           ( 1 9 )
```

```
W(I)=X(I)*Y(I)
```

```
10 CONTINUE
```

目的コード

```
L   LOAD  R1  M(R10+D1)
    LOAD  R2  M(R10+D2)
    ADDF   R3  R1  R2
    MULF   R4  R1  R2
    ADDI   R10 #11      (20)
    ADDI   R11 #11
    STORE  M(R10+D3) R3
    STORE  M(R10+D4) R4
    BRCN  M(PC-18)     Lに分岐
```

VLIW命令での実行

- ① LOAD IF D EX EX S
LOAD IF D EX EX S
- ② すべてNOP
- ③ ADDF IF D EX EX EX S
MULF IF D EX EX EX S
ADDI IF D EX S (2 1)
ADDI IF D EX S
- ④ すべてNOP
- ⑤ すべてNOP
- ⑥ STORE IF D EX EX S
STORE IF D EX EX S
BRCN IF D EX S
- ⑦ LOAD IF D EX EX S 遅延スロットの実行
LOAD IF D EX EX S (2回目の反復)
- ⑧ すべてNOP 遅延スロットの実行

これを先の乱発行乱終了スカラで行うと、

```
LOAD IF D EX EX S
LOAD IF D EX EX S
ADDF IF D - EX EX EX S
MULF IF D - EX EX EX S
ADDI IF D EX S
ADDI IF D EX S
STORE IF D EX EX S (2 2)
STORE IF D EX EX S
BRCN IF D EX S
2回目の反復
LOAD IF D EX EX S
LOAD IF D EX EX S
```

スーパースカラでは

LOAD IF D EX EX S

LOAD IF D EX EX S

ADDF IF D - EX EX EX S

MULF IF D - EX EX EX S

ADDI IF D EX S

ADDI IF D EX S (2 3)

STORE IF D - - EX EX S

STORE IF D - - EX EX S

BRCN IF D EX S

2回目の反復

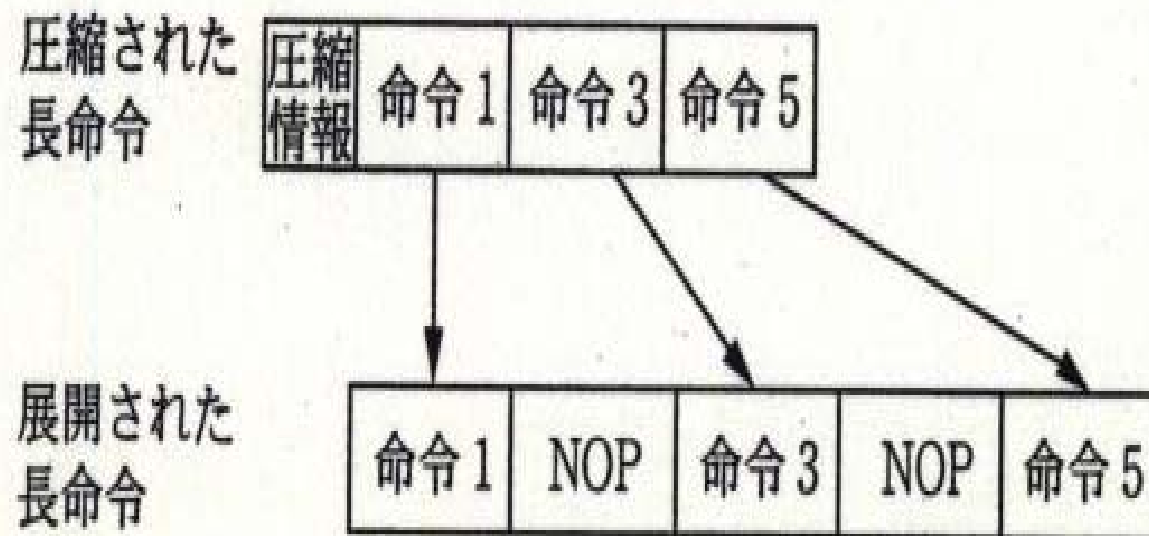
LOAD IF D EX EX S

LOAD IF D EX EX S

6.5.2 VLIW方式の改良

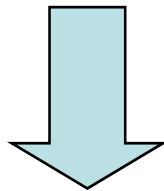
(1) 全NOP命令の削除

(2) 長命令の圧縮

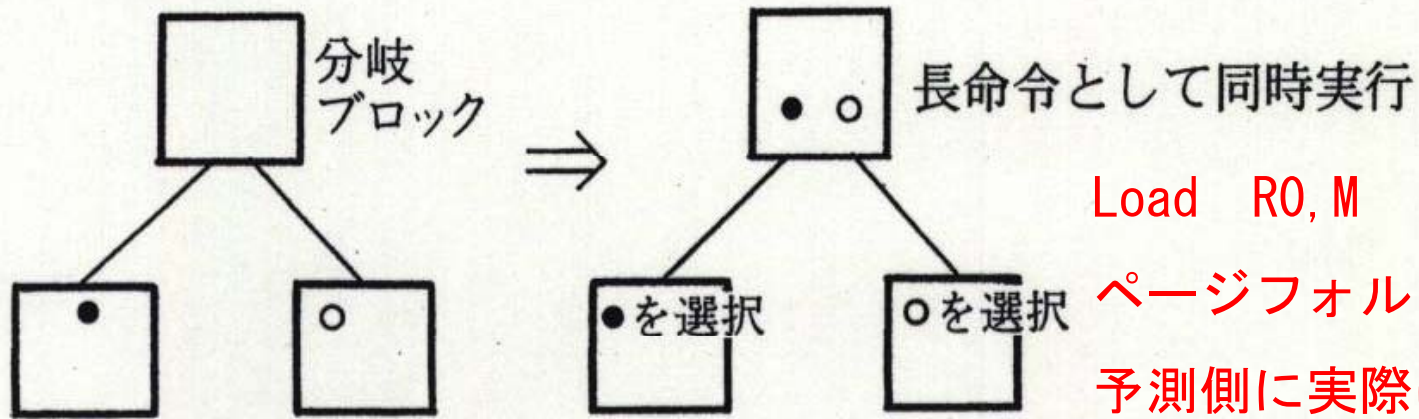


(3) 分岐命令の高速処理

分岐成立／不成立側の同時実行
ブースティングと遅延例外処理
プレディケート付き演算
ソフトウェアパイプライン支援



Itaniumのアーキテクチャ



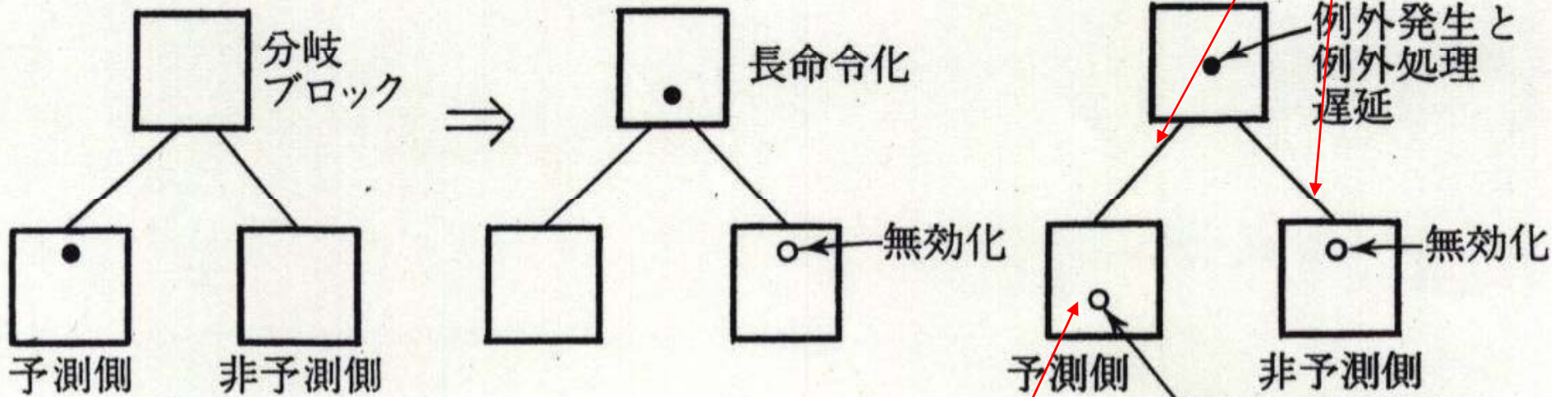
Load R0, M

ページフォルト

予測側に実際に分岐：仕方なし

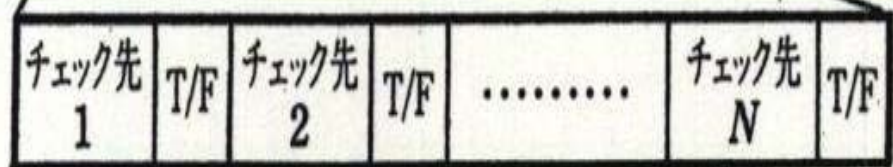
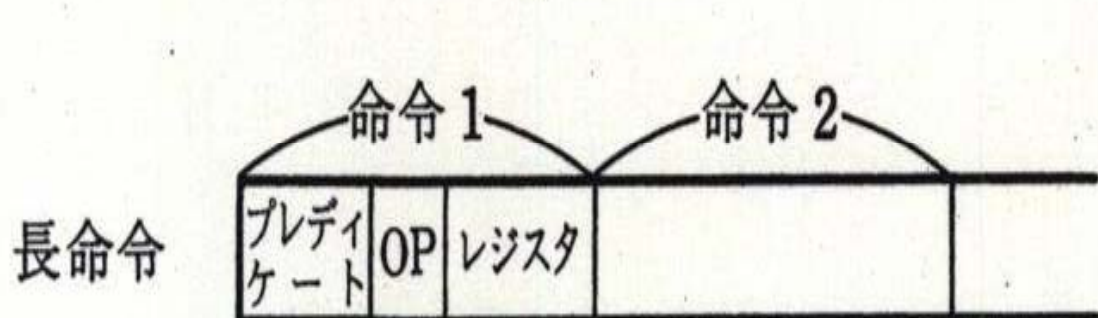
非予測側に分岐：被害甚大

(a) 条件成立側/不成立側同時実行

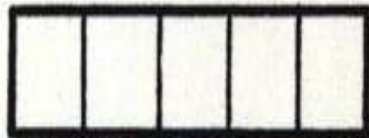


(b) ブースティングと遅延例外処理

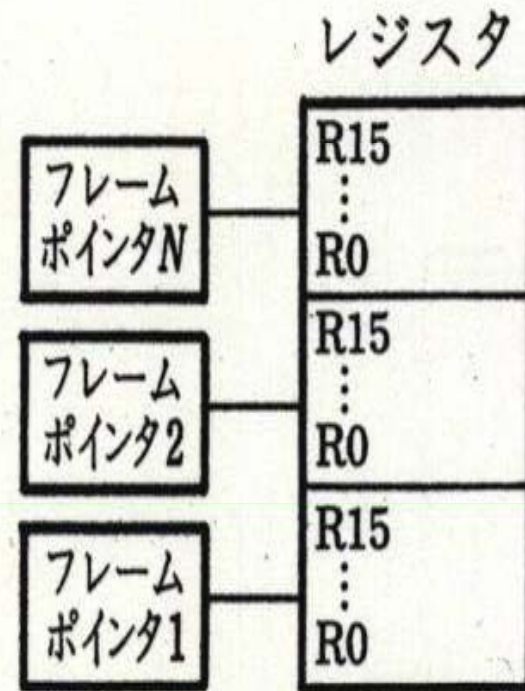
R0を利用するところでページフォルト発生



条件コードレジスタ



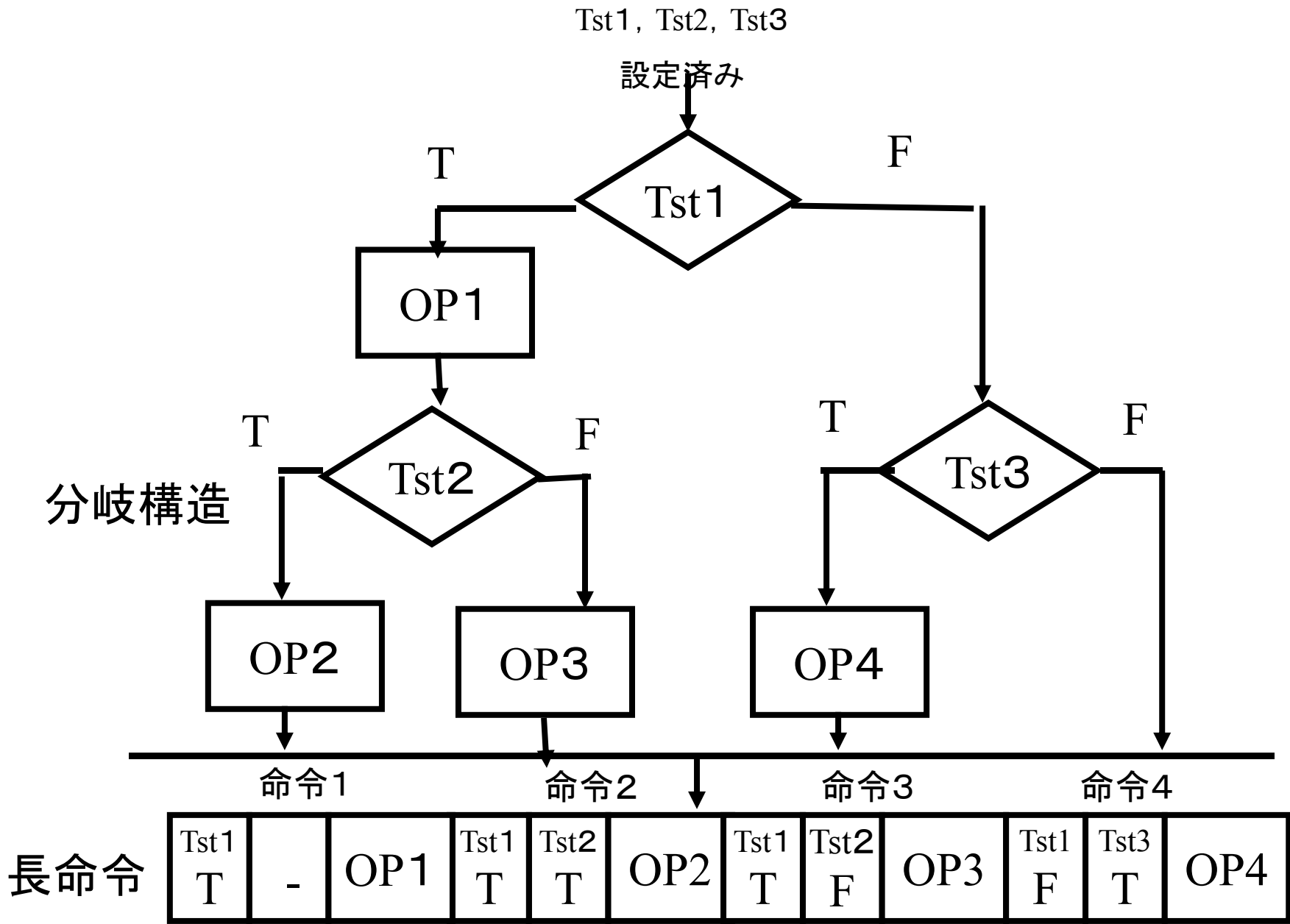
(c) プレディケート付き演算

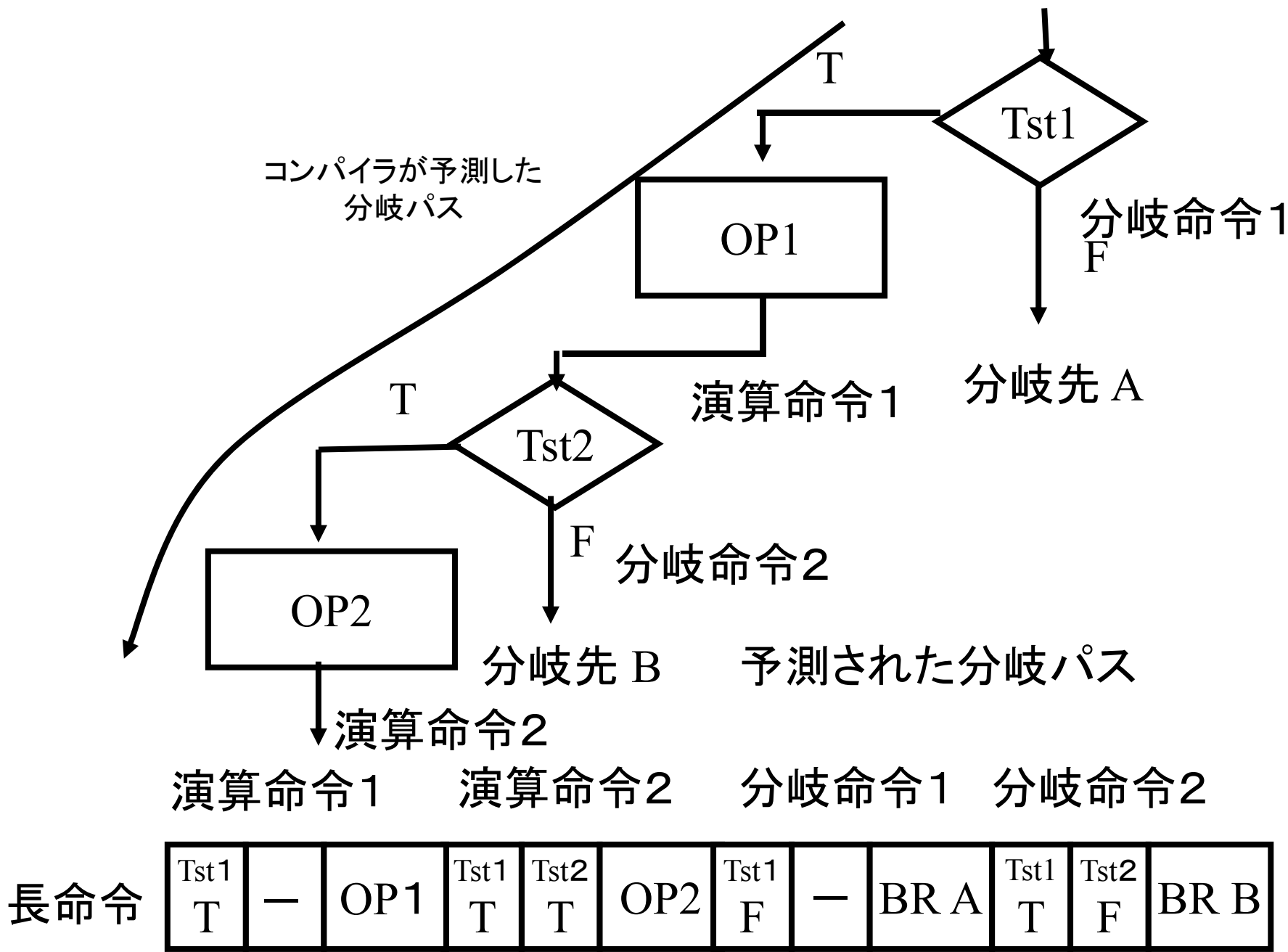


(d) フレームレジスタ

T : チェック先の論理値 1
 ならチェック結果 1
 F : チェック先の論理値 0
 ならチェック結果 1

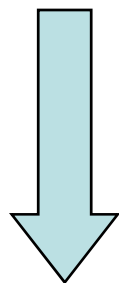
図 6.13 VLIW 方式の分岐高速化機構





VLIW方式における投機実行

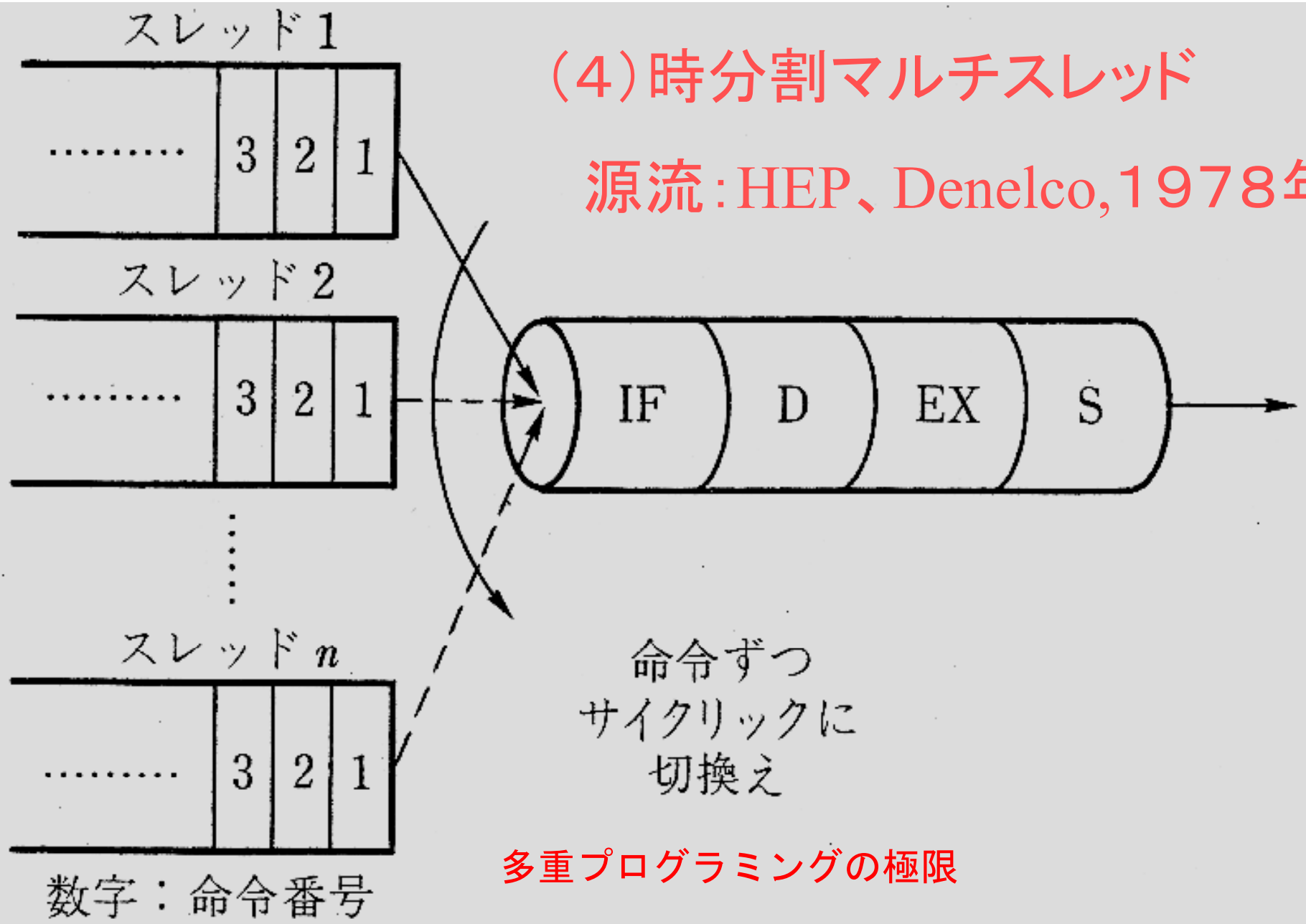
複雑な分岐構造を1つの長命令に埋め込める



多数の長命令のフェッチがなくなる
長命令のビット使用効率が向上する

(4) 時分割マルチスレッド

源流: HEP、Denelco, 1978年



数字: 命令番号

共有命令パイプライン方式

④	ADDF	IF	D	EX	EX	EX	S	DCN=0
	MULF	IF	D	EX	EX	EX	S	
	ADDI	IF	D	EX	S			DCN=2
	ADDI	IF	D	EX	S			
	スレッド 3							
⑤	LOAD		IF	D	EX	EX	S	DCN=1
	LOAD		IF	D	EX	EX	S	
	スレッド 1							
⑥	STORE		IF	D	EX	EX	S	DCN=2
	STORE		IF	D	EX	EX	S	
	BRCN		IF	D	EX	S		

スレッド 2

⑦

STORE

IF D EX EX S

DCN=2

STORE

IF D EX EX S

BRCN

IF D EX S

スレッド 1

2 回目の反復

⑧

LOAD

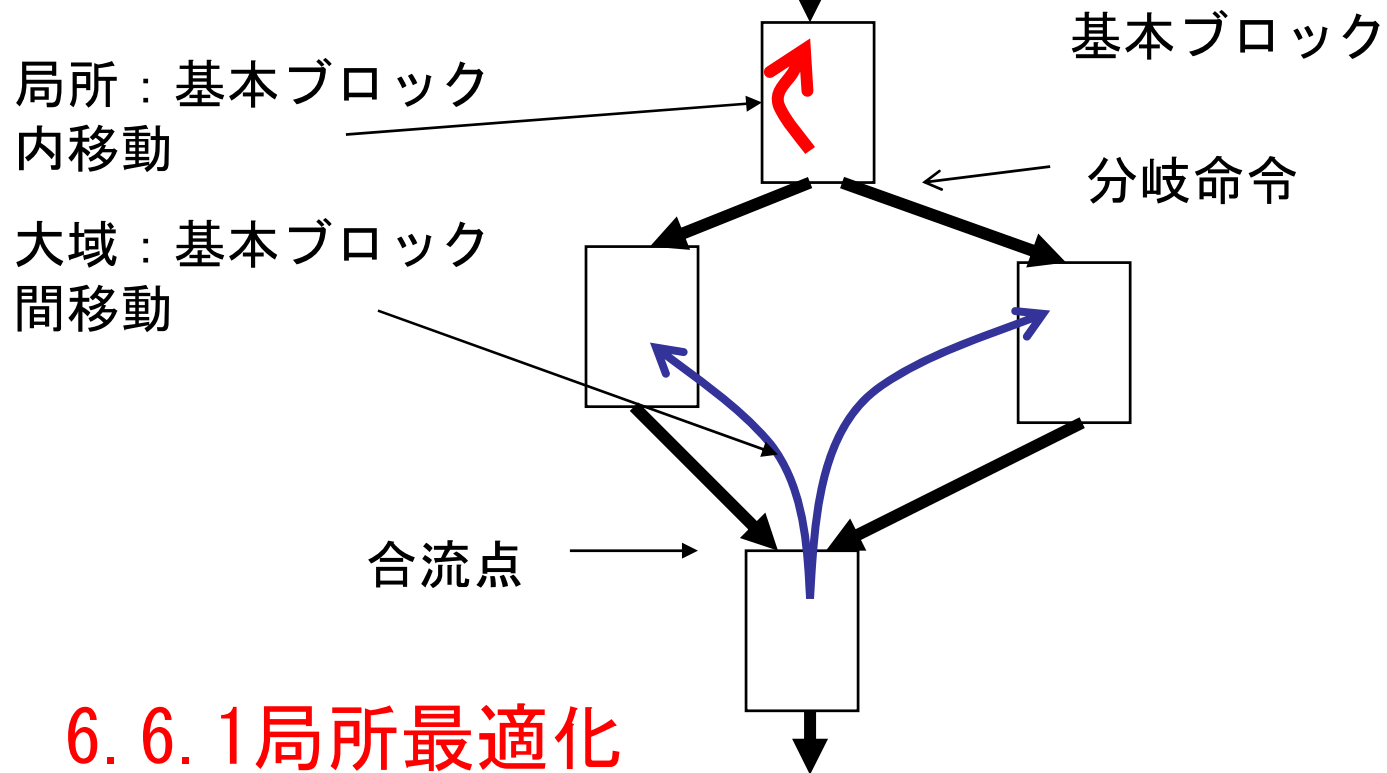
IF D EX EX S

DCN=1

LOAD

IF D EX EX S

6.6最適化コンパイラ



6.6.1局所最適化

リストスケジューリング

- ①基本ブロック内のデータ依存グラフの作成
- ②クリティカルパスの決定
- ③リストスケジューリング

命令	デスティネーションレジスタ	ソースレジスタ	実行時間
m_1	R3	R1, R2	1
m_2	R5	R3, R4	1
m_3	R6	R3	2
m_4	R7	R5, R6	1
m_5	R8	R7, R3	2
m_6	R10	R9, R3	1
m_7	R11	R9	1
m_8	R21	R10	1
m_9	R13	R12	1
m_{10}	R15	R10	2
m_{11}	R20	R13, R14	3
m_{12}	R17	R16, R11	1
m_{13}	R18	R17, R5	2
m_{14}	R19	R18	1

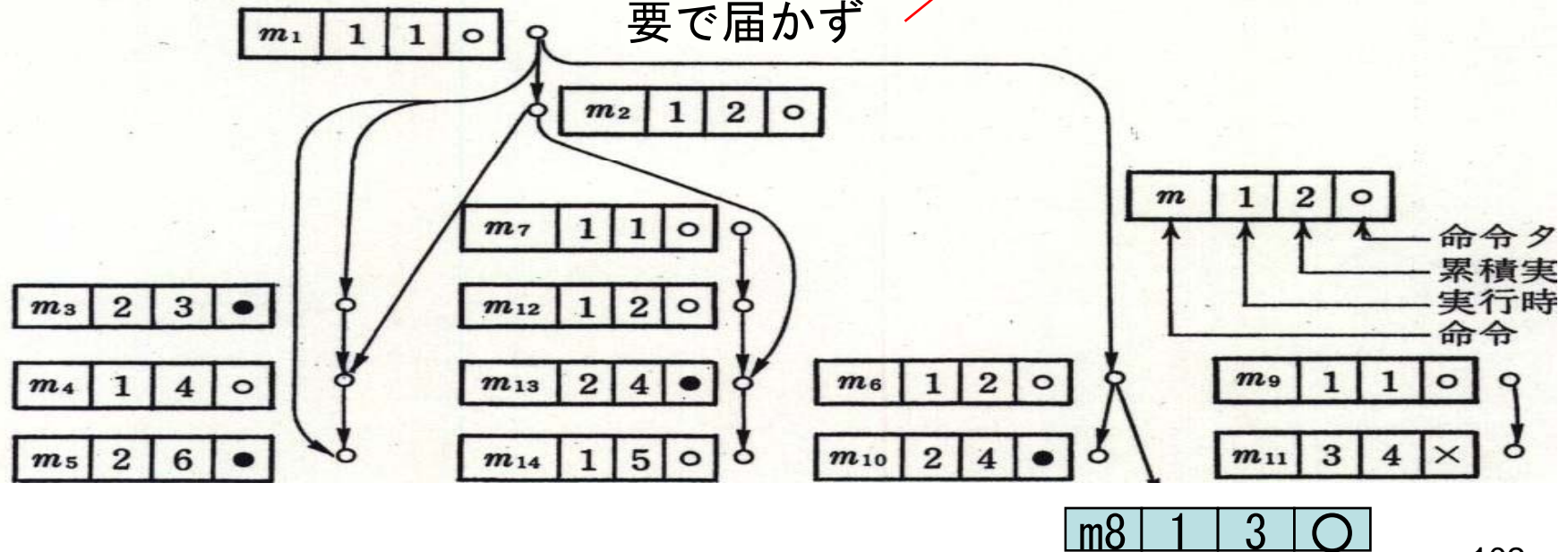
クリティカルパスの長い終端命令から割付

長命令			
●	○	○	×
m_5	m_{14}	m_8	m_{11}
m_{10}	m_4	m_9	
m_{13}	m_6		
m_3	m_{12}	m_2	
	m_1	m_7	

(a) プログラム例

(c) 生成された長命令

M13は2サイクル必要で届かず



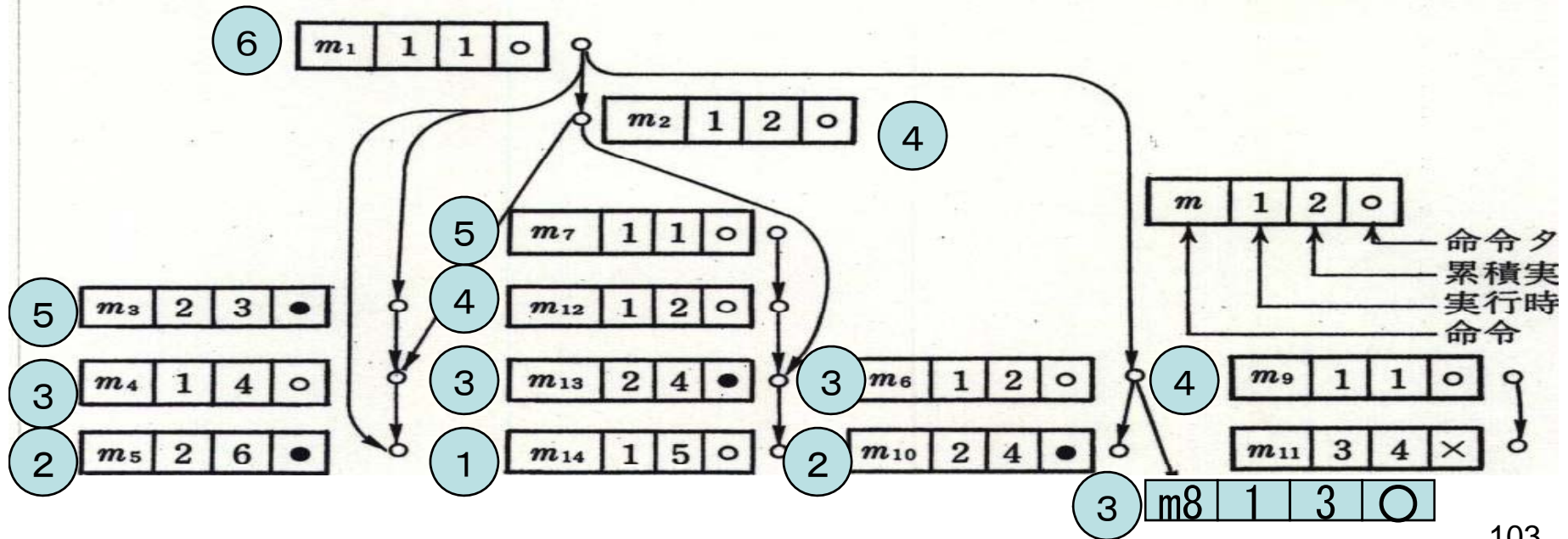
命令	デスティネーションレジスタ	ソースレジスタ	実行時間
m_1	R3	R1, R2	1
m_2	R5	R3, R4	1
m_3	R6	R3	2
m_4	R7	R5, R6	1
m_5	R8	R7, R3	2
m_6	R10	R9, R3	1
m_7	R11	R9	1
m_8	R21	R10	1
m_9	R13	R12	1
m_{10}	R15	R10	2
m_{11}	R20	R13, R14	3
m_{12}	R17	R16, R11	1
m_{13}	R18	R17, R5	2
m_{14}	R19	R18	1

クリティカルパスの長い緒端命令から割付

長命令			
タイムスロット	●	○	×
0		m_1	m_7
1	m_3	m_2	m_{12}
2	m_{13}	m_6	m_9
3	m_{10}	m_4	m_8 m_{11}
4	m_5	m_{14}	

(a) プログラム例

(c) 生成された長命令



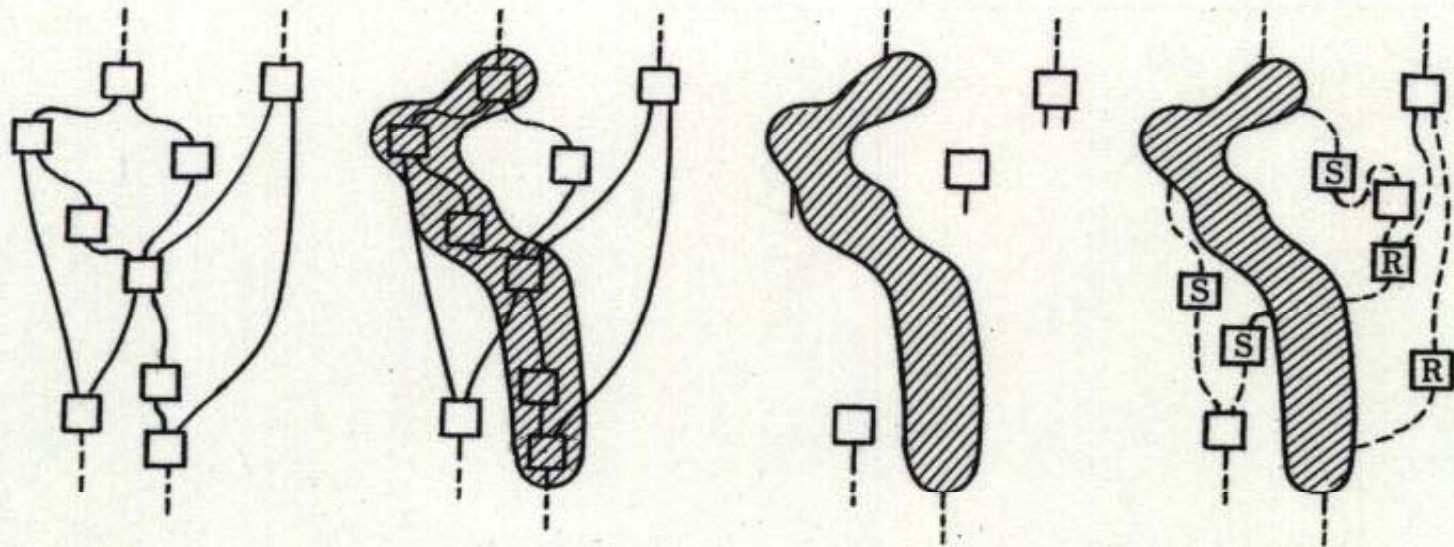
6.5.2 広域最適化

(1) トレーススケジューリング

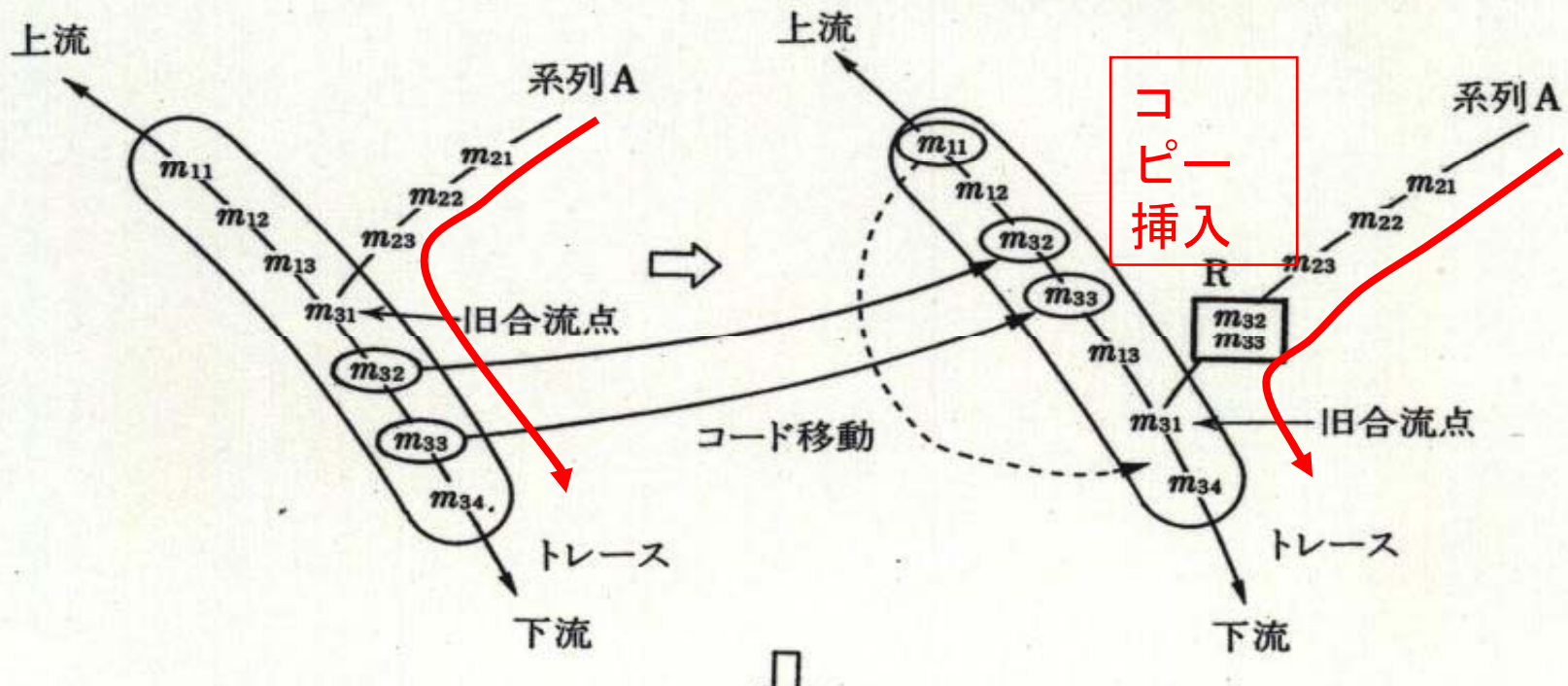
1981年 : J, Fisher

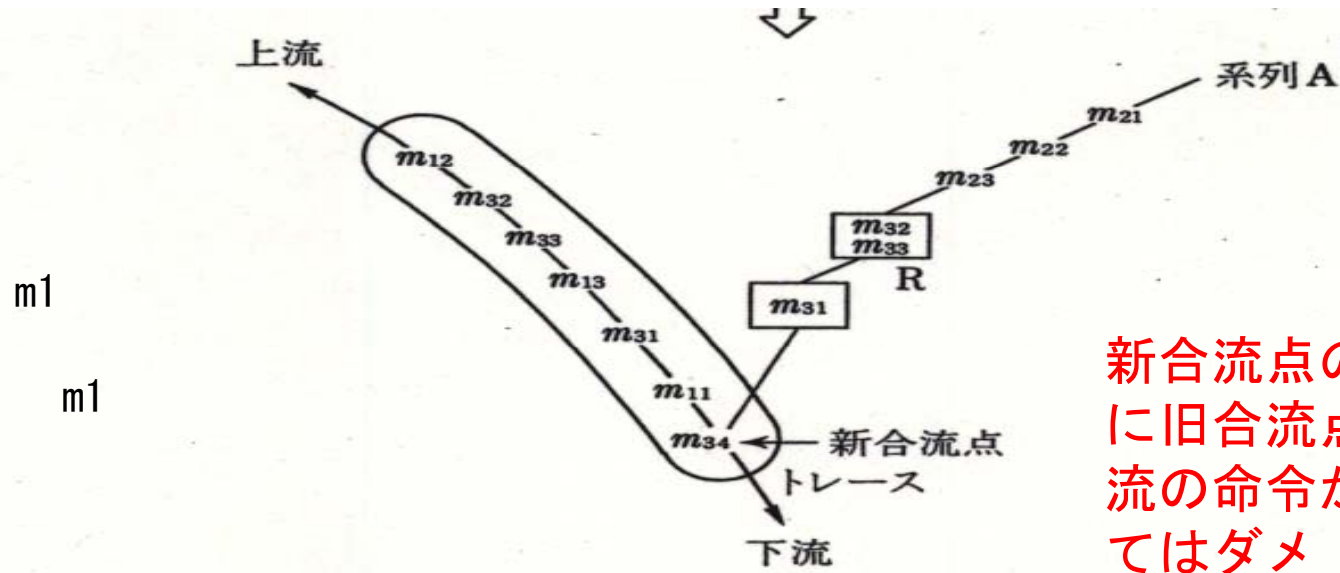
分岐に偏りのあるプログラムに適用

- ① トレースの選択
- ② リストスケジューリングでコード移動
- ③ 命令挿入
- ④ 次のトレース処理



(a) フローグラフ (b) トレースの決定 (c) トレース内最適化 (VLIW命令の合成) (d) トレース外に対する後処理

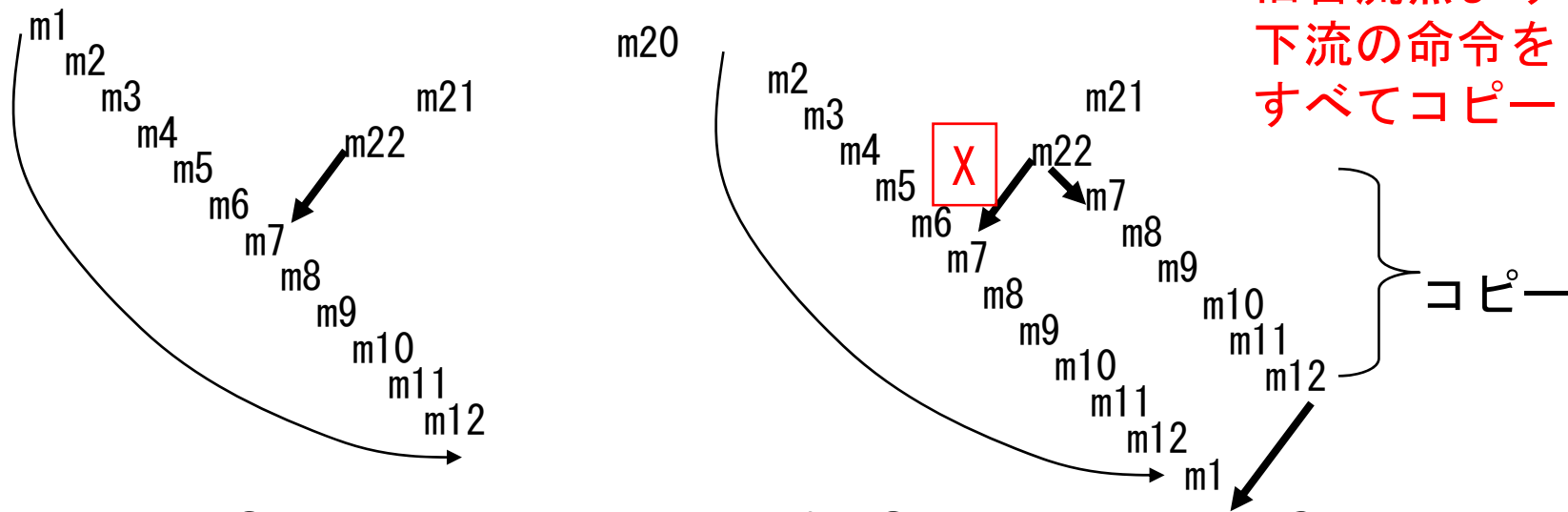




(e) 合流点での処理

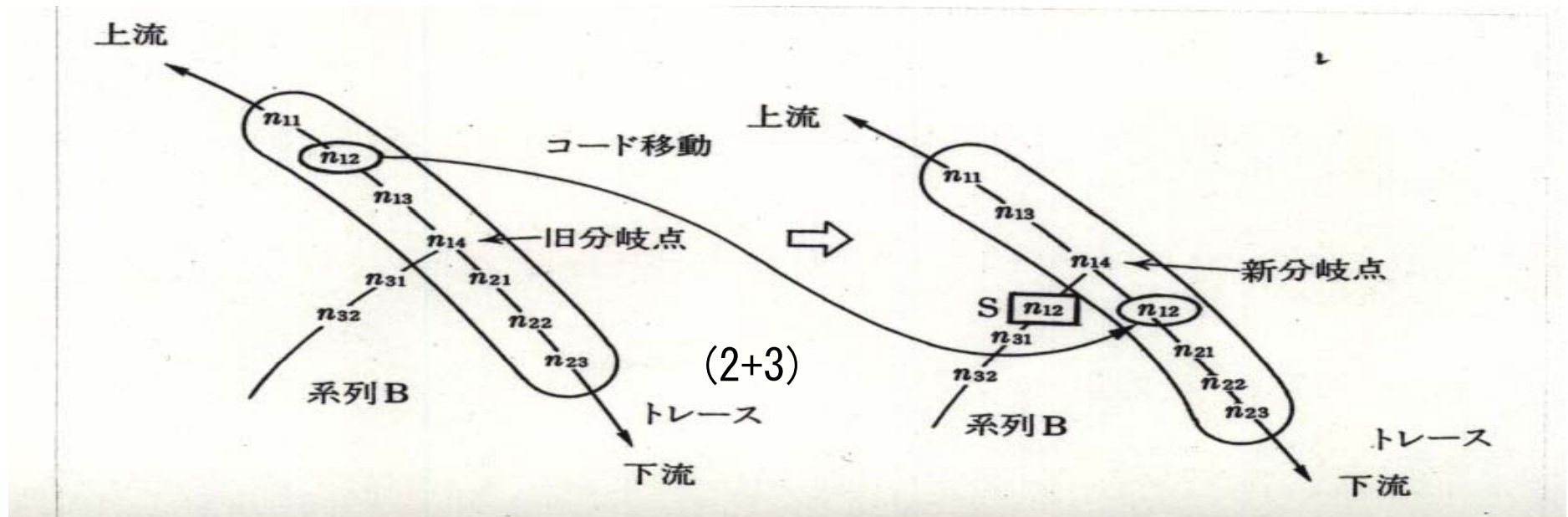
図 6.18 トレーススケジューリング

新合流点の下流
に旧合流点の上
流の命令があっ
てはダメ m11がトレ
ースの最後尾に
移動したら、
旧合流点より
下流の命令を
すべてコピー



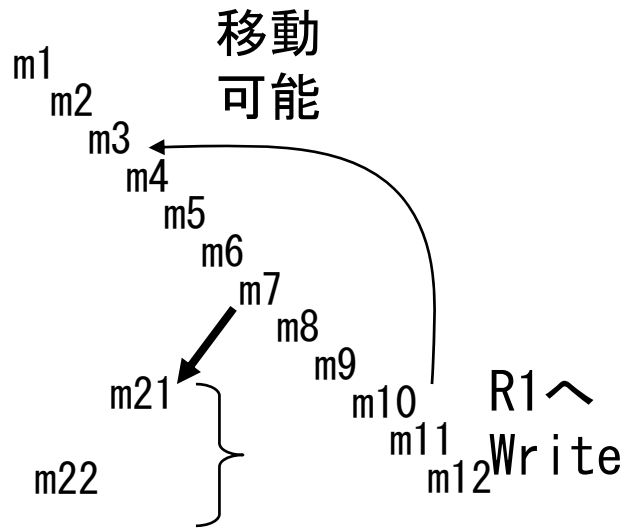
・J.Fisher:Trace Scheduling: A Technique for Global Microcode Compaction, IEEE Trans. Computers, C-30,7, pp.478-490(1981)

トレースの上流から分岐点を
を越えて移動の場合

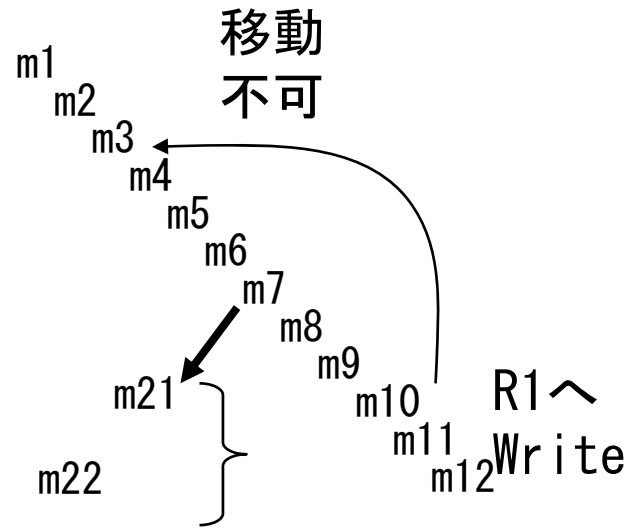


・J.Fisher:Trace Scheduling: A Technique for Global Microcode Compaction,
IEEE Trans. Computers, C-30,7, pp.478-490(1981)

分岐命令の下流から 上流への移動

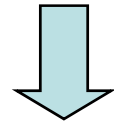


R1のリードなし (R1がDead)



R1のリードあり (R1がLive)

トレース上で
m7からm10への依存をつける



m10がm7より上流に移動するのを阻止する

基本演算 操作番号	ブロック 番号	デスティ ネーション レジスタ	ソース レジスタ	分岐先
エントリ点 → m_1	B ₁ :	R3	R1, R2	
m_2		R5	R3, R4	
m_3		R6	R3	
m_4		R7	R5, R6	
m_5		R8	R7, R3	m_6
m_6	B ₂ :	R10	R9, R3	
m_7		R11	R9	
m_8			R10	m_9, m_{17}
m_9	B ₃ :	R13	R12	
m_{10}		R15	R10	
m_{11}		R16	R13, R14	
m_{12}		R17	R16, R11	
m_{13}		R18	R17, R5	
m_{14}		R19	R18	Exit
エントリ点 → m_{15}	B ₄ :	R5	R3, R4	
m_{16}		R8	R7, R3	m_6
m_{17}	B ₅ :	R12	R13	

R20の
誤り



(a) プログラム例

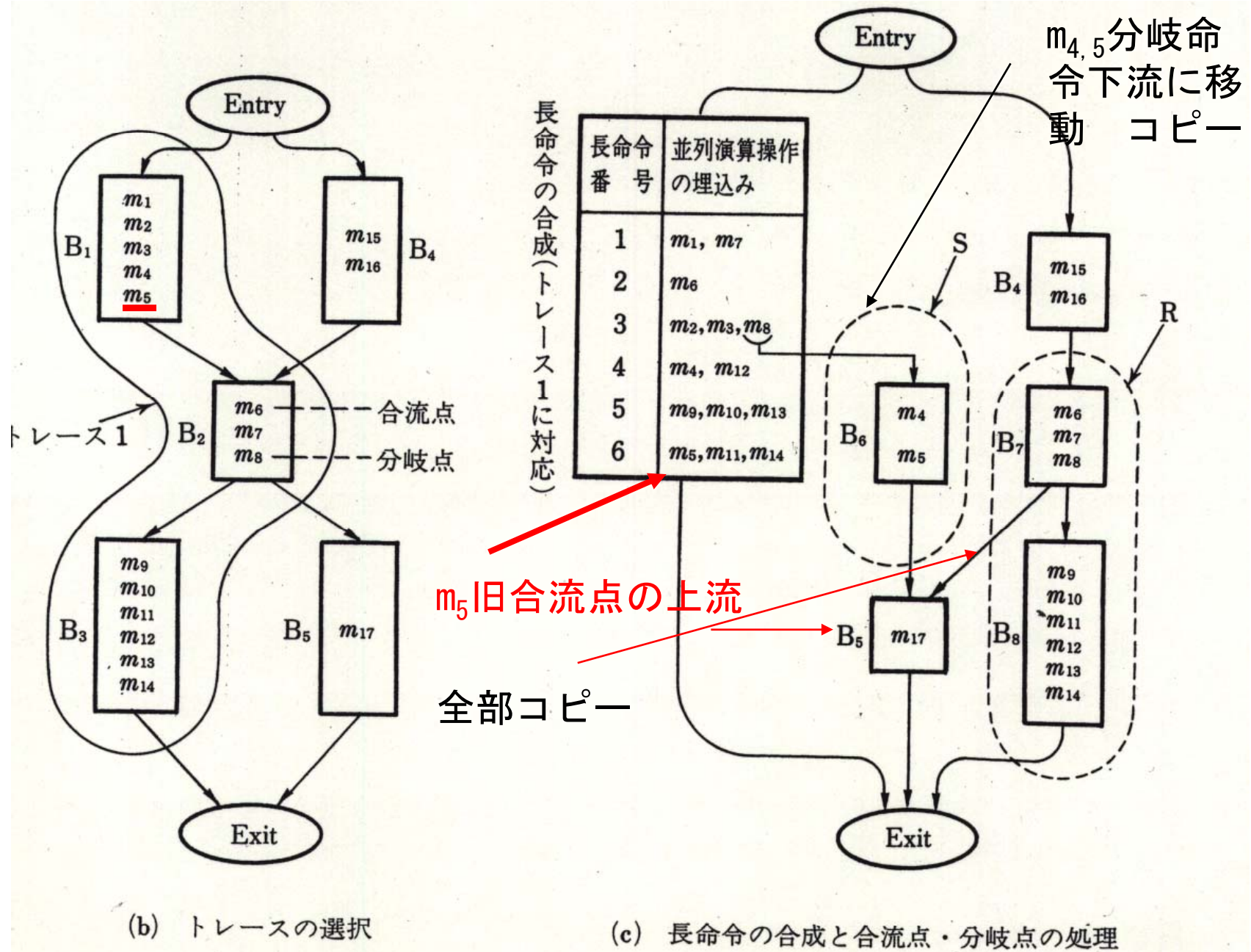
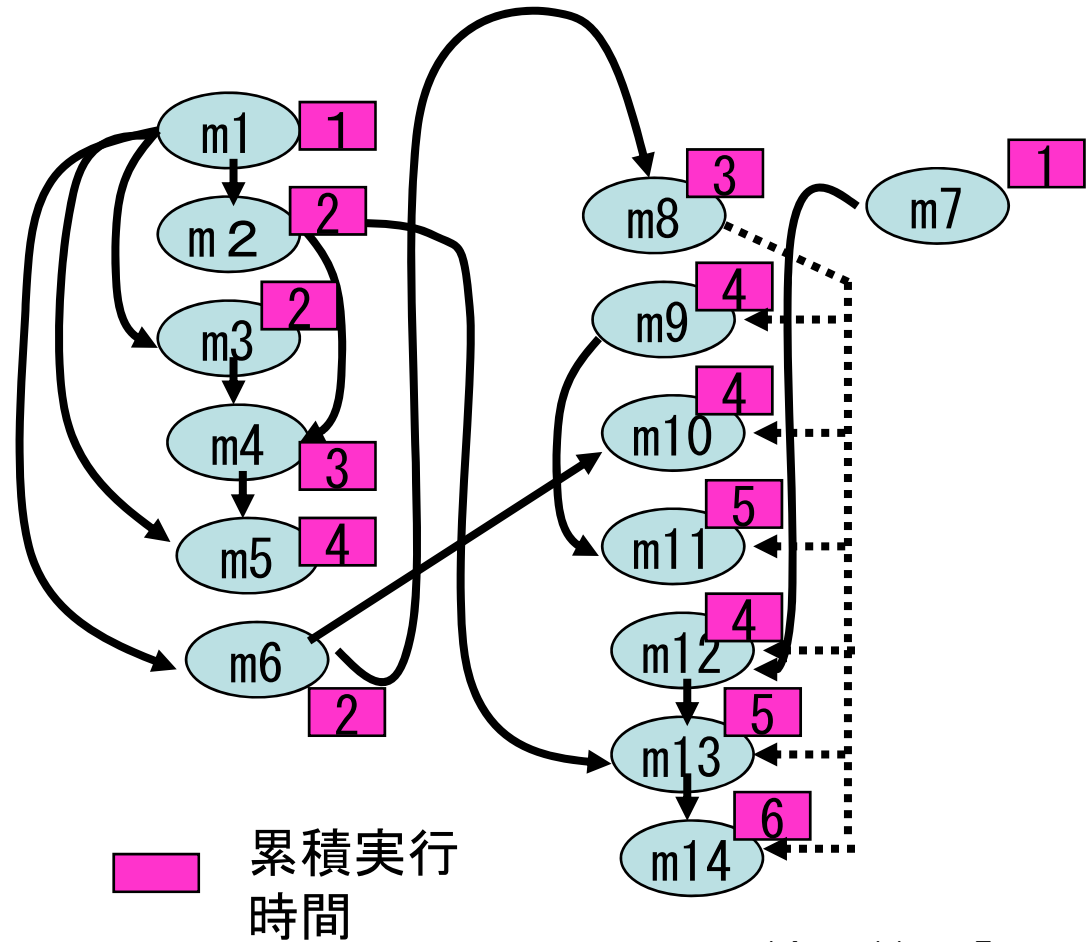


図 6.19 トレーススケジューリングの適用例³⁰⁾

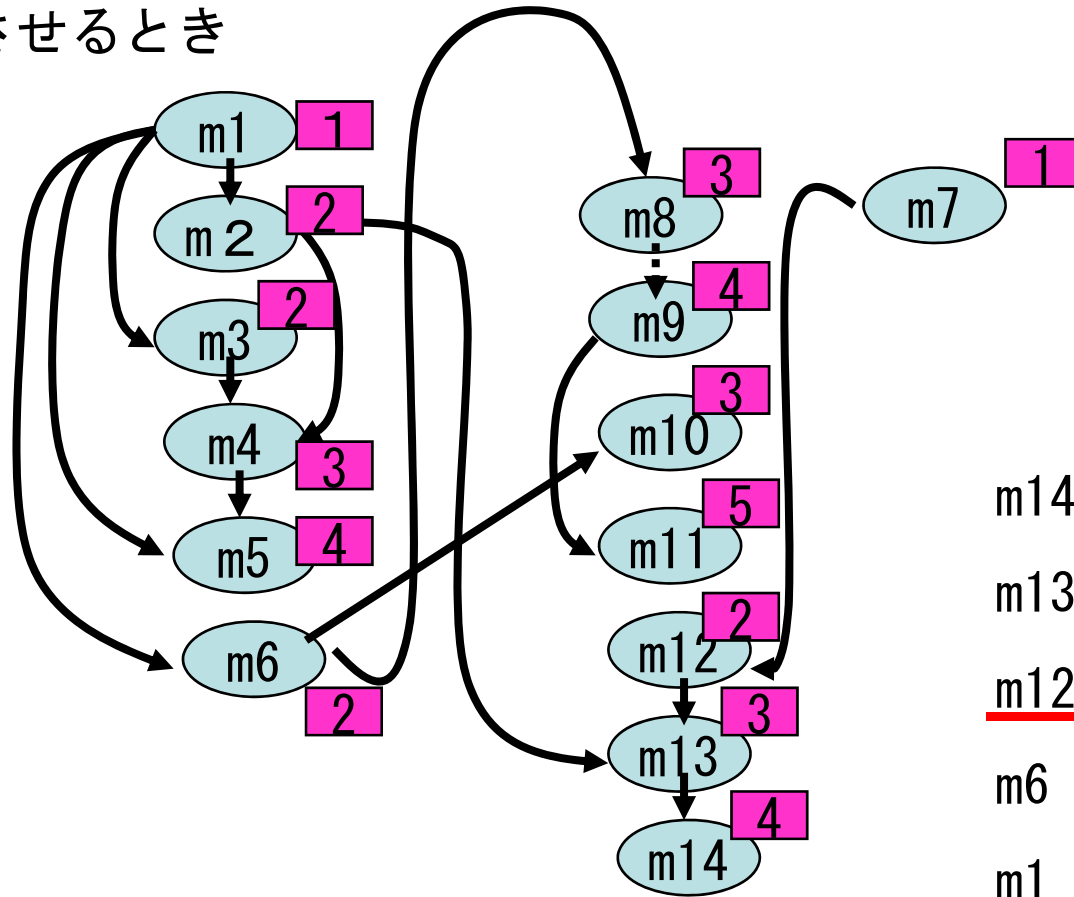
分岐命令の下流から上流
に移動をさせないとき

基本演算 操作番号	ブロック 番号	デスティ ネーション レジスタ	ソース レジスタ
→ m ₁	B ₁ :	R3	R1, R2
m ₂		R5	R3, R4
m ₃		R6	R3
m ₄		R7	R5, R6
m ₅		R8	R7, R3
m ₆	B ₂ :	R10	R9, R3
m ₇		R11	R9
m ₈			R10
m ₉	B ₃ :	R13	R12
m ₁₀		R15	R10
m ₁₁		R16	R13, R14
m ₁₂		R17	R16, R11
m ₁₃		R18	R17, R5
m ₁₄		R19	R18
→ m ₁₅	B ₄ :	R5	R3, R4
m ₁₆		R8	R7, R3
m ₁₇	B ₅ :	R12	R13



m14 m11 m5
 m13 m9 m10
 m12 m4
 m8 m3 m2
 m6
 m1 m7 111

分岐命令の下流から上流
に移動をさせるとき



m14	m11	m5
m13	m9	m10
<u>m12</u>	m4	<u>m8</u>
m6	m3	m2
m1	m7	

m12とm8が同時実行

基本演算 操作番号	ブロック 番号	デスティ ネーション レジスタ	ソース レジスタ	分岐先
エントリ点 → m_1	B ₁ :	R3	R1, R2	
m_2		R5	R3, R4	
m_3		R6	R3	
m_4		R7	R5, R6	
m_5		R8	R7, R3	m_6
m_6	B ₂ :	R10	R9, R3	
m_7		R11	R9	
m_8			R10	m_9, m_{17}
m_9	B ₃ :	<u>R13</u>	R12	
m_{10}		R15	R10	
m_{11}		R16	R13, R14	
m_{12}		R17	R16, R11	
m_{13}		R18	R17, R5	
m_{14}		R19	R18	Exit
エントリ点 → m_{15}	B ₄ :	R5	R3, R4	
m_{16}		R8	R7, R3	m_6
m_{17}	B ₅ :	R12	<u>R13</u>	

R20の
誤り



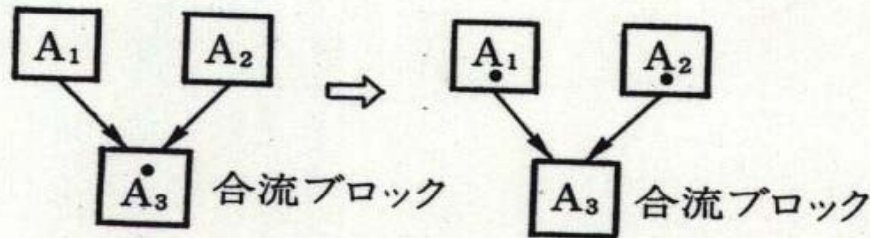
(a) プログラム例

(2) パーコレーションスケジューリング

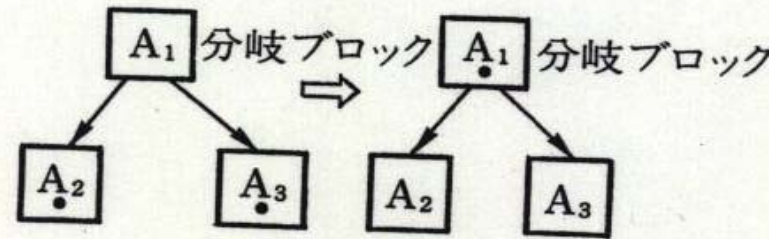
命令をどんどん前方のブロックへ：漏斗

①一般命令移動, ②分岐命令移動

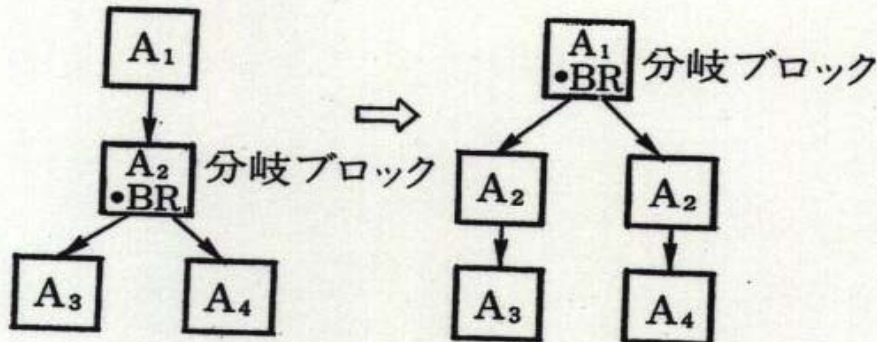
③単一化, ④削除



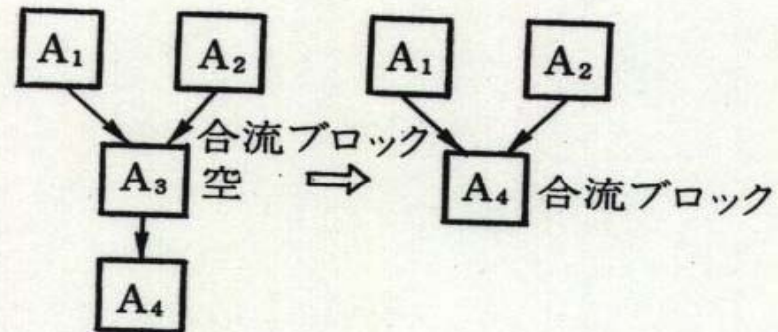
(a) 一般命令移動



(c) 単一化



(b) 分岐移動



(d) 削除

図 6.20 パーコレーションスケジューリングの基本移動

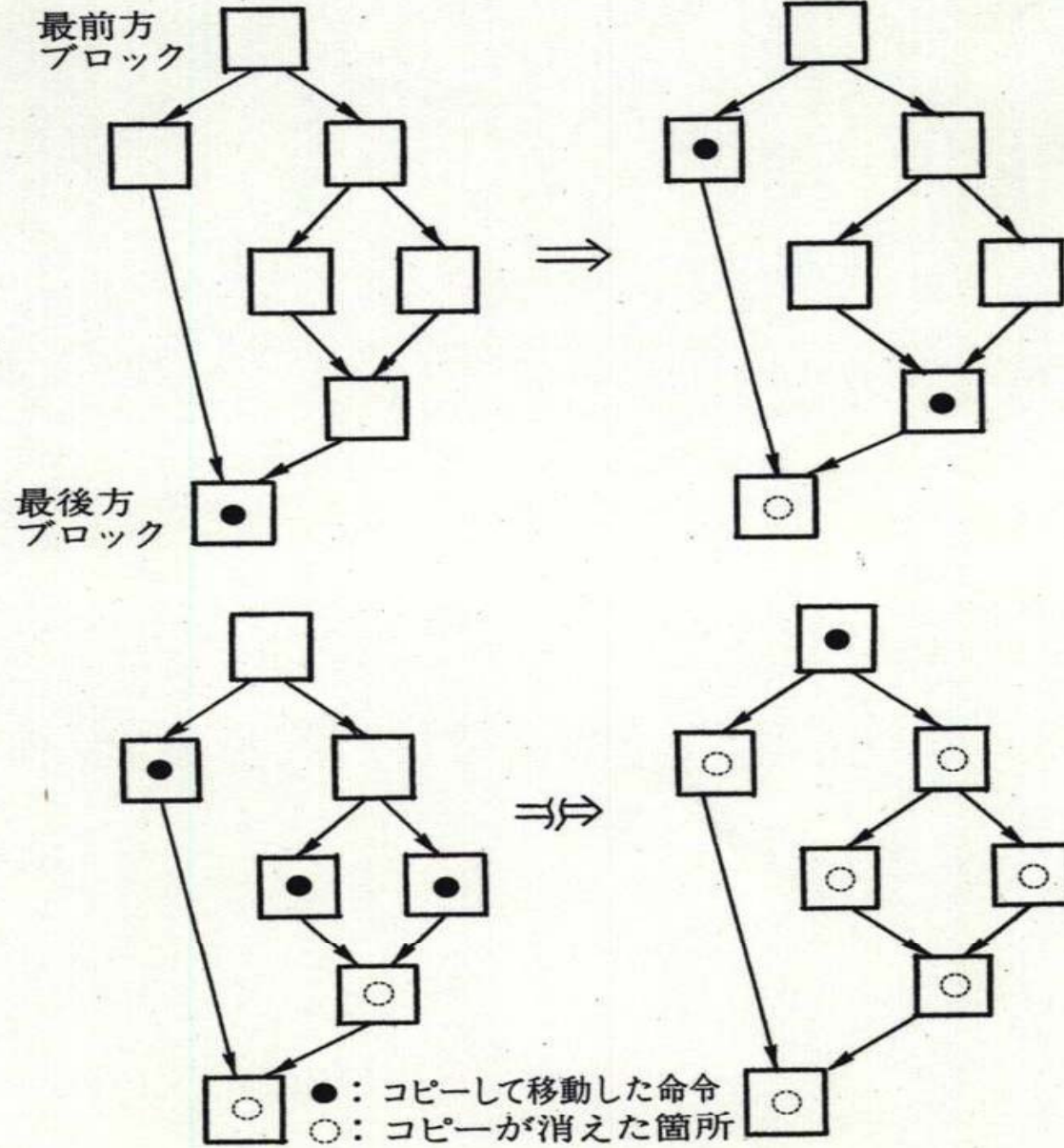
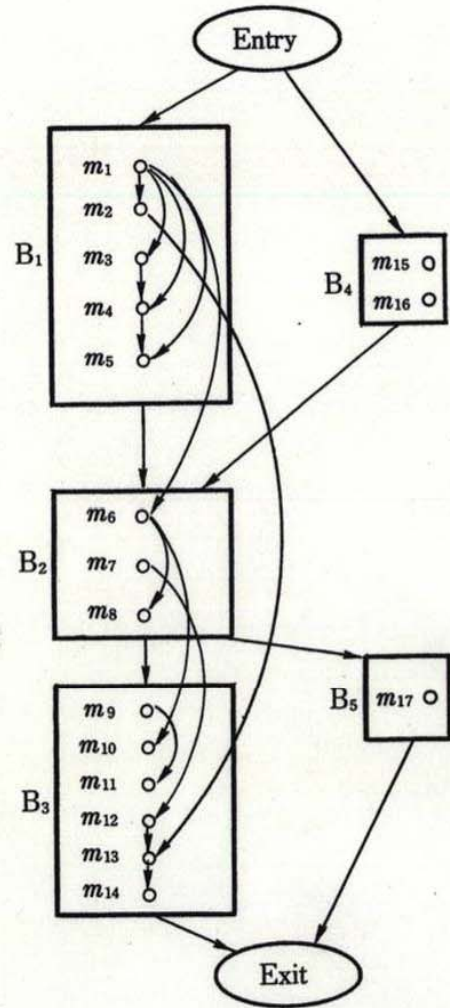
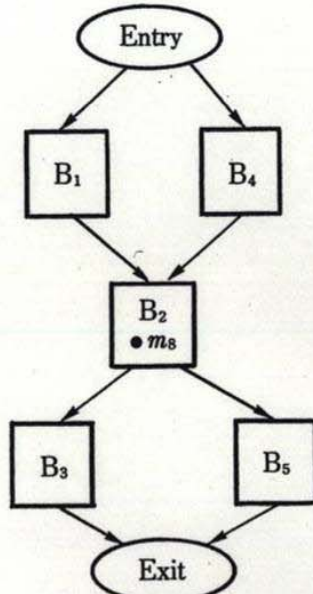


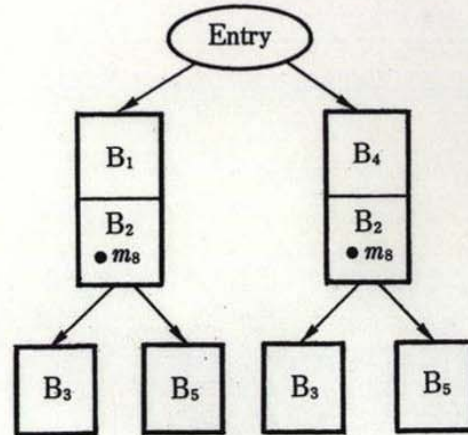
図 6.21 単一化移動の効果



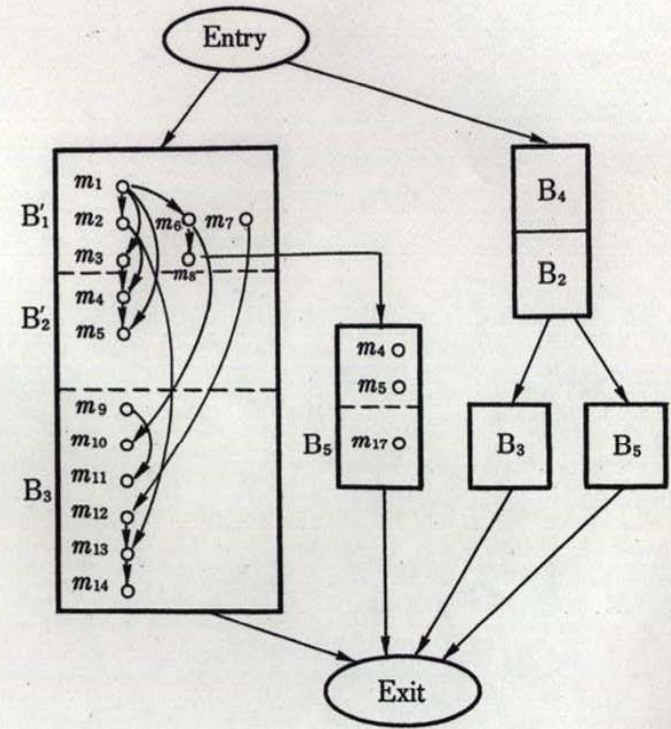
(a) 依存グラフ



(b) 初期依存グラフ



(c) 移動後依存グラフ



(d) 長命令の合成

図 6.22 パーコレーションスケジューリングの例

(3) ループアンローリング

ソースプログラム

```
DO 10 I=1, N  
  Z(I)=X(I)+Y(I)          ( 1 9 )  
  W(I)=X(I)*Y(I)  
10 CONTINUE
```

目的コード

```
L   LOAD  R1  M(R10+D1)
      LOAD  R2  M(R10+D2)
      ADDF  R3  R1  R2
      MULF  R4  R1  R2
      ADDI  R10 #11          (20)
      ADDI  R11 #11
      STORE M(R10+D3) R3
      STORE M(R10+D4) R4
      BRCN  M(PC-18)       Lに分岐
```

VLIW命令での単純実行：NOP多し

- ① LOAD IF D EX EX S
LOAD IF D EX EX S
- ② すべてNOP
- ③ ADDF IF D EX EX EX S
MULF IF D EX EX EX S
ADDI IF D EX S (2 1)
ADDI IF D EX S
- ④ すべてNOP
- ⑤ すべてNOP
- ⑥ STORE IF D EX EX S
STORE IF D EX EX S
BRCN IF D EX S
- ⑦ LOAD IF D EX EX S 遅延スロットの実行
LOAD IF D EX EX S (2回目の反復)
- ⑧ すべてNOP 遅延スロットの実行

ループアンローリング：1つの反復内に旧反復を2つ入れ、反復回数を半分にする

例 (19) のプログラム。アンローリングを1回

DO 10 I=1, N, 2

I=1, 3, 5, ...,

Z(I)=X(I)+Y(I)

Z(I+1)=X(I+1)+Y(I+1) (25)

W(I)=X(I)*Y(I)

W(I+1)=X(I+1)*Y(I+1)

10 CONTINUE

- ① LOAD IF D EX EX S
LOAD IF D EX EX S
- ② LOAD IF D EX EX S
LOAD IF D EX EX S
- ③ ADDF IF D EX EX EX S
MULF IF D EX EX EX S
- ④ ADDF IF D EX EX EX S
MULF IF D EX EX EX S (26)
ADDI IF D EX S
ADDI IF D EX S
- ⑤ すべてNOP
- ⑥ STORE IF D EX EX S
STORE IF D EX EX S

⑦	STORE	IF	D	EX	EX	S
	STORE	IF	D	EX	EX	S
	BRCN	IF	D	EX	S	

遅延スロットの実行

(2回目の反復)

⑧	LOAD	IF	D	EX	EX	S
	LOAD	IF	D	EX	EX	S

アンローリングを2回

DO 10 I=1, N, 3

Z(I)=X(I)+Y(I)

Z(I+1)=X(I+1)+Y(I+1)

Z(I+2)=X(I+2)+Y(I+2) (2 7)

W(I)=X(I)*Y(I)

W(I+1)=X(I+1)*Y(I+1)

W(I+2)=X(I+2)*Y(I+2)

10 CONTINUE

①	LOAD	IF	D	EX	EX	S						
	LOAD	IF	D	EX	EX	S						
②	LOAD		IF	D	EX	EX	S					
	LOAD		IF	D	EX	EX	S					
③	LOAD			IF	D	EX	EX	S				
	LOAD			IF	D	EX	EX	S				
④	ADDF				IF	D	EX	EX	EX	S		
	MULF				IF	D	EX	EX	EX	S		
⑤	ADDF					IF	D	EX	EX	EX	S	
	MULF					IF	D	EX	EX	EX	S	
⑥	ADDF						IF	D	EX	EX	EX	S
	MULF						IF	D	EX	EX	EX	S
	ADDI							IF	D	EX	S	

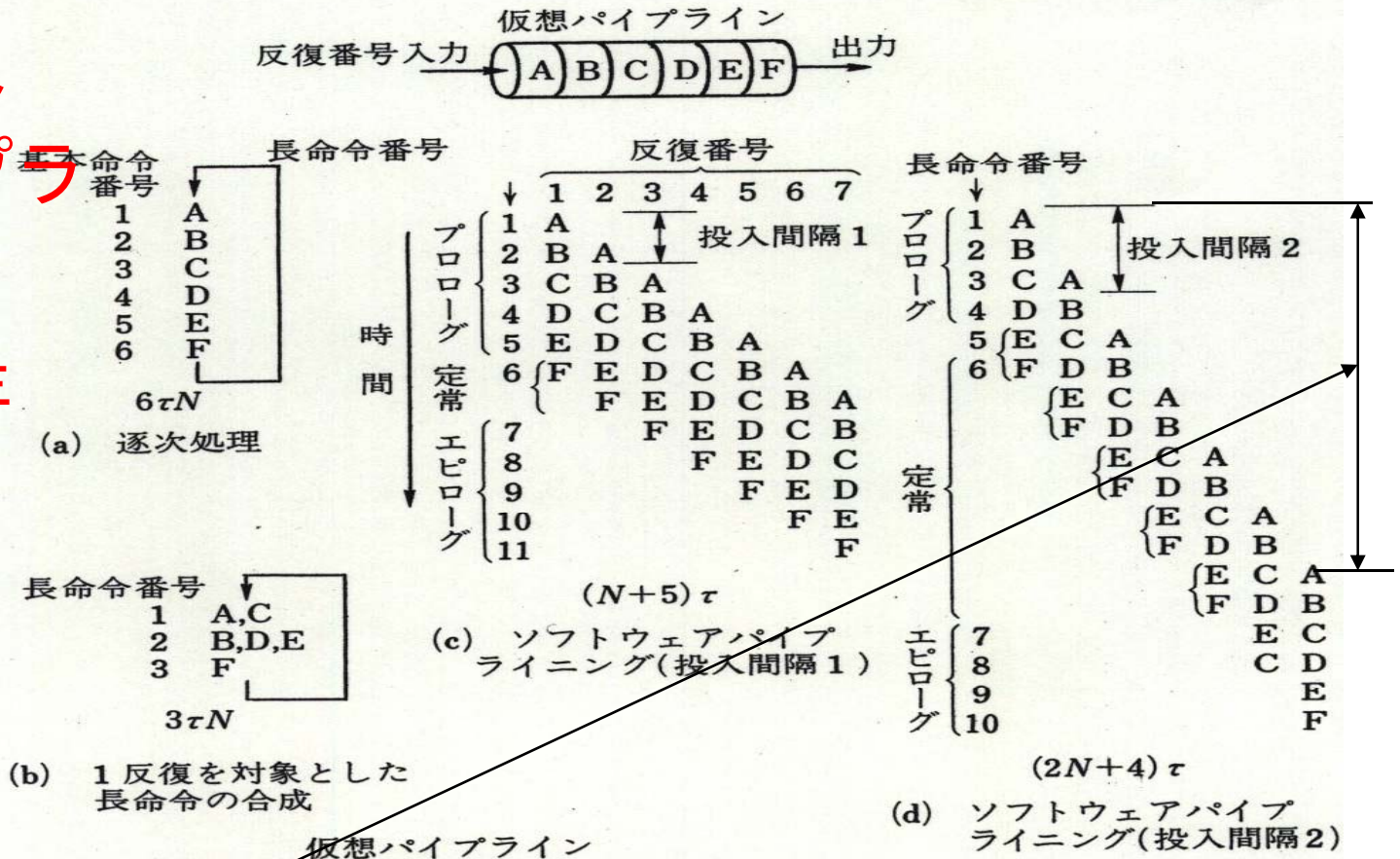
	ADDI	IF D	EX	S	(28)
⑦	STORE	IF D	EX	EX	S
	STORE	IF D	EX	EX	S
⑧	STORE		IF D	EX	EX S
	STORE		IF D	EX	EX S
⑨	STORE			IF D	EX EX S
	STORE			IF D	EX EX S
	BRCN			IF D	EX S

遅延スロットの実行
(2回目の反復)

⑩	LOAD			IF D	EX	EX	S
	LOAD			IF D	EX	EX	S

(4) ソフトウェアパイプライン

M. Lam: 1988年

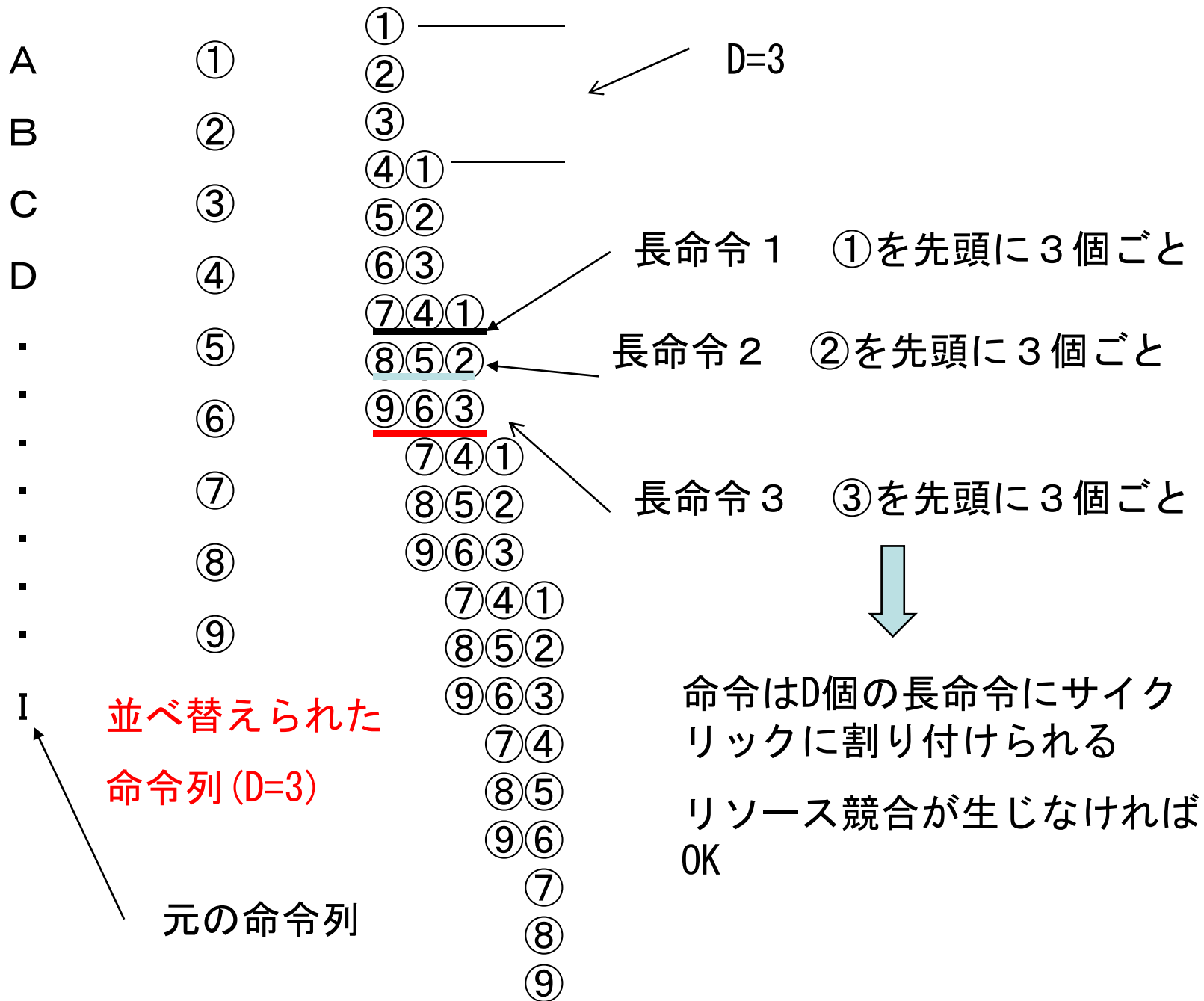


投入間隔D

実行時間:

$$D * (N-1) + \text{命令数}$$

• M.Lam: Software Pipelining: An Effective Scheduling Technique for VLIW Machines, Proc of Conf on Programming Language Design and Implementation, pp.318-328(1988)



リストスケージュeringを適用
投入間隔をDに設定
モジュール予約テーブル
Dごとに循環
資源の競合発生 : D+1

Dの最小値は？

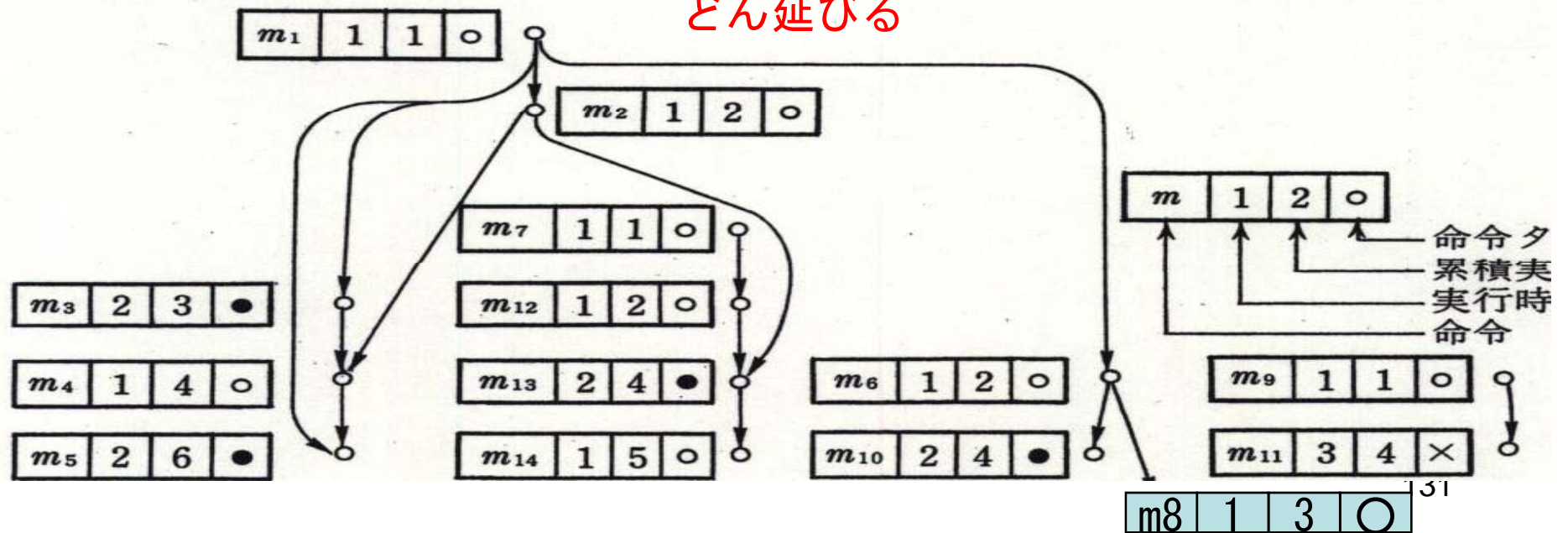
命令	デスティネーションレジスタ	ソースレジスタ	実行時間
m_1	R3	R1, R2	1
m_2	R5	R3, R4	1
m_3	R6	R3	2
m_4	R7	R5, R6	1
m_5	R8	R7, R3	2
m_6	R10	R9, R3	1
m_7	R11	R9	1
m_8	R21	R10	1
m_9	R13	R12	1
m_{10}	R15	R10	2
m_{11}	R20	R13, R14	3
m_{12}	R17	R16, R11	1
m_{13}	R18	R17, R5	2
m_{14}	R19	R18	1

(a) プログラム例

長命令			
●	○	○	×
m_5	m_{14}	m_8	m_{11}
m_{10}	m_4	m_9	
m_{13}	m_6		
m_3	m_{12}	m_2	
	m_1	m_7	

(c) 生成された長命令

スロット数どんどん延びる



タイムスロット →

命令	ロード ストア	ロード ストア	整数	整数	浮動 小数点	浮動 小数点	分岐
0							
1							
2							

D {

(c) モジュール予約テーブル

図 6.24 ソフトウェアパイプラインの方法

スロット
数はDに
固定

リストスケジューリングでD
ごとに折り畳む

Dの最小値の決定法

①反復間の依存性

$$D \geq (d(V) - d(U) + I(V)) / P1 \quad (29)$$

$$D \geq (d(U) - d(V) + I(U)) / P2 \quad (30)$$

$$D \geq (I(V) + I(U)) / (P1 + P2) \quad (31)$$

反復内での依存性

$$d(V) - d(U) \geq I(U) \quad (32)$$

②資源競合

反復内で同一の演算装置を使用する命令数NI、
長命令で指定できる同一の演算器の数をNVとすると、

$$D \geq NI / NV \quad (33)$$

D個の長命令

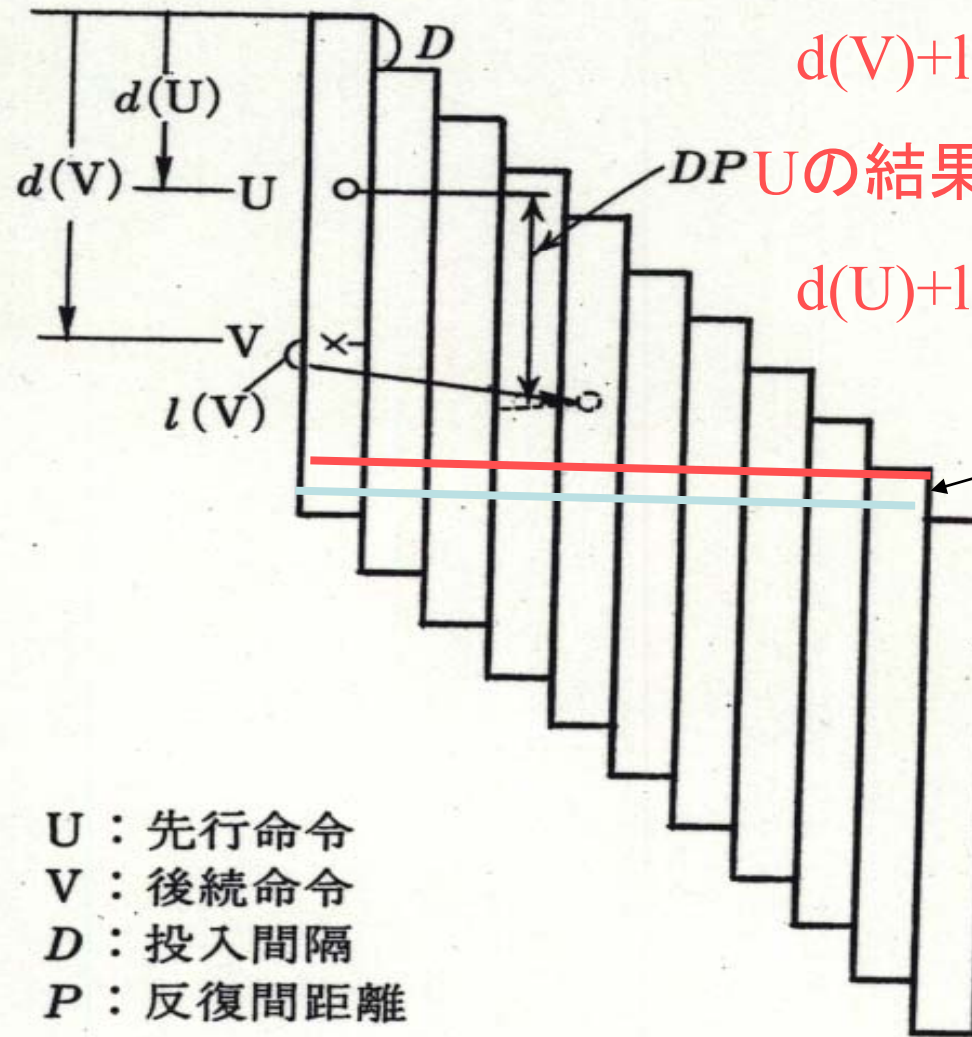
演算指定個数 : $D * NV \geq$ 命令数NI

Vの結果をP1後Uで利用

$$d(V)+l(V)-d(U)\leq DP1$$

Uの結果をP2後Vで利用

$$d(U)+l(U)-d(V)\leq DP2$$



D個の長命令

- U : 先行命令
- V : 後続命令
- D : 投入間隔
- P : 反復間距離

(b) 反復間での依存がある時

ソフトウェアパイプラインの手順

- ①投入間隔Dの最小値、最大値の決定
- ②投入間隔Dを最小値に設定, だめなら $D + 1$

命令の並び換えの手順

- ①データ依存グラフの作成
- ②ループ運搬依存（サイクル）のないとき
資源競合チェック：モジュロ予約テーブル
- ③ループ運搬依存（サイクル）のあるとき
ループ運搬依存にある2つの命令の優先的並換え
大ノードからなるデータ依存グラフに対して、②
を実行する。
- ④②、③で得られた結果に対して、逆依存、出力依存
を解消するレジスタリネーミングを施す。

サイクルのない場合の例 ソースプログラム

```
DO 10 I=1, N
```

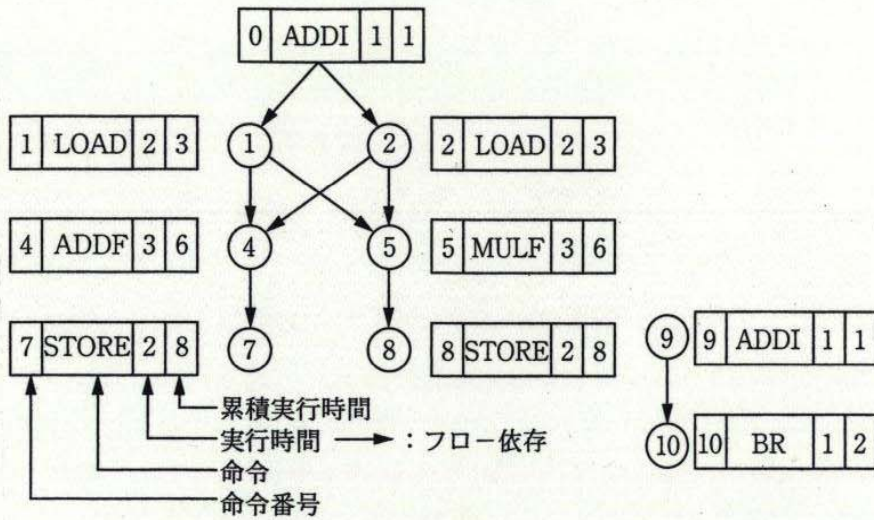
```
  Z(I)=X(I)+Y(I)          ( 1 9 )
```

```
  W(I)=X(I)*Y(I)
```

```
10 CONTINUE
```

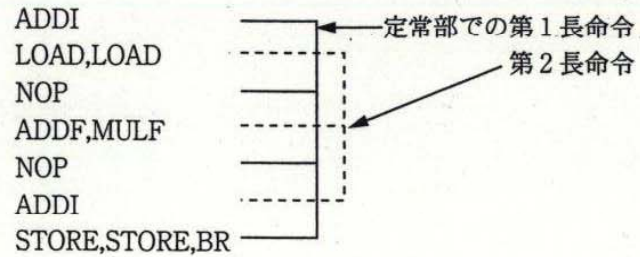
ソフトウェアパイプラインの例： 反復間にサイクルなしの例

0	ADDI	R10	#11	
1	LOAD	R1	M(R10+D1)	資源の制約から
2	LOAD	R2	M(R10+D2)	Dの最小値：2
3	NOP			
4	ADDF	R3	R1 R2	
5	MULF	R4	R1 R2	
6	NOP			(3 4)
7	STORE	M(R10+D3)	R3	
8	STORE	M(R10+D4)	R4	
9	ADDI	R9	#11	
10	BRCN	M(PC-18)		



(a) データ依存グラフ

		ロード ストア	ロード ストア	整数	整数	浮動 小数点	浮動 小数点	分岐	
ステップ 1	D	0	0					10	並び換えられた命令 ↑ 後 ↓ 前
	D	1	1						
ステップ 2	D	0	0	7	8			10	
	D	1	1				9		
ステップ 3	D	0	0	7	8			10	
	D	1	1				9		
ステップ 4	D	0	0	7	8			10	
	D	1	1				9	4 5	
ステップ 5	D	0	0	7	8			10	
	D	1	1				9	4 5	
ステップ 6	D	0	0	7	8			10	
	D	1	1	1	2		9	4 5	
ステップ 7	D	0	0	7	8			10	
	D	1	1				9	4 5	



(b) モジュール予約テーブルの変化

図 6.25 ソフトウェアパイプラインの一例
(データ依存グラフにサイクルのない場合)

並び換えられた命令列

0 ADDI R10 #11

1 LOAD R1 M(R10+D1), LOAD R2 M(R10+D2)

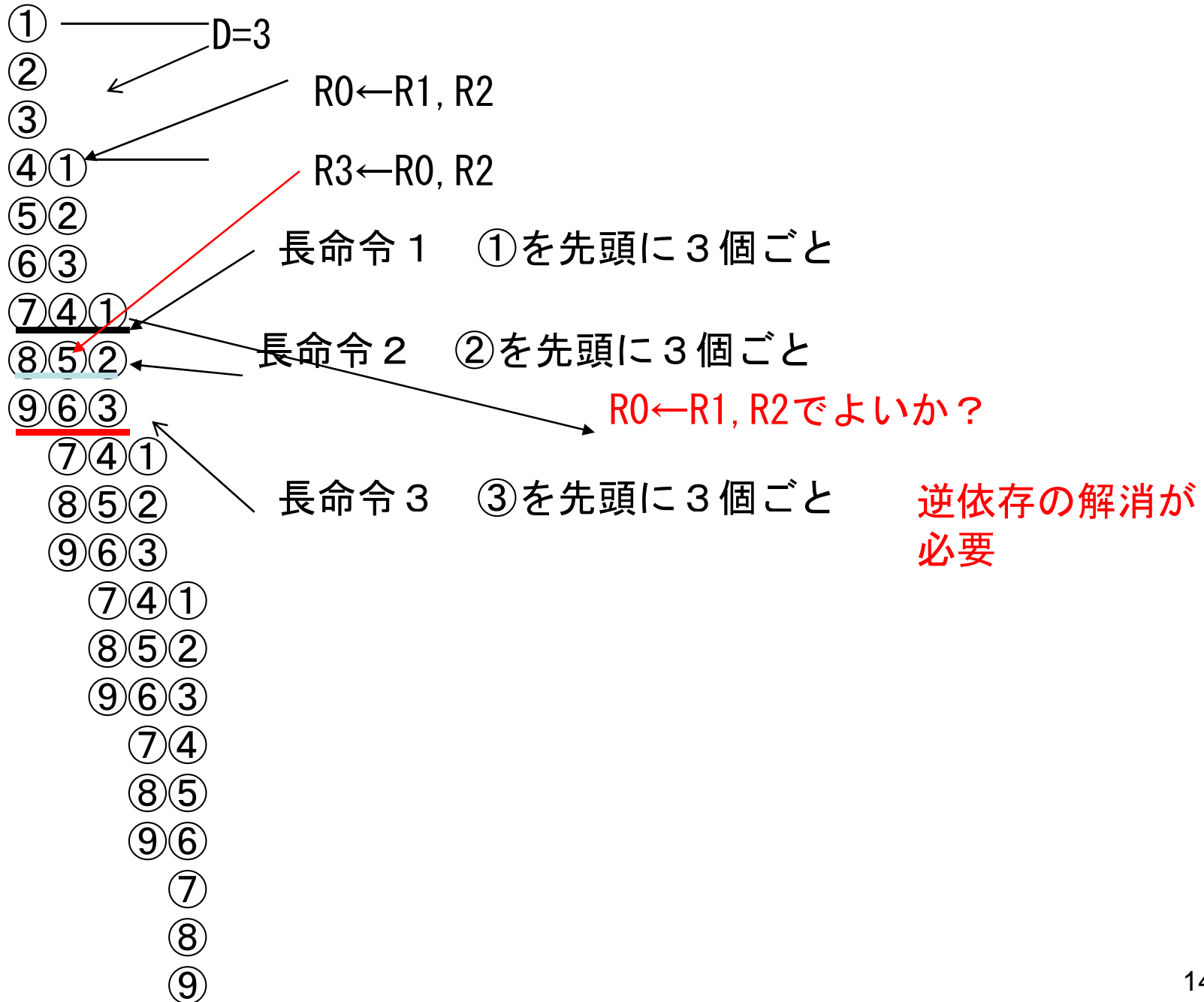
2 NOP

3 ADDF R3 R1 R2 , MULF R4 R1 R2

4 NOP

5 ADDI R9 #11 (3 5)

6 STORE M(R10+D3) R3 , STORE M(R10+D4) , BR



逆依存の解消

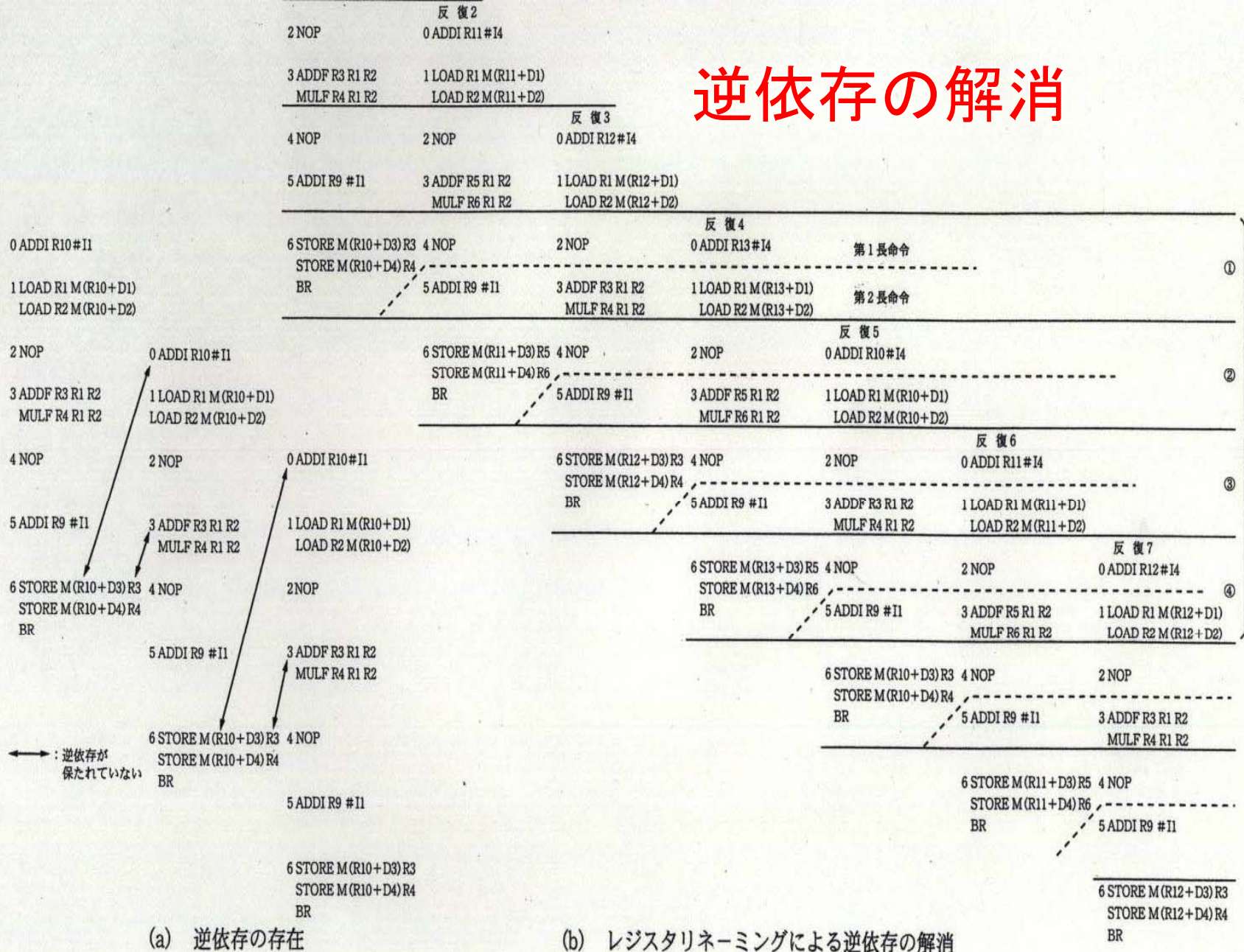
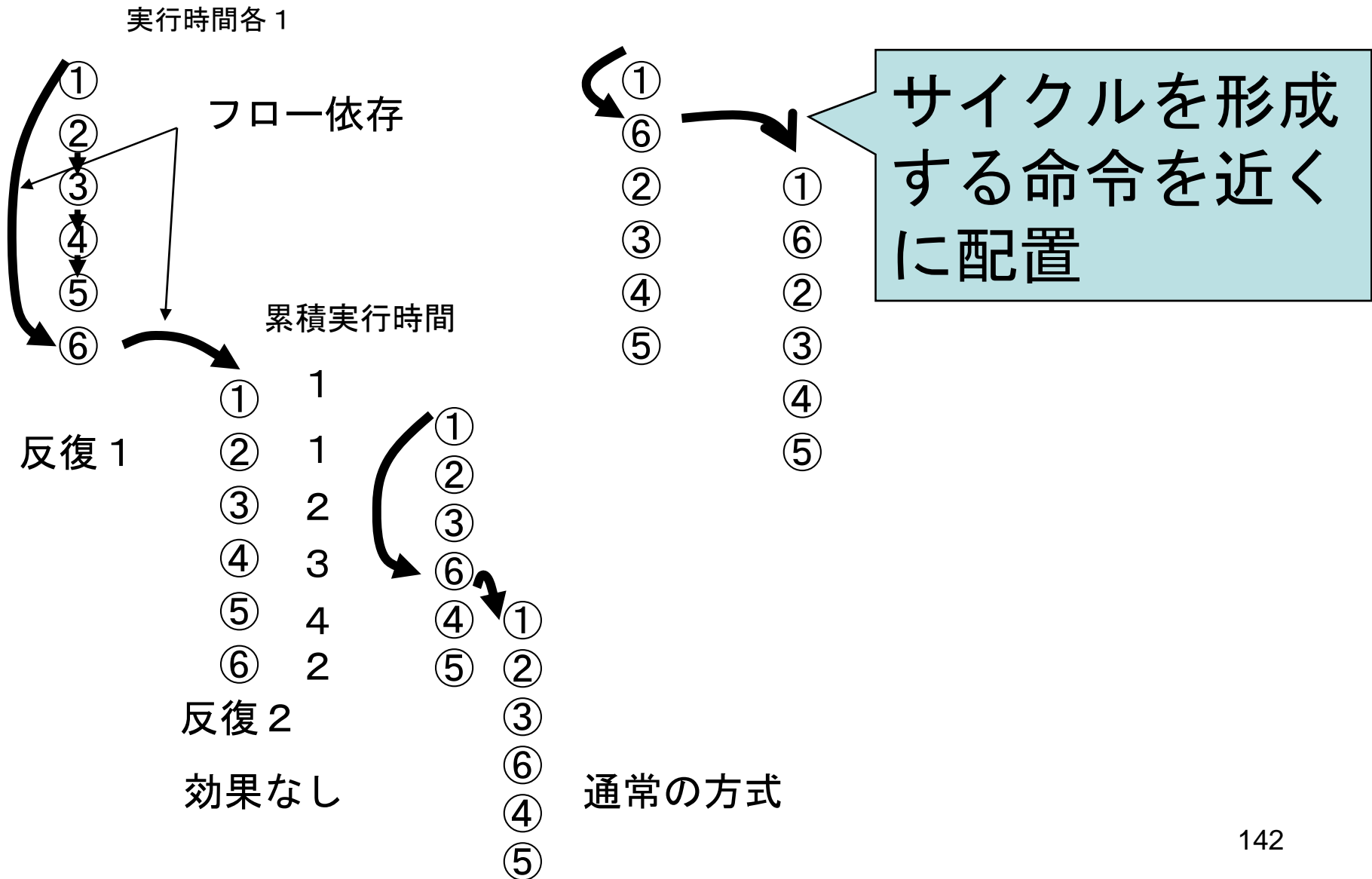


図 6.26 レジスタリネーミングの適用

反復間にサイクルがある場合



サイクルがある場合

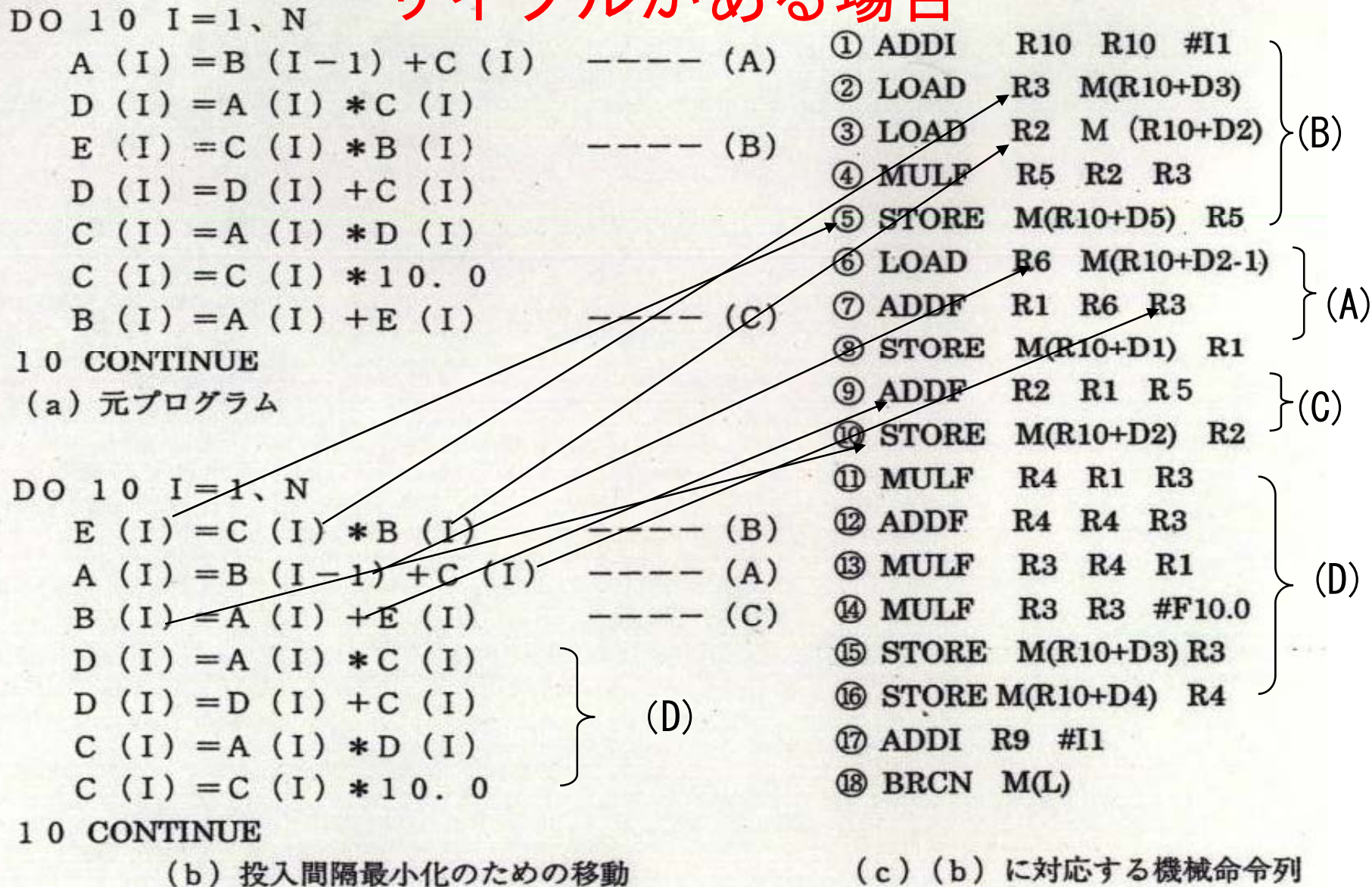
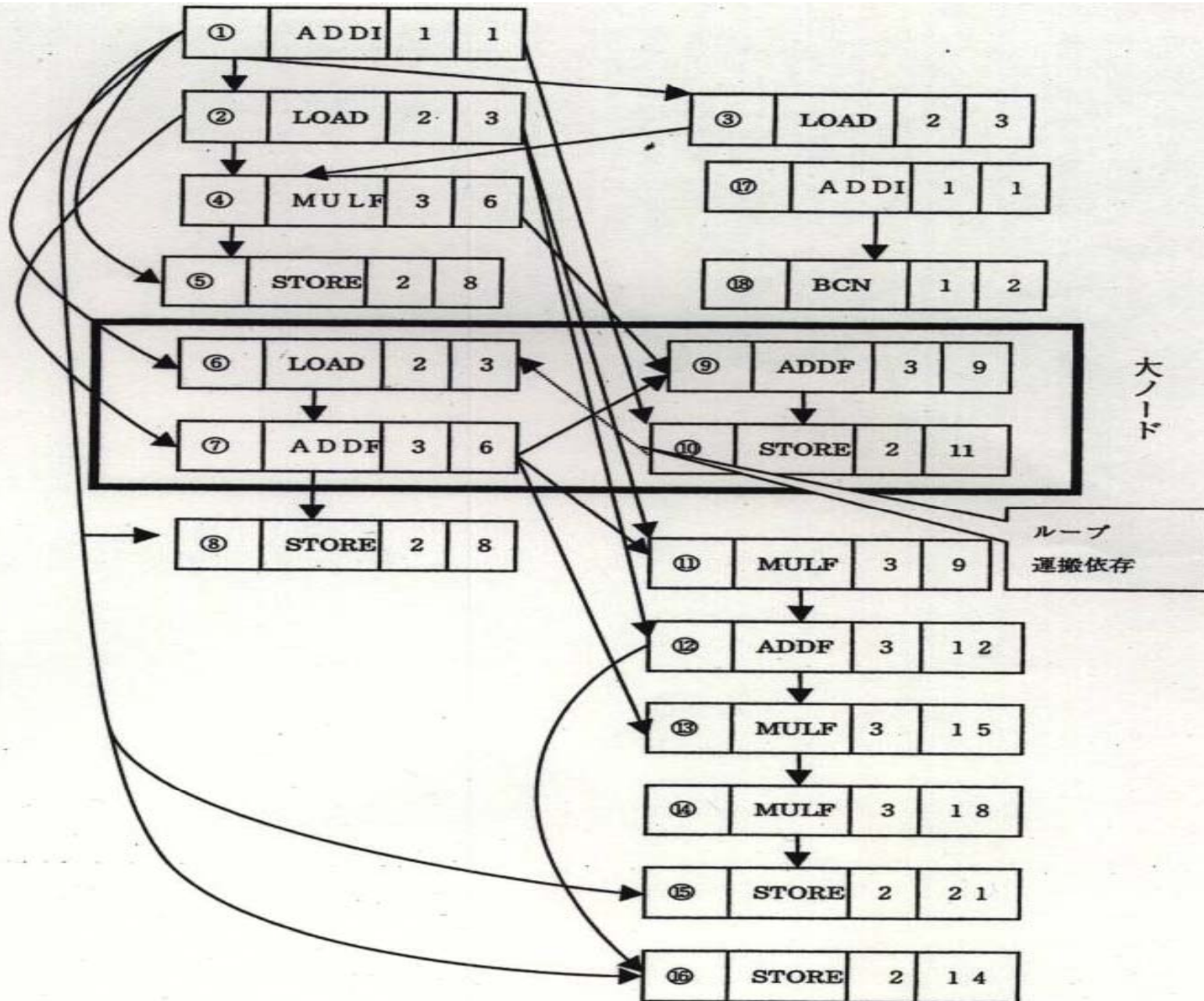


図 6.27 ソフトウェアパイプラインニングの例(デー

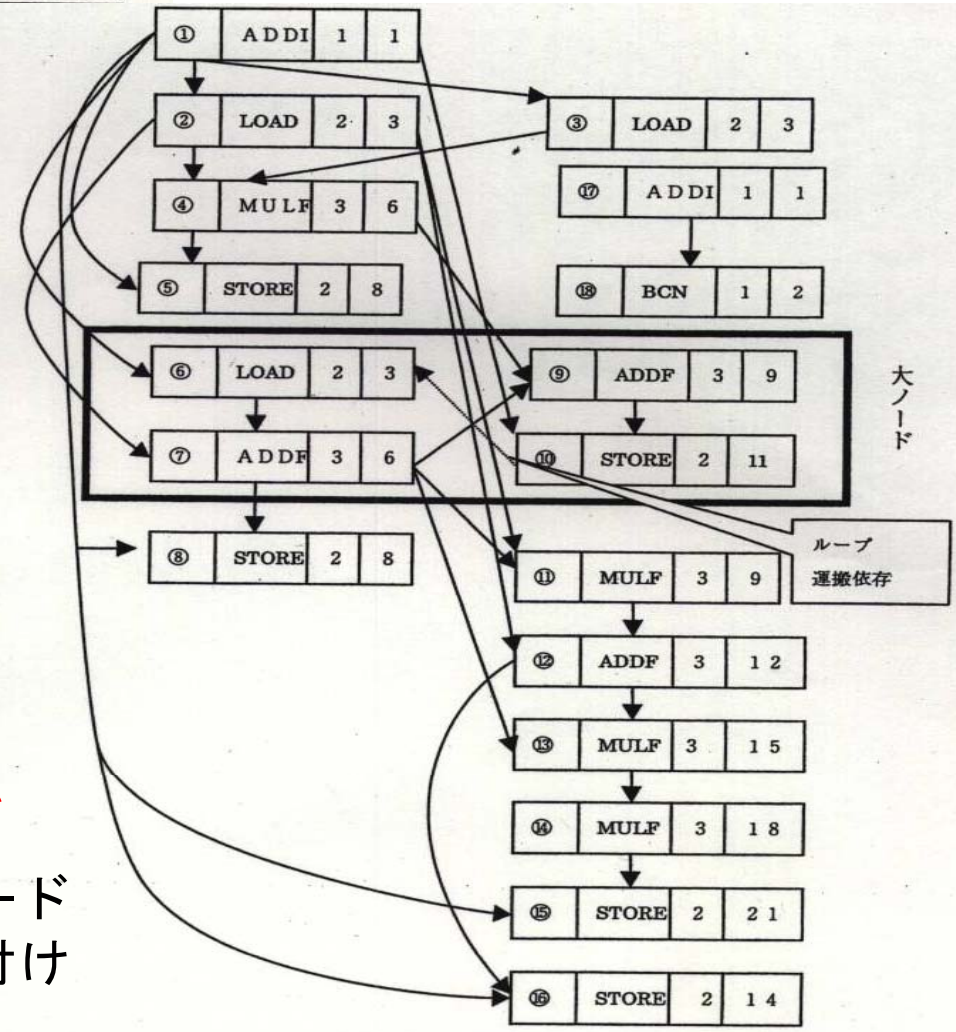
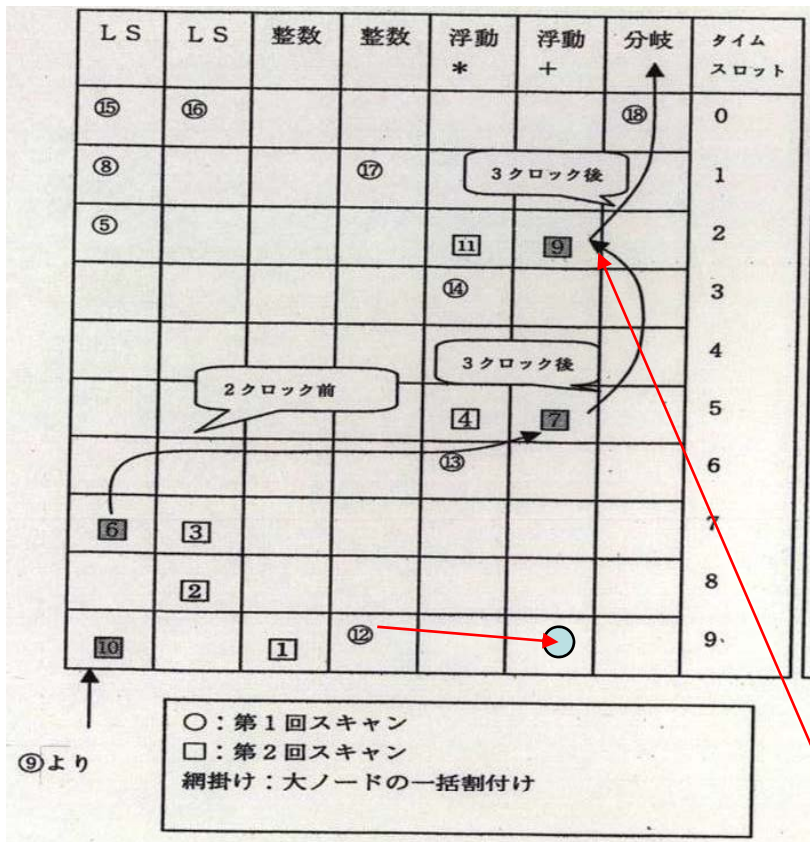


(d) データ依存グラフ

LS	LS	整数	整数	浮動 *	浮動 +	分岐	タイム スロット	並替えら れた命令
⑮	⑯					⑱	0	① ②
⑧			⑰				1	③ ⑥ —
⑤				⑪	⑨		2	④ ⑦ —
				⑭			3	— ⑨ ⑪
							4	— —
				④	⑦		5	⑫ ⑩ —
				⑬			6	— ⑬
⑥	③						7	— —
	②						8	⑭ ⑤
⑩		①	⊗		⑫		9	⑧ ⑰ ⑮ ⑯ ⑱

⑨より

○：第1回スキャン
□：第2回スキャン
網掛け：大ノードの一括割付け



大ノード
の割付け
可能

(d) データ依存グラフ

コンパイラ最適化の効果

ソースプログラム

```
DO 10 I=1, N  
  Z(I)=X(I)+Y(I)      ( 1 9 )  
  W(I)=X(I)*Y(I)  
10 CONTINUE
```

	方式	例番号	サイクル数
①	乱発行乱終了スカラ	(2 2)	1 0
②	乱発行乱終了スーパスカラ	(2 3)	6
③	単純 V L I W	(2 2)	6
④	マルチスレッド V L I W	(2 4)	3.5
⑤	アンローリング 2 回	(2 6)	3.5
⑥	アンローリング 3 回	(2 8)	3
⑦	ソフトウェアパイプライン	(3 5)	2

IntelのItanium

発表：2000年

EPIC

(Explicitly Parallel
Instruction Computing)

バンドルの導入

純粹VLIWの欠点を解消

コンパイラ支援を得つつ、

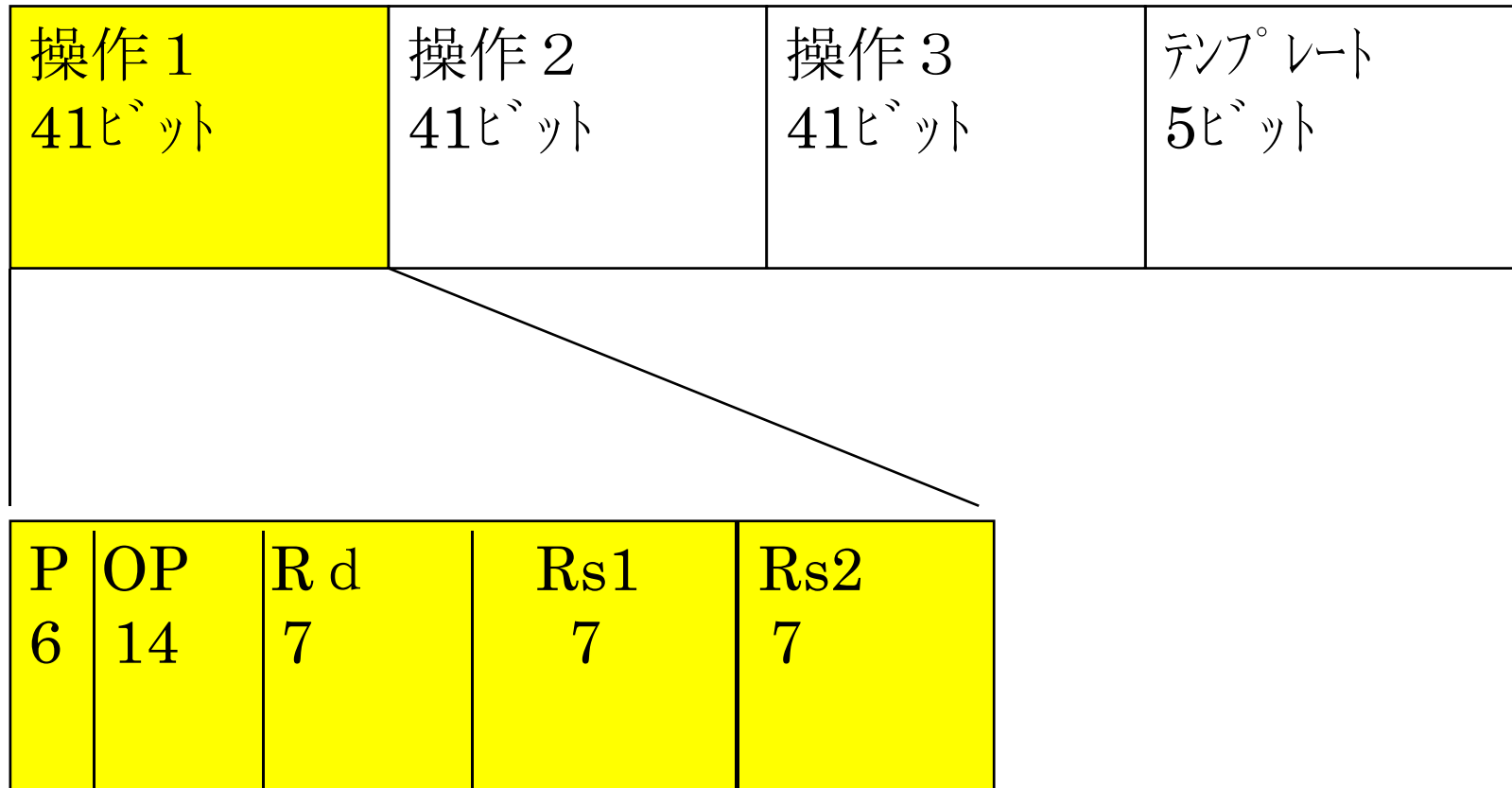
命令長の圧縮、NOP削除

互換性の確保、拡張性の確保

128ビット命令：バンドル

41ビット×3操作＋5ビットテンプレート

バンドル



テンプレート：5ビット

ストップビット（1ビット）

0：前バンドルと並列化可
互換性，拡張性の確保

1：前バンドルと並列化不可
長命令NOPなし

操作パターンの指定（4ビット）

3つの操作の組み合わせに制約

第2，3操作と並列化不可パターン
長命令内NOP削減

表 2.2 バンドルの種類

テンプレート	00	01	02	03	04	05	06	07
命令スロット	MII	MI_I	MLX	(未定義)	MMI	M_MI	MFI	MMF

テンプレート	08	09	0A	0B	0C	0D	0E	0F
命令スロット	MIB	MBB	(未定義)	BBB	MMB	(未定義)	MFB	(未定義)

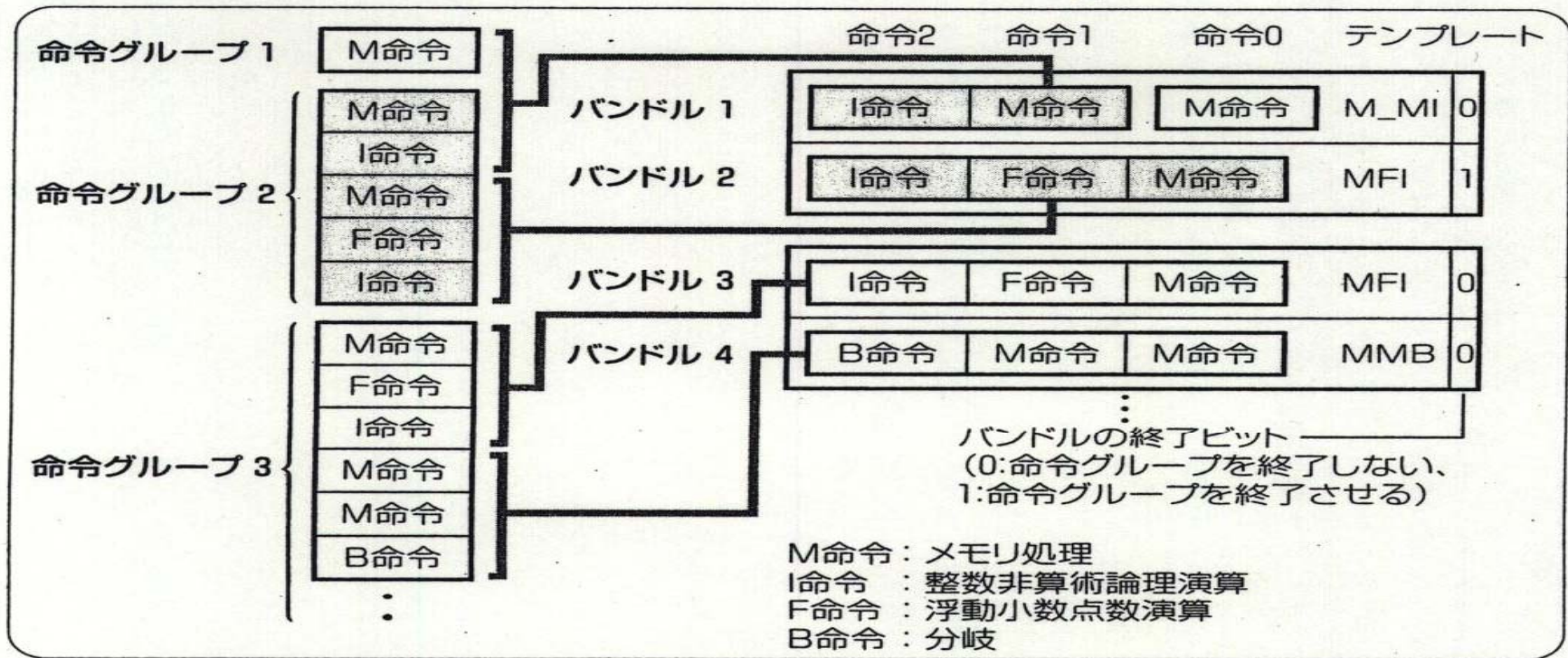


図 2.15 バンドルの構成例

プレディケート：6ビット

プレディケートレジスタを指示

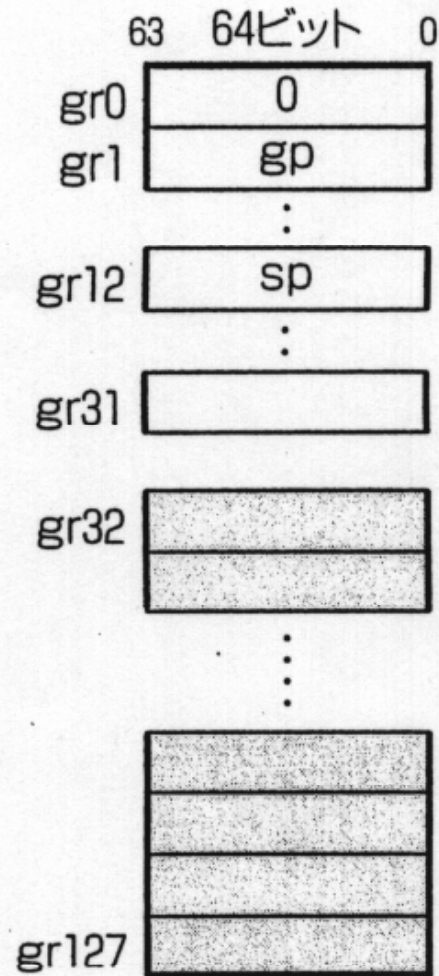
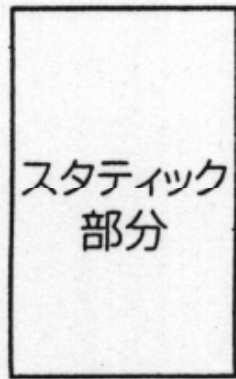
その内容が1であれば操作実行完了

True,False側同時実行と選択

レジスタ64個，6ビットで指定

投機ロードと遅延例外

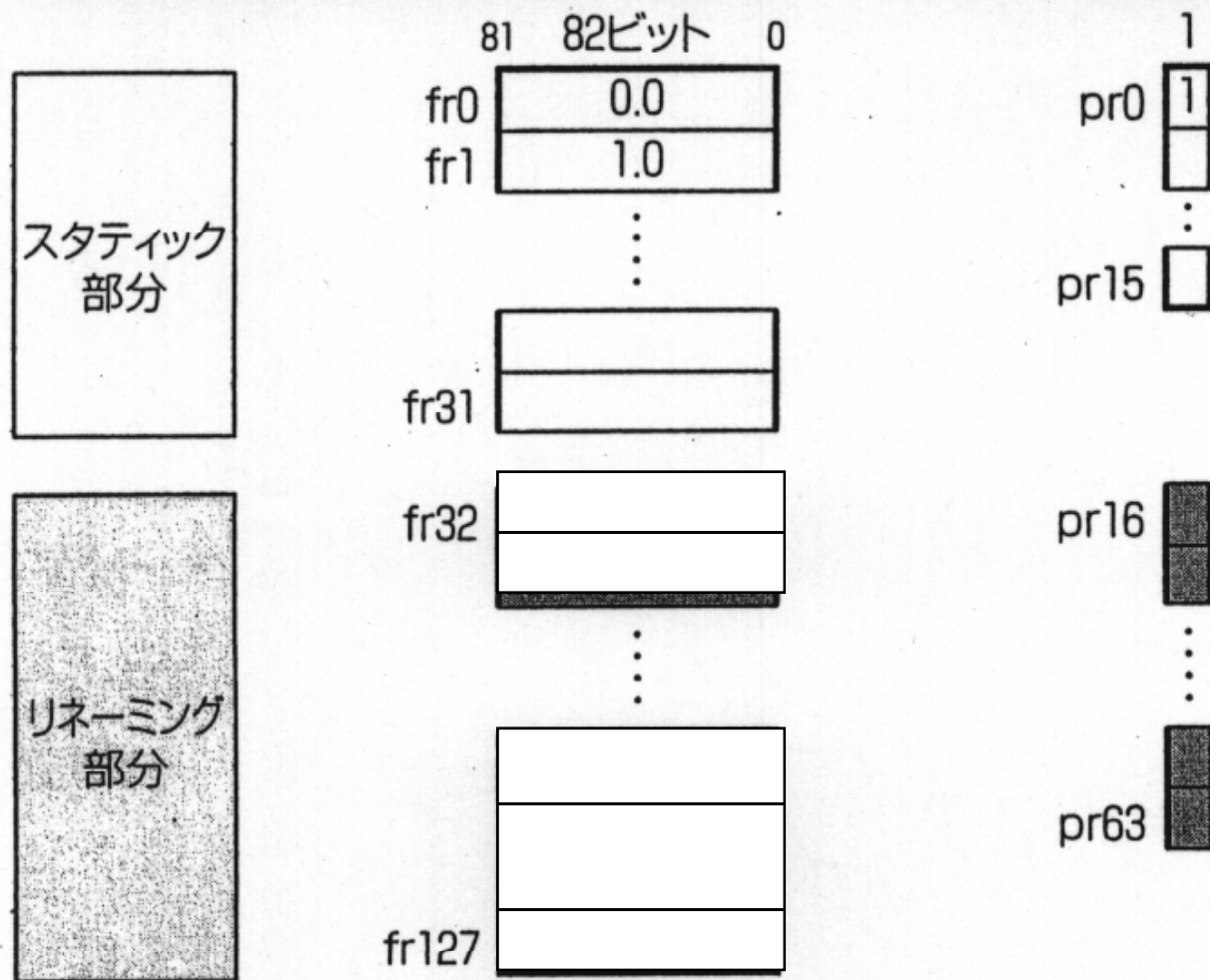
汎用レジスタ



レジスタ・スタック
+ローテーション

gp:グローバル・ポインタ NaT:Not a Thing
sp:スタック・ポインタ

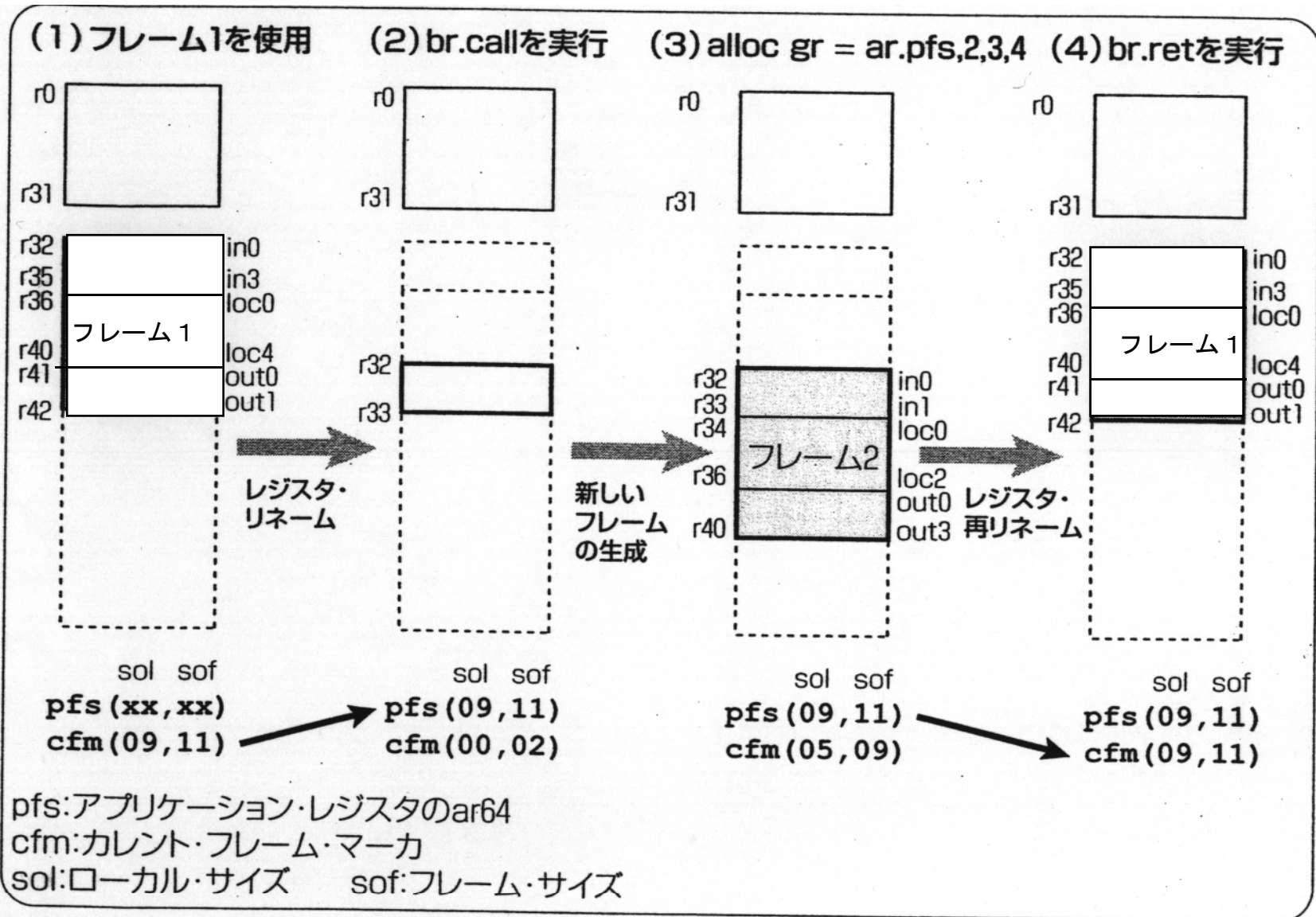
浮動小数点、プレディケートレジスタ



ローテーション

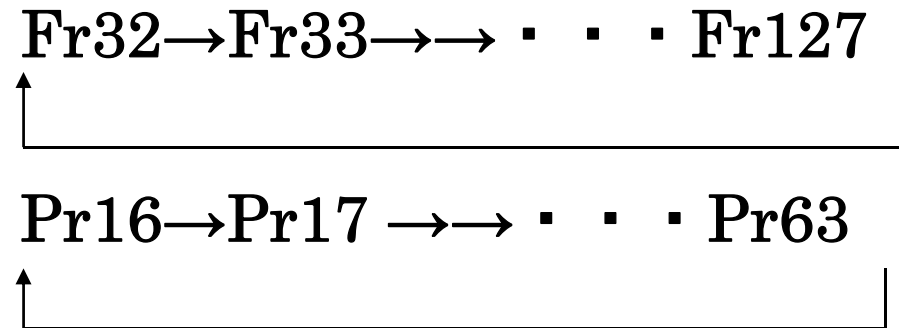
0.0,0.1:それぞれ数値の定数

レジスタスタック



RISCIのOverlap Register Window と同様

レジスタローテーション



モジュロスケジュール命令を実行時
レジスタ内容が回転する

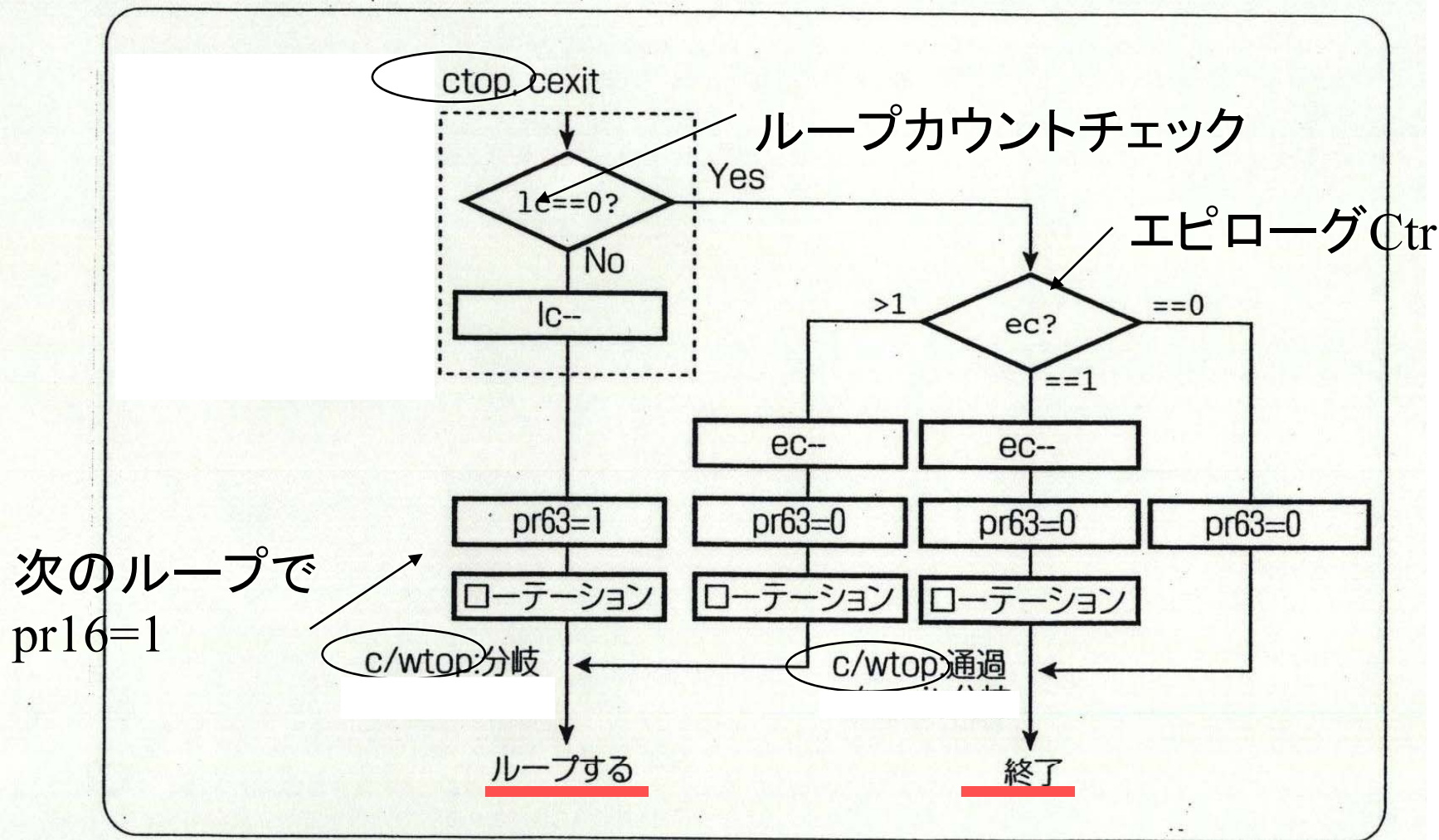


図 4.30 モジュール・スケジュール・ループ命令

(1)配列を定数倍するC言語プログラム

```
float a[20],b[20],c;  
for( i=0; i<20; i++){  
    b[i]=a[i]*c  
}
```

通常の目的プログラム



(2)IA-64アセンブラプログラム

```
mov ar.lc=19;; //ループカウンタに回数-1をセット  
loop:ldfs f32=[r15],4;; //命令グループ0 (4サイクル)  
fmpy.s f33=f32,f17;; //命令グループ1 (2サイクル)  
stfs [r16]=f33,4 //命令グループ2 (1サイクル)  
br.cloop.sptk loop //命令グループ2 (1サイクル)
```

```
mov pr.rot=0 //プレディケート・レジスタをクリア
cmp.eq p16=r0,r0 //pr16に1をセット
mov ar.lc=19 //ループ・カウンタに回数-1をセット
mov ar.ec=7 //エピローグ・カウンタにステージ数

loop:
(p16)ldfs f32=[r15],4
(p20)fmpy.s f37=f36,f17
(p22)stfs [r16]=f39,4
br,ctop.sptk loop
br,ctop
```

ロード:4サイクル
MUL:2サイクル

図4.31 ソフトウェア・パイプライン

プレディケート モジュロスケジュール命令

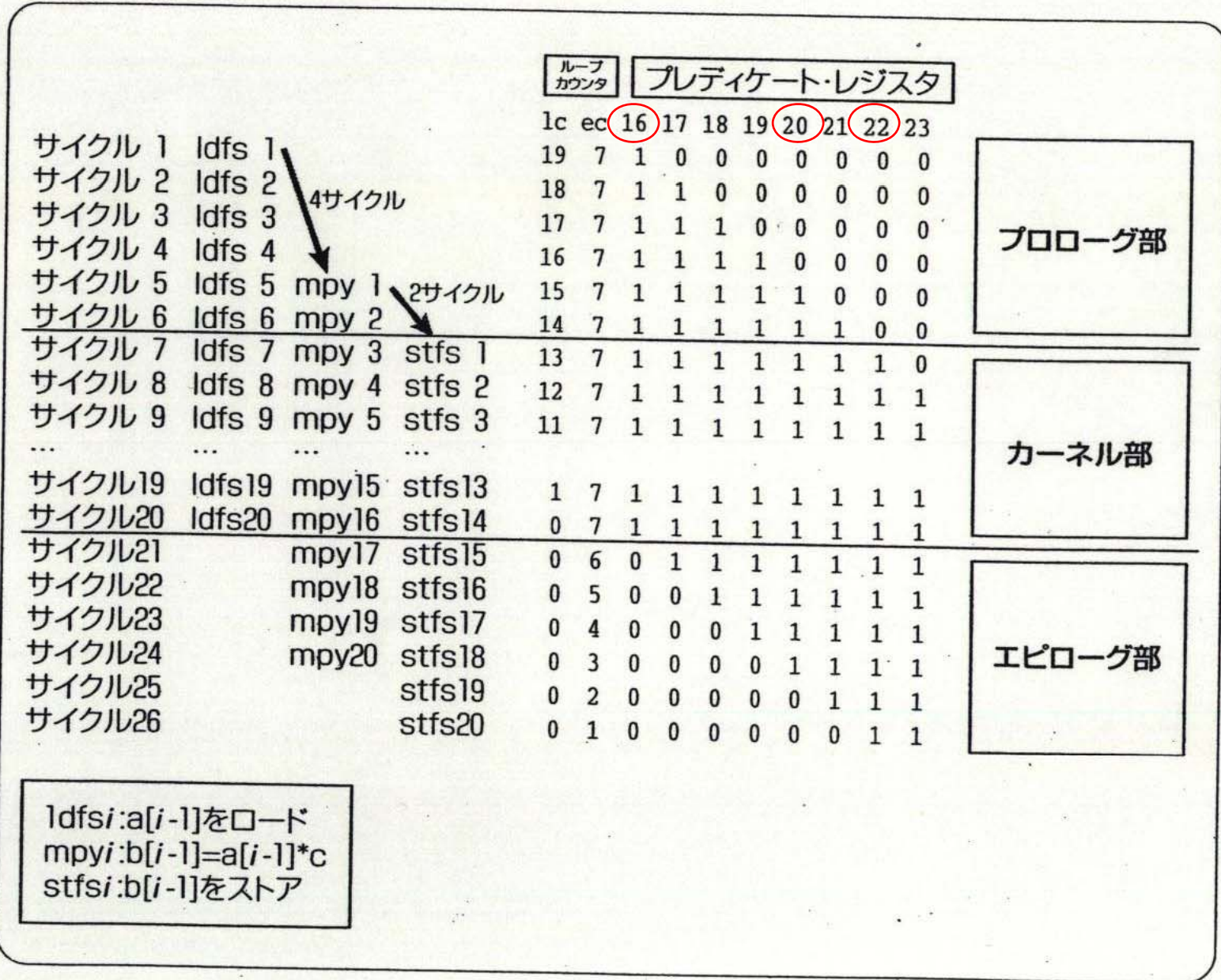


図 4.32 プレディケート・レジスタと実行サイクル

```

mov pr.rot=0 //プレディケート・レジスタをクリア
cmp.eq p16=r0,r0 //pr16に1をセット
mov ar.lc=19 //ループ・カウンタに回数-1をセット
mov ar.ec=7 //エピローグ・カウンタにステージ数

```

				ループ カウンタ	プレディケート・レジスタ								
				lc	ec	16	17	18	19	20	21	22	23
サイクル 1	ldfs 1			19	7	1	0	0	0	0	0	0	0
サイクル 2	ldfs 2			18	7	1	1	0	0	0	0	0	0
サイクル 3	ldfs 3			17	7	1	1	1	0	0	0	0	0
サイクル 4	ldfs 4			16	7	1	1	1	1	0	0	0	0
サイクル 5	ldfs 5	mpy 1		15	7	1	1	1	1	1	0	0	0
サイクル 6	ldfs 6	mpy 2		14	7	1	1	1	1	1	1	0	0
サイクル 7	ldfs 7	mpy 3	stfs 1	13	7	1	1	1	1	1	1	1	0
サイクル 8	ldfs 8	mpy 4	stfs 2	12	7	1	1	1	1	1	1	1	1
サイクル 9	ldfs 9	mpy 5	stfs 3	11	7	1	1	1	1	1	1	1	1
...										
サイクル19	ldfs19	mpy15	stfs13	1	7	1	1	1	1	1	1	1	1
サイクル20	ldfs20	mpy16	stfs14	0	7	1	1	1	1	1	1	1	1
サイクル21		mpy17	stfs15	0	6	0	1	1	1	1	1	1	1
サイクル22		mpy18	stfs16	0	5	0	0	1	1	1	1	1	1
サイクル23		mpy19	stfs17	0	4	0	0	0	1	1	1	1	1
サイクル24		mpy20	stfs18	0	3	0	0	0	0	1	1	1	1
サイクル25			stfs19	0	2	0	0	0	0	0	1	1	1
サイクル26			stfs20	0	1	0	0	0	0	0	0	1	1

```

loop:
(p16)ldfs f32=[r15],4
(p20)fmpy.s f37=f36,f17
(p22)stfs [r16]=f39,4
br.ctop.sptk loop

```

・ 投機ロード

STORE	M (R4)	R12	
LOAD	R6	M (R8)	
ADD	R5	R6	R7
STORE	M (R18)	R5	
	↓		
LOADAdv	R6	M (R8)	

STORE	M (R4)	R12	
LOADChk	R6	M (R8)	
ADD	R5	R6	R7
STORE	M (R18)	R5	

他の命令を埋め込み

R8=R4の時、投機失敗

ALAT (Advanced Load Address Table)で管理

ALATのR6のエントリをLOADAdvでセット

R8=R4ならSTOREでエントリ無効化

LOADChkでチェックOKなら投機成功、XならLOADの再実行

Itanium2の命令パイプライン

2002年7月発表

8ステージ

2バンドル発行

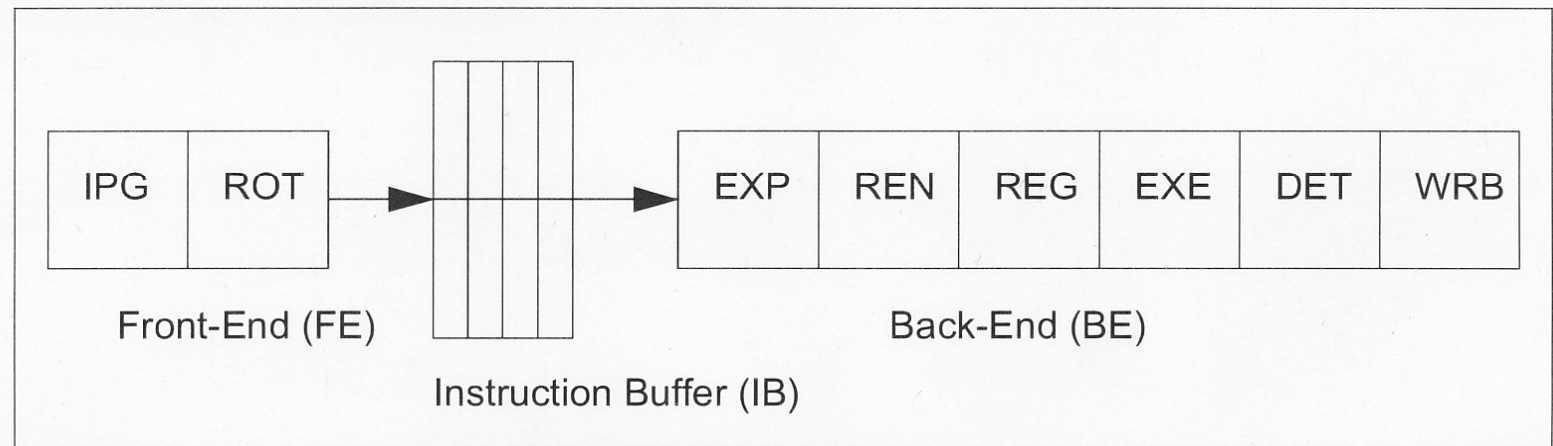
順発行順終了

分岐予測ミス：6サイクル

A.1 Core Pipeline

The core pipeline is separated into a front end (FE) and a back end (BE). The FE and BE are separated by an Instruction Buffer (IB).

Figure A-1. Core Pipeline of the Itanium[®] 2 Processor



The core pipeline consists of 8 stages:

IPG: Instruction pointer generation

ROT: instruction rotation

EXP: Instruction template decode, expand, and disperse

REN: Rename (for register stack and rotating registers) and decode

REG: Register file read

EXE: ALU execution

DET: Last stage for exception detection

WRB: Write back

A.4.1 FPU Micro-Pipeline

The FPU pipeline is four stages deep (*FP1* to *FP4*), with write back performed in the fifth stage (*FWB*). The FPU is fully pipelined. In the *FP1* stage, an early examination of the numeric operands is performed to determine if the instruction can be numeric exception free.

Table A-1. FPU Pipeline

Core Pipeline	REG	EXE	DET	WRB		
FPU Pipeline		FP1	FP2	FP3	FP4	FWB

A.4.2 L1D Micro-Pipeline

In the *L1M* stage, the L1 data, tag and the L1 DTLB are accessed in parallel and deliver data to the execution units.

Table A-2. L1D Micro-Pipeline

Core Pipeline	REN	REG	EXE	DET	WRB
L1D Pipeline		L1I	L1M	L1D	WRB

A.4.3 L2 Micro-Pipeline

The first stage is used for L2 TLB accesses. The L2A stage arbitrates for the data array accesses. Demand fetches for instructions have the highest priority, followed by loads and prefetches. Data array access occurs in the L2M stage. The L2D stage is for way selection, and data delivery. The L2C stage is used for correction of ECC errors and for error detection.

Table A-3. L2 Micro-Pipeline

Core Pipeline	REG	EXE	DET	WRB			
L2 Pipeline		L2L	L2A	L2M	L2D	L2C	L2W

キャッシュメモリ: 3階層、すべて実アドレスキャッシュ

Table 6-4. Cache Summary

	L1I	L1D	L2	L3
Size	16 KB	16 KB	256 KB	3 MB or 1.5 MB
Associativity	4-way	4-way	8-way	12-way
Line size	64 Bytes	64 Bytes	128 Bytes	128 Bytes
Latency	1 cycle	1 cycle	Minimum 5 cycles integer load use Minimum 6 cycles floating-point load use 7 cycles with 6 cycle stall penalty in <i>ROT</i> stage for instruction load use	Minimum 12 cycles load use
Tag Read bandwidth	2 / cycle	4 / cycle	4 / cycle	1 / cycle
Data Read bandwidth	1 X 32B / cycle	2 X 8B / cycle	4 x 8B / cycle	1 x 32B / cycle
Data banks	NA	8 bytes/bank (store only)	16 bytes/bank	NA
Write bandwidth	NA	2 x 8B / cycle	4 x 16B / cycle	1 x 32B / cycle
Fill bandwidth	64 bytes assembly 2 cycles write - 1 cycle	64 bytes assembly 2 cycles write - 1 cycle	128 bytes assembly 4 cycles write - 1 cycle	128 bytes in 4 cycles
Outstanding misses	7 prefetches	8 unique lines	16 unique lines	22 (16 read shared with L2, 6 write)
Line size	64 Bytes	64 Bytes	128 Bytes	128 Bytes

ストアスルー

ストアイン

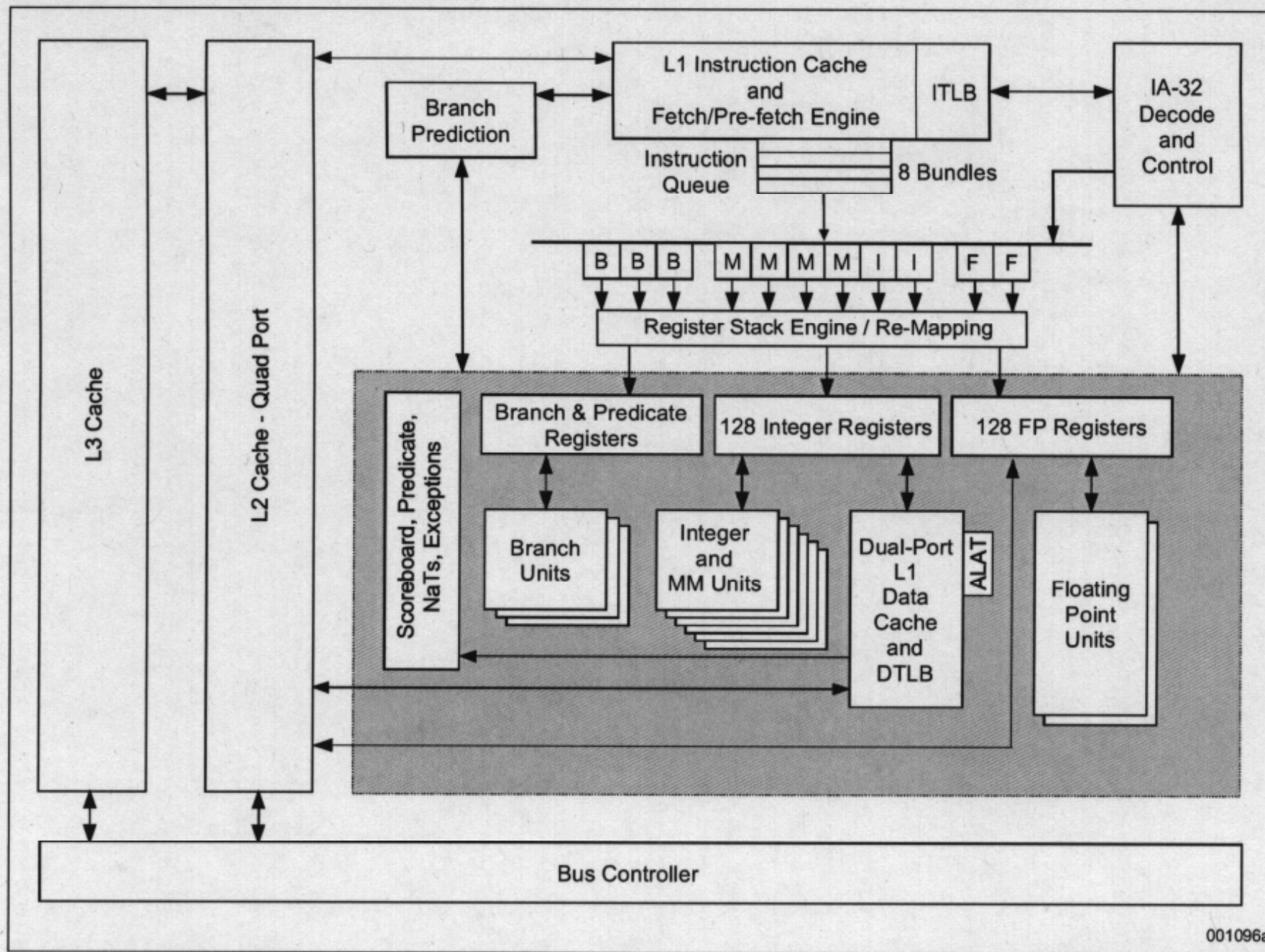
ストアイン

分岐予測

Table 2-8. Branch Prediction Latencies (in cycles)

	Itanium [®] 2 Processor	Itanium [®] Processor
Correctly predicted taken IP-relative branch	0	1
Correctly predicted taken indirect branch	2	0
Correctly predicted taken return branch	1	1
Last branch in perfect loop prediction	0	2
Misprediction latency	6+	9

Figure 2-3. Itanium[®] 2 Processor Block Diagram



かってデータフローコンピュータ という魅惑的な研究があった

1974年：J. Dennis (MIT)、静的データフロー

1978年：J. Gurd (Manchester大)、動的データフロー

1970年代後半から80年代前半

スーパスカラ方式のルーツ

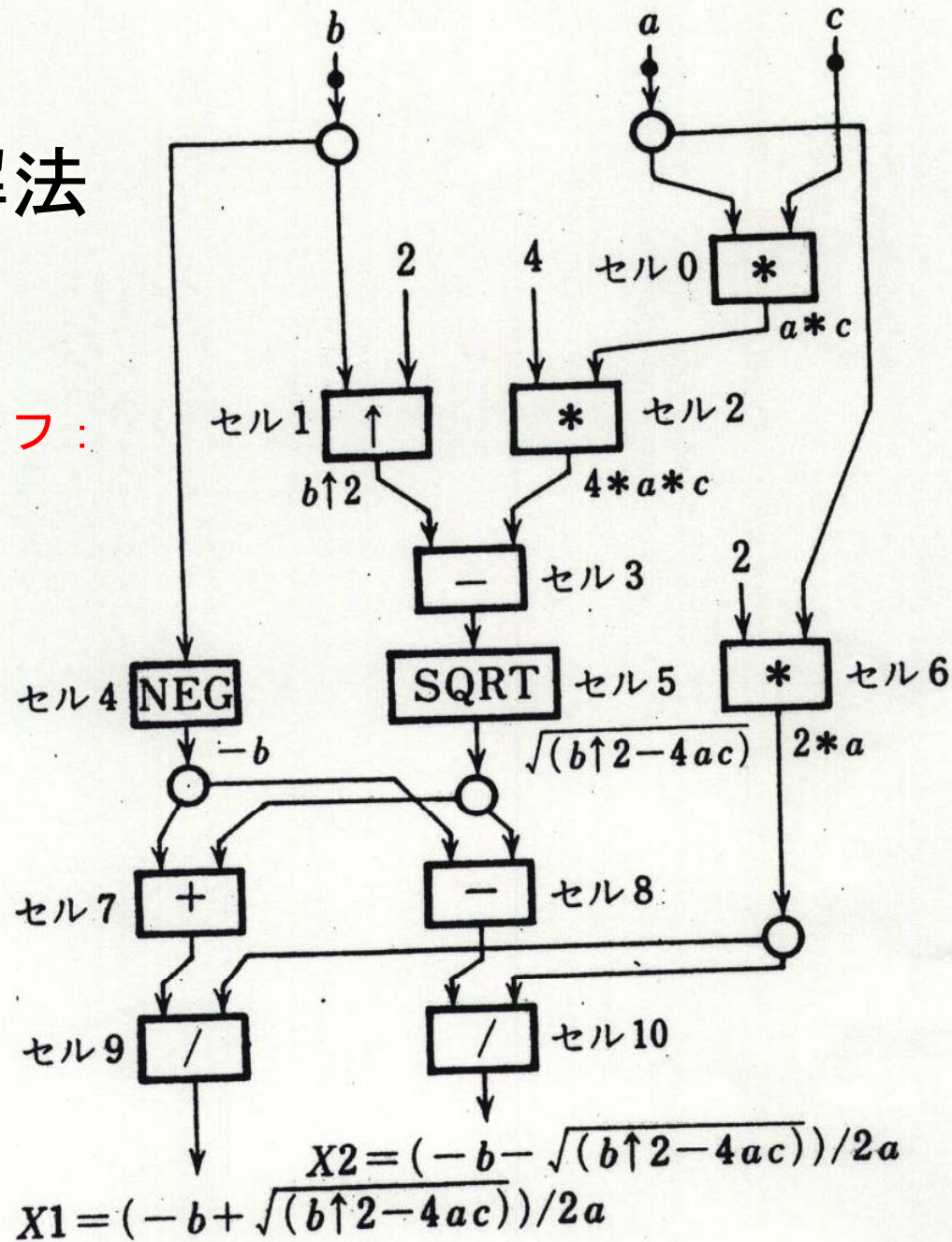
特徴

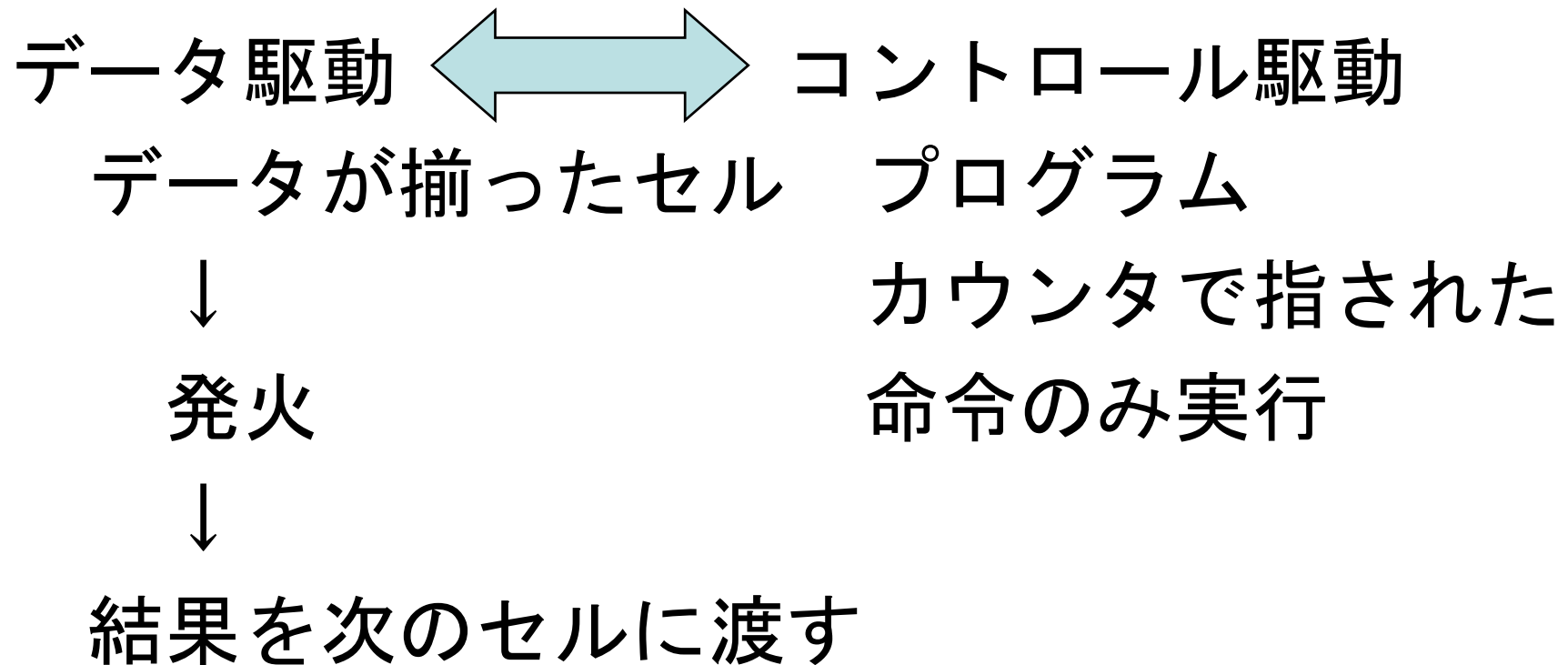
- ・プログラムカウンタがない
- ・プログラムに内在するすべての並列性を実行時に取り出せる

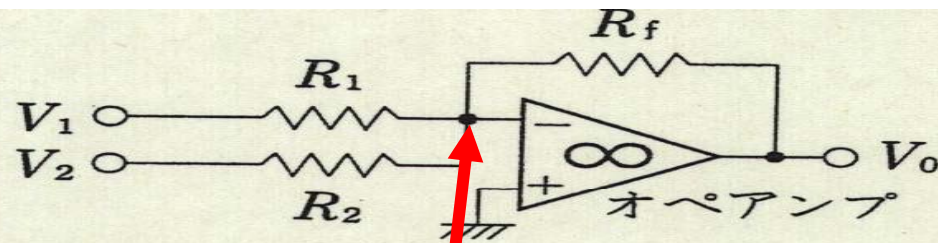
VLIW方式のルーツ：水平型マイクロプログラム

2次方程式の解法

データ依存を示すグラフ：
データフローグラフ

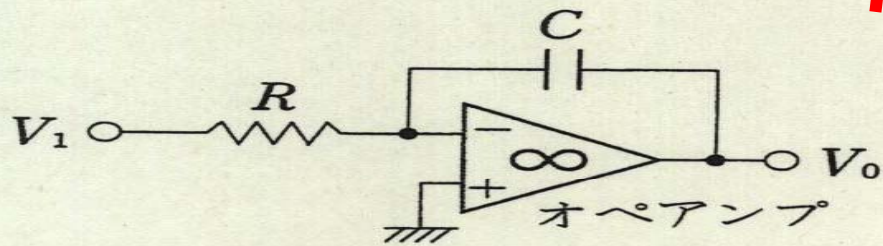




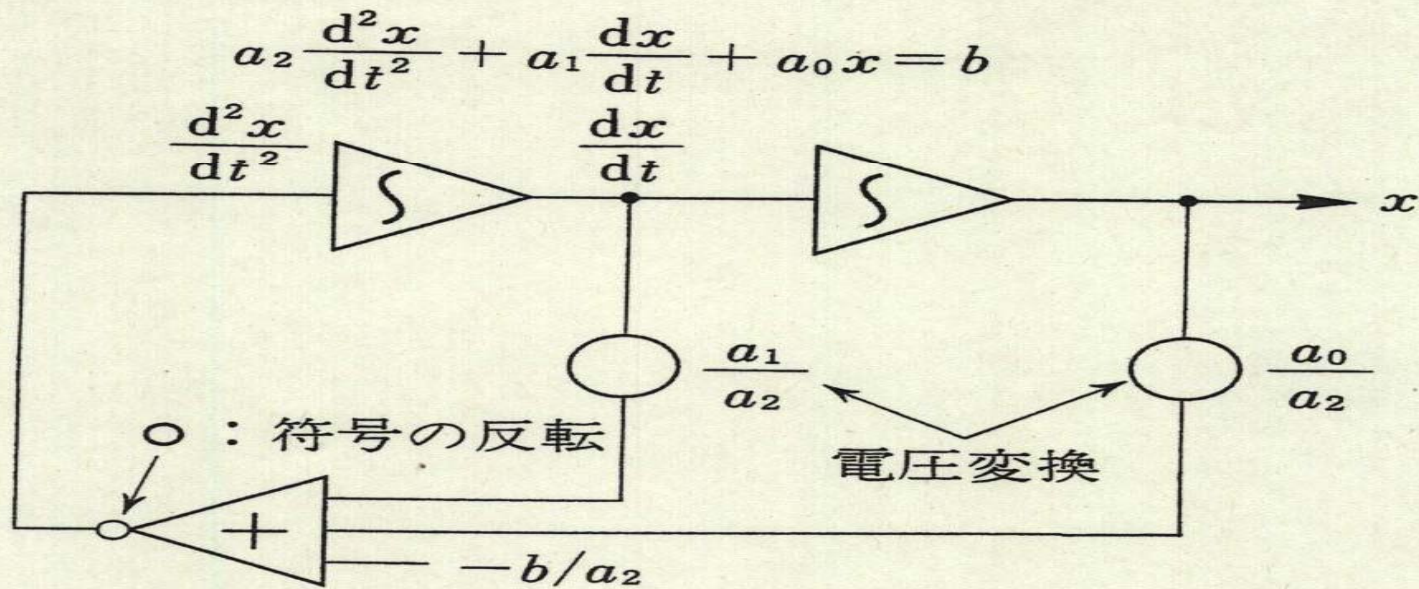


(a) 加算器 $V_0 = -R_f \left(\frac{V_1}{R_1} + \frac{V_2}{R_2} \right)$

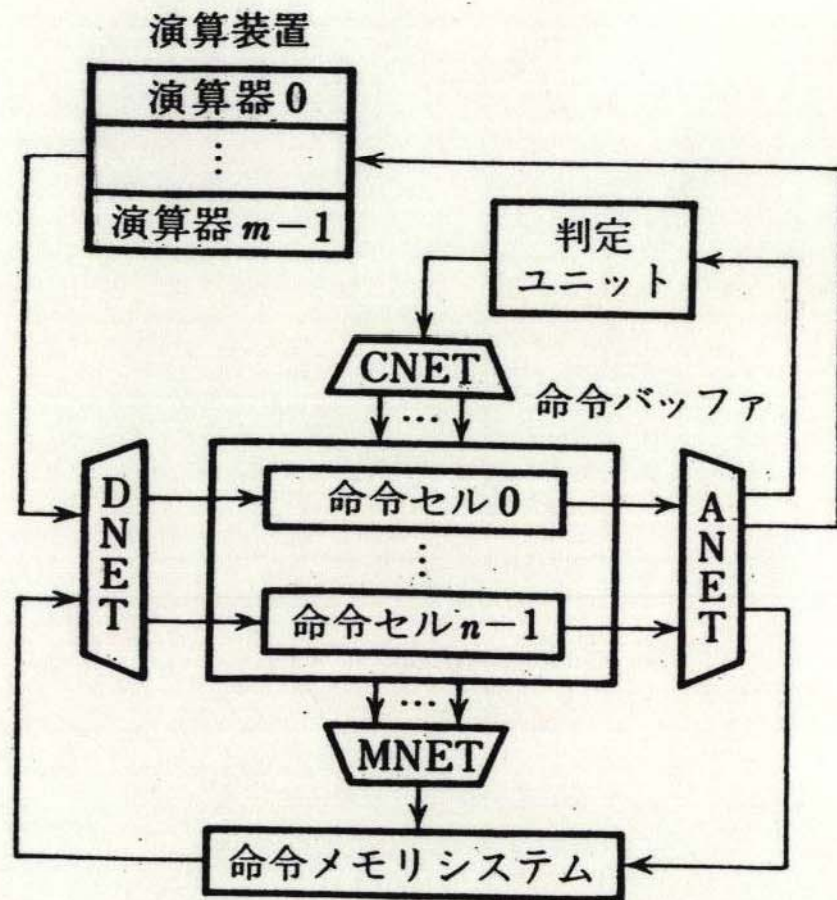
仮想接地: 0V



(b) 積分器 $V_0 = -\frac{Q}{C} = -\frac{1}{CR} \int V_1 dt$



(c) 常微分方程式を解く回路



(a) ハードウェア構造. ANET:アービトレーションネットワーク, DNET:ディストリビューションネットワーク, CNET:制御ネットワーク, MNET:メモリコマンドネットワーク



(b) 各命令セルの構造. d1, d2: デスティネーションセルを指定.

図 5.23 データ駆動型計算機の構成 (J. B. Dennis, *et al.*: 'A Preliminary Architecture for a Basic Data-Flow Processor', *Proc. of 2nd Annual Int. Symp. on Computer Architecture*, pp, 126-132 (1975) による)

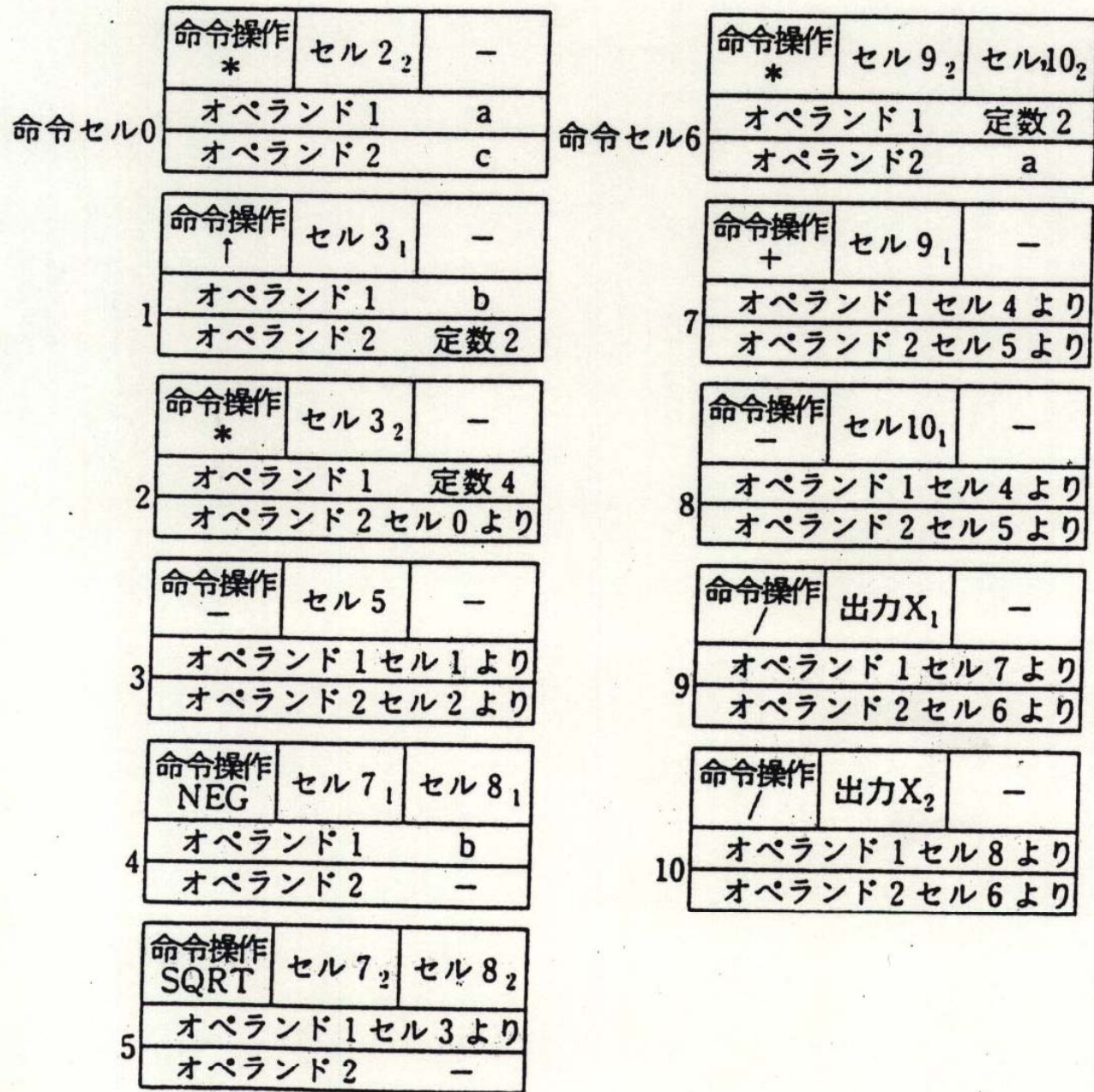
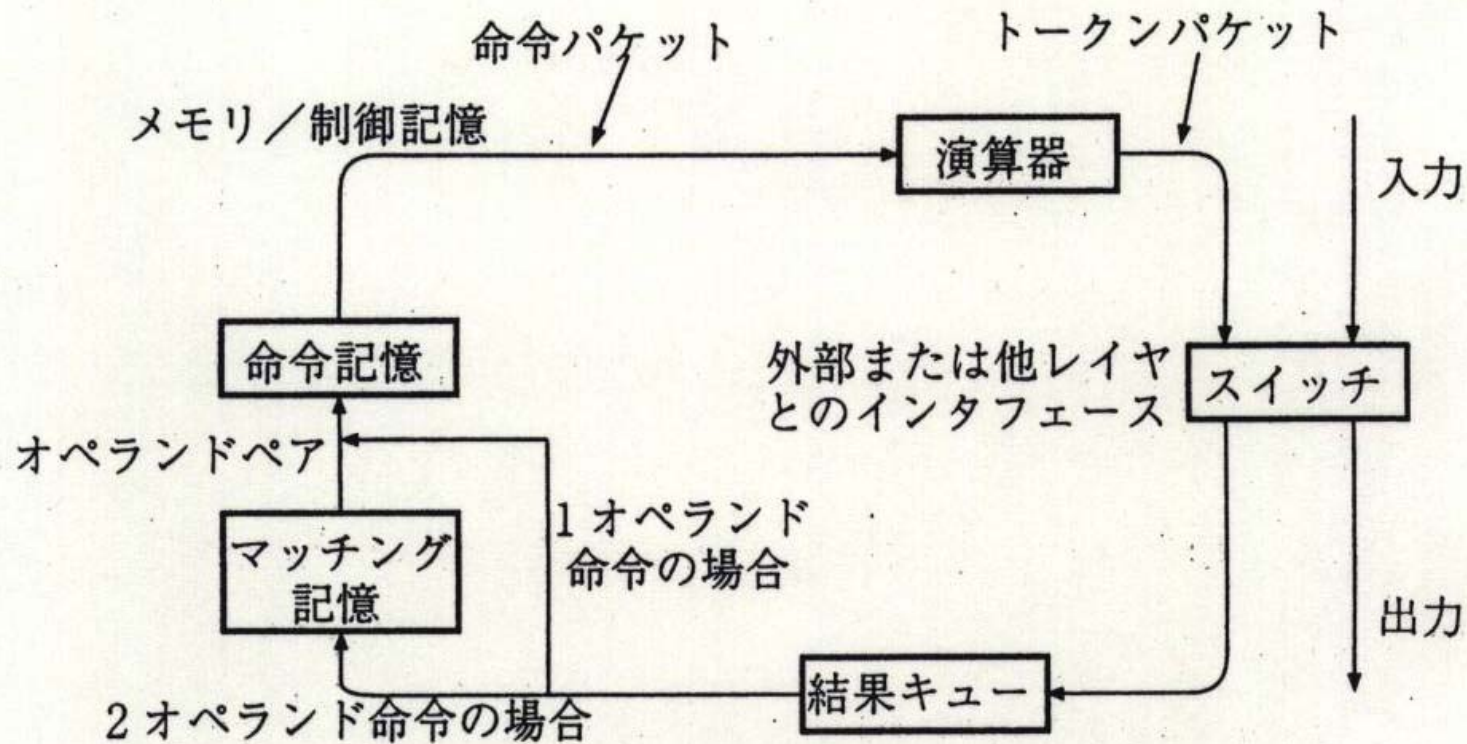


図 5.24 2次方程式の根を求めるデータフロープログラムに対するパケット群



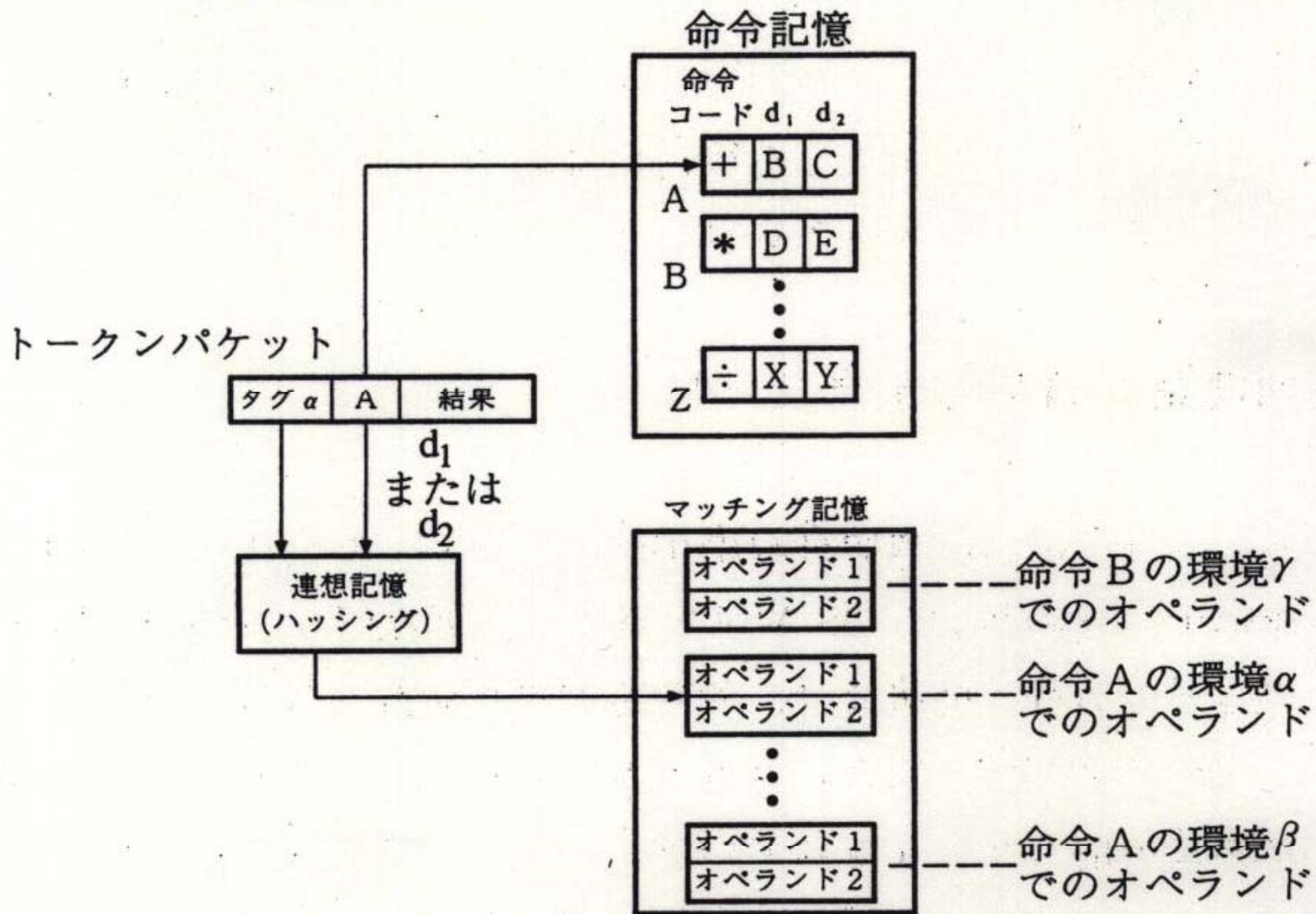
命令パッケージ

タグ	命令	あて先1 (d1)	あて先2 (d2)	オペランド1	オペランド2
----	----	--------------	--------------	--------	--------

トークンパッケージ

タグ	あて先 (d)	結果
----	------------	----

(a) 循環パイプラインの構造



(b) マッチング記憶

図 5.25 循環パイプライン型データ駆動コンピュータ

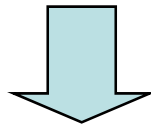
第 1 3 章 将来展望

- 13. 1 汎用プロセッサの高速化・省電力化
- 13. 2 メディアプロセッサ
- 13. 3 省電力化への並列処理の利用
- 13. 4 超高信頼、セキュアなプロセッサ
- 13. 5 再構成可能素子による
可変構造コンピュータ

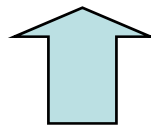
①大域並列の利用

- ・パイプライン：時間並列
- ・乱実行、投機実行による時間並列の高速化
- ・局所並列：スーパスカラ、VLIW
- ・非常に複雑な構造

IPCの頭打ち



- ・大域並列：マルチコア型プロセッサ



- ・周波数向上が困難

深いパイプラインで性能向上：小さくなった

プログラムカウンタの
近傍にある命令の
並列実行

②省電力化

動的消費電力 $\propto f^3$

リーク電流の増大

③高信頼化・セキュア化

キャッシュ、分岐予測ミス

ステージ内ゲート段数₁₇₉

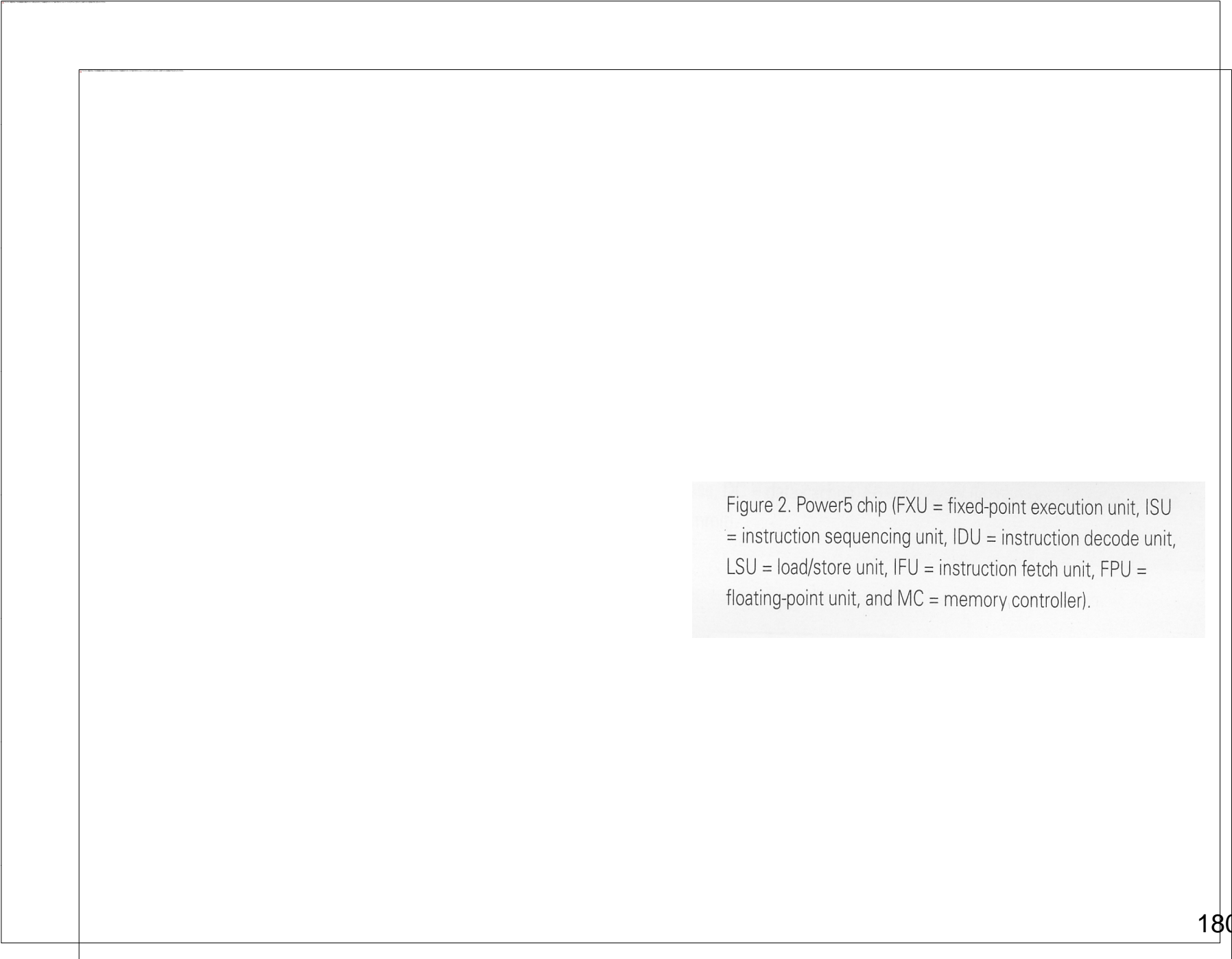


Figure 2. Power5 chip (FXU = fixed-point execution unit, ISU = instruction sequencing unit, IDU = instruction decode unit, LSU = load/store unit, IFU = instruction fetch unit, FPU = floating-point unit, and MC = memory controller).

Intelのマイクロプロセッサの 発展過程

Intel Processor	Date Introduced	Max. Clock Frequency at Introduction	Transistors	Register Sizes ¹	Ext. Data Bus Size ²	Max. Extern. Addr. Space	Caches
8086	1978	8 MHz	29 K	16 GP	16	1 MB	None
Intel 286	1982	12.5 MHz	134 K	16 GP	16	16 MB	Note 3
Intel386 DX Processor	1985	20 MHz	275 K	32 GP	32	4 GB	Note 3
Intel486 DX Processor	1989	25 MHz	1.2 M	32 GP 80 FPU	32	4 GB	L1: 8 KB
Pentium Processor	1993	60 MHz	3.1 M	32 GP 80 FPU	64	4 GB	L1:16 KB
Pentium Pro Processor	1995	200 MHz	5.5 M	32 GP 80 FPU	64	64 GB	L1: 16 KB L2: 256 KB or 512 KB
Pentium II Processor	1997	266 MHz	7 M	32 GP 80 FPU 64 MMX	64	64 GB	L1: 32 KB L2: 256 KB or 512 KB
Pentium III Processor	1999	500 MHz	8.2 M	32 GP 80 FPU 64 MMX 128 XMM	64	64 GB	L1: 32 KB L2: 512 KB
Pentium III and Pentium III Xeon Processors	1999	700 MHz	28 M	32 GP 80 FPU 64 MMX 128 XMM	64	64 GB	L1: 32 KB L2: 256 KB

Intel Processor	Date Introduced	Microarchitecture	Clock Frequency at Introduction	Transistors	Register Sizes ¹	System Bus Bandwidth	Max. Extern. Addr. Space	On-Die Caches ²
Pentium 4 Processor	2000	Intel NetBurst Microarchitecture	1.50 GHz	42 M	GP: 32 FPU: 80 MMX: 64 XMM: 128	3.2 GB/s	64 GB	12K μ op Execution Trace Cache; 8KB L1; 256-KB L2
Intel Xeon Processor	2001	Intel NetBurst Microarchitecture	1.70 GHz	42 M	GP: 32 FPU: 80 MMX: 64 XMM: 128	3.2 GB/s	64 GB	12K μ op Trace Cache; 8-KB L1; 256-KB L2

Intel Processor	Date Introduced	Microarchitecture	Clock Frequency at Introduction	Transistors	Register Sizes¹	System Bus Bandwidth	Max. Extern. Addr. Space	On-Die Caches²
Intel Xeon Processor	2002	Intel NetBurst Microarchitecture; Hyper-Threading Technology	2.20 GHz	55 M	GP: 32 FPU: 80 MMX: 64 XMM: 128	3.2 GB/s	64 GB	12K μ op Trace Cache; 8-KB L1; 512-KB L2
Intel Xeon Processor MP	2002	Intel NetBurst Microarchitecture; Hyper-Threading Technology	1.60 GHz	108 M	GP: 32 FPU: 80 MMX: 64 XMM: 128	3.2 GB/s	64 GB	12K μ op Trace Cache; 8-KB L1; 256-KB L2; 1-MB L3
Intel Pentium 4 Processor Supporting Hyper-Threading Technology	2002	Intel NetBurst Microarchitecture; Hyper-Threading Technology	3.06 GHz	55 M	GP: 32 FPU: 80 MMX: 64 XMM: 128	4.2 GB/s	64 GB	12K μ op Execution Trace Cache; 8-KB L1; 512-KB L2
Intel Pentium M Processor	2003	Intel Pentium M Processor	1.60 GHz	77 M	GP: 32 FPU: 80 MMX: 64 XMM: 128	3.2 GB/s	4 GB	L1: 64 KB L2: 1 MB
Intel Pentium 4 Processor Supporting Hyper-Threading Technology at 90 nm process	2004	Intel NetBurst Microarchitecture; Hyper-Threading Technology	3.40 GHz	125 M	GP: 32 FPU: 80 MMX: 64 XMM: 128	6.4 GB/s	64 GB	12K μ op Execution Trace Cache; 16 KB L1; 1 MB L2
Intel Pentium M Processor 755 ³	2004	Intel Pentium M Processor	2.00 GHz	140 M	GP: 32 FPU: 80 MMX: 64 XMM: 128	3.2 GB/s	4 GB	L1: 64 KB L2: 2 MB
64-bit Intel Xeon Processor with 800 MHz System Bus	2004	Intel NetBurst Microarchitecture; Hyper-Threading Technology; Intel Extended Memory 64 Technology	3.60 GHz	125 M	GP: 32, 64 FPU: 80 MMX: 64 XMM: 128	6.4 GB/s	64 GB	12K μ op Execution Trace Cache; 16 KB L1; 1 MB L2

Intel Processor	Date Introduced	Microarchitecture	Clock Frequency at Introduction	Transistors	Register Sizes ¹	System Bus Bandwidth	Max. Extern. Addr. Space	On-Die Caches ²
64-bit Intel Xeon Processor MP with 8MB L3	2005	Intel NetBurst Microarchitecture; Hyper-Threading Technology; Intel Extended Memory 64 Technology	3.33 GHz	675M	GP: 32, 64 FPU: 80 MMX: 64 XMM: 128	5.3 GB/s ⁴	1024 GB (1 TB)	12K μ op Execution Trace Cache; 16 KB L1; 1 MB L2, 8 MB L3
Intel Pentium 4 Processor Extreme Edition Supporting Hyper-Threading Technology	2005	Intel NetBurst Microarchitecture; Hyper-Threading Technology; Intel Extended Memory 64 Technology	3.73 GHz	164 M	GP: 32, 64 FPU: 80 MMX: 64 XMM: 128	8.5 GB/s	64 GB	12K μ op Execution Trace Cache; 16 KB L1; 2 MB L2
Intel Pentium Processor Extreme Edition 840	2005	Intel NetBurst Microarchitecture; Hyper-Threading Technology; Intel Extended Memory 64 Technology; Dual-core ⁵	3.20 GHz	230 M	GP: 32, 64 FPU: 80 MMX: 64 XMM: 128	6.4 GB/s	64 GB	12K μ op Execution Trace Cache; 16 KB L1; 1MB L2 (2MB Total)

周波数向上: 40%/年

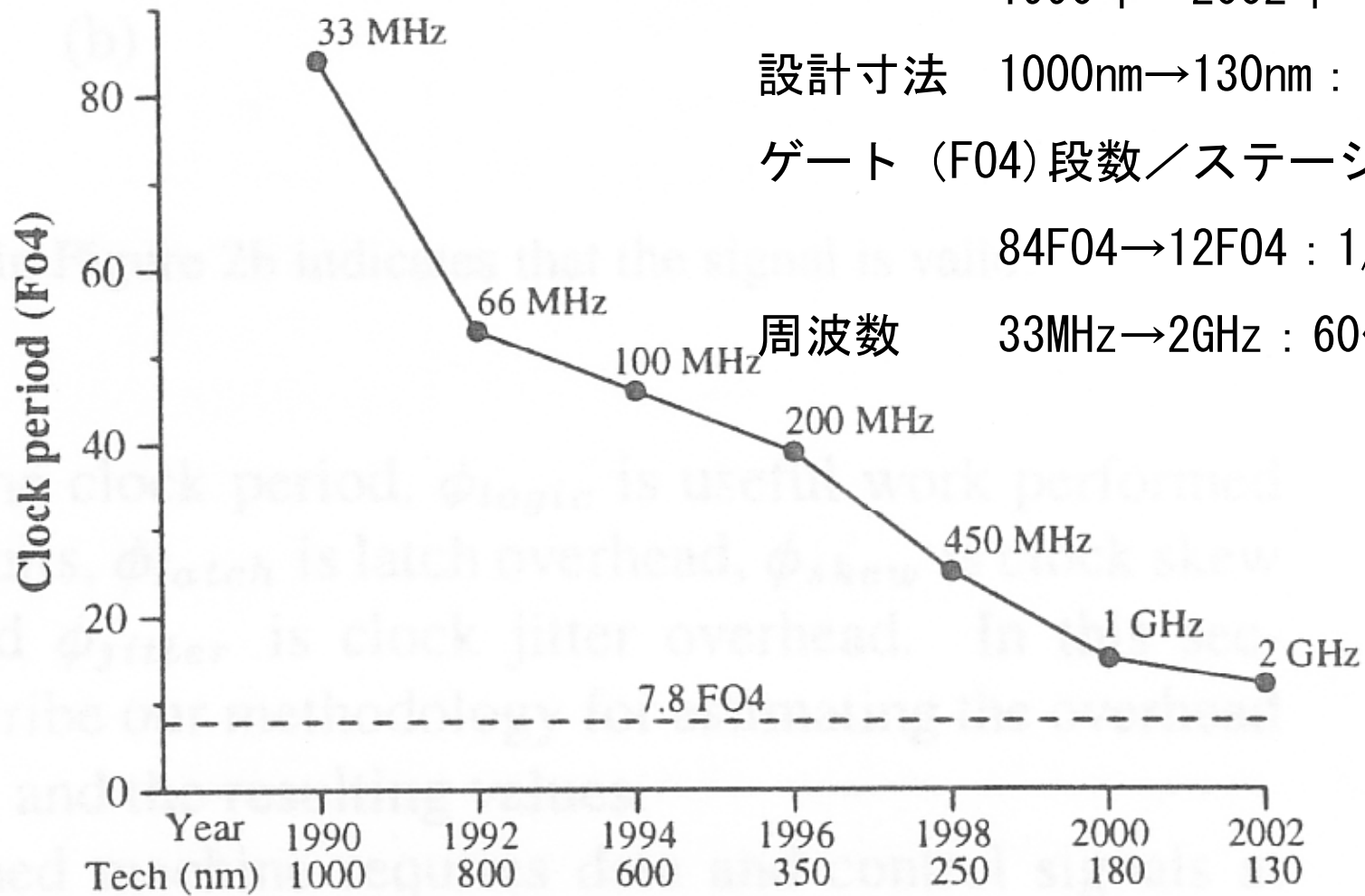
1990年 2002年

設計寸法 1000nm→130nm : 1/8

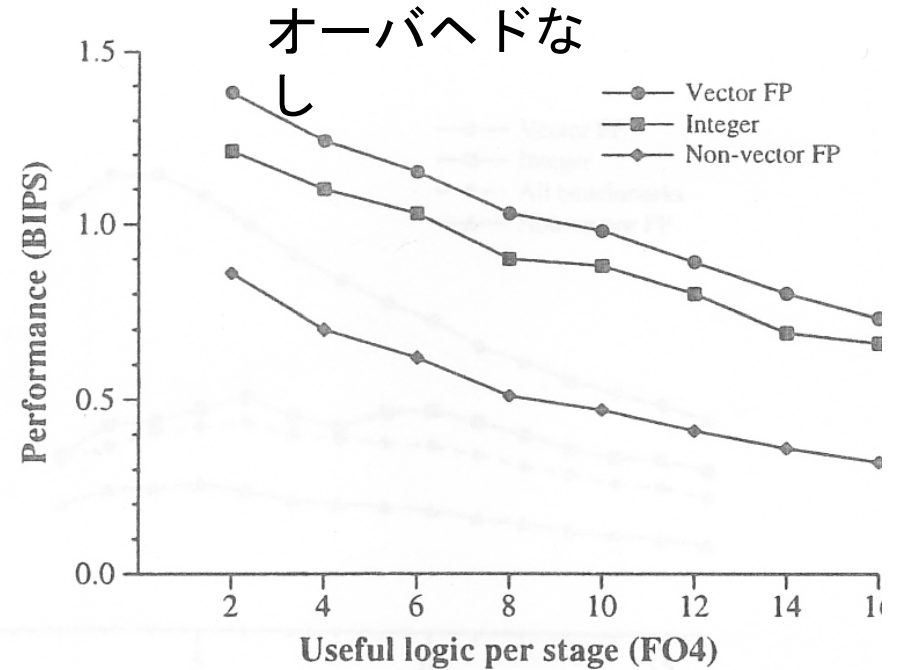
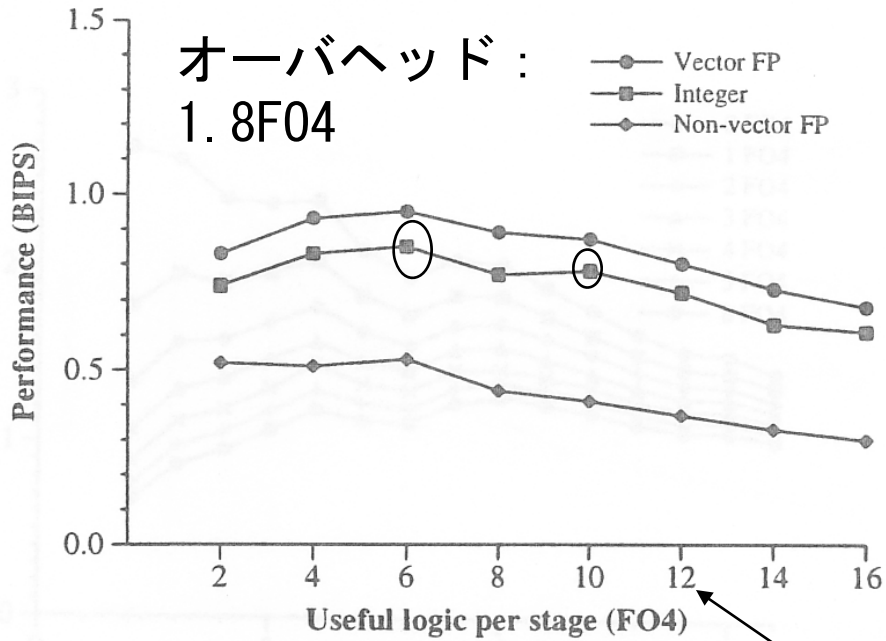
ゲート (F04) 段数/ステージ

84F04→12F04 : 1/7

周波数 33MHz→2GHz : 60倍



N. Joppi et. al., ISCA, 2002

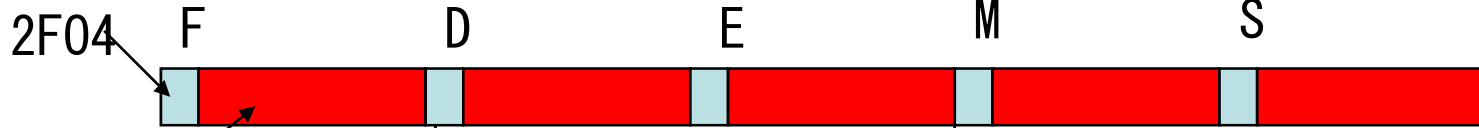


(b) 1ステージの中の有効ゲート数

10F04 → 6F04 : 9%性能向上

11.8F04 → 7.8F04 : 周波数1.5倍

オーバーヘッド



12F04 ↓ スタージ数2倍、周波数1.75倍

分岐予測ミス : 28F04 → 32F04



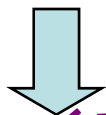
6F04

13.1汎用プロセッサの高速化・省電力化

(1) 大域並列の利用:

マルチスレッド／マルチプロセッサ

スーパースカラ、VLIW：単一プロセス／
スレッドの高速化



複数プロセス／スレッドの高速化

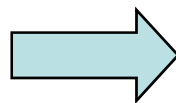
パソコン内：数10個のプロセス

Intel Hyperthreading,

NEC MUSCAT, 名大 SKY,

Wisconsin Multiscalar

IBM POWER4



オンチップマルチプロ
セッサ (CMP)

マルチスレッド型プロセッサ

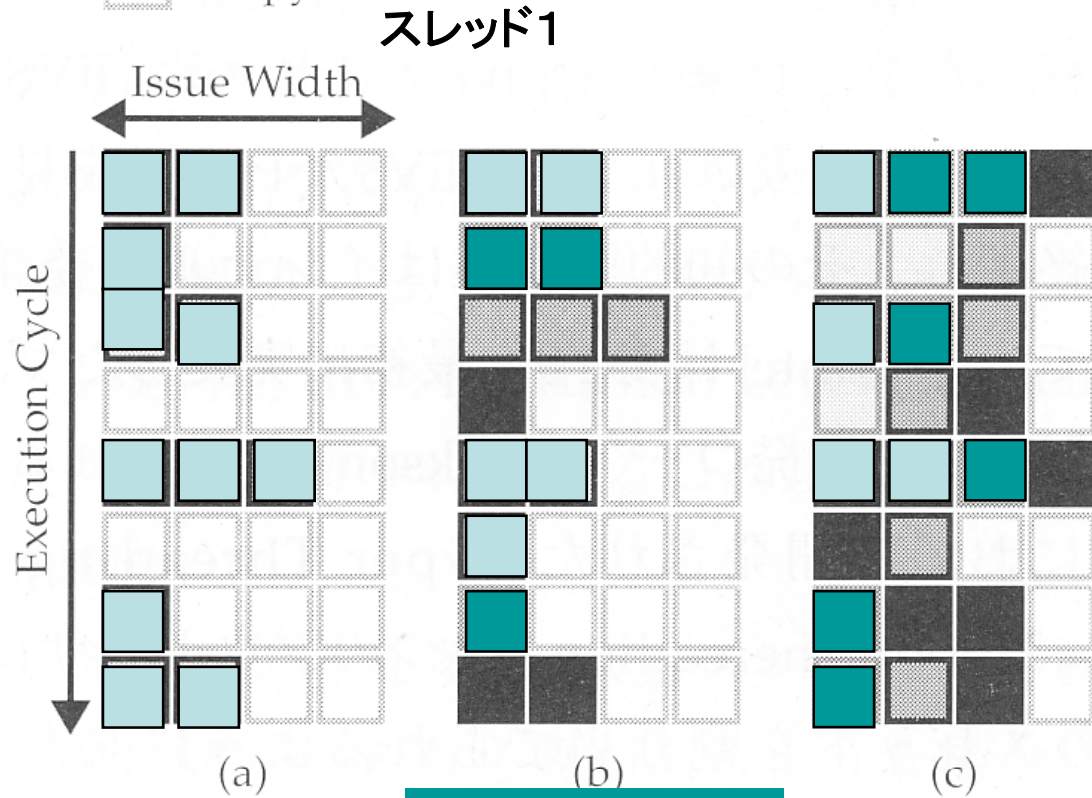
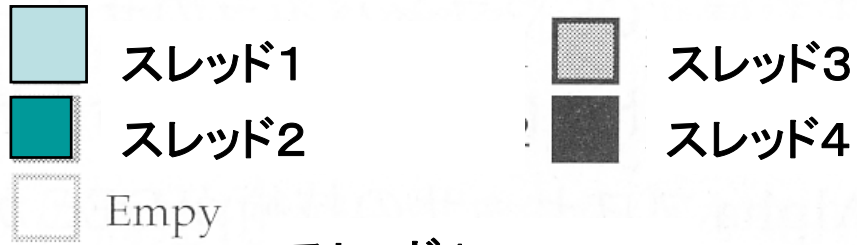
①命令パイプ共有時分割多重マルチスレッド
(粗粒度、細粒度)

Temporal Multi-Threading (TMT)

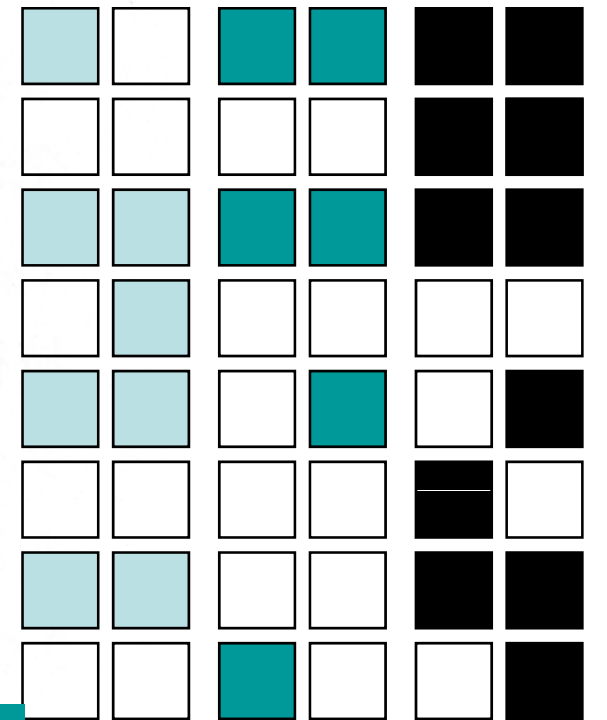
②命令パイプ共有同時多重マルチスレッド
Simultaneous Multi-Threading(SMT)

③命令パイプ多重マルチスレッド

④CMP(チップマルチプロセッサ)



軽量命令パイプライン



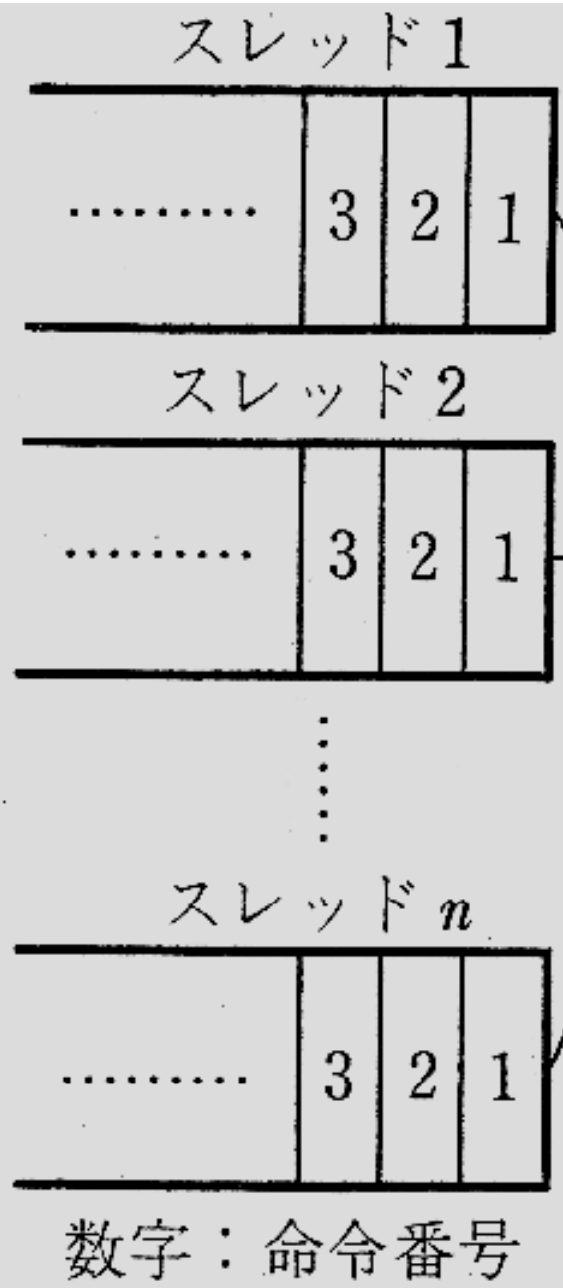
スレッド1 スレッド2 スレッド3

スーパースカラ

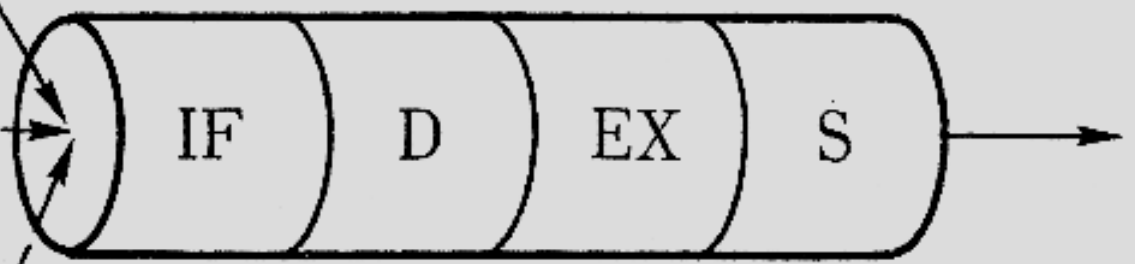
命令パイプ共有
時分割多重
マルチスレッド

命令パイプ共有
同時多重マルチ
スレッド(SMT)

命令パイプ多重
マルチスレッド



①命令パイプ共有
時分割多重マルチスレッド



命令ずつ
サイクリックに
切換え

原型：HEP (B. Smith, Tera
Computer)：1974

共有命令パイプライン方式

②SMT Simultaneous Multithreading

命令パイプ共有同時多重マルチスレッド

基本命令パイプライン：1つ
＋複数のレジスタ、PCなど
多数のスレッドの実行

Intel Hyper Threading

15-25%性能向上、
チップサイズ5%増
2スレッド実行

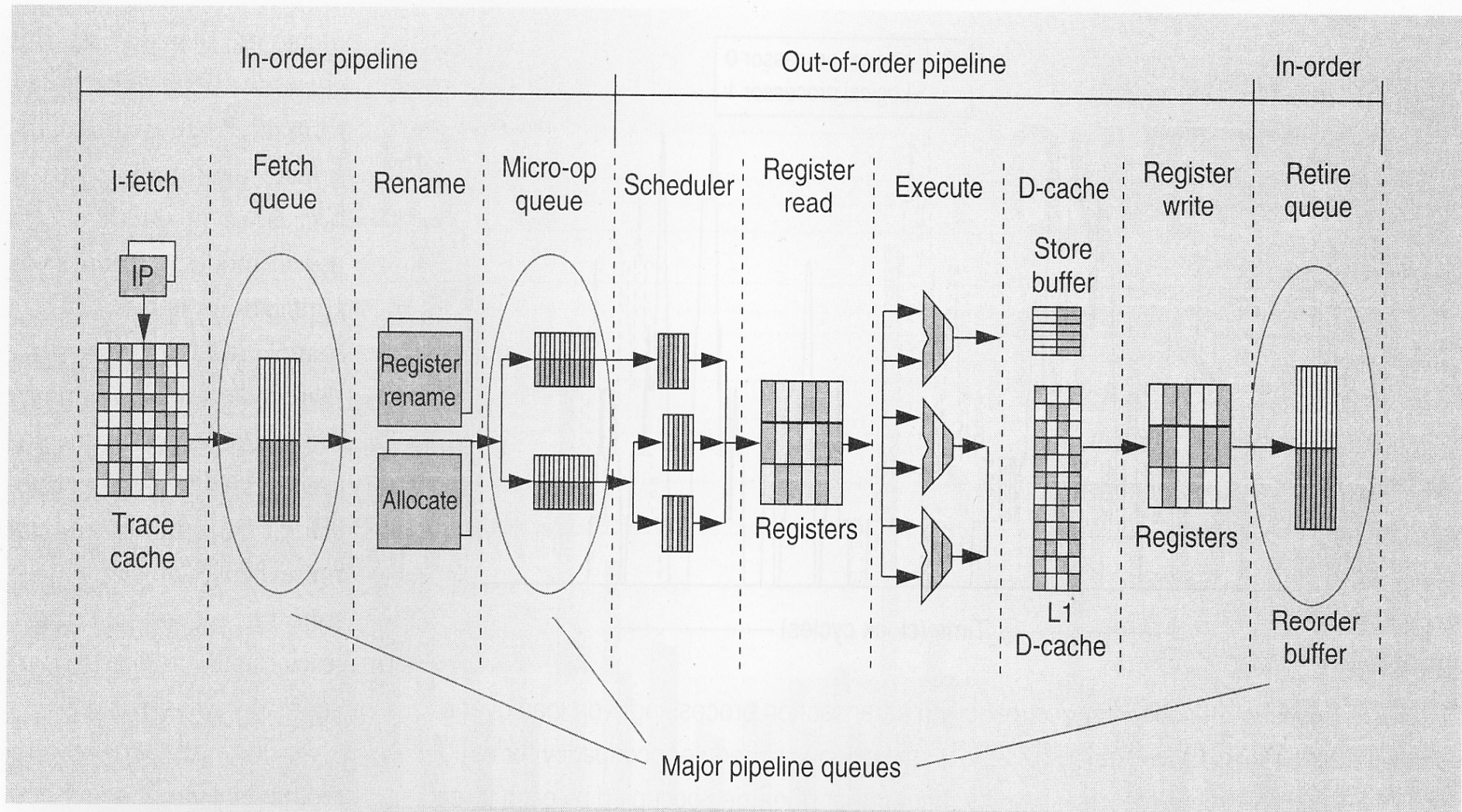


Figure 4. In this view of a Netburst microarchitecture's execution pipeline, the light and dark areas indicate the resource utilization of the two software threads running on the two logical processors.

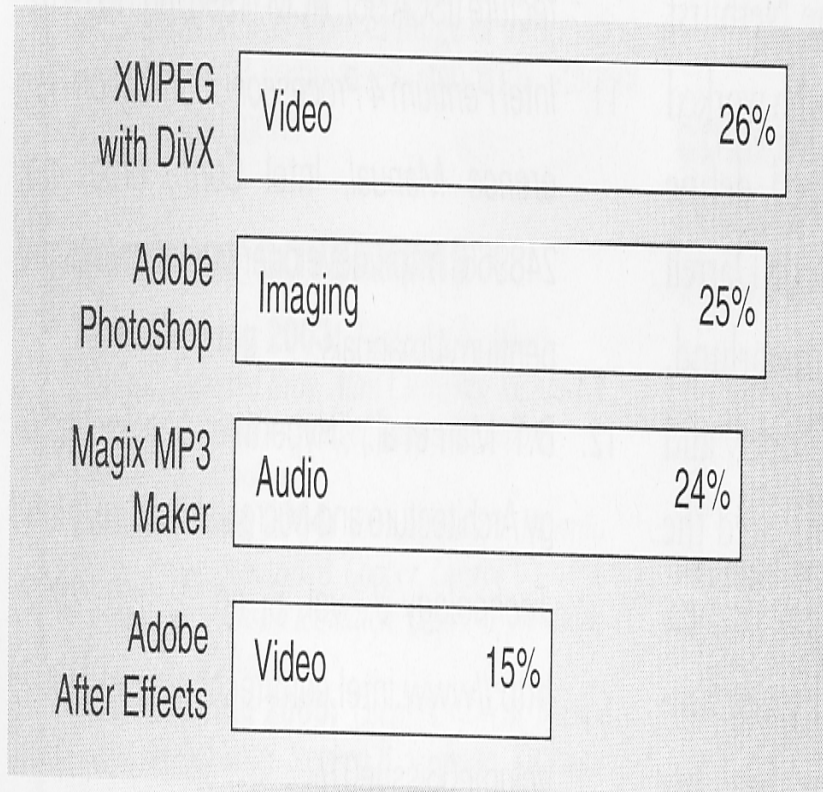


Figure 7. Hyperthreading technology performance gains on several popular multithreaded software packages.

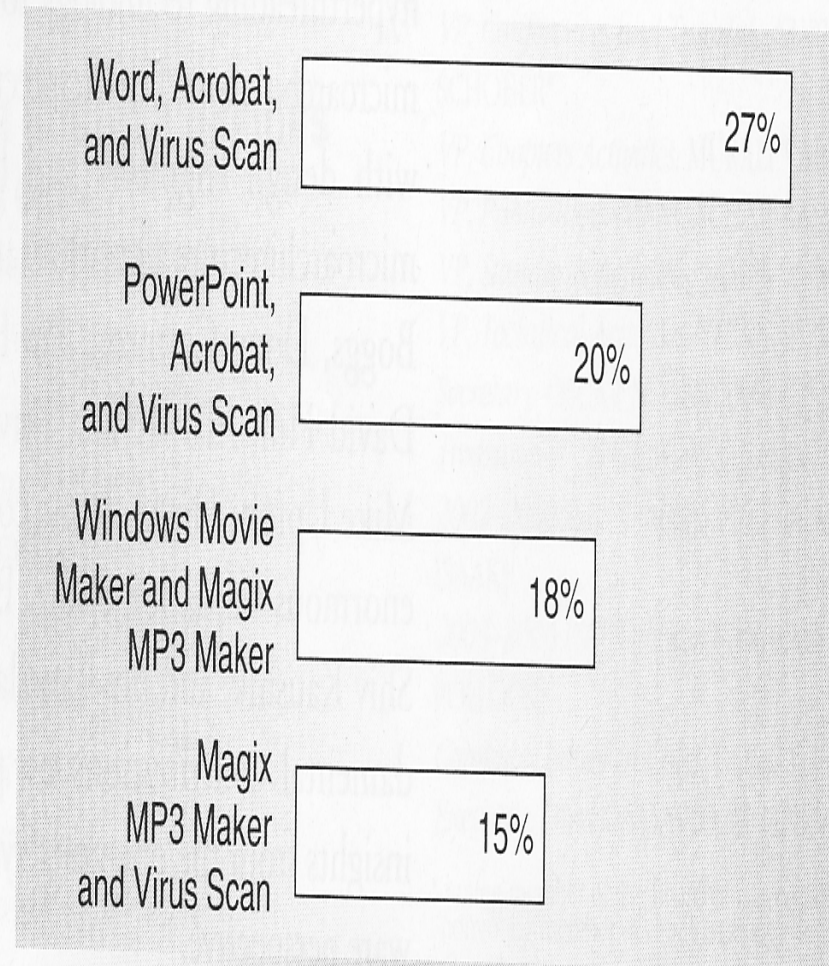
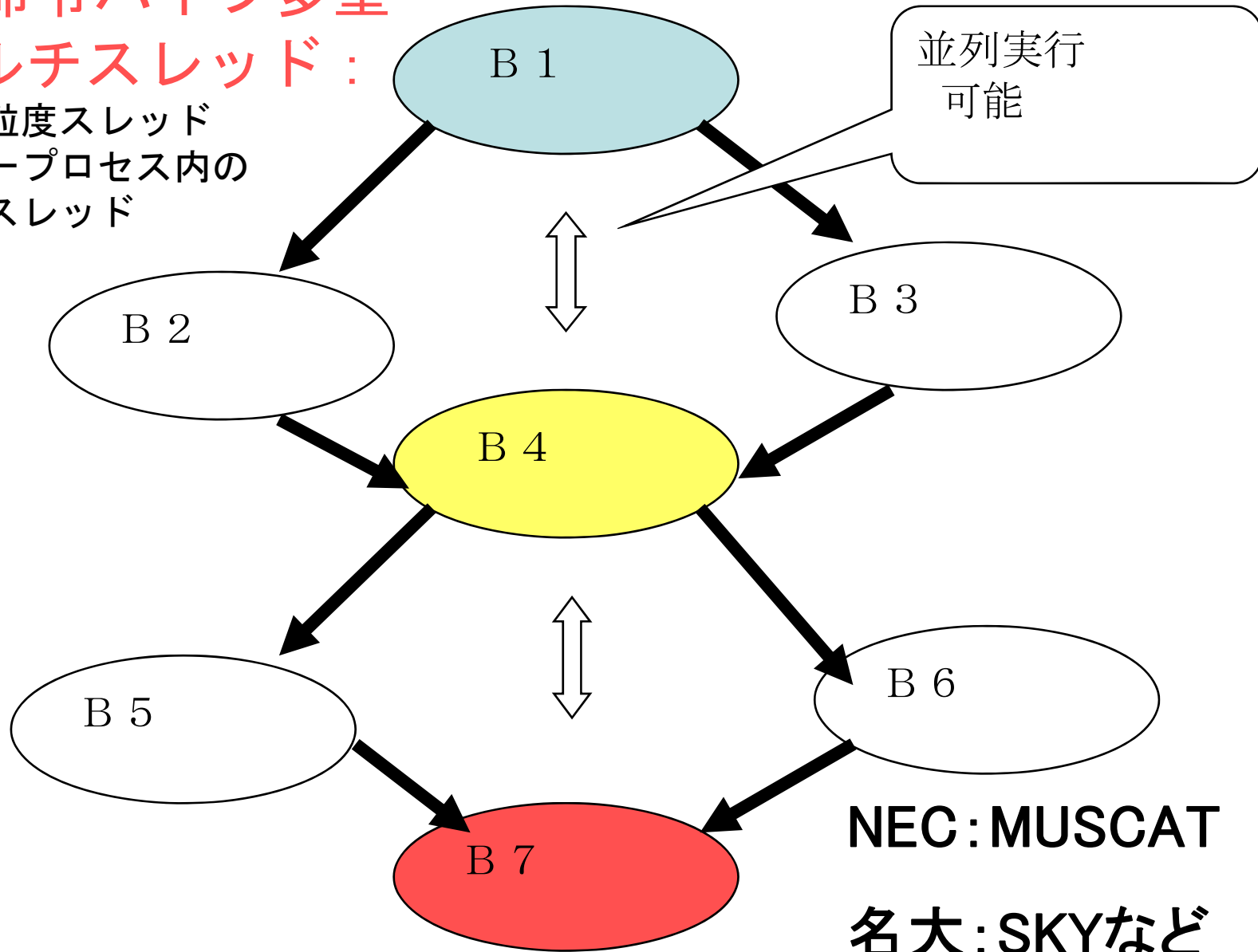
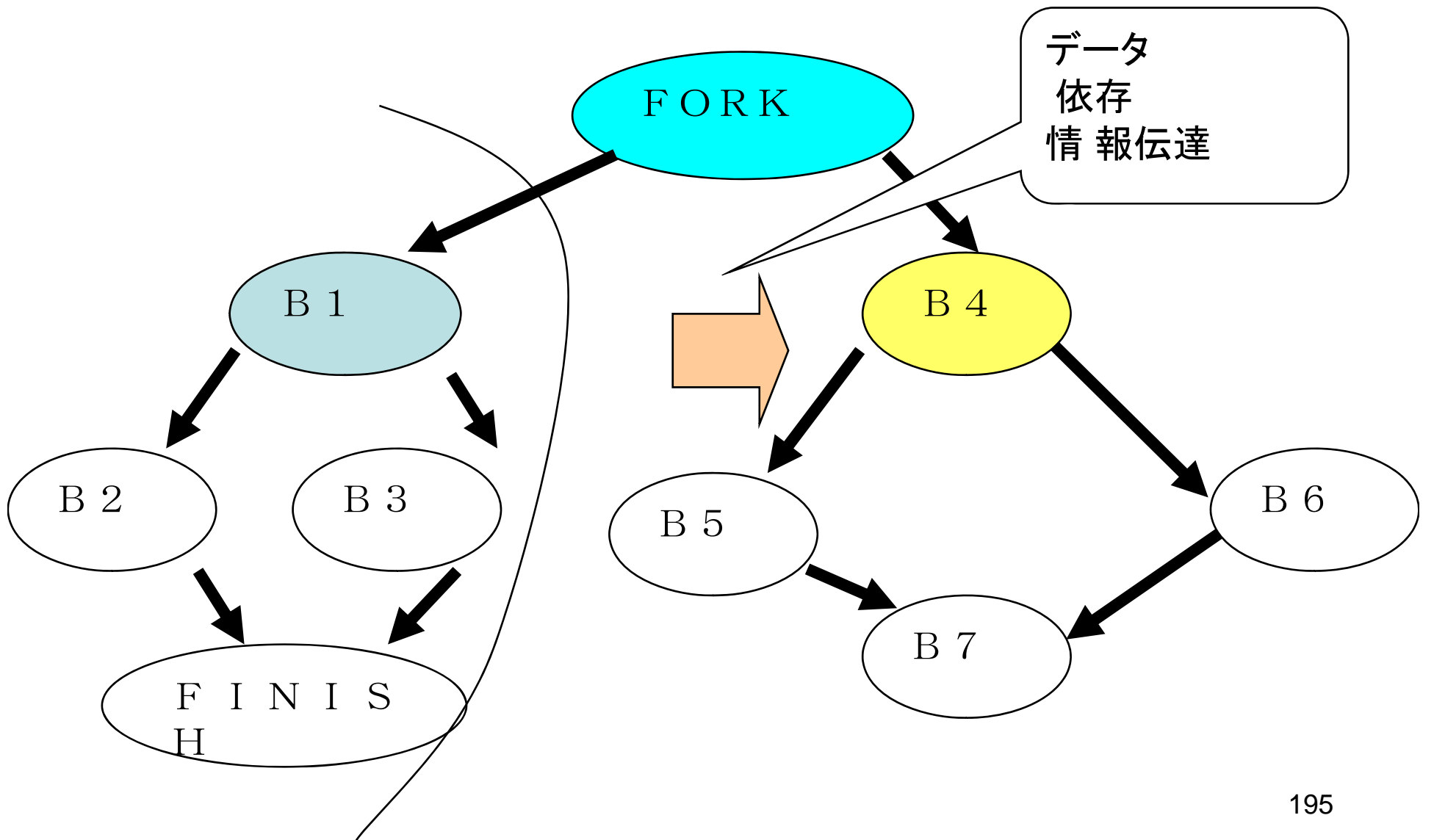


Figure 8. Hyperthreading technology performance boost on multitasking workloads.

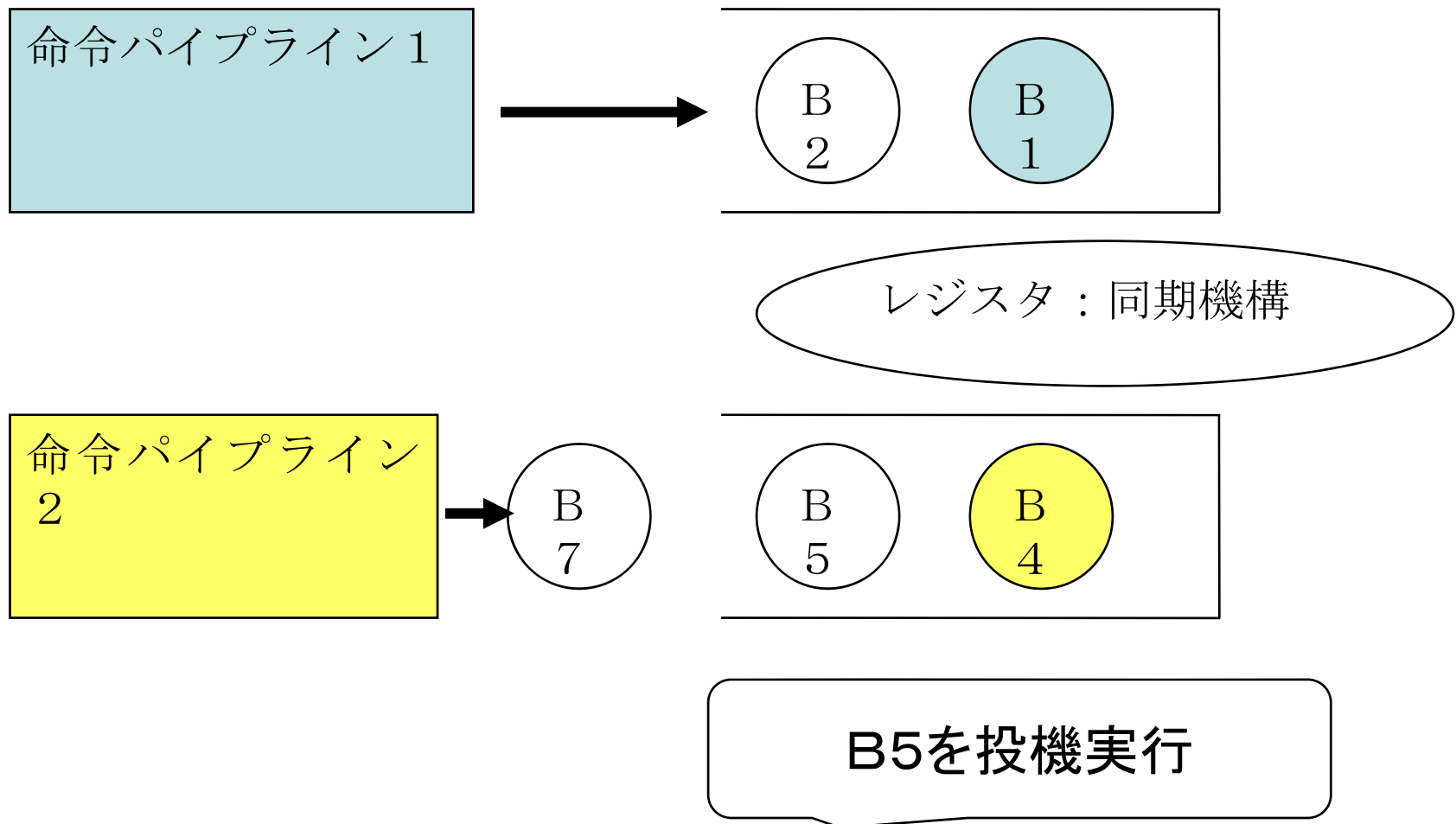
③命令パイプ多重 マルチスレッド:

細粒度スレッド
単一プロセス内の
スレッド





命令パイプ多重マルチスレッド実行



NEC MUSCAT

表 2: シミュレーションパラメタ

項目	パラメタ
各 PE の構造	
パイプライン	IF, ID, Issue/Reg, EX, WB, Graduate Issue/Reg から WB まで out-of-order 実行
命令ウィンドウ	PE 毎 32 命令 (整数 / ロードストア各 16)
演算リソース	ALU 2×PE 数 L/S パイプ 1×PE 数
ロード / ストア	3 サイクルレイテンシ
スーパスカラ度	4 命令同時デコード / 終了
分岐投機	4 分岐まで仮実行
分岐履歴	2048entry 4 状態 (PE 間共有)
キャッシュ	
キャッシュ方式	命令 / データ分離
キャッシュ容量	各 32Kbyte (64byte × 512entry)
マッピング方式	4Way Set Associative LRU 追い出し

鳥居、近藤、本村、西、小長谷: On Chip Multiprocessor 指向制御並列アーキテクチャMUSCATの提案、JSPP97、pp.229-236、1997

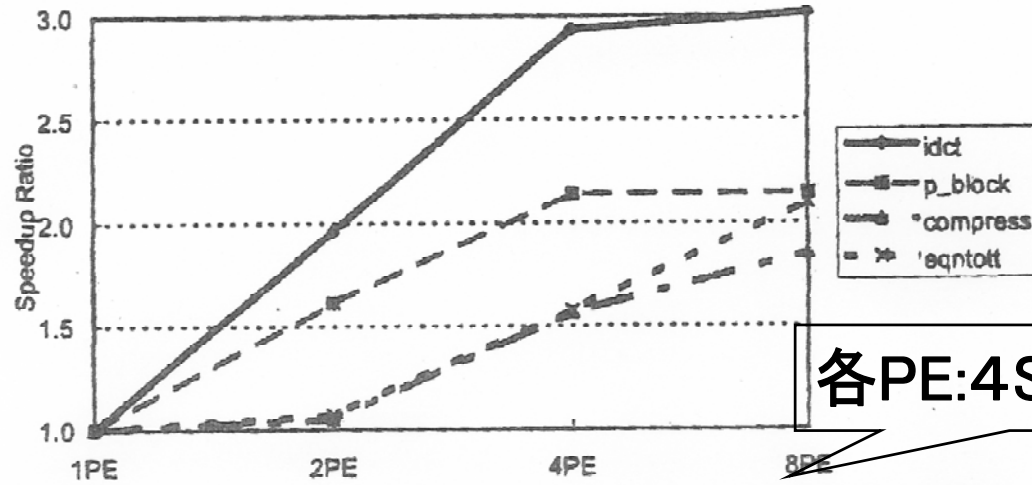


図 9: MUSCAT の性能向上率

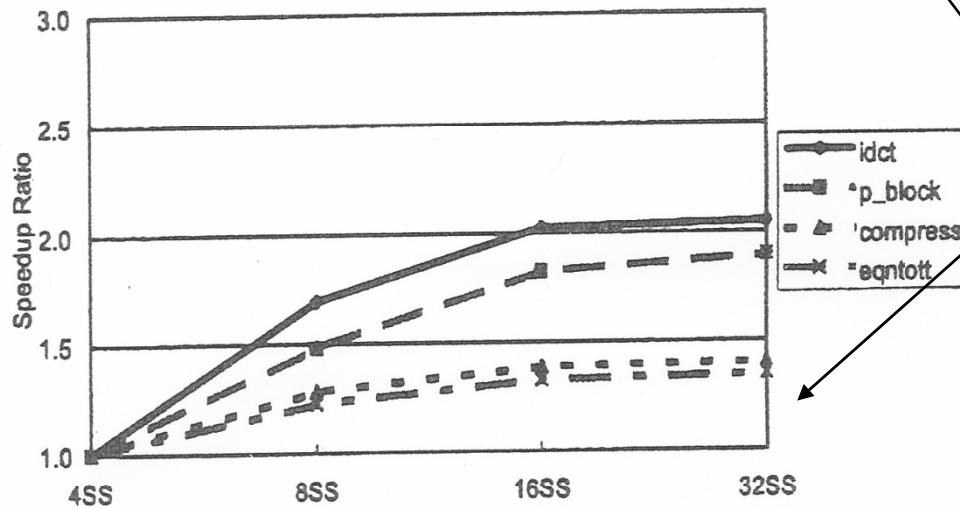


図 11: スーパスカラ強化モデルの性能向上率

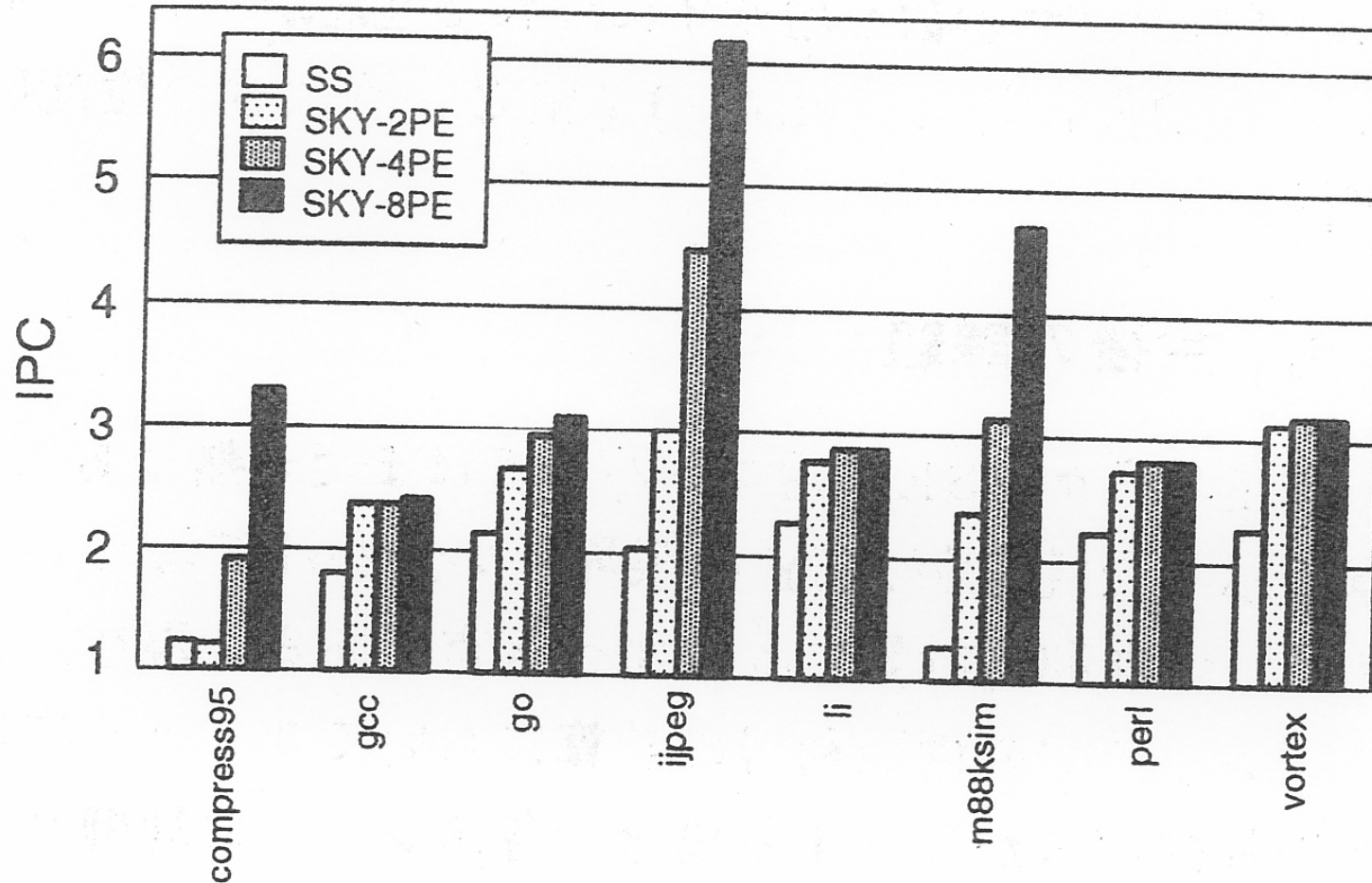


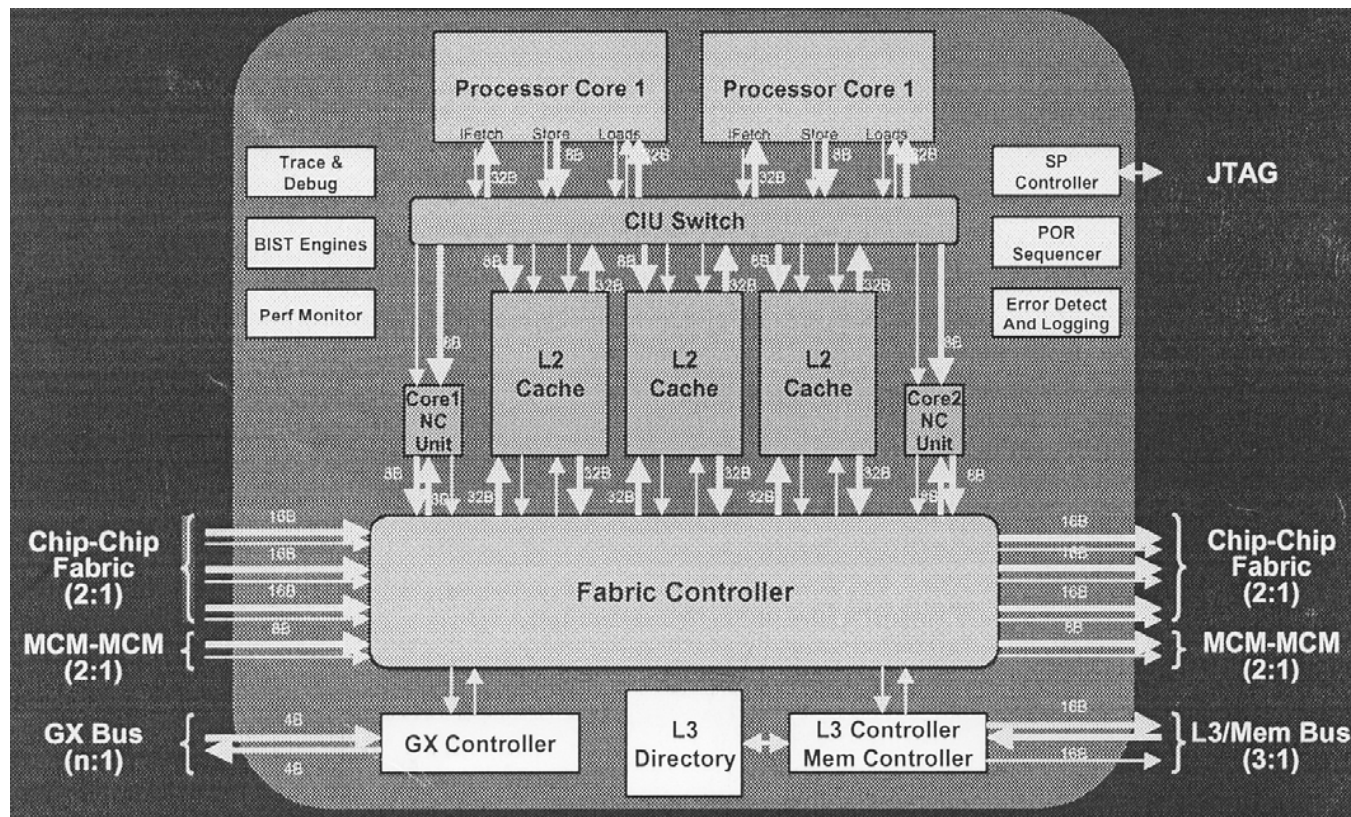
図 6 SKY の台数効果

SS:8 多重

小林、岩田、安藤、島田：非数値計算プログラムのスレッド
間命令レベル並列を利用するプロセッサアーキテクチャSKY、
JSP98、pp. 87-94、1998

④ オンチップマルチプロセッサ CMP

- IBM POWER 4 : 2 台のSMP、スヌープキャッシュ
- MCM : 8 台までのSMP、最大構成 : 4 MCM (3 2 台)
- L1 (I: 6 4 KBx2, D: 3 2 KBx2) : 2 状態、L2 (1. 5 MB) 7 状態、L3 (3 2 MB) : 5 状態



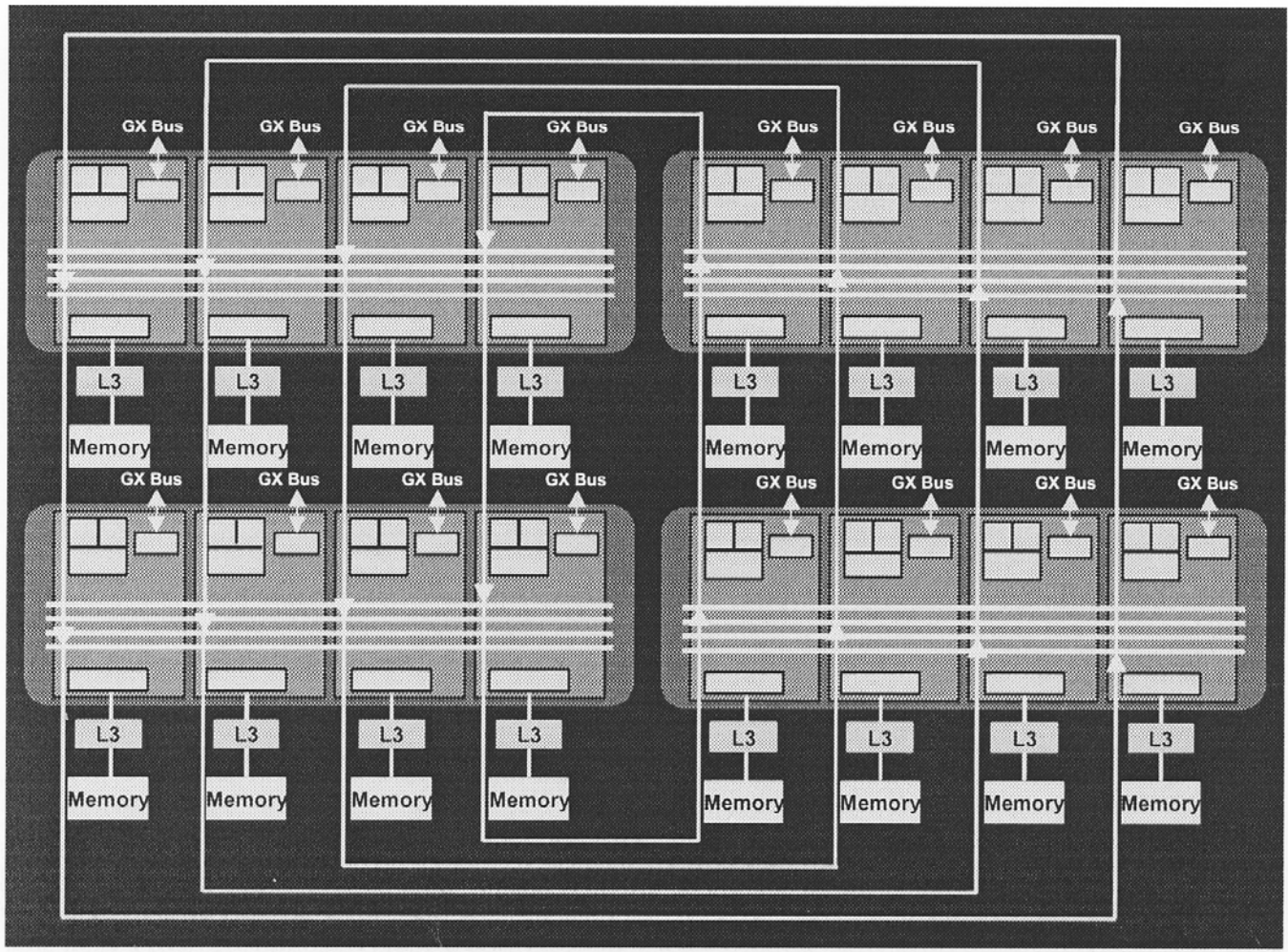


Figure 9: Multiple POWER4 multi-chip module interconnection

▪ Pentium EE840

デュアルコアプロセッサ

各プロセッサ：ハイパースレッディング

 4個のプロセッサ

90nmデザインルール

3.2GHz

各プロセッサ

L1データキャッシュ：16KB

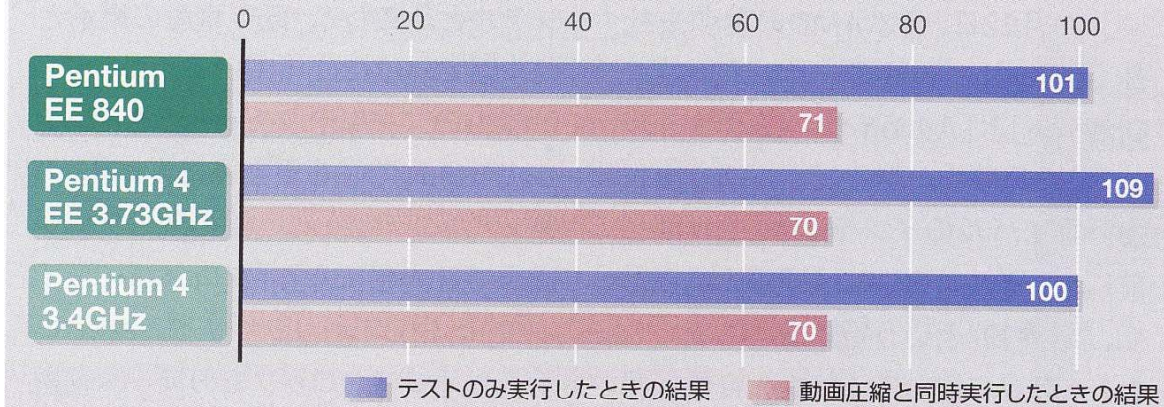
L1トレースキャッシュ12KB

L2キャッシュ：1MB

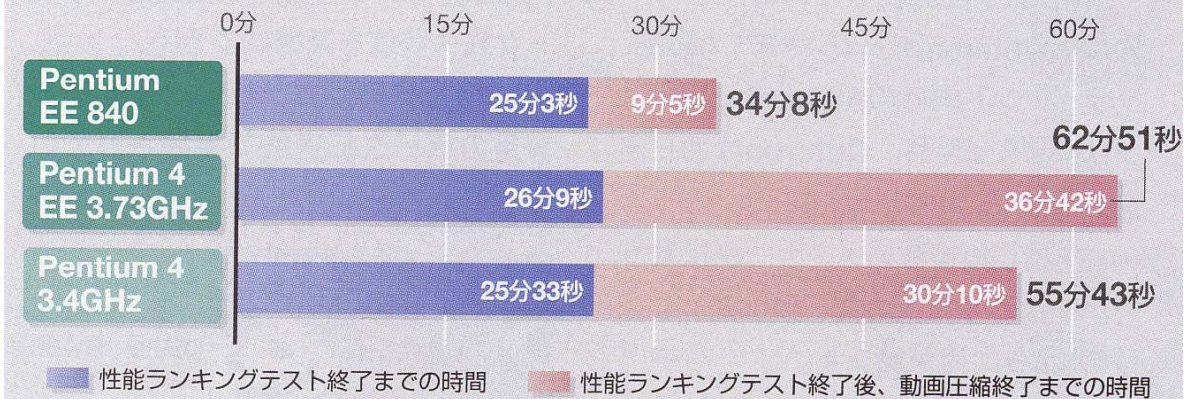
SSE3

●複数ソフトの同時実行に威力を発揮

性能ランキングテストの結果（標準機の結果を100とした相対値）



ソフト同時実行テストの処理時間



デュアルコアはテスト単体を動かしたときは、その真価は発揮されない。複数のソフトを同時実行すると、全処理時間がほかよりも短くなる。いろいろなソフトを同時に動かしたいユーザーにお勧めだ

テストの詳細

Pentium EE 840機: CPUはPentium EE 840 (3.2GHz)、メモリーはDDR 2 667対応1GB、HDDは7200回転/分の250GB、グラフィックスはRADEON X850 XT 256MB

Pentium 4 EE機: CPUはPentium 4 EE (3.73GHz)、他はPentium EE 840機と同じ

Pentium 4機: CPUはPentium 4 (3.4GHz)、メモリーはDDR 2 533対応1GB、HDDは7200回転/分の300GB、グラフィックスはRADEON X600Pro 128MB

上の図は性能ランキングテストのみと、動画圧縮をしながら性能ランキングテストを動かしたときの結果。下の図は動画圧縮しながら性能ランキングテストを動かしたときにかかった時間

日経パソコン
2005. 5. 9

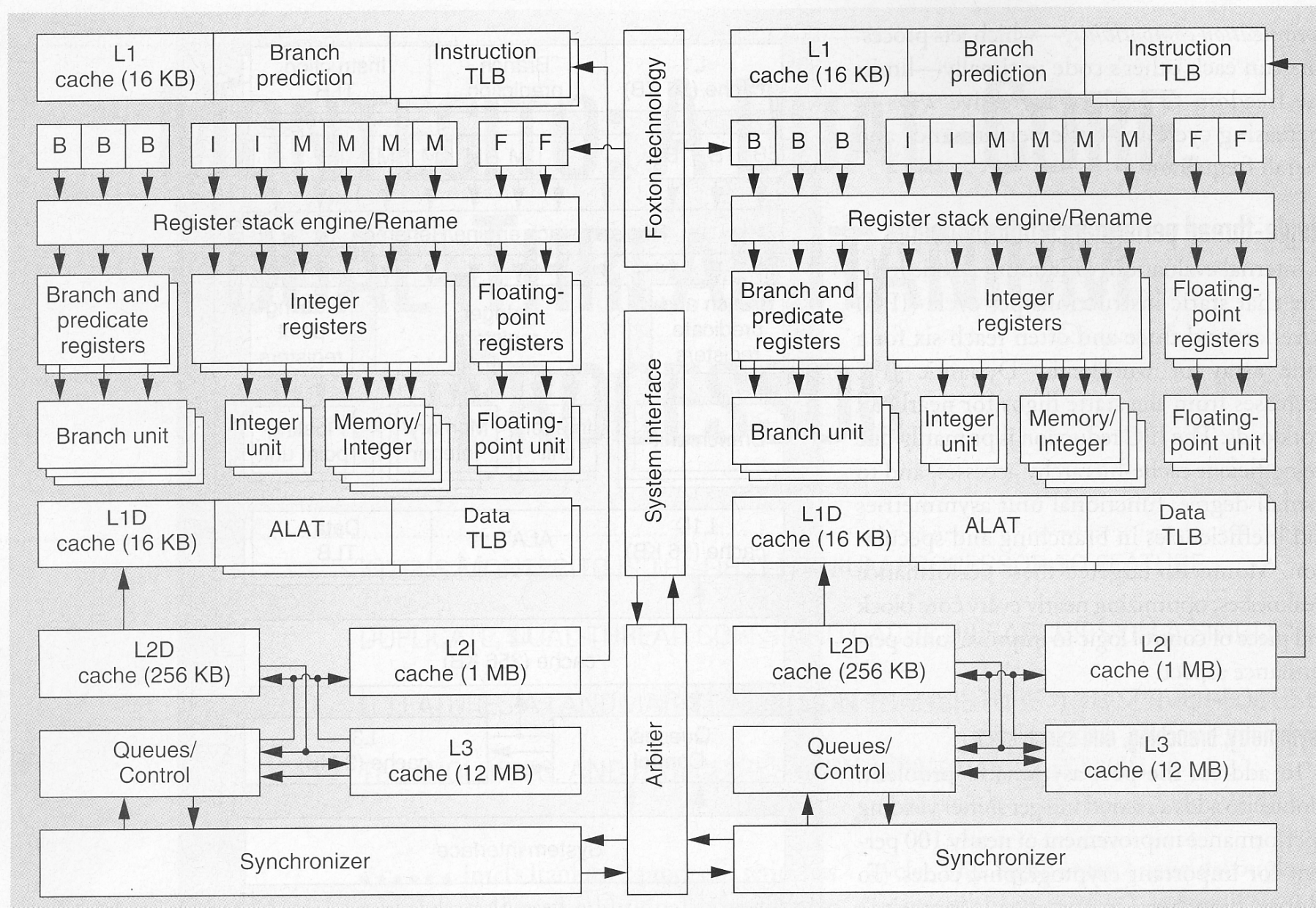


Figure 2. Block diagram of Intel's Montecito. The dual cores and threads realize performance unattainable in the Itanium 2 processor. Montecito also addresses Itanium 2 port asymmetries and inefficiencies in branching, speculation, and cache hierarchy.

Itanium2-Montecito

2個のItanium2プロセッサ、2スレッド
時分割多重マルチスレッド命令パイプ共有 (TMT)
各クロセッサ

L1キャッシュ16KB、1サイクル

L2キャッシュ1MB、5サイクル

L3キャッシュ12MB、14サイクル

スレッドスイッチ：15サイクル (粗粒度TMT)

L3キャッシュミス、タイムアウトなど5つの事象

1.72億TR

消費電力：100W

周波数：1.8GHz

IEEE Micro
Vol25 No2 2005

SPARC : Niagara

1 多重で 6 ステージの SPARC プロセッサ 8 台
各プロセッサ : 4 スレッド

時分割多重マルチスレッド命令パイプ共有
(細粒度TMT)

L1 キャッシュ : 8 KB、Write Through

L2 キャッシュ : 3 MB、4 バンク、クロスバSWで共有

対応するバンクにL1キャッシュディレクトリのシャドウ
命令レベル並列が小さく、

スレッドレベル並列の大きなWEBやデータベース応用向き

Table 1. Commercial server applications.

Benchmark	Application category	Instruction-level parallelism	Thread-level parallelism	Working set	Data sharing
Web99	Web server	Low	High	Large	Low
JBB	Java application server	Low	High	Large	Medium
TPC-C	Transaction processing	Low	High	Large	High
SAP-2T	Enterprise resource planning	Medium	High	Medium	Medium
SAP-3T	Enterprise resource planning	Low	High	Large	High
TPC-H	Decision support system	High	High	Large	Medium

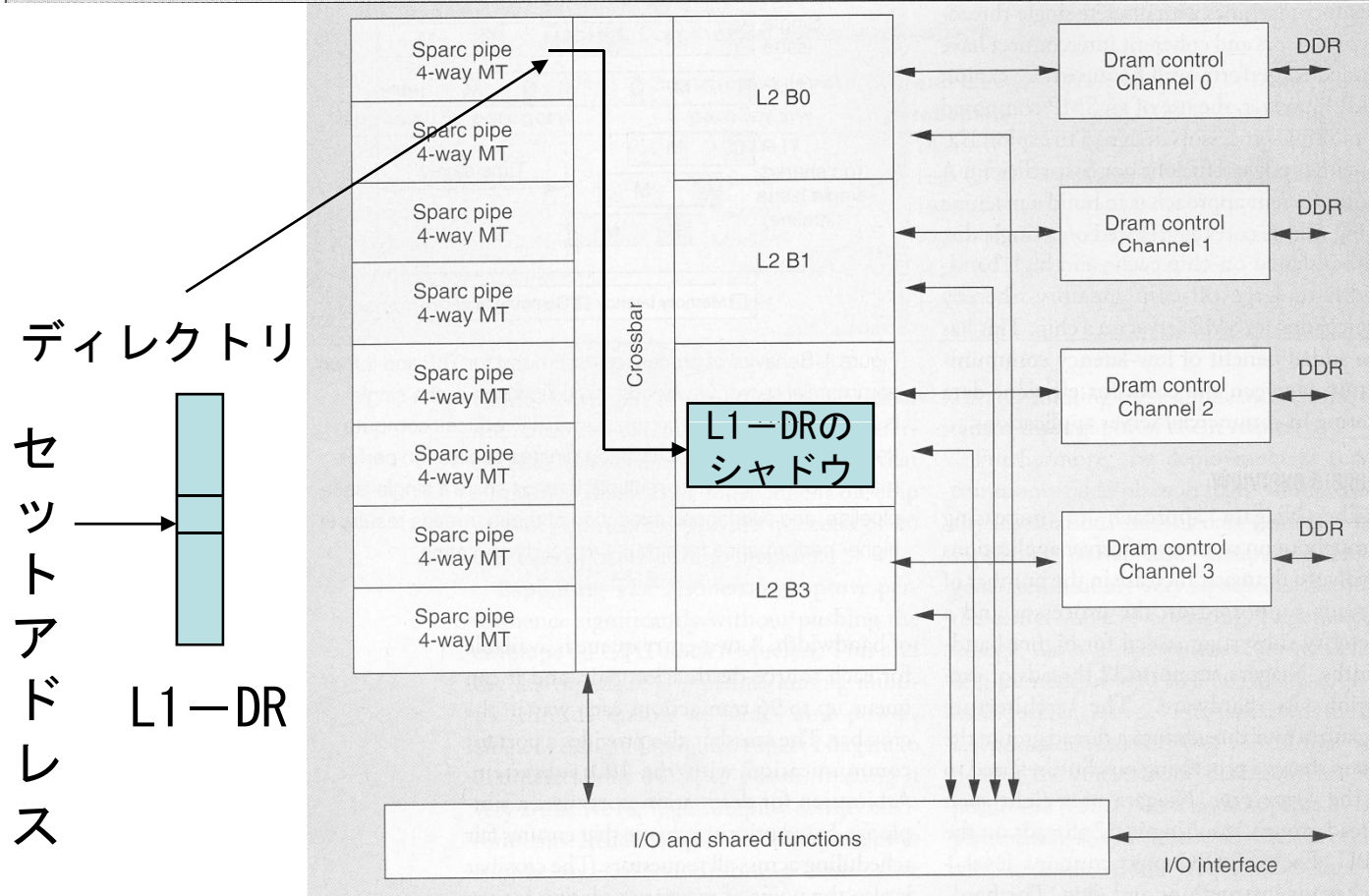
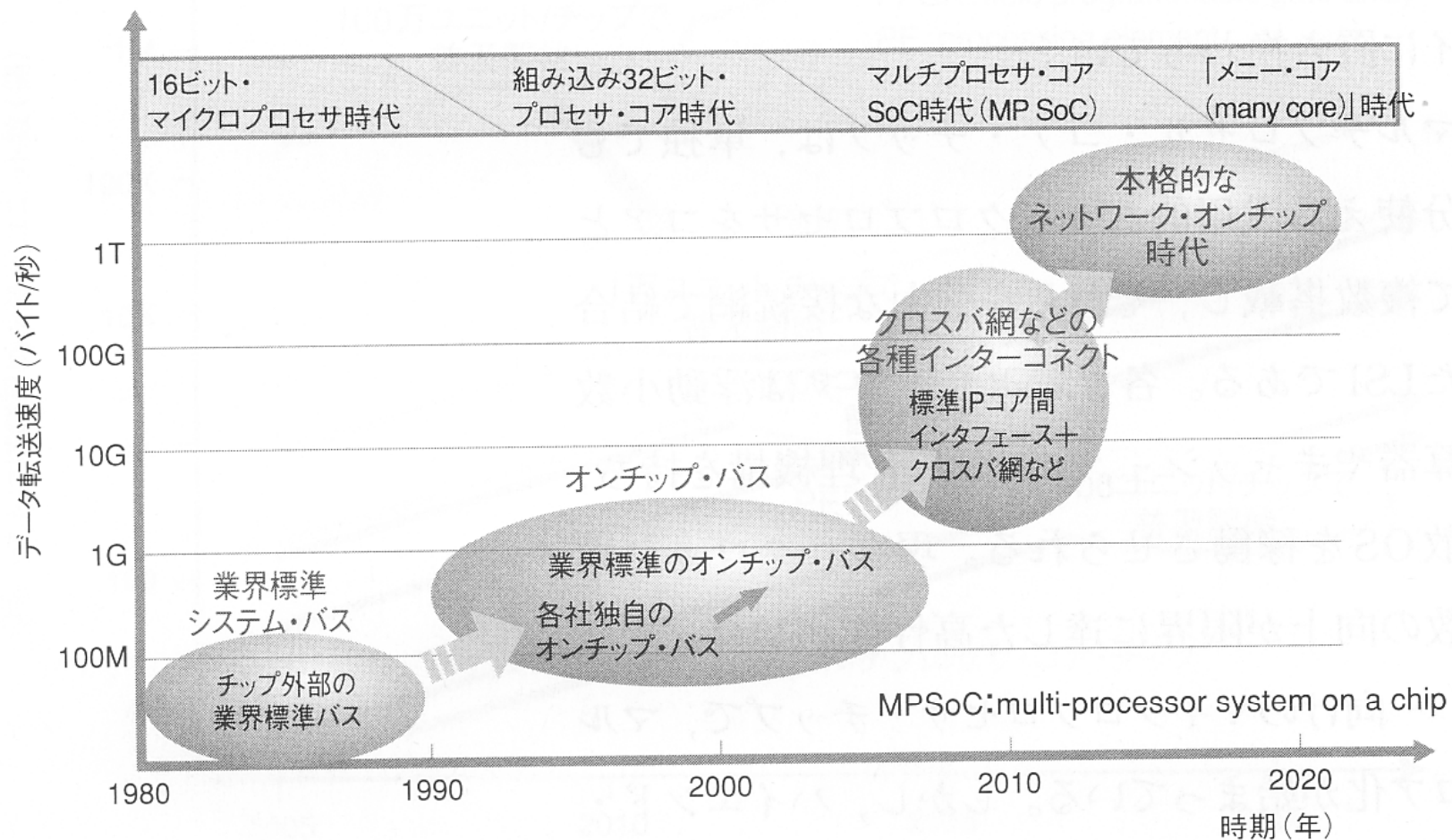


Figure 2. Niagara block diagram.



森：日経Micro devices, No242, 2005

図1 ● チップ内通信のデータ転送速度が向上

チップに集積できるプロセッサの演算能力の向上にあわせて、チップ内通信のバンド幅が拡大する。それを実現するために、通信手段が変わる。著者のデータ。

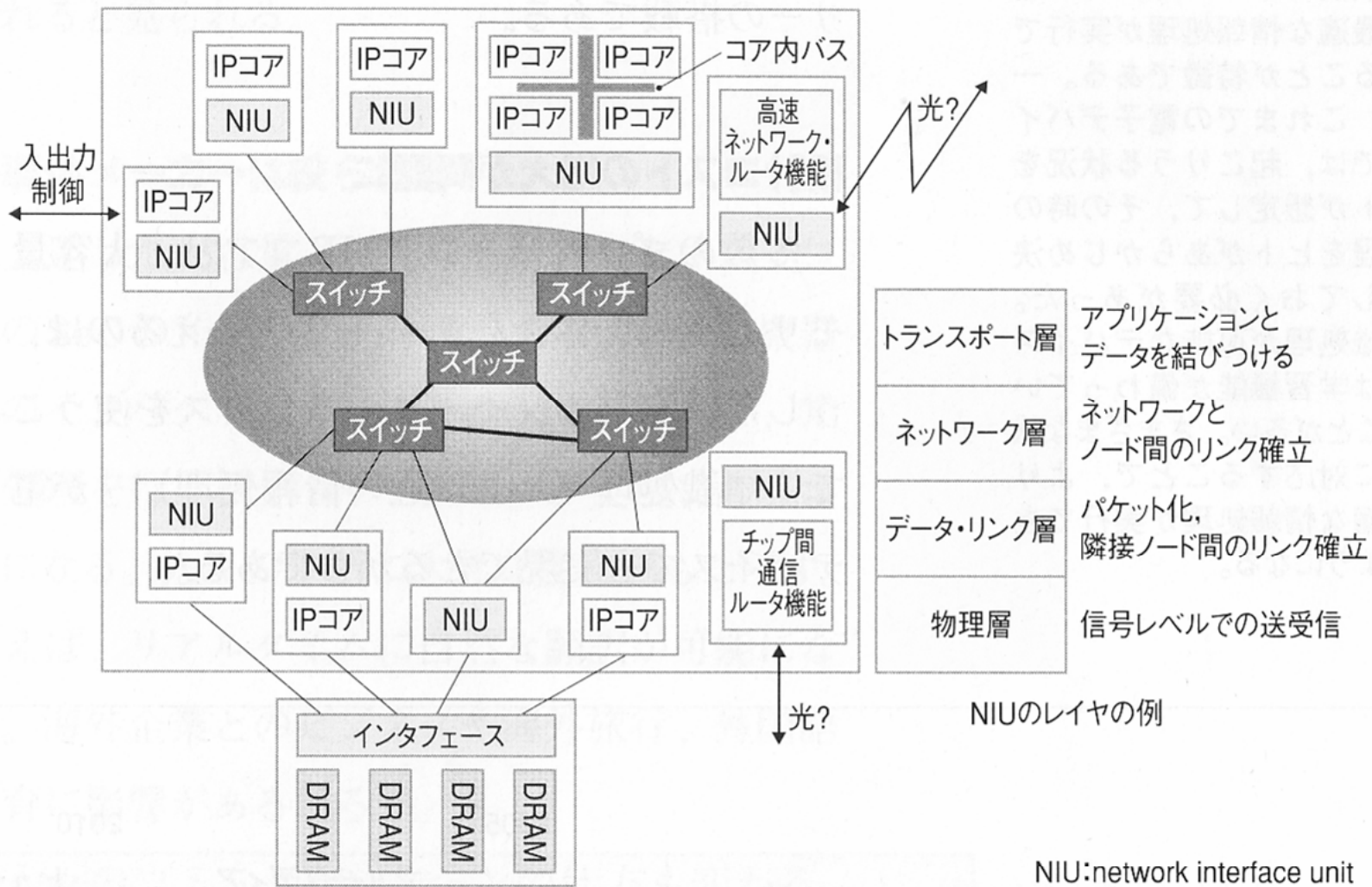


図3 ● IPコアの通信機能が向上

2015年頃になると、チップに多数のプロセサ・コアやIPコアが載る。各IPコアが通信機能を担う、NIU（network interface unit）を内蔵するようになる。著者のデータ。

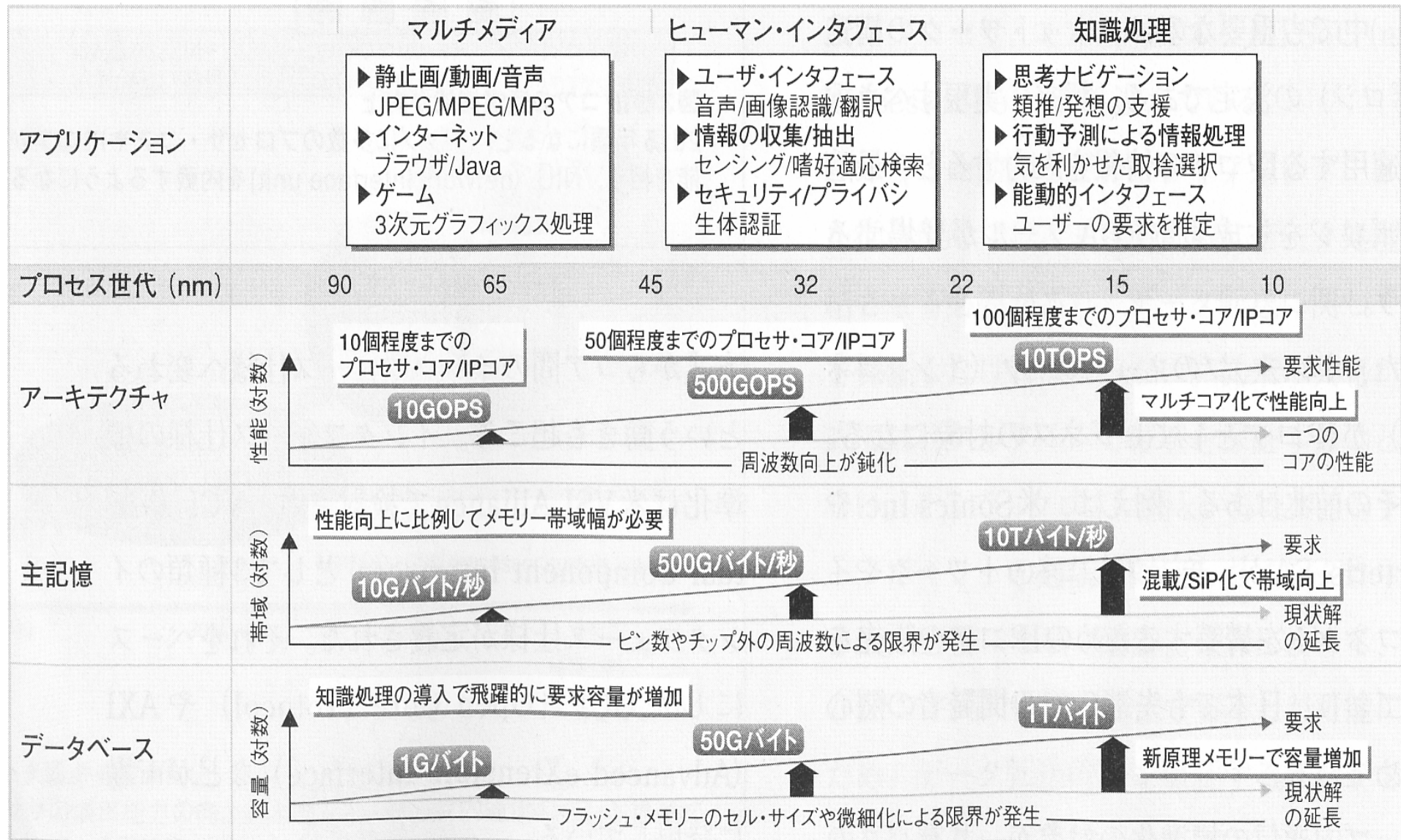
時期(年)
2020

2005

2010

2015

2020



川崎、小沢：日経Micro devices, No242, 2005

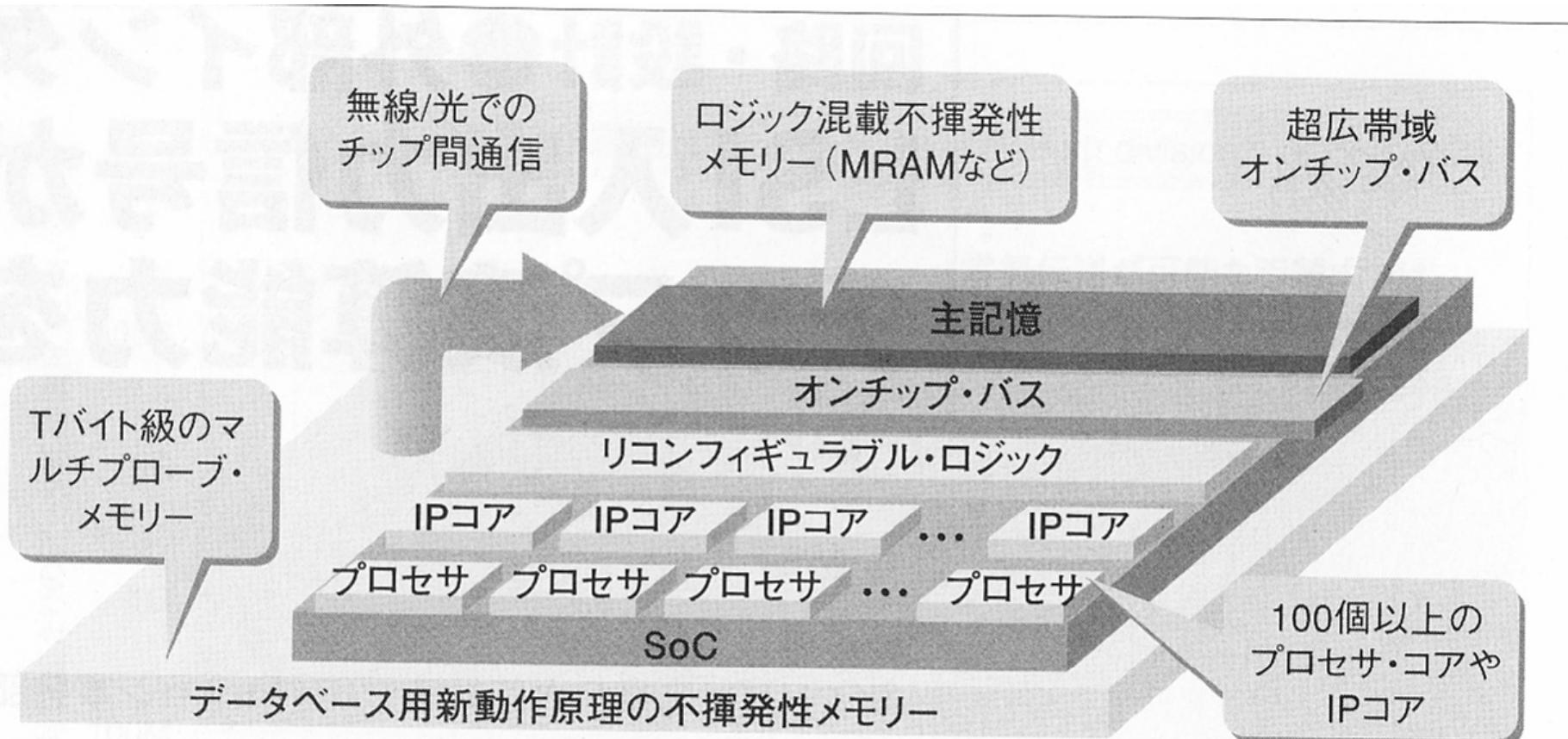


図2 ● 2015年のSoCアーキテクチャ

ローカル (SoC 付近) に必要なデータベースの実現には、マルチプローブ・メモリーなどの次世代不揮発性メモリーが必要になる。著者のデータ。

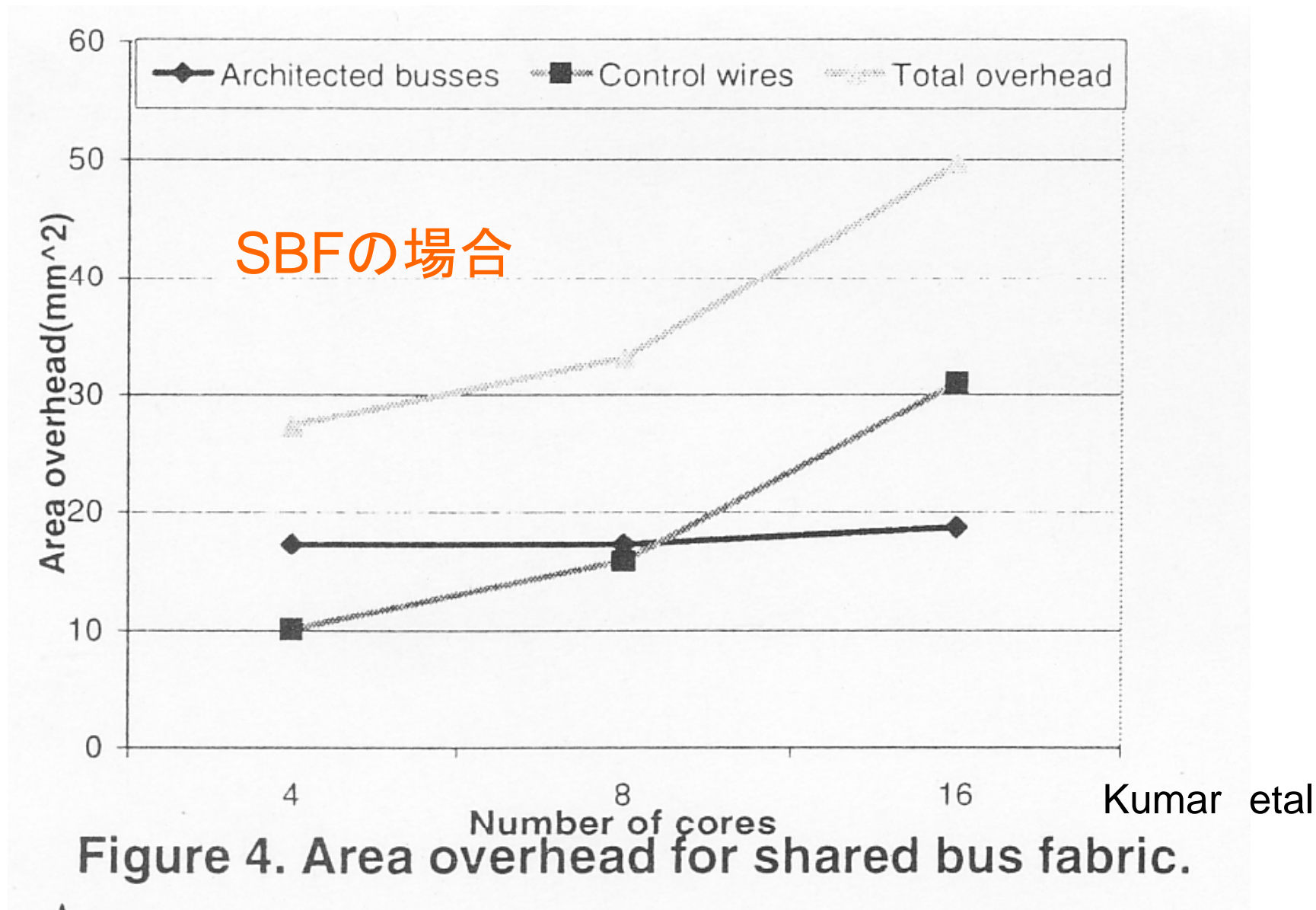
マルチコアの相互結合網

評価項目

- ①スループット
- ②レイテンシ
- ③消費電力： スイッチ＋配線
- ④面積
- ⑤配線層数

・P.P.Pande et al: Performance Evaluation and Design Trade-offs for Network-on-Chip Interconnect Architectures, IEEE Trans. Computers, Vol54, No.8,pp.1025-1040,2005

・R.Kumar et al: Interconnections in Multi-core Architectures: Understanding Mechanisms, Overhead, and Scaling, pp.408-419, ISCA, 2005

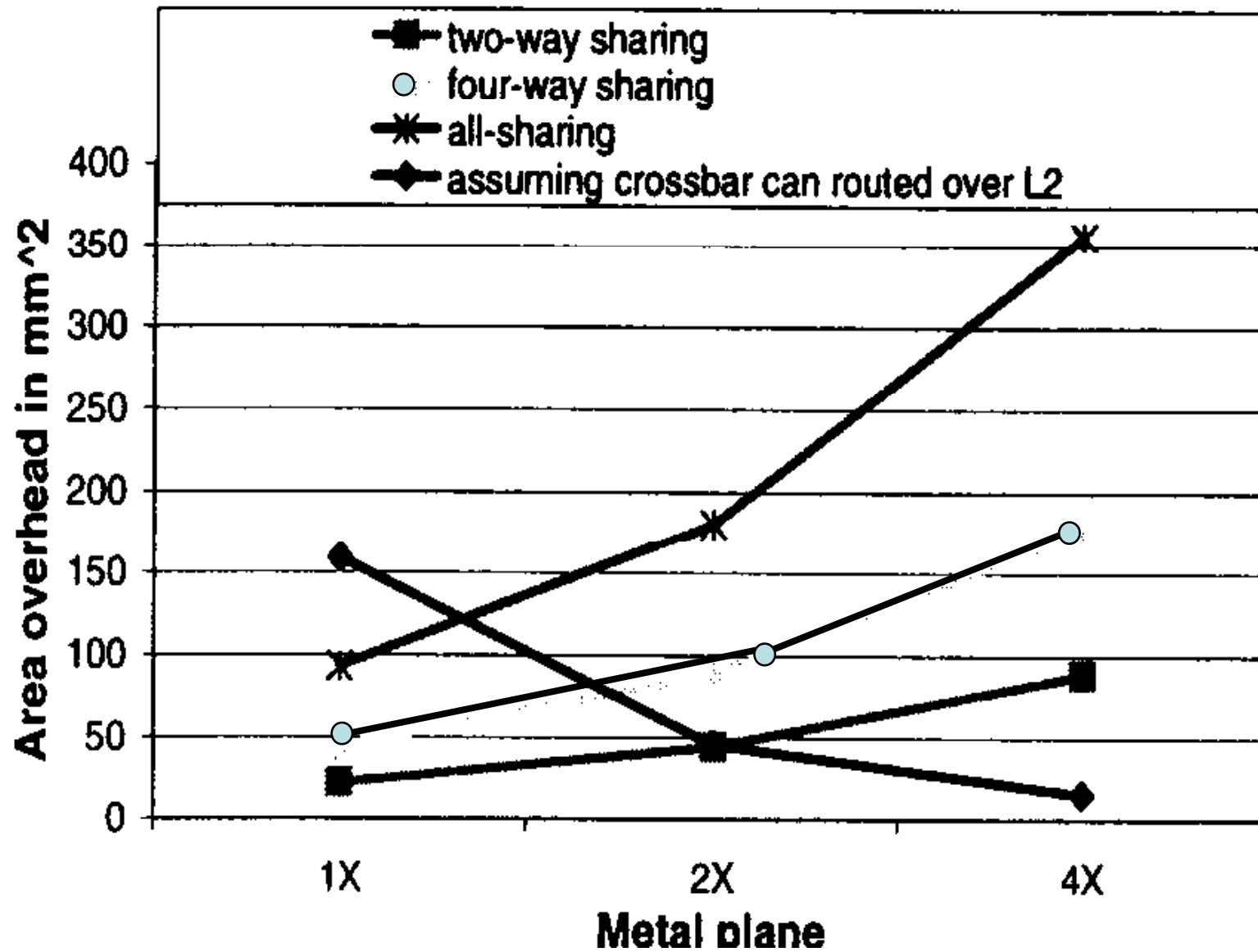


16コアで配線面積50mm² ⇔65nmデザイン、ダイサイズ: 400mm²、1コア: 10mm²、L2キャッシュ: 0.125MB/mm²を仮定

213

ロジック(L2の下): 5.6mm²(4コア)、8.6mm²(8コア)、17.94mm²(16コア)

8コアX8L2キャッシュバンクのクロスバススイッチ



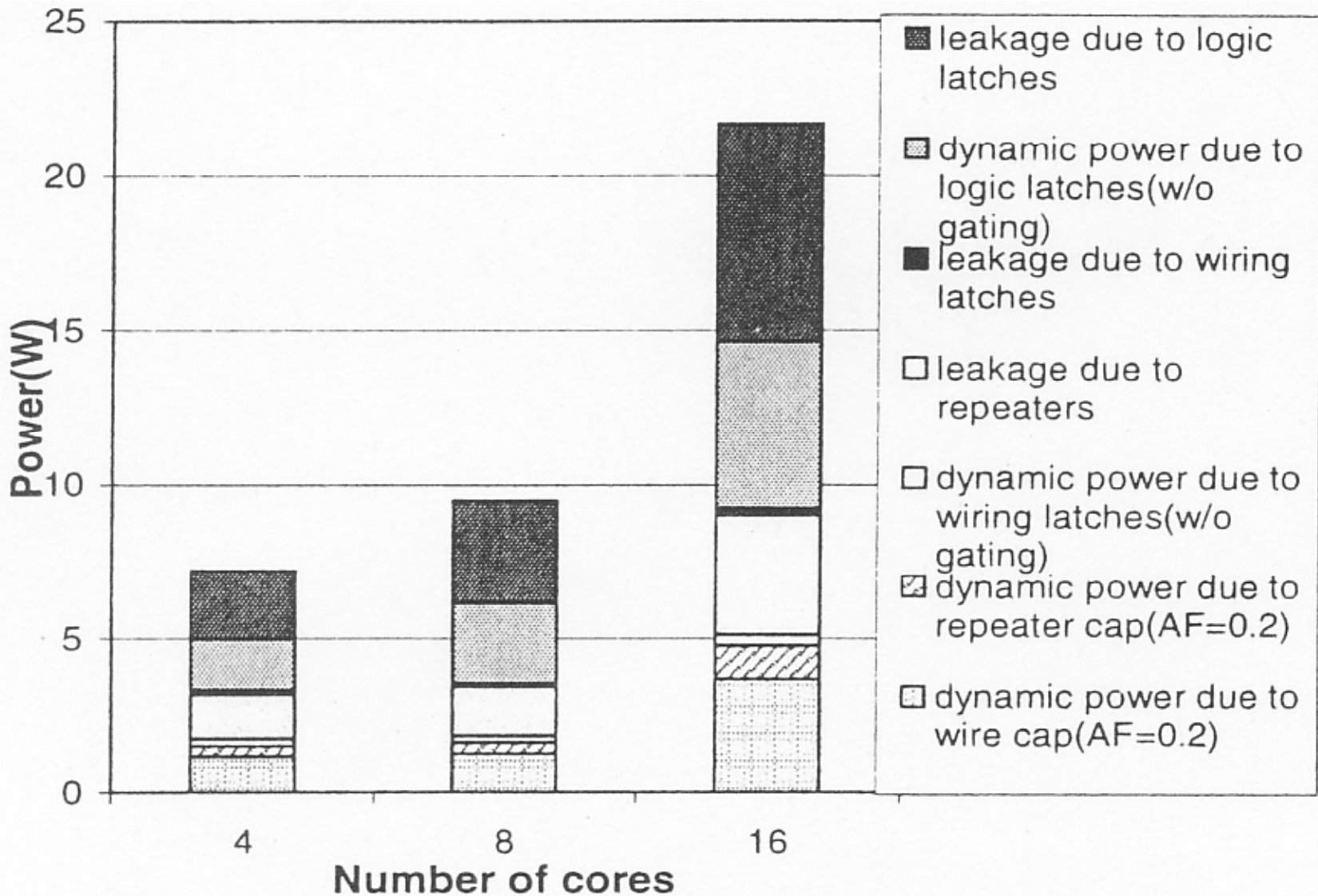
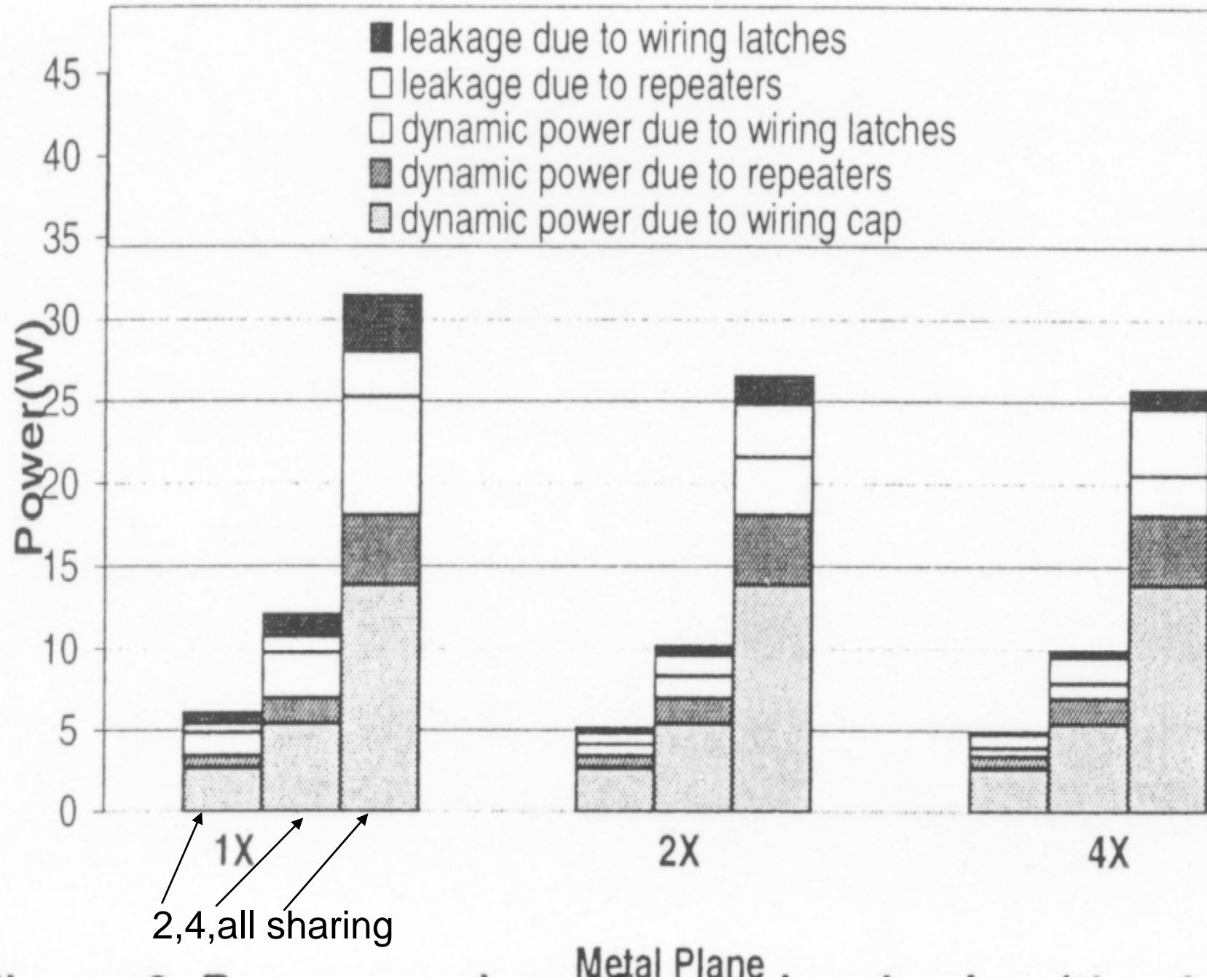
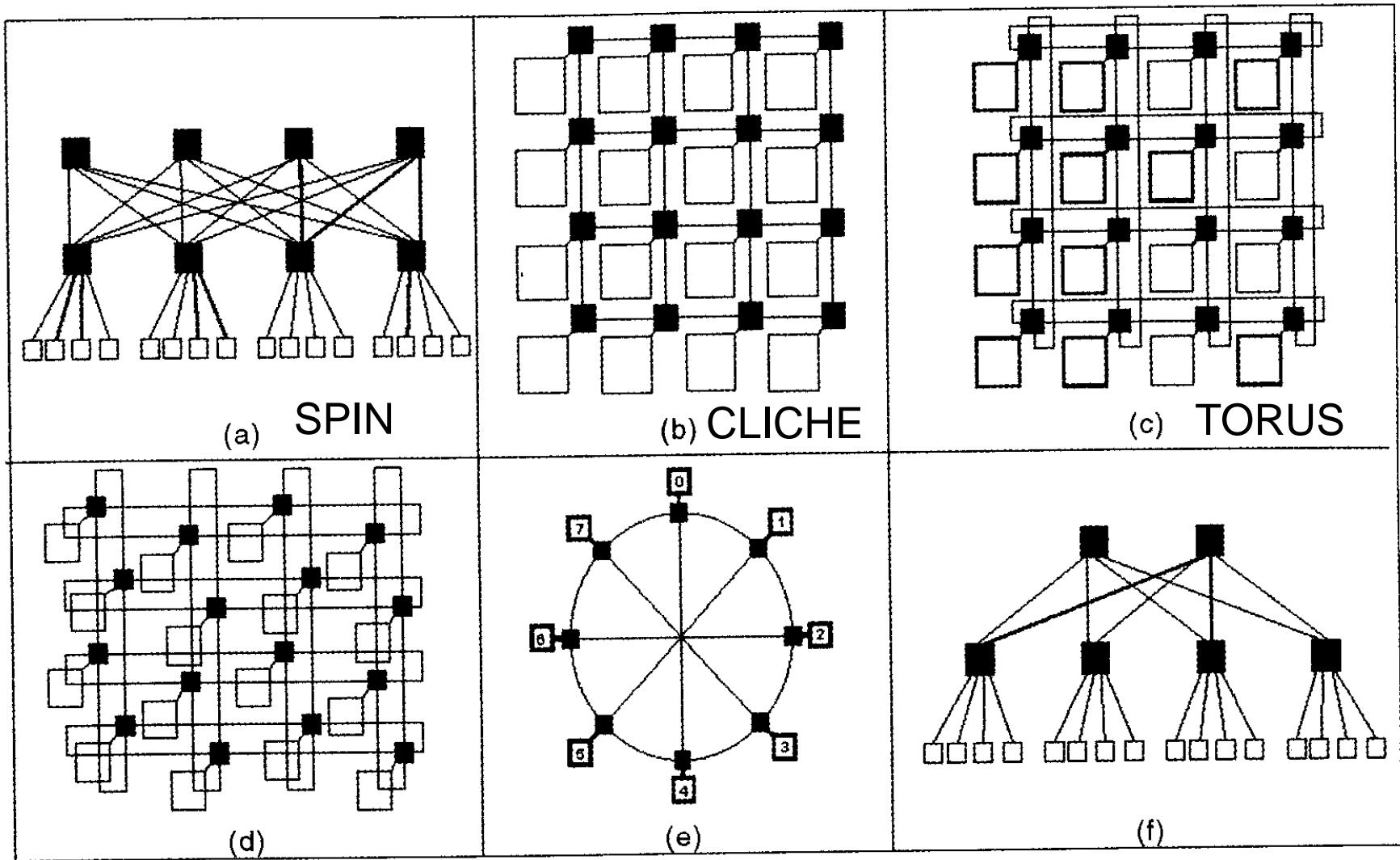


Figure 5. Power overhead for shared bus fabric.

1コア (Power4) : 10W

オーバーヘッド: 2コア分に相当 (22mm² 16コアの場合)





(a) SPIN

(b) CLICHE

(c) TORUS

(d)

(e)

(f)

Folded TORUS

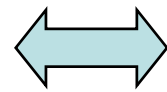
OCTAGON

BFT Pande et al

SPIN: Scalable, Programmable, Integrated Network, CLICHÉ: Chip-Level Integration of Communicating Heterogeneous Elements, BFT: Butterfly Fat Tree

(2) データ投機実行

未商用化



分岐予測

データ依存を超えて

制御依存を超えて

①値予測

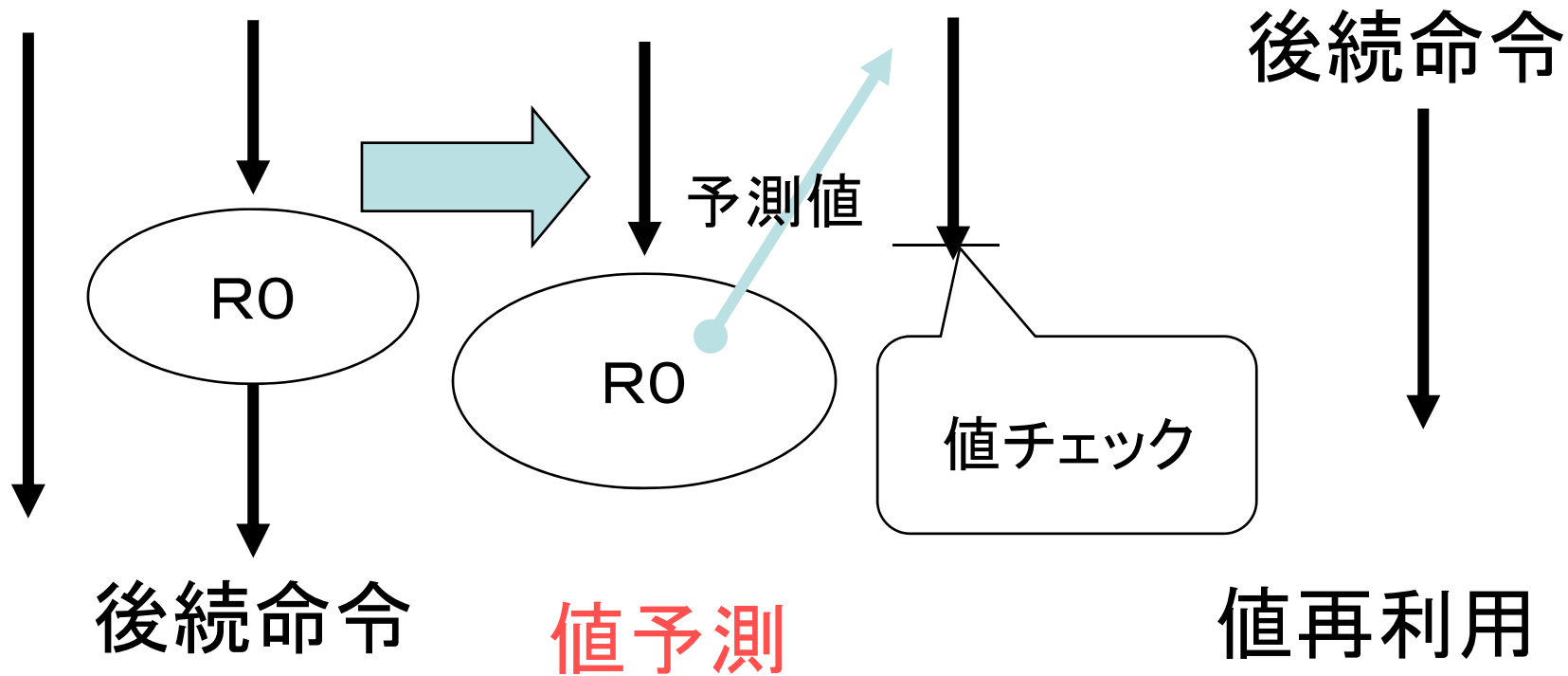
②値再利用

フロー依存

同時実行

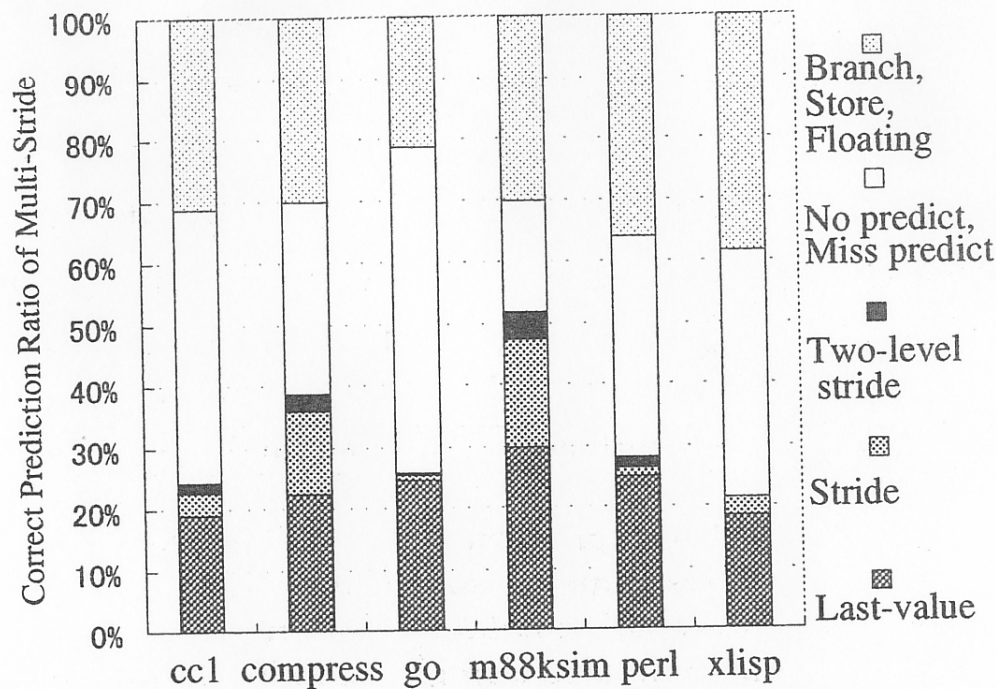
先行命令実行せず

先行命令 先行命令 後続命令



①値予測

Last-Value値、ストライド値、2レベルストライド値などで予測



吉瀬、坂井、
田中: JSPP99

図 9: Multi-stride 値予測機構の予測成功率

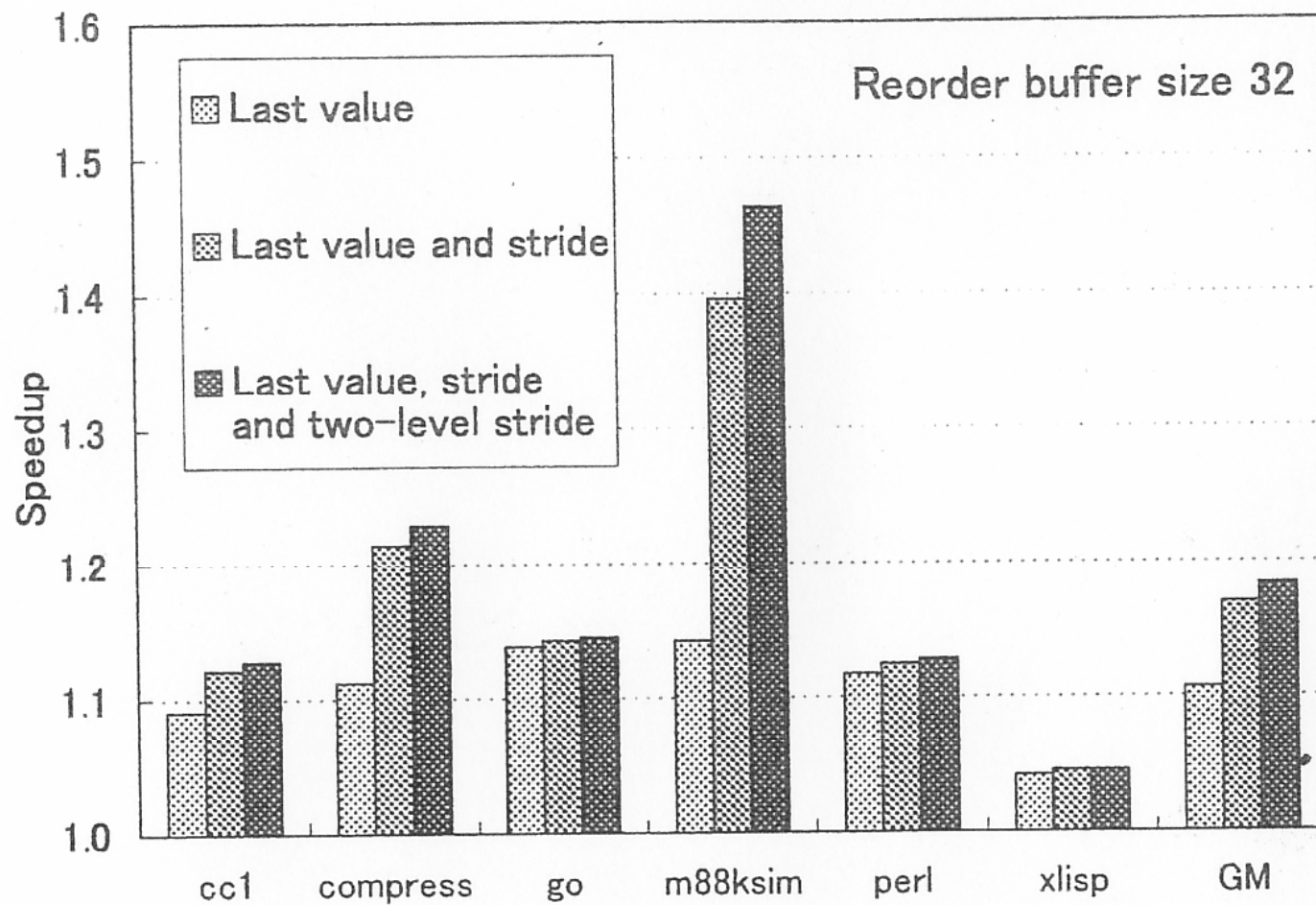


図 10: Multi-stride 値予測機構による性能向上率

吉瀬、坂井、
田中: JSPP99

②値再利用

- ・ 同一パラメータによる関数・ループ
実行自体の省略
- ・ 事前実行による値再利用の可能性の増大化
- ・ データ値の許容範囲設定による値再利用の
可能性の増大化

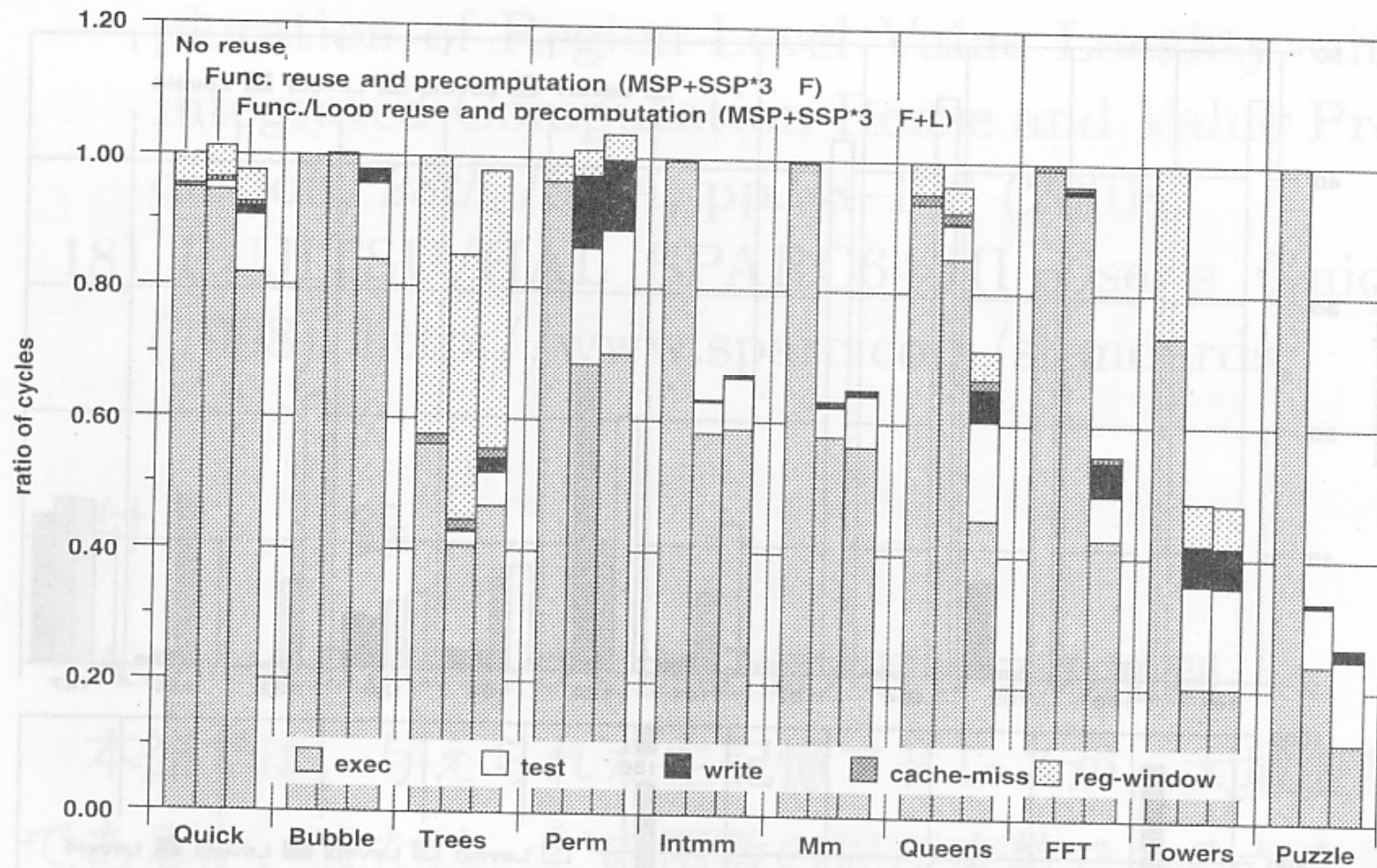


図 12 MSP が実行したサイクル数 (Stanford-Integer).
 Fig. 12 Executed cycles on MSP (Stanford-Integer).

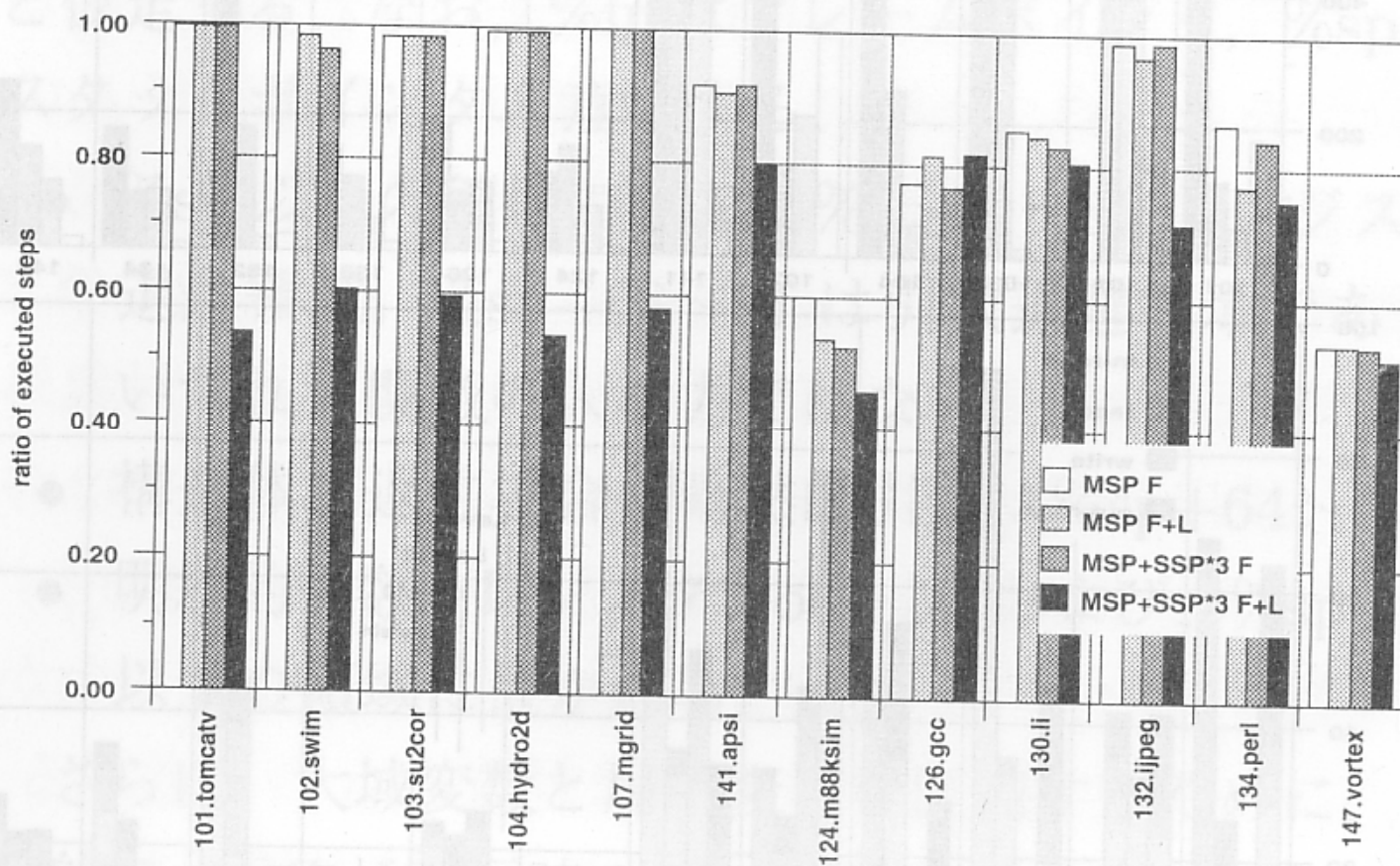


図 13 MSP が実行した命令ステップ数 (SPEC95).

Fig. 13 Executed steps on MSP (SPEC95).

13.2 メディアプロセッサ

CG : 浮動小数点演算、 4×4 行列演算

SH-4 : 500万ポリゴン/秒

プレステ2 : 1600万ポリゴン/秒

画像 : MPEG

符号化 : 100GOPS,

復号化 : 10GOPS

(1) Sony:Cell

マルチコア：ヘテロジニアス構成

1個のPPE:Power Processor Element

IBM Power Architecture

2 多重スーパスカラ、乱実行なし、

分岐予測なし、非常にシンプル

8個のSPE：Synergistic Processing Element

4 SIMD並列積和演算

4 GHzX 8：256GFLOPS

相互結合：リングバス、256GB/s

応用：マルチメディア

これがCell

4GHz

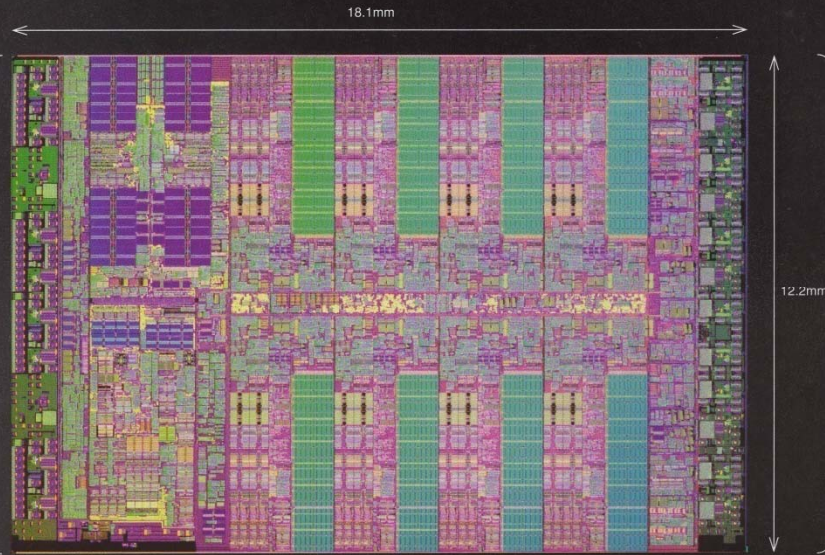
(動作周波数)
現行「Pentium 4」の最大動作周波数である3.8GHzを上回る。

25.6Gバイト/秒

(メモリ・インタフェースの最大データ転送速度)
米Rambus, Inc.の「XDRインタフェース」を2チャンネル備える。

9個

(マルチコア構成)
PowerアーキテクチャのCPUコアに加えSPEと呼ぶ信号処理プロセッサを8個搭載している。各コアは最大データ転送速度が256Gバイト/秒のリング状のバスで結ぶ。



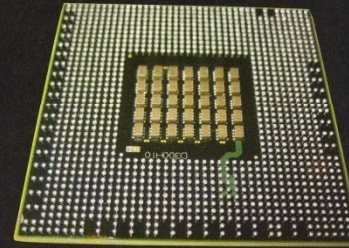
日経エレクトロニクス
2005. 2. 28

256GFLOPS

(SPEの演算性能)
各SPEはSIMD命令によって32ビット浮動小数点データの積和演算を4個並列に1サイクルのスループットで処理可能。

1236端子

(パッケージ)
「Pentium 4」の約1.6倍。チップ上には2965個のC4バンパを形成してある。



2億3400万個

(トランジスタの数)
「Pentium 4」の約2倍。このうち8個のSPEの論理回路に合計5600万個を使う。

マルチスレッド

(メモリ管理の強化による支援)
各SPEに集積したDMAコントローラがメモリ管理を行う。メモリ管理ユニットのアドレス変換テーブルをCPUコアが制御することで、並列処理するスレッドの動作を保証する。SPEが専用メモリ空間へのCPUコアのアクセスを禁じる動作モードを用意。

46nm

(ゲート長)
90nmルールのSOI技術で製造。8層のCu配線技術とひずみSi技術を使う。

仮想化

(複数OSの並列実行)
複数のOSを並列に動かすソフトウェアの実行環境を実装。演算資源の割り当てを保証しリアルタイム処理を可能に。

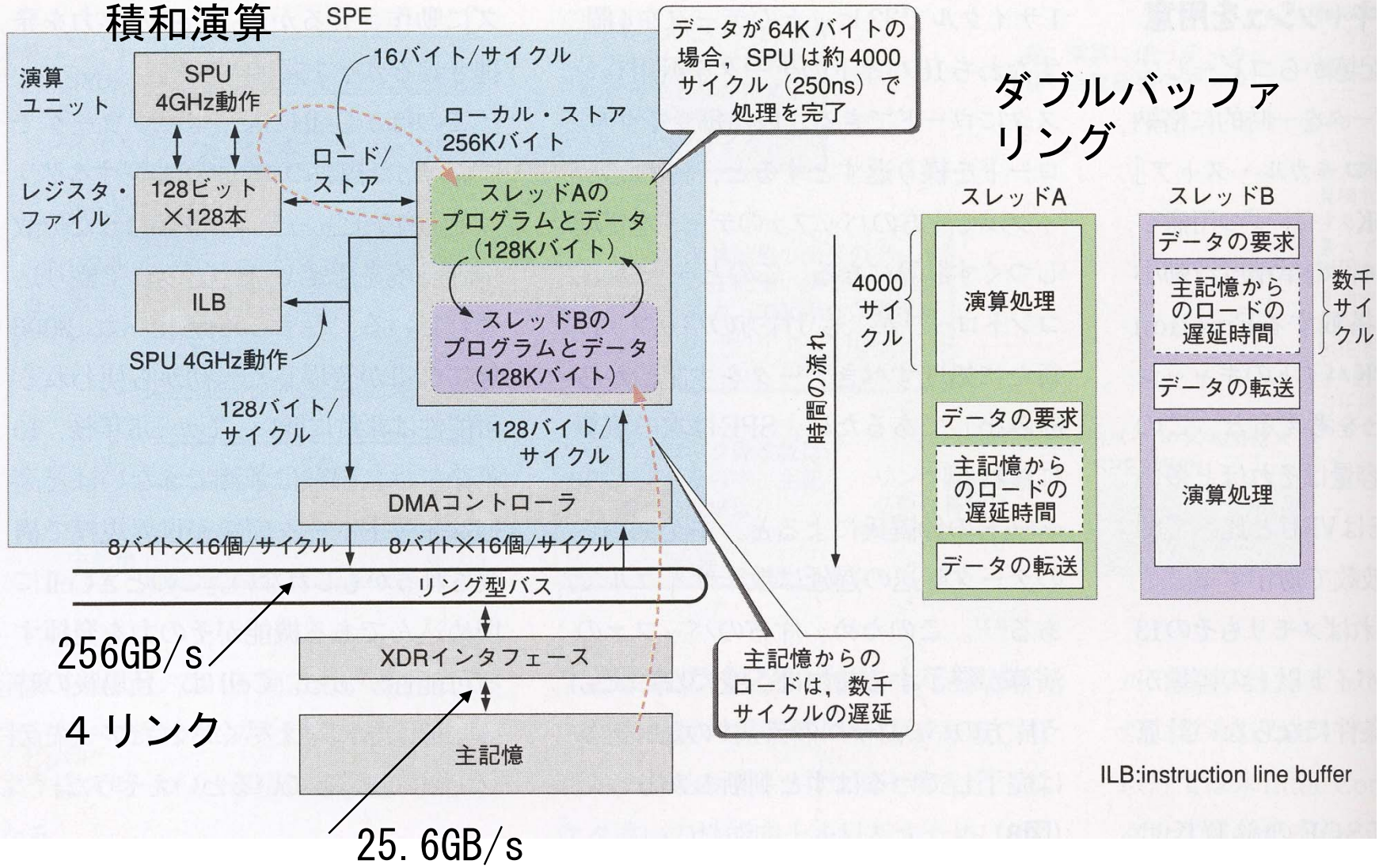
76.8Gバイト/秒

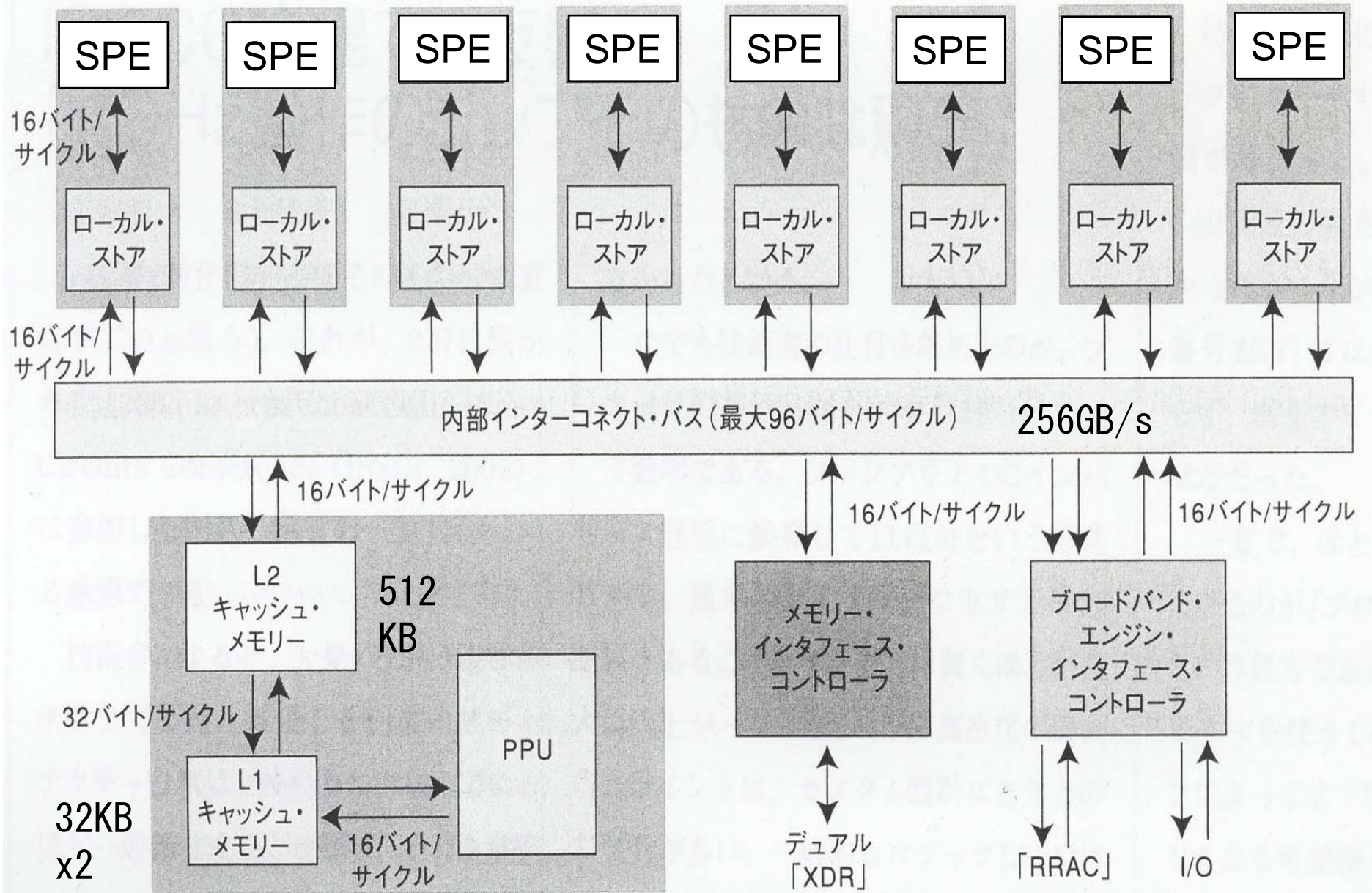
(システム・インタフェースの最大データ転送速度)
米IBM Corp.の「POWER5」の64Gバイト/秒を凌駕する。「Pentium 4」や「Itanium 2」に比べて10倍以上に当たる。物理層には米Rambus, Inc.の「FlexIO」を用いる。

	Cell	Emotion Engine
発表時期	2005年	1999年
動作周波数	4GHz	300MHz
32ビット浮動小数点データの演算性能	256GFLOPS	6.2GFLOPS
トランジスタ数	2億3400万個	1350万個
発売時の設計ルール	90nm	250nm
基板	SOI	バルクSi
チップ面積	221mm ² (18.1mm×12.2mm)	226mm ² (15.02mm×15.04mm)
CPUコア	Power (2命令同時発行)	MIPS (2命令同時発行)
信号処理プロセッサ	128ビットのSIMD型×8個	128ビットのVLIW型×2個
メモリ・インタフェースのデータ転送速度	最大25.6Gバイト/秒	最大3.2Gバイト/秒
システム・インタフェースのデータ転送速度	最大76.8Gバイト/秒	最大1.2Gバイト/秒
パッケージ	1236端子のBGA	540端子のプラスチックBGA

	Cell	Emotion Engine
発表時期	2005年	1999年
動作周波数	4GHz	300MHz
32ビット浮動小数点データの演算性能	256GFLOPS	6.2GFLOPS
トランジスタ数	2億3400万個	1350万個
発売時の設計ルール	90nm	250nm
基板	SOI	バルクSi
チップ面積	221mm ² (18.1mm×12.2mm)	226mm ² (15.02mm×15.04mm)
CPUコア	Power (2命令同時発行)	MIPS (2命令同時発行)
信号処理プロセサ	128ビットのSIMD型×8個	128ビットのVLIW型×2個
メモリ・インタフェースのデータ転送速度	最大25.6Gバイト/秒	最大3.2Gバイト/秒
システム・インタフェースのデータ転送速度	最大76.8Gバイト/秒	最大1.2Gバイト/秒
パッケージ	1236端子のBGA	540端子のプラスチックBGA

4 SIMD並列 積和演算





PPE シンプルな2多重
スーパスカラ

日経エレクトロニクス²³¹
2005. 2. 28

項目	「Cell」の仕様
汎用プロセサ・コアの種類	64ビット「Power Architecture」(PPE), SIMD構造とローカル・ストアを持つRISCアーキテクチャ(SPE)
プロセサ・コアの数(個)	PPEが1, SPEが8
動作周波数(Hz)	最大4.6G
L1キャッシュ・メモリー[命令](バイト)	32K
L1キャッシュ・メモリー[データ](バイト)	32K
L2キャッシュ・メモリー(バイト)	512K
外付けメイン・メモリー	XDR DRAM
メモリーのバス・バンド幅(バイト/秒)	25.6G
浮動小数点演算ユニット(個)	36個[単精度], 9個[倍精度]
浮動小数点演算性能(FLOPS/チップ)	256G[単精度]
データ・バス幅(ビット)	1024[内部]
プロセス	90nm, SOI, 8層Cu配線
ゲート長(nm)	46
消費電力(W)	70~80W
トランジスタ数(個)	2億3400万
チップ面積(mm ²)	221

図3 ● 「Cell」の主な仕様

「ISSCC 2005」で発表された試作チップ。公表値やヒアリングを基に本誌が作成。

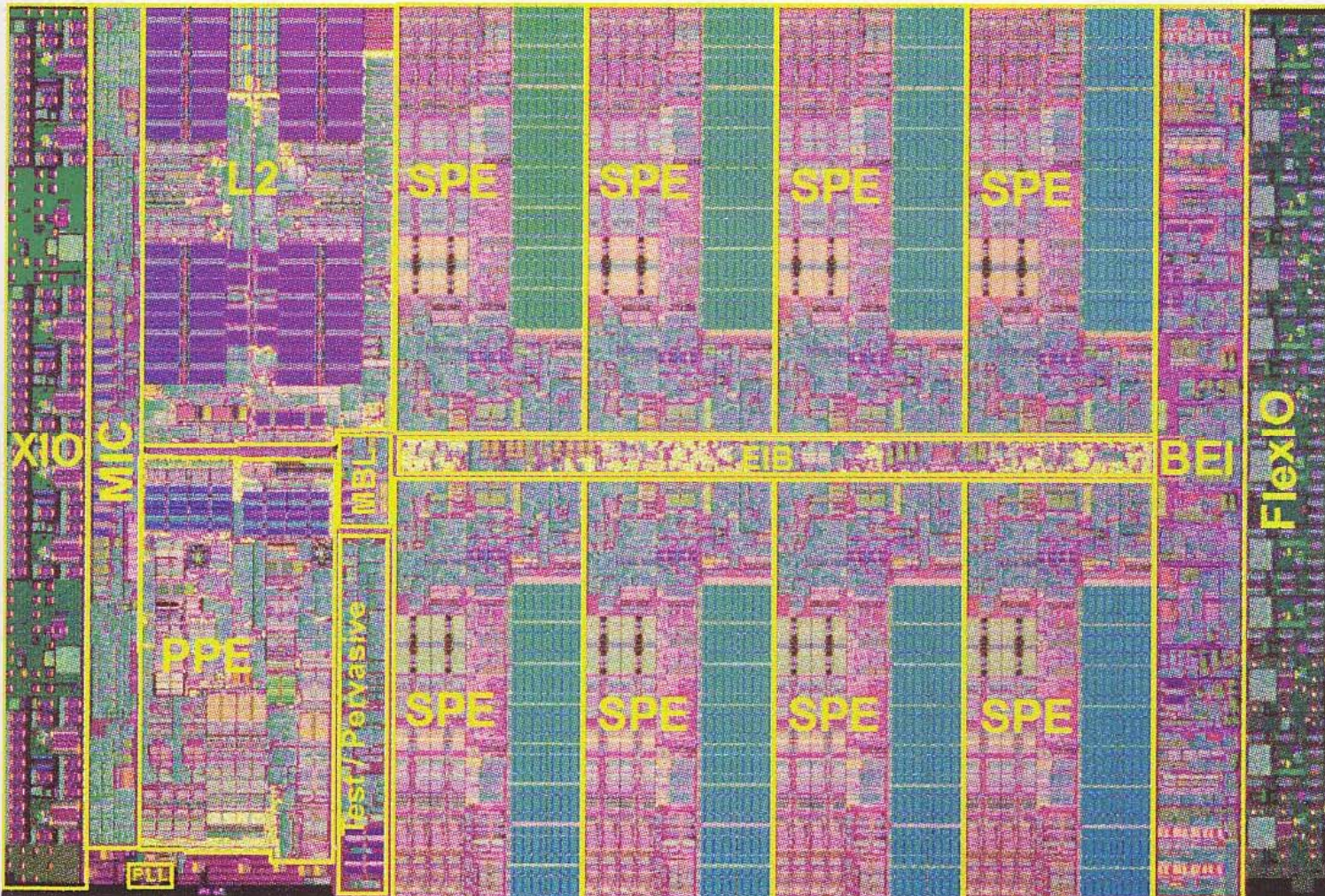


図1 試作チップのレイアウト

90nmルールのSOI技術を使い約2億3400万個のトランジスタを18.1mm×12.2mmのチップに集積した。

- BEI: broadband engine interface
- EIB: element interconnect bus
- MBL: MIC Bus Logic
- MIC: memory interface controller
- PPE: power processor element
- SPE: synergistic processing element
- XIO: XDR i/o interface

スパコンにPS3セル

搭頭脳に米、核実験で使用

IBMに発注

【ワシントン6日共同】米エネルギー省核安全保障局は6日、最大演算速度が「世界最高」というペタフロップス(一秒間に千兆回)のスーパーコンピュータの設計・製造を米IBMに発注したと発表した。核兵器の模擬実験などに使うのが目的で、スパコンの頭脳部にはソニー・コンピュータエンタテインメント(SCE)が今秋発売するゲーム機「プレイステーション3」にも使われる高性能半導体「セル」を採用。セルがスパコンに搭載されるのは初めてという。

スパコンは「ロードラや安全性を検証するため「シミュレーター」と命名。実際の核爆発を伴わない模擬実験などを通じて、米国が保有する核兵器の信頼性

セルはIBMがソニーグループや東芝などと共同開発。

処理能力に優れた

超小型演算処理装置(MPU)。演算処理などを行う中核部分(コア)が九つ搭載されており、スーパーコンピュータ並みの処理ができるという。最新のパソコン用プロセッサの10倍以上の性能を誇る。今秋発売の家庭用ゲーム機「プレイステーション3」に採用されるほか、デジタル家電など幅広い応用が期待されている。

(共同)

高い映像処理能力を持つ月一〇六年九月)に三、模擬実験などに適している」と判断されたとのみ

た。

ロスアラモス国立研究所(ニューメキシコ州)に二〇〇八年に設置される計画で、米議会は○六会計年度(〇五年十

SCE広報部は「コンピュータゲームでも登場人物の髪の毛や風向きなどの微妙な描写ができる高性能のプロセッサ

が必要とされている。セルはもともと汎用プロセッサとして開発され、最近ではタンパク質の構造解析など医療分野でも注目されており、普及を期待している」と話している。

(2) NEC IMAP-CE

省電力車載用画像処理プロセッサ

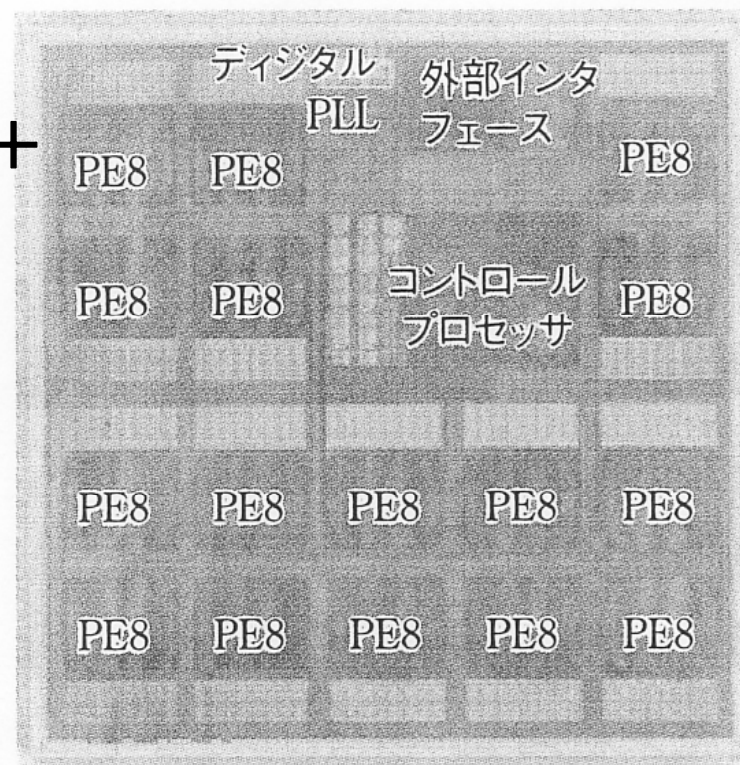
SIMD方式

PE:8ビット、2ALU+乗算器+

アドレス演算器

演算性能	最大51.2GOPS
消費電力	最大4W
PE数	8bit×128
動作周波数	100MHz
電源電圧	+1.8V(内部) +3.3V(入出力)
製造技術	0.18 μ mルール
トランジスタ数	3,270万個
チップ面積	11mm×11mm

(a) 性能概要



(b) チップ写真

図1 超高速並列プロセッサ

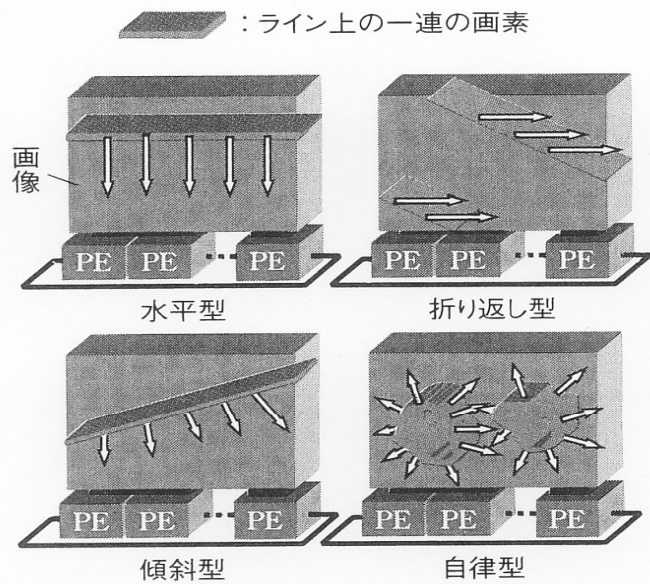


図2 4種類の並列画像認識処理

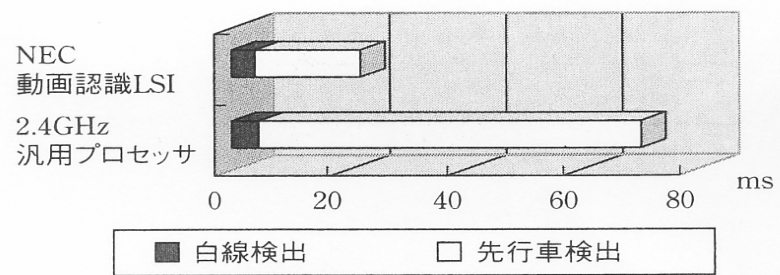


図3 白線と先行車の検出例

13.3 省電力プロセッサへの 並列処理利用

CMOSの電力消費

- ・ 動的

回路がON、OFFするとき $\alpha f C_L V^2$

- ・ 漏れ電流 $V I_{leak}$

- ・ 貫通電流 $\alpha f t_{sc} I_{short} V$

pMOS、nMOSがスイッチング時同時ON

α : ゲート動作率、 f : 周波数、 C : ゲート総容量、
 V : 電源電圧、 t_{st} : スイッチング時間

T.Mudge: Power: A First-Class Architectural Design
Constraint, IEEE Computer, pp.52-58, April 2001

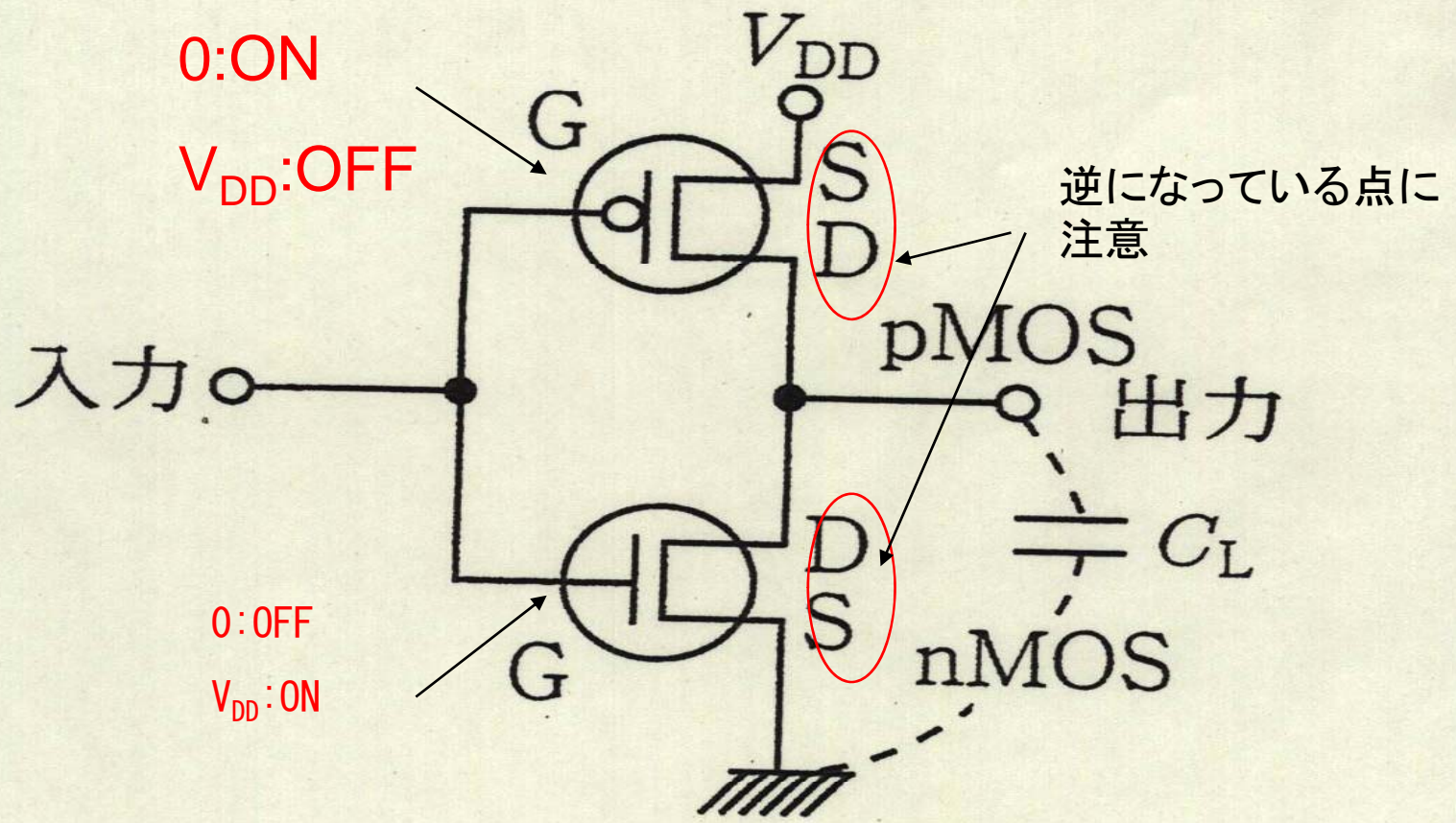
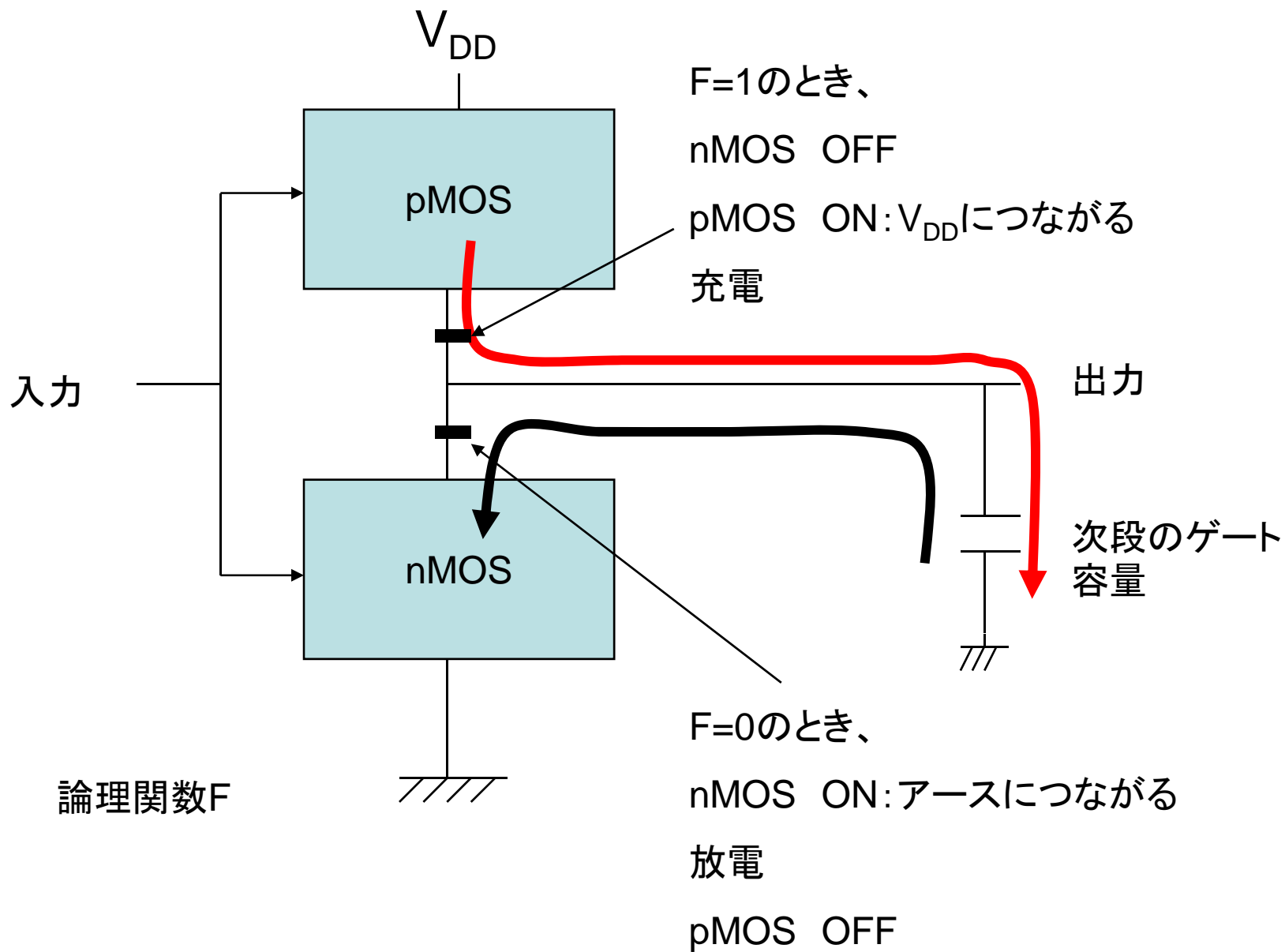


図 2.32 CMOS 否定回路



基本式

$$P = \alpha f C_L V^2 + V I_{\text{leak}} + \alpha f t_{\text{sc}} I_{\text{short}} V$$

$$F_{\text{max}} \propto (V - V_{\text{threshold}})^2 / V \doteq V$$

$$I_{\text{leak}} \propto \exp(-qV_{\text{threshold}}/kT)$$

各部での電力消費の割合 (%)

	クロック	データパス	メモリ	I/O
組み込み系	50	33	11	6
高速プロセッサ	30	25	40	5
MPEGASIC	22	34	22	22
ATMSWASIC	20	10	6	64

基本的な考え方

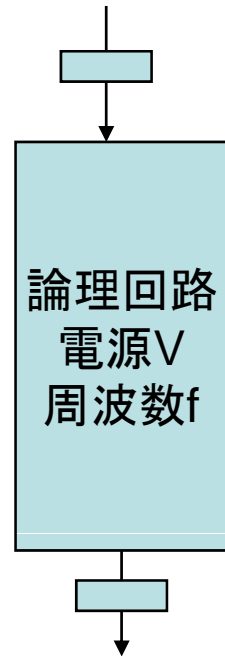
- ・ スイッチング回数を少なくする
- ・ 動作をしない（と予想される）回路には
 クロック供給しない
 電源を供給しない
- ・ 電源電圧を制御して、必要十分な処理速度で実行
- ・ 電源電圧、周波数を落として並列処理、パイプラインで行う
- ・ 基盤バイアス印加による閾値制御：
 リーク電流削減（サブスレッショルドリーク電流）
 高速部分：低閾値、
 低速部分や待機時：高閾値（バイアス印加）

- ・ デバイスレベル
 - 低電源電圧化、低ゲート容量化、基盤バイアス制御
- ・ 回路レベル
 - パストランジスタ論理
 - ゲート付きクロック
 - グリッチの削減
 - 動的電源遮断
 - 非同期回路
- ・ アーキテクチャレベル
 - データパスの最適化：必要な演算幅の決定など
 - 並列処理、パイプライン処理
 - キャッシュメモリ
 - バス：アドレスの反射2進符号化、データ圧縮
- ・ OS、コンパイラ、アルゴリズムレベル
 - 符号化
 - ビット変化の少ないコード生成
 - 動的電源電圧制御
 - 動的周波数制御

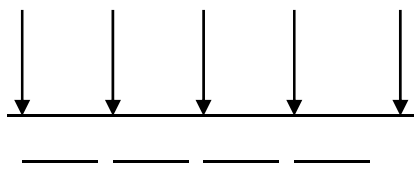
(1) 並列処理の導入

並列処理

電力 fCV^2



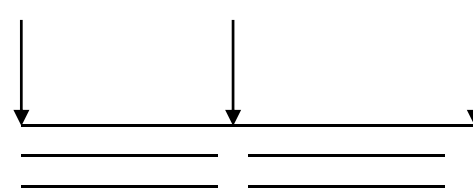
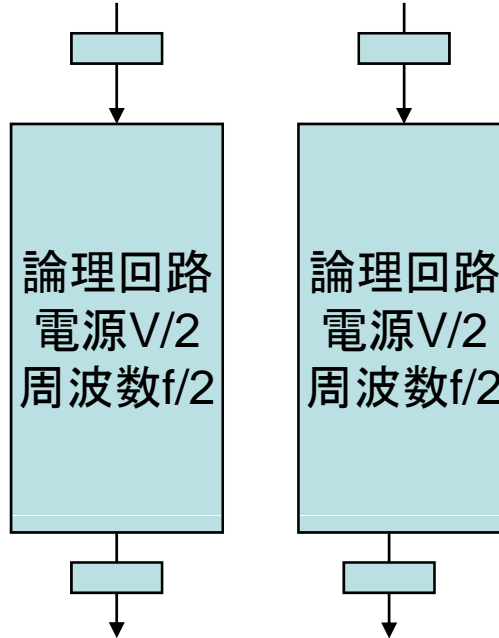
ラッチ



1/fごとに1つの結果

電力 $2(f/2C(V/2)^2)$

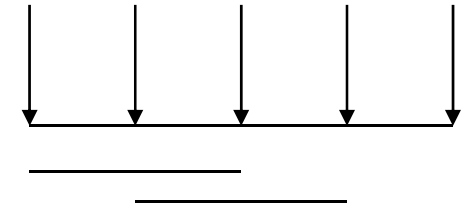
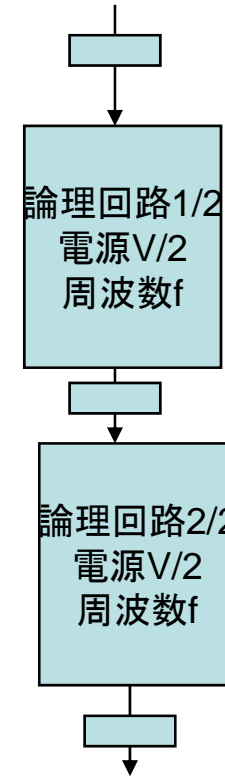
$fCV^2/4$



1/2fごとに2つの演算

パイプライン処理

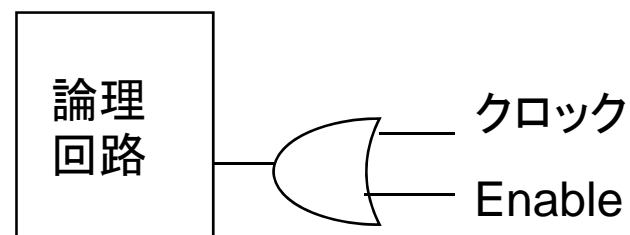
電力 $fCV^2/2$



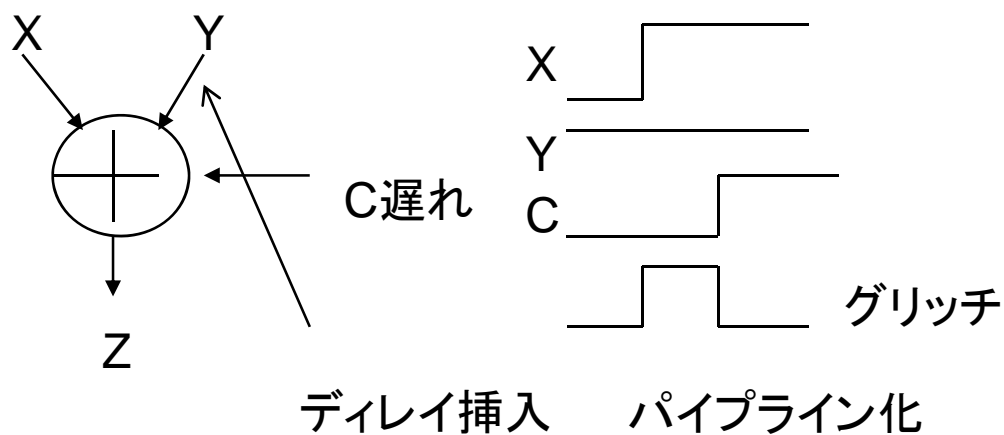
1/fごとに1つの結果

(2) 無駄なスイッチングを減らす

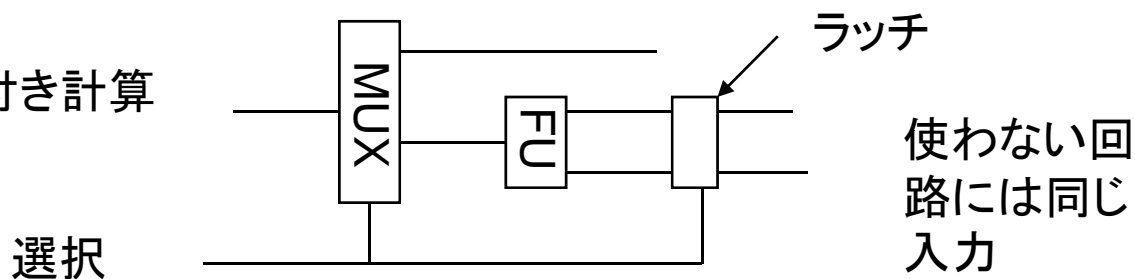
①ゲート付き
クロック



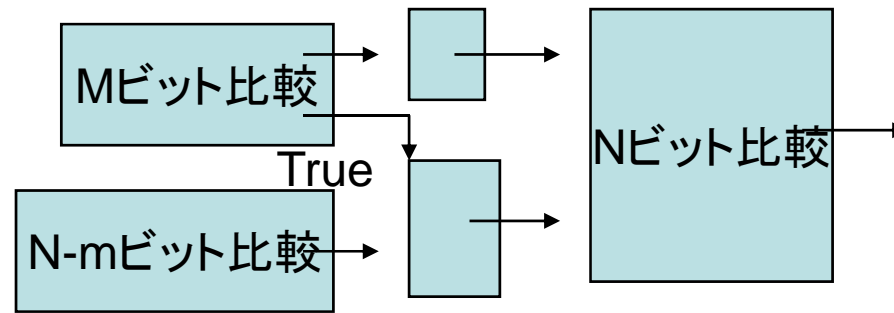
②グリッチの
削除



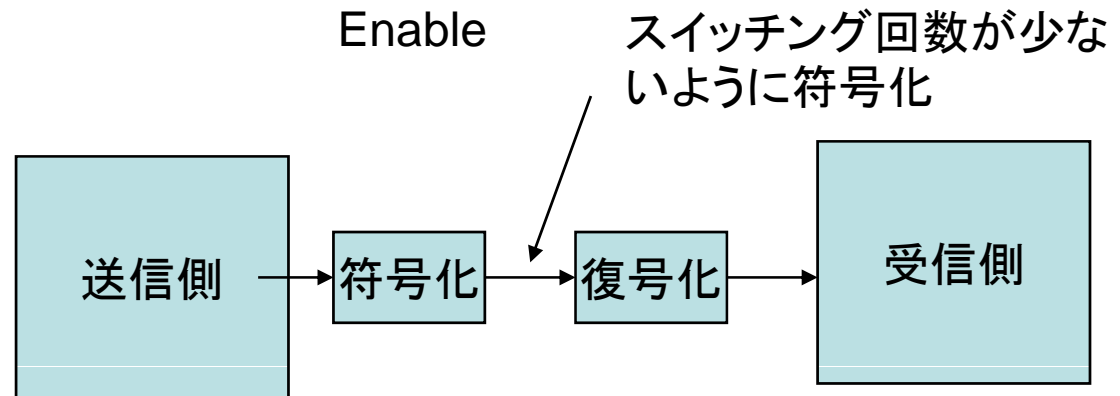
③ガード付き計算



④プレ計算

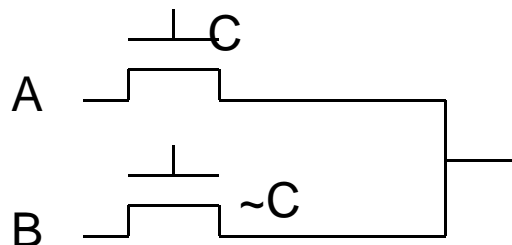


⑤符号化



⑥データの転送順、
処理順の変更

正の数のグループ先、負の数のグループ後



$$C \cdot A + \sim C \cdot B$$

⑦パストラ
ンジスタ

トランジスタ数の削減

(3) スイッチングの少ない符号化

2の補数表示：符号反転操作大変

正+5 負-5
0 0 1 0 1 → 1 1 0 1 1

絶対値表示：楽

0 0 1 0 1 → 1 0 1 0 1

カウンタ：反射2進符号

2進符号 000 001 010 011 100 101 110 111

反射2進符号 000 001 011 010 110 111 101 100

1ビットずつ変化

・ 2進符号と反射2進符号の相互変換

$$(b_n, b_{n-1}, b_{n-2}, \dots, b_1) \quad (g_n, g_{n-1}, g_{n-2}, \dots, g_1)$$

$$b_n=1, \quad g_n=1$$

$$b_n=0, \quad g_n=0$$

$$b_n=0, b_{n-1}=0 \text{ のとき } g_{n-1}=0$$

$$b_n=0, b_{n-1}=1 \text{ のとき } g_{n-1}=1$$

$$b_n=1, b_{n-1}=0 \text{ のとき } g_{n-1}=1$$

$$b_n=1, b_{n-1}=1 \text{ のとき } g_{n-1}=0$$

一般に

$$b_i \oplus b_{i-1} = 0 \text{ のとき } g_{i-1} = 0$$

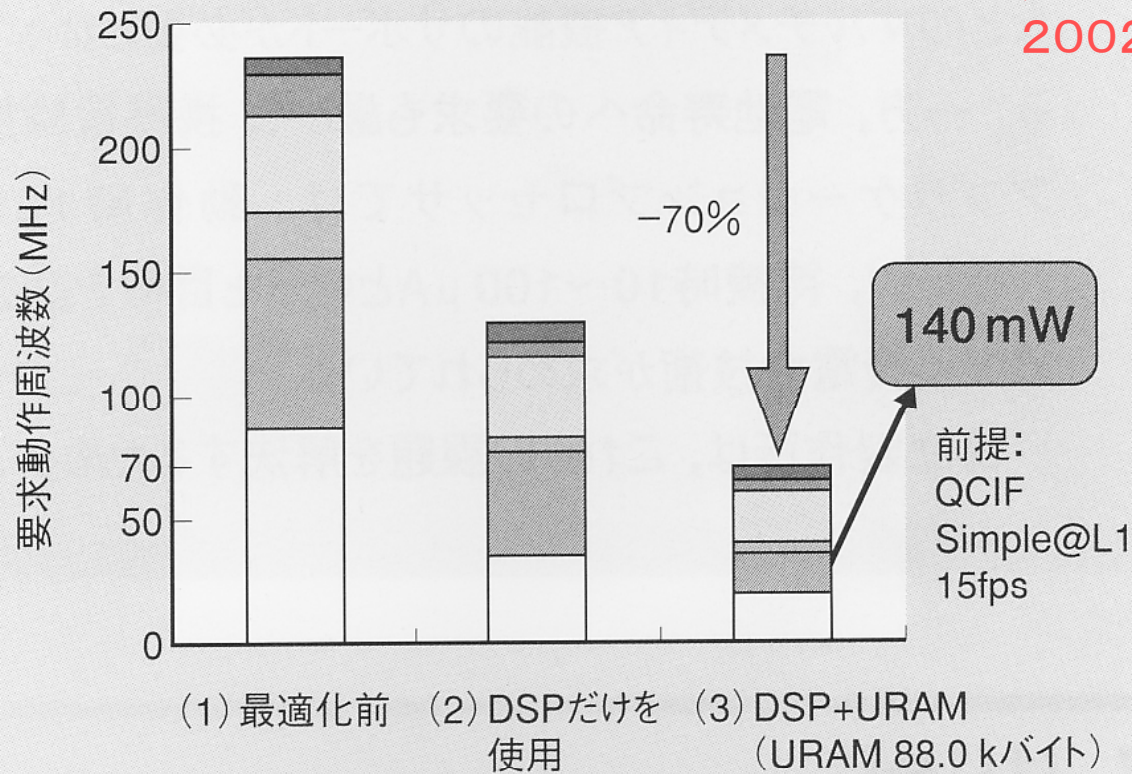
$$b_i \oplus b_{i-1} = 1 \text{ のとき } g_{i-1} = 1$$

$$b_i \oplus b_{i-1} = g_{i-1}$$

(4) メモリ・キャッシュメモリの省電力化

- ①フィルタキャッシュ：L1キャッシュ前にバッファ設置
- ②メモリのバンク化
- ③目的別領域化
- ④リプレースなし領域：画像処理など
- ⑤選択的ウェイアクセス、投機的

入江ほか：日立評論、
2002年10月



注1: ■MC(動き補償), ■IDCT(逆離散コサイン変換), □Q(量子化)/IQ(逆量子化)
+VLC(可変長符号化), ■DCT(離散コサイン変換), ■Ctl(制御), □ME(動き
予測)

注2: 略語説明 QCIF(176×144画素のビデオ信号フォーマット), fps(Frames per
Second), Simple@L1(低解像度で単純な動画像)

図1 MPEG-4符号化処理における要求動作周波数の低減

DSPとURAMを併用することで、一般的なCPUに比べて70%の動作周波数を低減することができる。

(5) 電源電圧と周波数制御

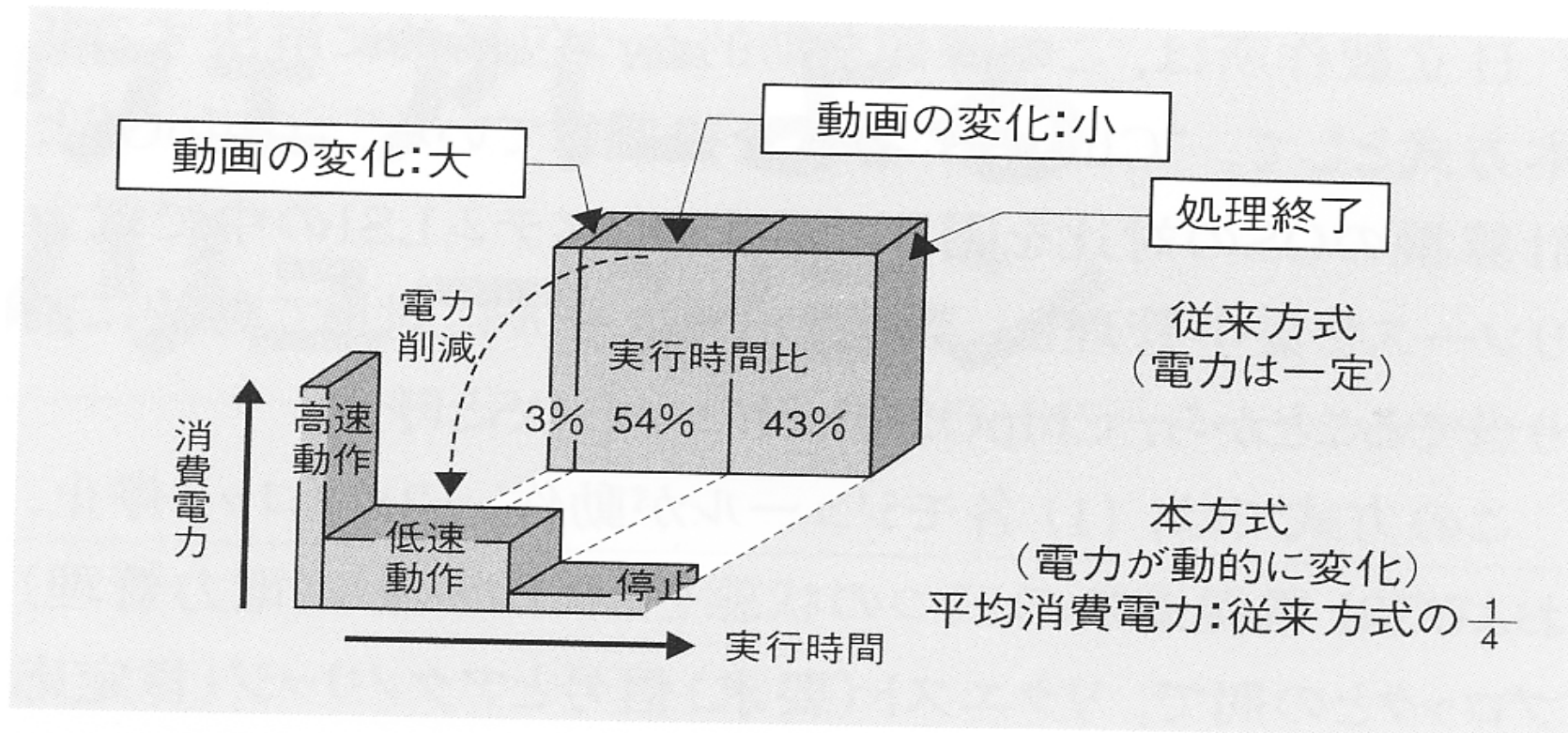
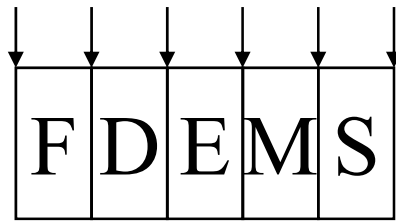
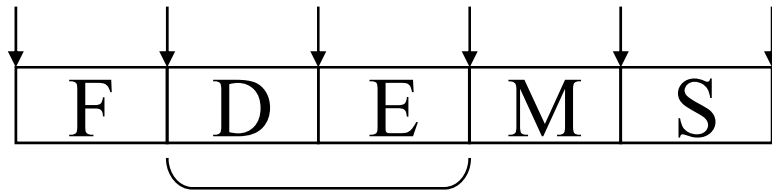


図2 MPEG-4動画デコーダでの消費電力削減

大半の処理は、低速・低電圧か停止(スリープ)である。このため、高速・高電圧を常に保持する処理の $\frac{1}{4}$ の電力に低減することができる。



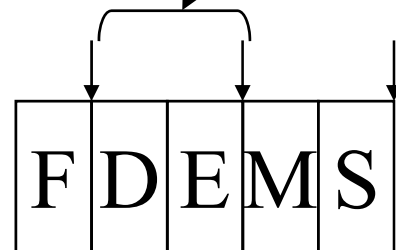
周波数 f 、電源 V :電力: fV^2



ボルテージスケールリング

周波数 $f/2$ 、電源 $V/2$:電力: $1/8$

分岐ミスするとき



パイプラインステージ統合

周波数 $f/2$ 、電源 V :電力: $1/2$

低電源電圧化が困難なとき、有効(リーク電流)

各メーカーでの名称

- Transmeta : Longrun
- Intel : SpeedStep
- AMD : PowerNow
- VIA : LongHaul

13.4 高信頼、セキュアなプロセッサ

高信頼、セキュアなプロセッサ

要因

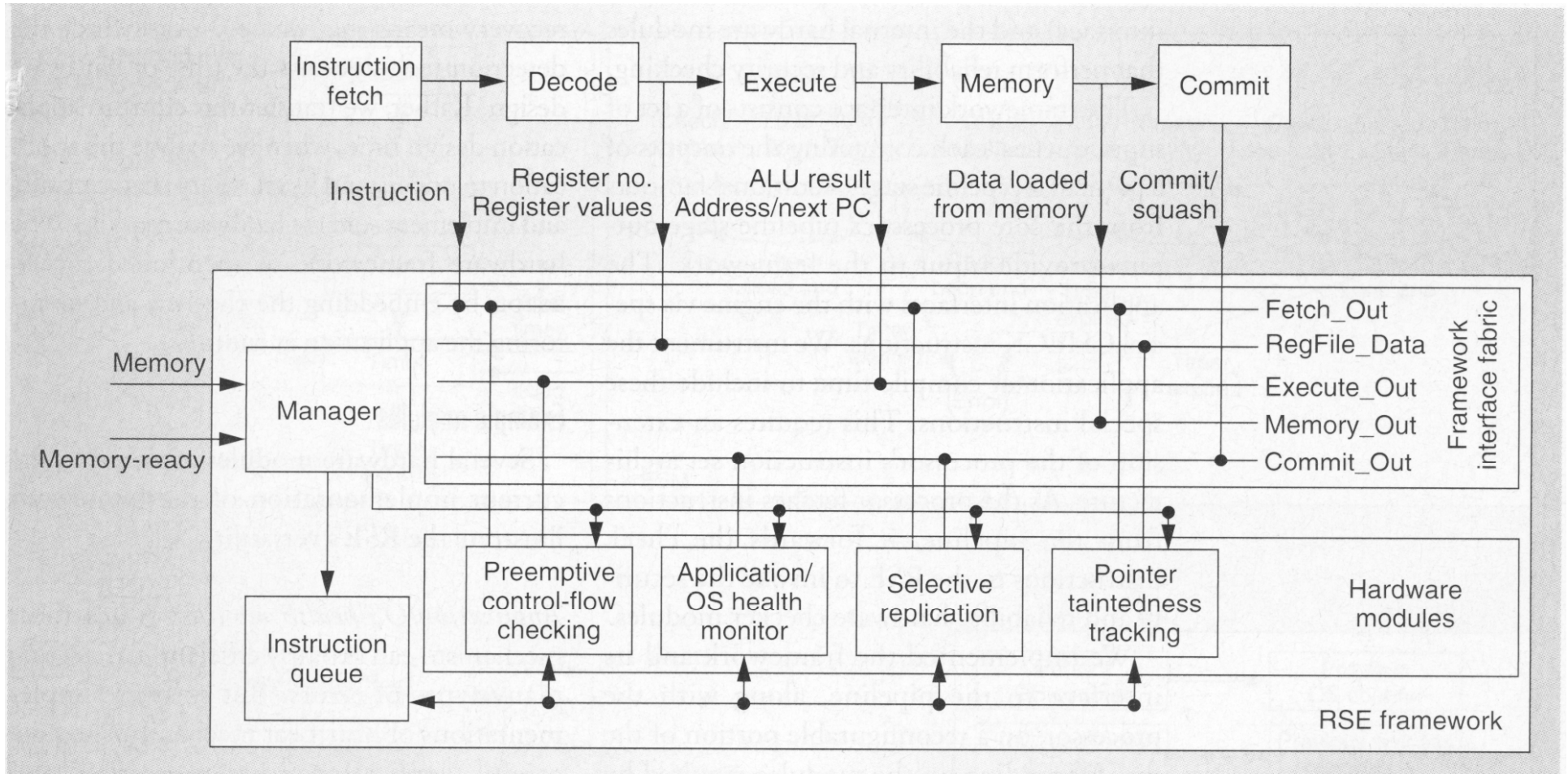
- ・製造バラツキ
- ・ソフトウェア
- ・経年変化
- ・セキュリティ・アタック

高信頼性

- ・ デバイスレベル
- ・ 論理回路レベル
 - パリティ、ECC、ラッチ／フリップフロップの2重化
- ・ アーキテクチャレベル
 - 機能装置の2重化、マルチスレッドでの2重実行、データパスの2重化
 - ランタイムチェッカ
 - イリノイ大学RSE (Reliability and Security Engine)

高セキュア化

- ・ 暗号化
- ・ スタックオーバフローアタック対策



R.K.Iyer et al, IEEE Micro Vol.25,
No.6,2005

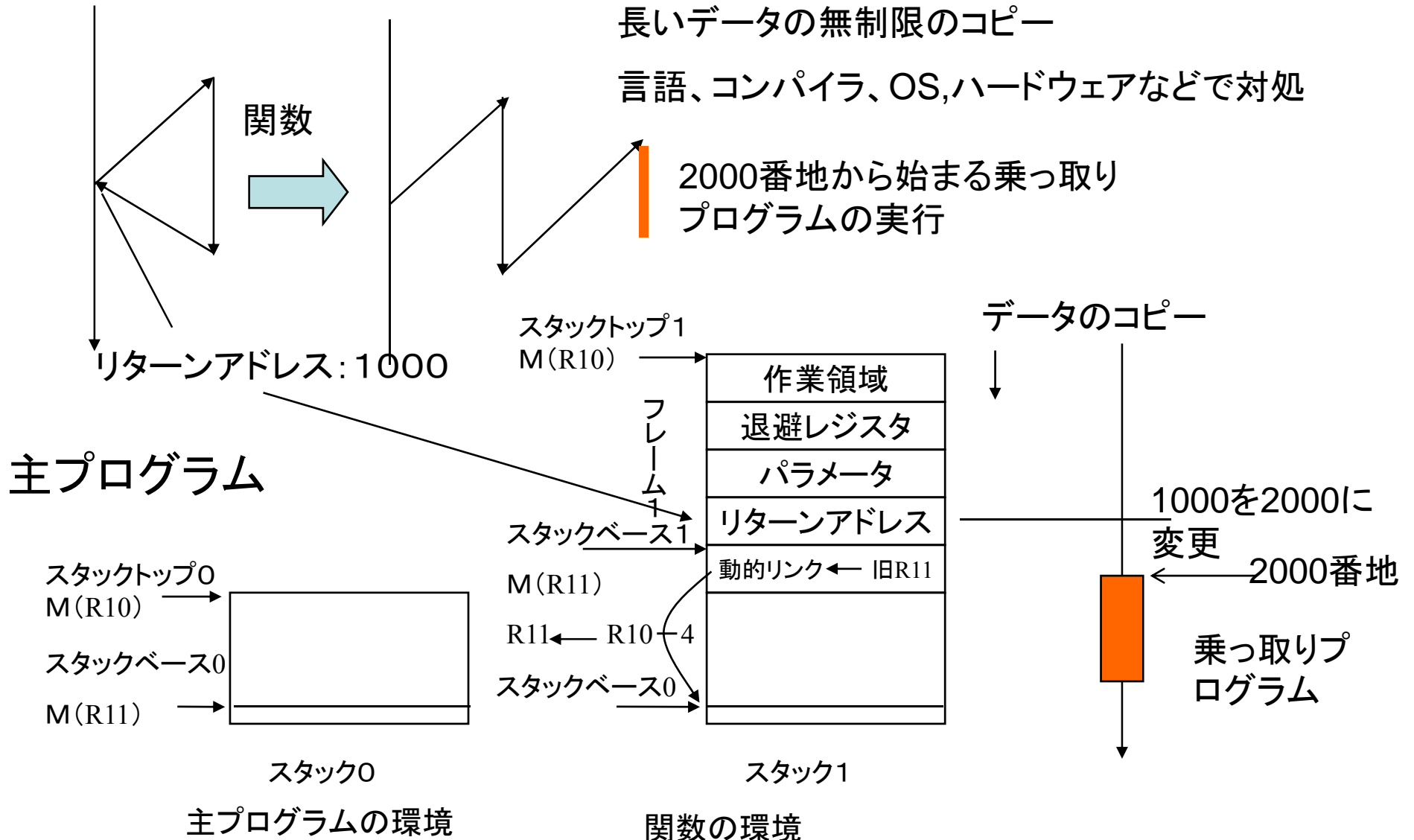
ソフトウェアの脆弱性の例

バッファオーバーフロー攻撃問題

長いデータの無制限のコピー

言語、コンパイラ、OS,ハードウェアなどで対処

2000番地から始まる乗っ取りプログラムの実行



関数呼出しの制御

13.5 再構成可能素子による 可変構造型コンピュータ

可変構造型

演算ビット幅

様々な演算器の実装

データ通信路



個々の応用に合致した

速度、電力の達成

(1) FPGA (Field Programmable Gate Array)

Xilinx社: Virtex II Pro (PowerPC)、

Altera社: Excalibur (ARM)

(2) Dynamically Reconfigurable Processor

NEC: DRP-1

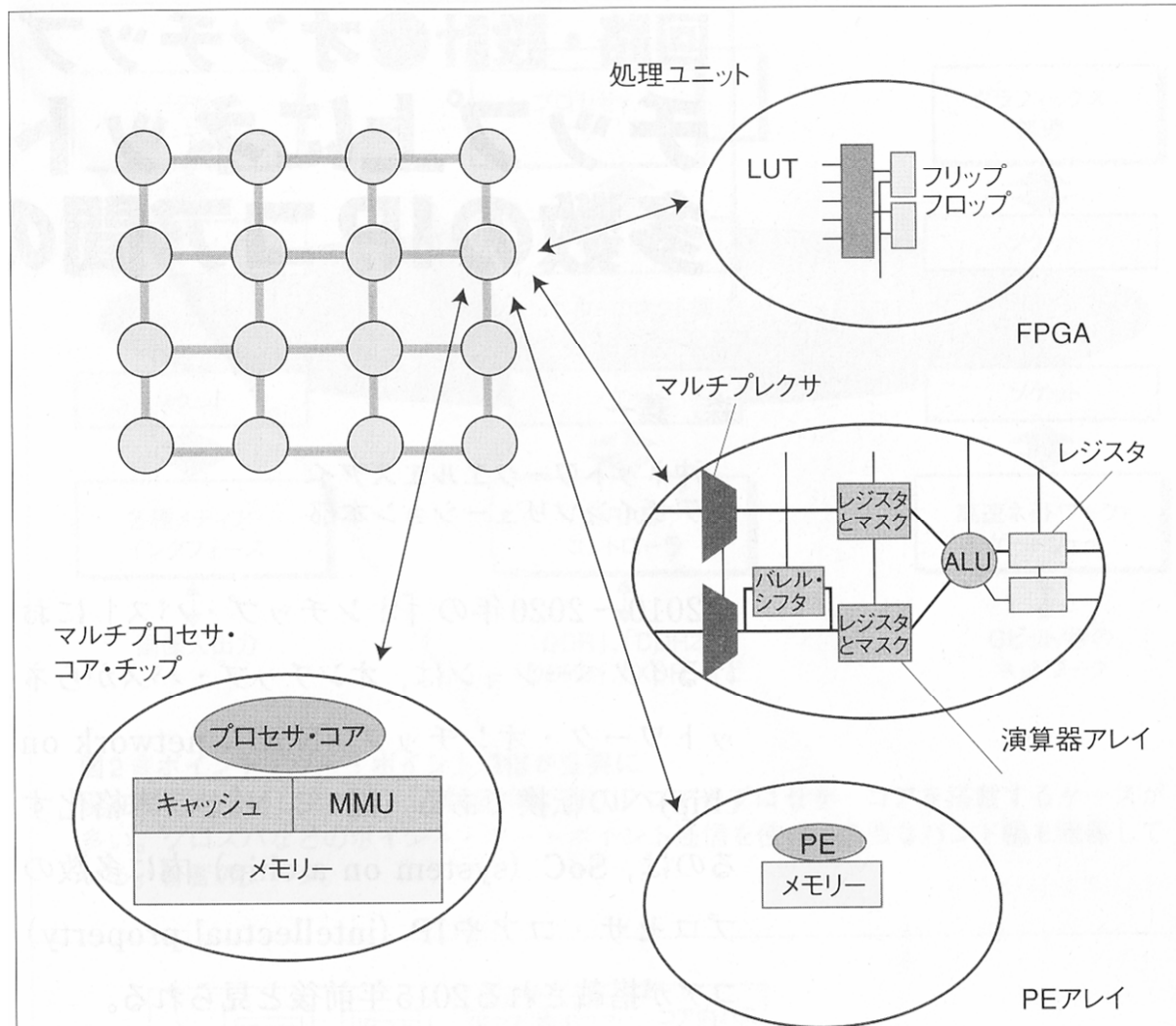
PEアレイ状配置

粗粒度: 演算器中心

動的に構造を切り替え可能 (16コンテキスト)

FFT, MPEGなど多様な応用に適用

IPFLEX社 DAPDNA-2



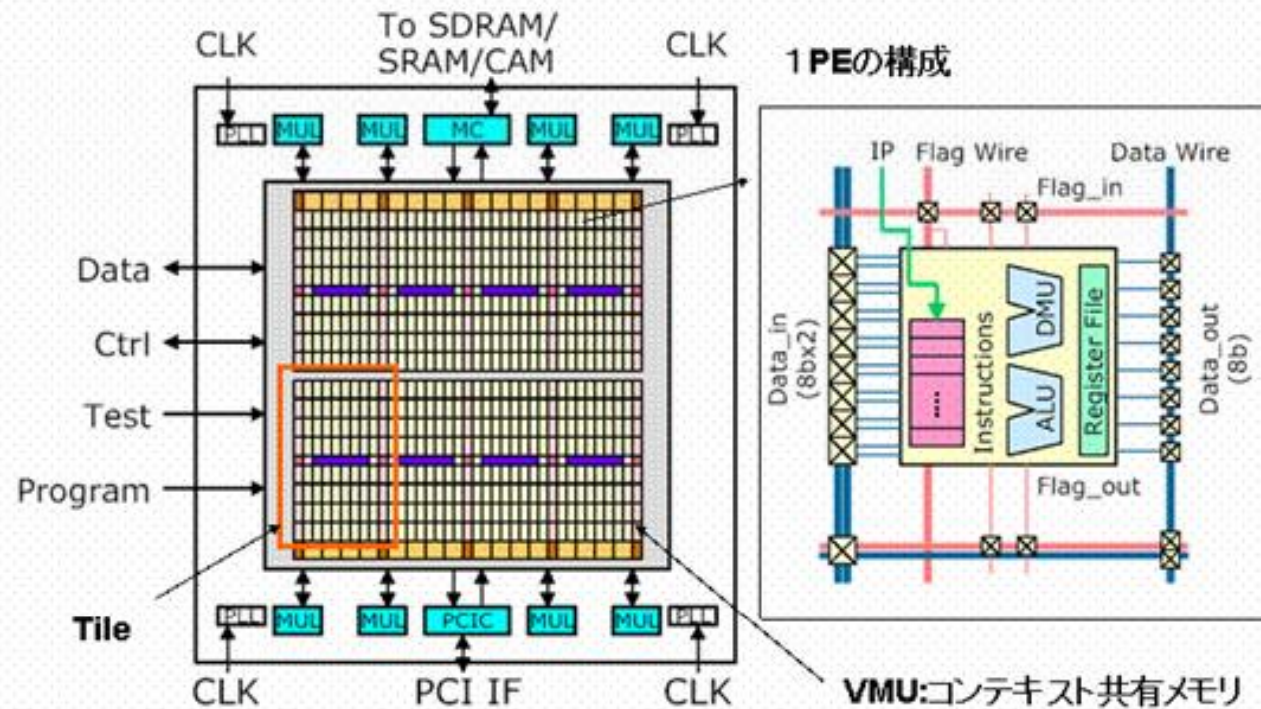
ALU : arithmetic logic unit
 FPGA : field programmable gate array
 LUT : look up table
 MMU : memory management unit
 PE : processing element

天野 : 日経Micro devices, No242, 2005

図2 ● 処理ユニットの例

ここではプロセサ・アレイを4種類に分類して、その具体例を示した。今後、さまざまな処理ユニットを持つプロセサ・アレイが登場してくると予想される。著者のデータ。

DRP



主な仕様

150nm CMOSプロセス

パッケージ：696ピンTBGA

動作電圧：外部3.3V、内部1.5V

PE数：512、PEアレイ：64

クロック周波数：最大133MHz

搭載メモリ：2Mビット

2ポートメモリ（VMEM：各タイルの左右に配置）

1ポートメモリ（HMEM：DRPコアの上下に配置）

IPFLEX社 :
DAPDNA-2

DAP		高性能 32 ビット RISC プロセッサ 命令キャッシュ 8K バイト、 データキャッシュ 8K バイト
DNA	DNA	動的再構成可能な 32 ビット PE の 2 次元アレイ
	PE 数	376 個 (演算エレメント 168 個)
	RAM 容量	576K バイト (RAM エレメント計 512KB + DNA バッファ計 64KB)
	コンフィギュレーション数	4 バンク (フォアグラウンド 1 バンク + バックグラウンド 3 バンク) 4 バンク以上は外部メモリからロード可
外部 インタ フェ ース	DNAダイレクト I/O	166MHz(最大、外部クロックに同期可)、 32 ビット幅、 入出力計 6 チャンネル : 32Gbps (DAPDNA-2 の複数接続も可能)
	DDR-SDRAM	166MHz、 64 ビット幅 DDR-SDRAM インタフェース 最大容量 512M バイト
	PCI	33MHz、 32 ビット幅 PCI インタフェース (3.3V トレラント)
	ROM	ブート用およびプログラム用 Serial ROM(SPI) インタフェース
	外部割込み	8 本
	その他	UART 2 チャンネル、 GPIO 16 チャンネル、 同期シリアル (マスタ) 1 チャンネル
動作周波数		166MHz
電源		2 電源 : 2.5V(I/O), 1.2V(コア)
パッケージ		FC-BGA パッケージ、 1156 ピン

文献

- (1) 富田眞治：コンピュータアーキテクチャ 第2版, 丸善, 2000
- (2) J. R. Goodman, D. Burger: Billion Transistor Architectures, IEEE Computer, pp. 46-93, Sep. 1997
- (3) J. R. Goodman, D. Burger: Billion Transistor Architectures, There and Back Again, IEEE Computer, pp. 22-28, March 2004
- (4) 富田眞治：計算機アーキテクチャの過去, 現在, 未来, 情報処理, 41, 5, 2000