講義内容

目的：仮想的なプロセッサ(データパス)を設計

単一サイクル方式

- データパスの実現
- 制御論理の設計
- 単一サイクル方式の欠点

パイプライン処理

- 方式比較
- データパス/制御部のパイプライン化
- パイプラインハザード
- 例外／割り込み
- 更なる高速化手法

基本知識1～データパス設計の重要性

- マシン性能(CPU時間)を決定する3つの要因

CPU時間＝実行命令数×ＣＰＩ×クロックサイクル時間

命令セット・アーキテクチャ
(狭義のアーキテクチャ)

実現方法(Implementation)
(広義のアーキテクチャ)

性能は実現方法(Implementation)に大きく影響される

ＣＰＩ (Clock cycle Per Instruction) : 命令を実行するために必要なクロックサイクル数

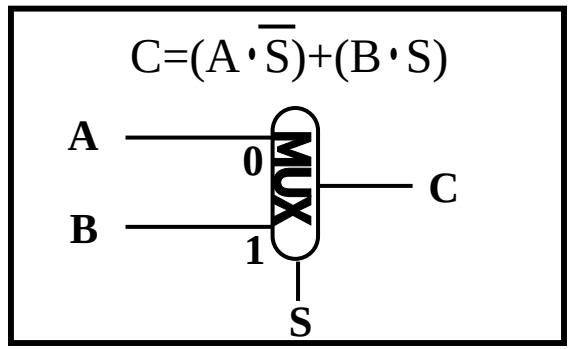
基本知識2～論理回路の種類

① 組合せ論理回路 (Combinational logic)

=> **データを記憶できない回路**

- ANDｹﾞｰﾄ、ORｹﾞｰﾄ、マルチプレクサ
- 上記回路で構成されるALU

マルチプレクサ (セレクタ)

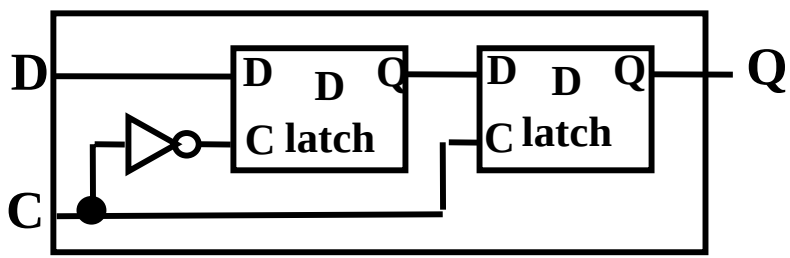


② 順序回路・状態論理要素 (Sequential logic)

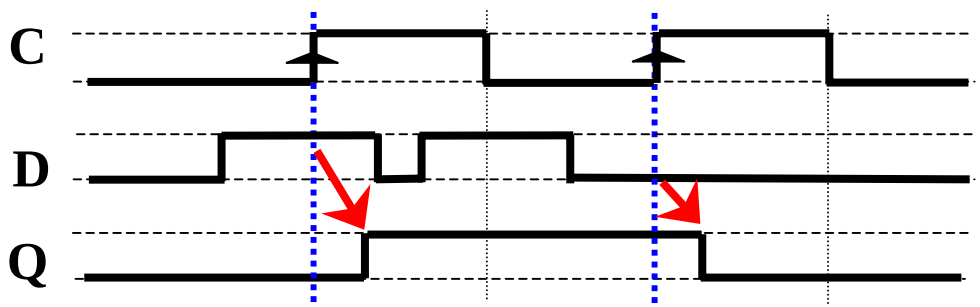
=> **データを記憶できる回路**

- フリップフロップ、ラッチ
- 複数の記憶素子で構成されたレジスタファイル、メモリ

フリップフロップ

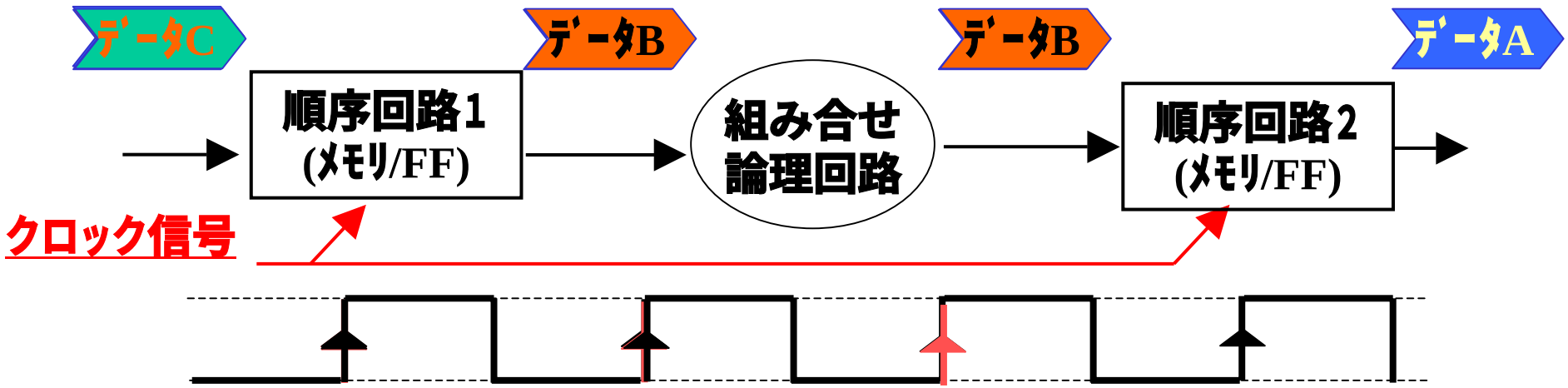


クロックのエッジでデータを取り込む



基本知識3：クロック方式と同期回路

エッジトリガクロック方式



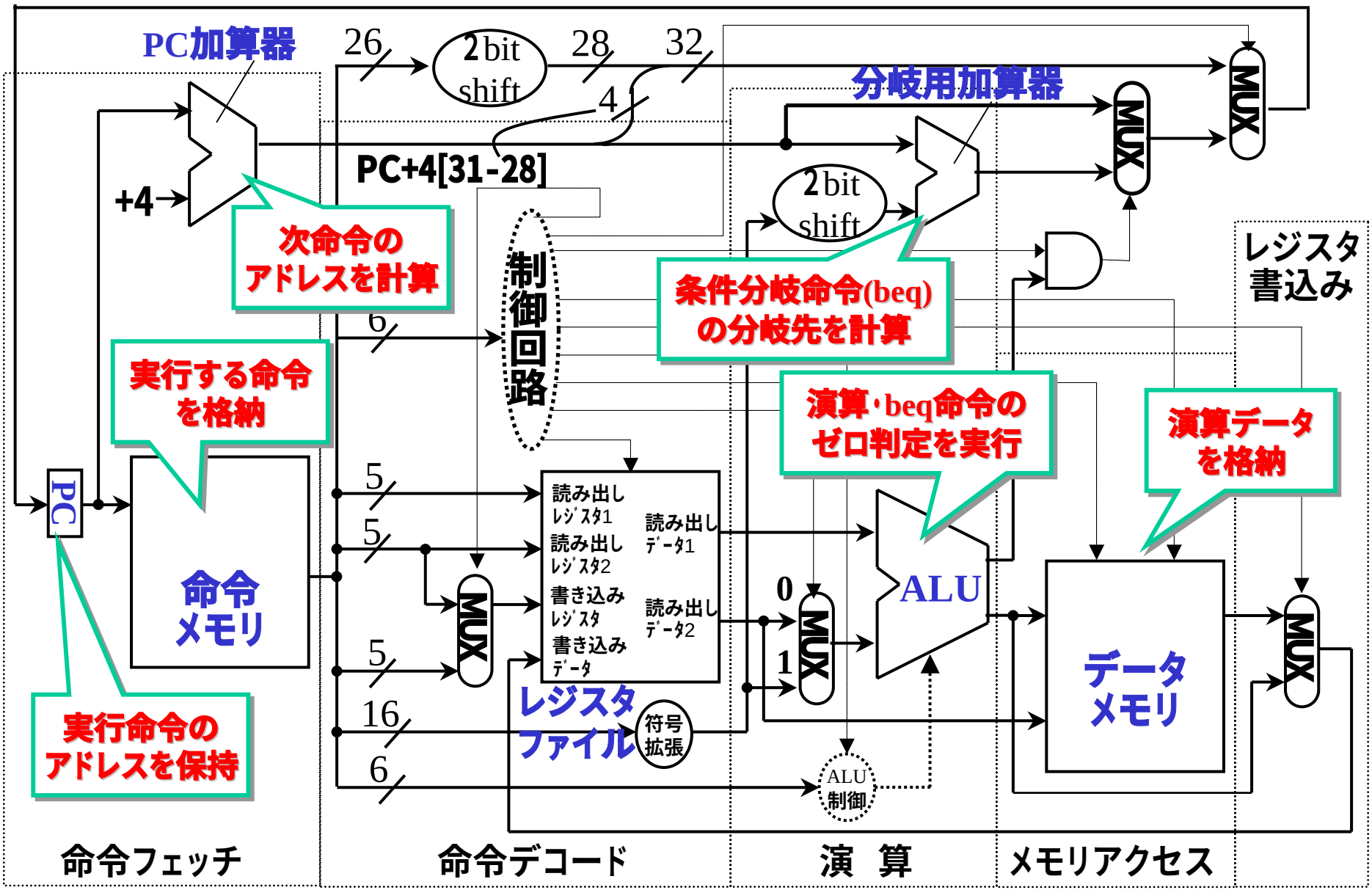
- 順序回路の間に組み合わせ回路が挟まれる。
- クロック信号の立ち上がりで順所回路のデータが更新される。
- 順序回路1から2まで、1クロックサイクル以内に信号を伝播しなければならない。
  =>この伝播時間がクロックサイクル時間を決める。
- 一般に、論理機能図ではクロック信号は省略する

本講義で実現するデータパスの命令セット一覧

		31	25	20	15	10	5	0	
命令		op	rs	rt	rd	shamt	funct		意　味
R形式	add	0	$被加数	$加数	$和	0	32		$和=$被加数+$加数
	sub	0	$被減数	$減数	$差	0	34		$差=$被減数-$減数
I形式命令	lw	35	$index	$転送先	転送元相対ｱﾄﾞﾚｽ				$転送先＝ﾒﾓﾘ [$index+転送元相対ｱﾄﾞﾚｽ]
	sw	43	$index	$転送元	転送先相対ｱﾄﾞﾚｽ				ﾒﾓﾘ [$index+ 転送元相対ｱﾄﾞﾚｽ] ＝$転送元
	beq	4	$A	$B	ｼﾞｬﾝﾌﾟ先Offset				$A=$B → go toｼﾞｬﾝﾌﾟ先
J	j	2	ｼﾞｬﾝﾌﾟ先 (擬似直接ｱﾄﾞﾚｽ)						go toｼﾞｬﾝﾌﾟ先 (擬似直接ｱﾄﾞﾚｽ)

6命令を実現したプロセッサ(データパス) を設計

単一サイクル・データパスと部品

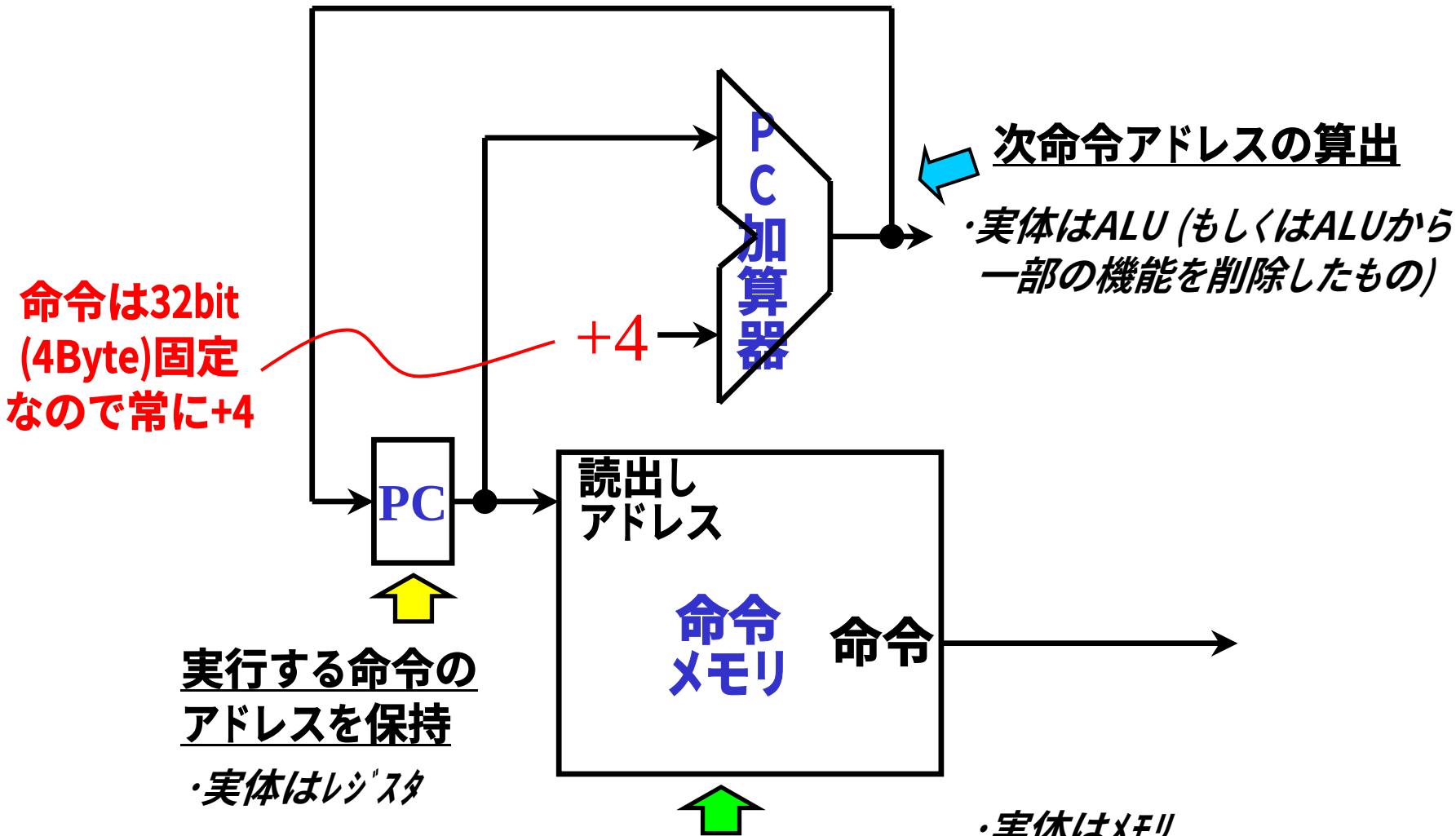


プロセッサを構成する部品

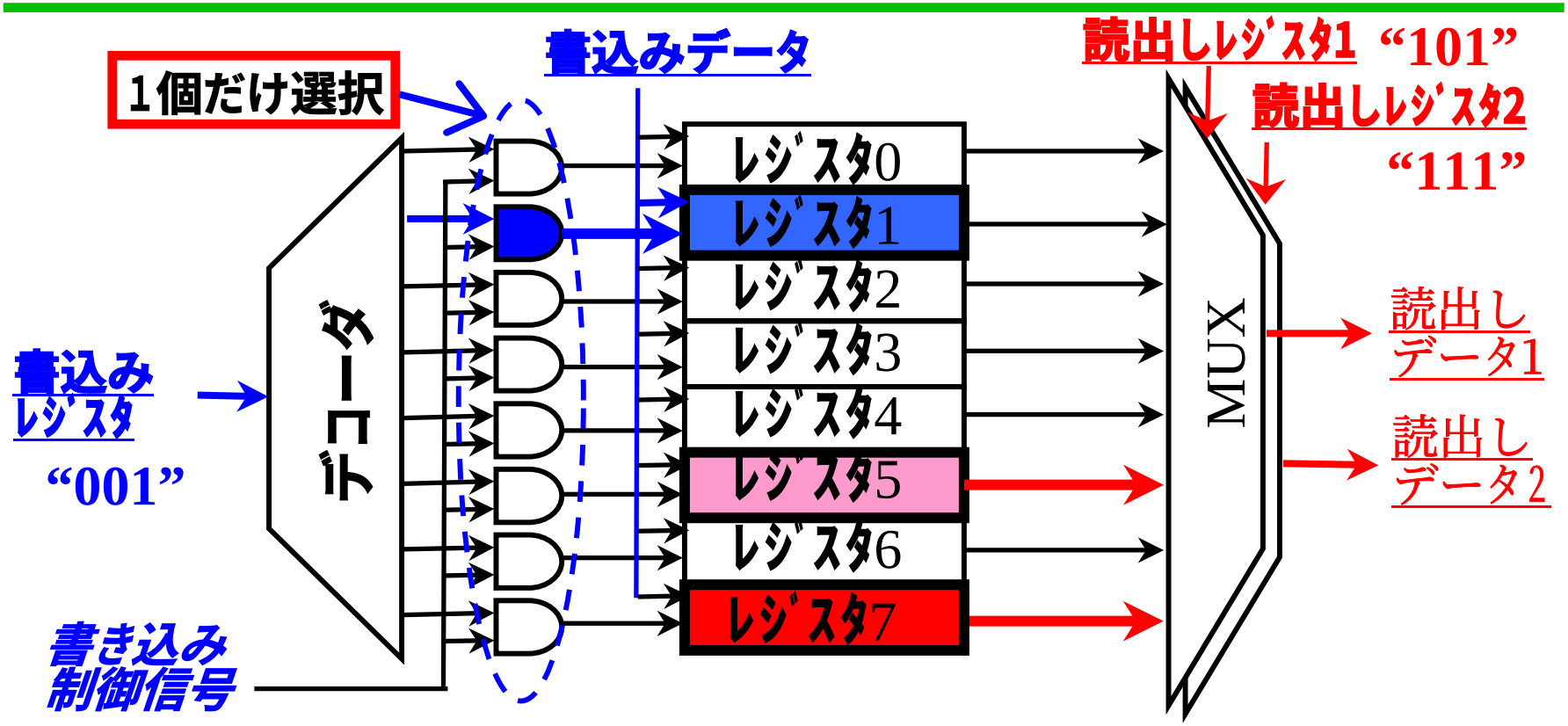
部品	S/C	機能
命令メモリ	(S)	実行する命令を格納
プログラムカウンタ (PC)	S	実行命令のアドレスを保持
PC加算器 (PC Adder)	C	次命令アドレスを計算
レジスタファイル	S	演算データを一時保持
ALU	C	演算･ゼロ判定を実行
分岐用加算器	C	条件分岐命令の分岐先を計算
データメモリ	S	演算データを格納
制御回路	C	実行する命令を制御

C：組み合わせ論理回路　S:順序回路

プロセッサを構成する部品(命令フェッチ関連)



プロセッサを構成する部品(レジスタファイル)

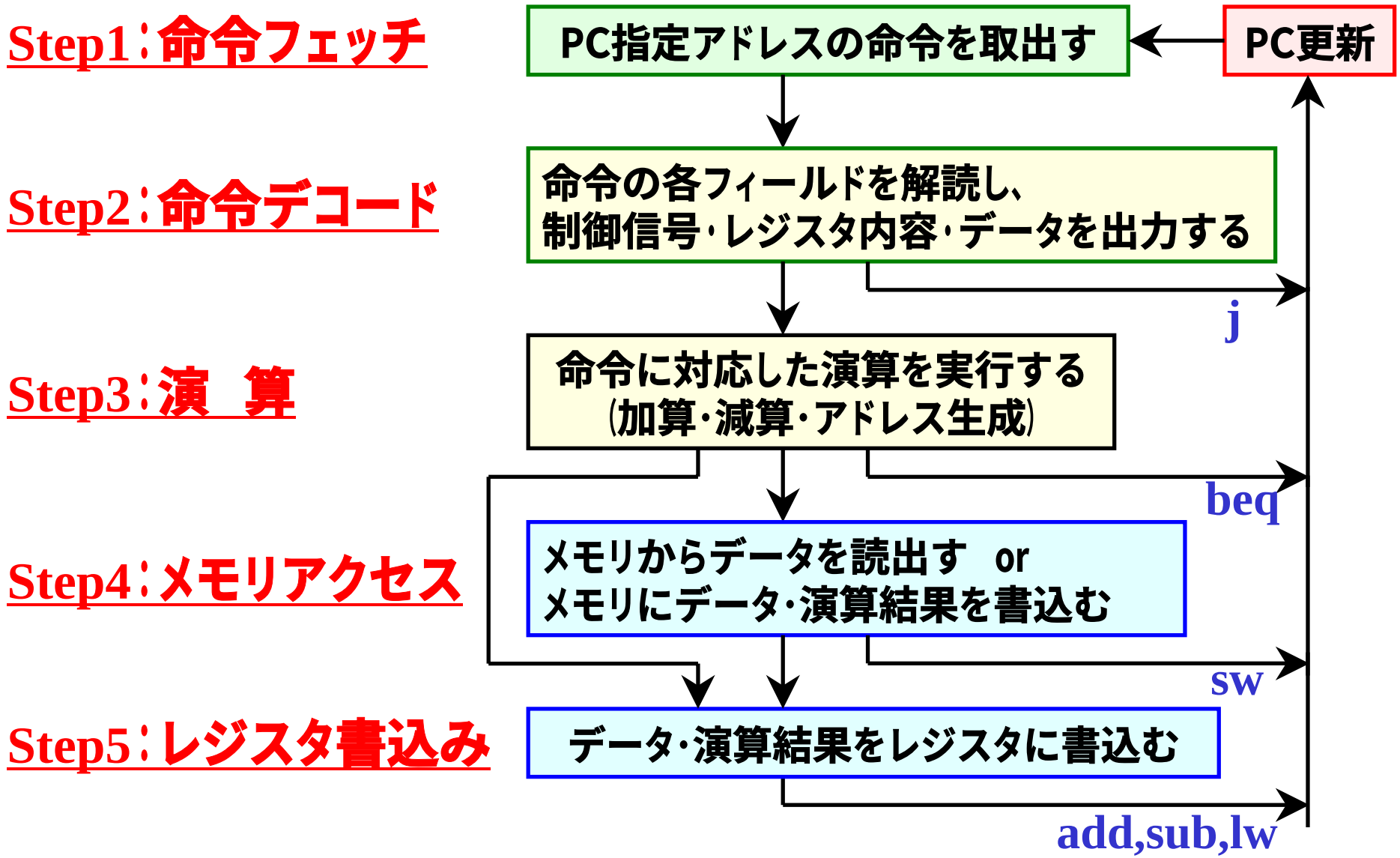


レジスタ：元々はプログラマから見える記憶領域のこと。実態はフリップフロップの集まり

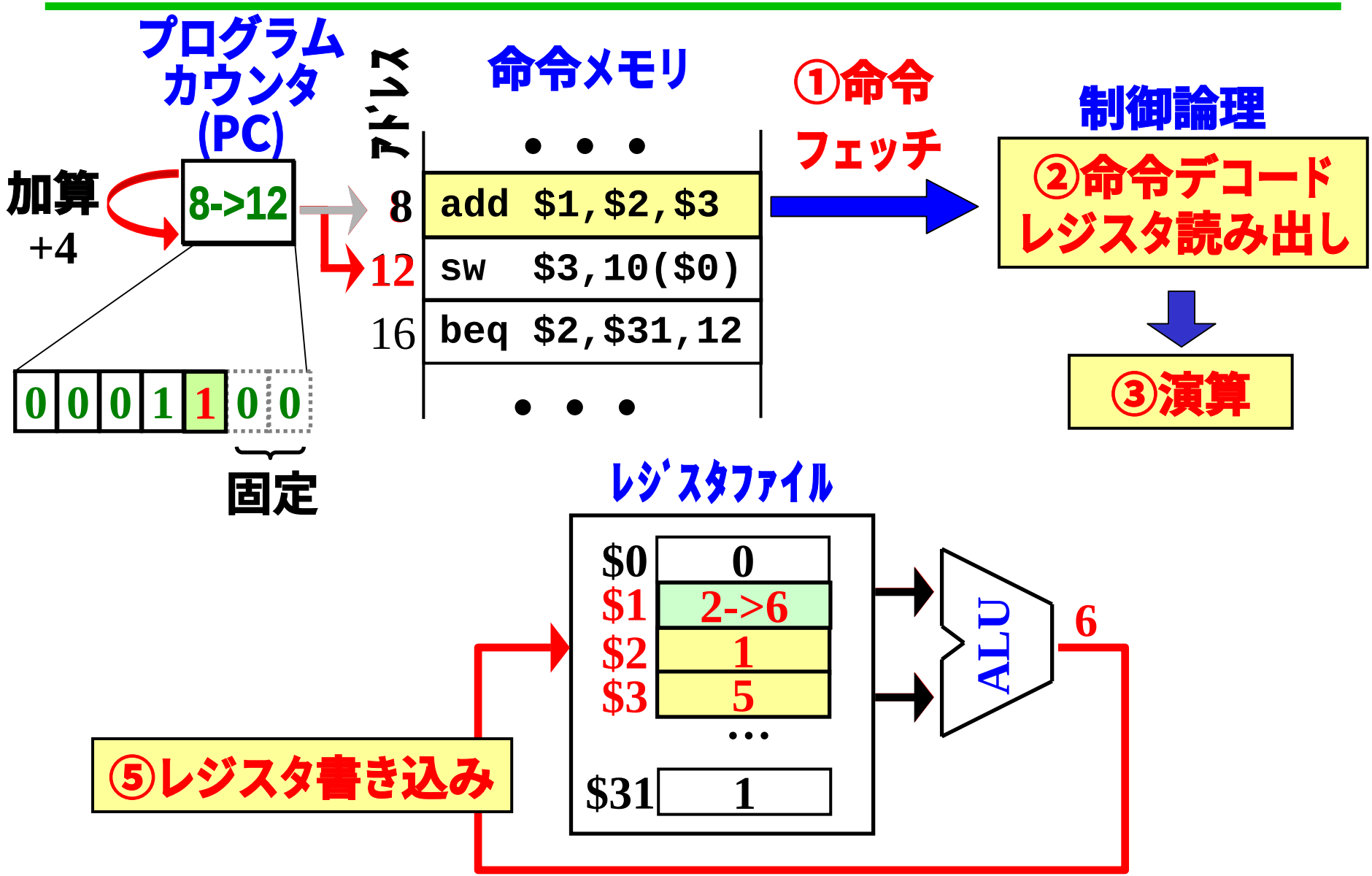
レジスタファイル：レジスタの集まり。(レジスタファイルをレジスタと呼ぶ事もある)

- 1クロックサイクル内で書き込みと読み出しの同時動作が可能
- 高速動作が可能（今回設計するデータパスでは、演算するデータをデータメモリから一度レジスタファイルにロードしてから演算が実行される。）

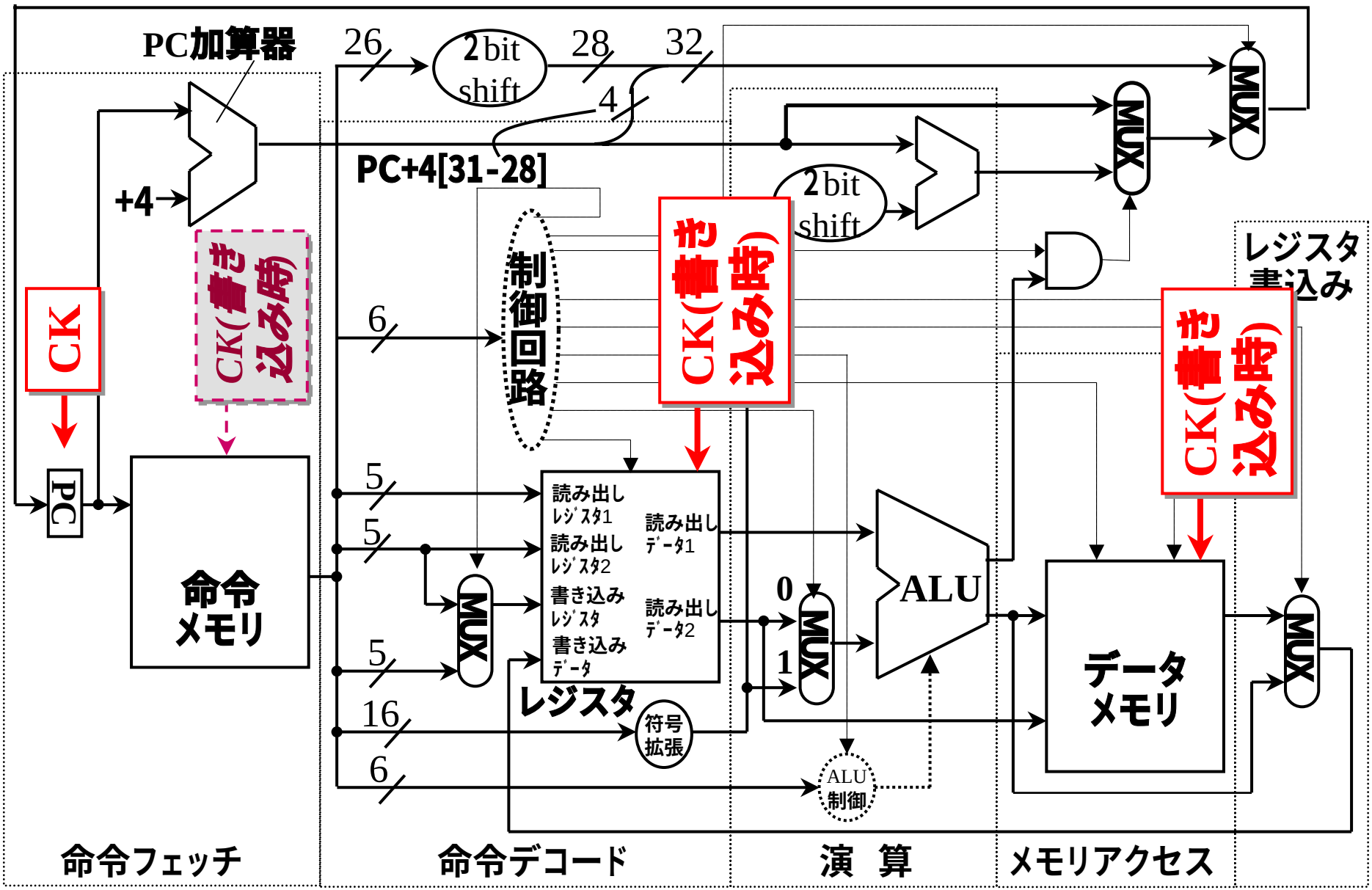
プロセッサの実行ステップと命令の関係



プロセッサの処理イメージ1

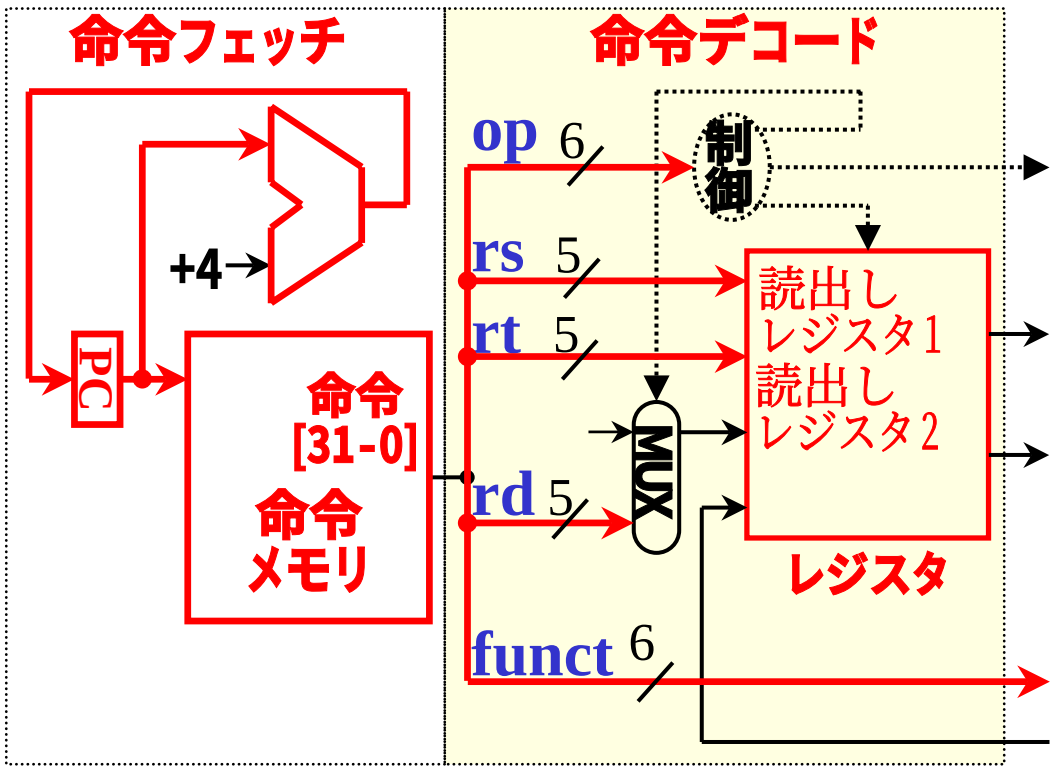


単一サイクル・データパス



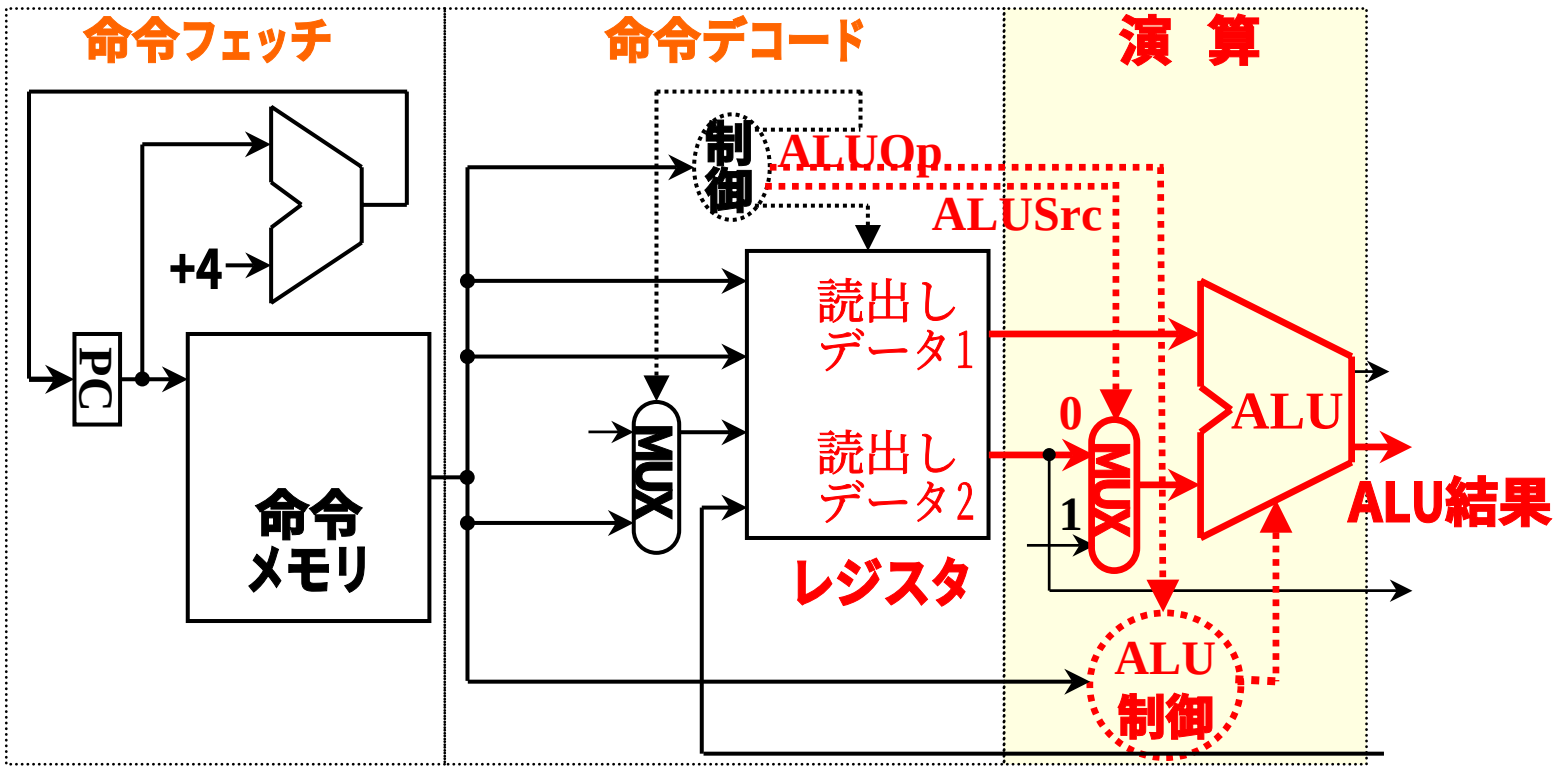
R形式命令のデータパス：命令デコード

	31	25	20	15	10	5	0	
R形式命令	op	rs	rt	rd	shamt	funct		意　味
add	0	$被加数	$加数	$和	0	32		$和＝$被加数＋$加数
subtract	0	$被減数	$減数	$差	0	34		$差＝$被減数－$減数



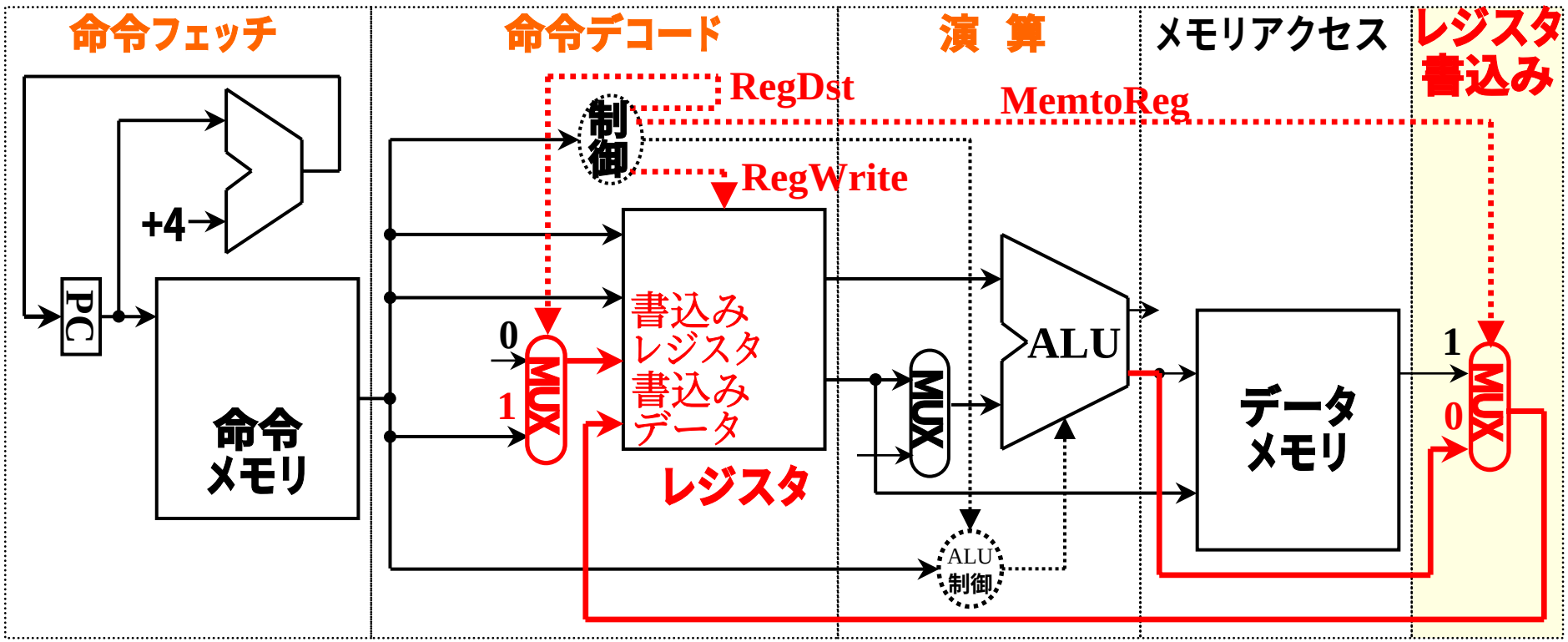
R形式命令のデータパス：演　算

	31	25	20	15	10	5	0	
R形式命令	op	rs	rt	rd	shamt	funct		意　味
add	0	$被加数	$加数	$和	0	32		$和＝$被加数＋$加数
subtract	0	$被減数	$減数	$差	0	34		$差＝$被減数－$減数



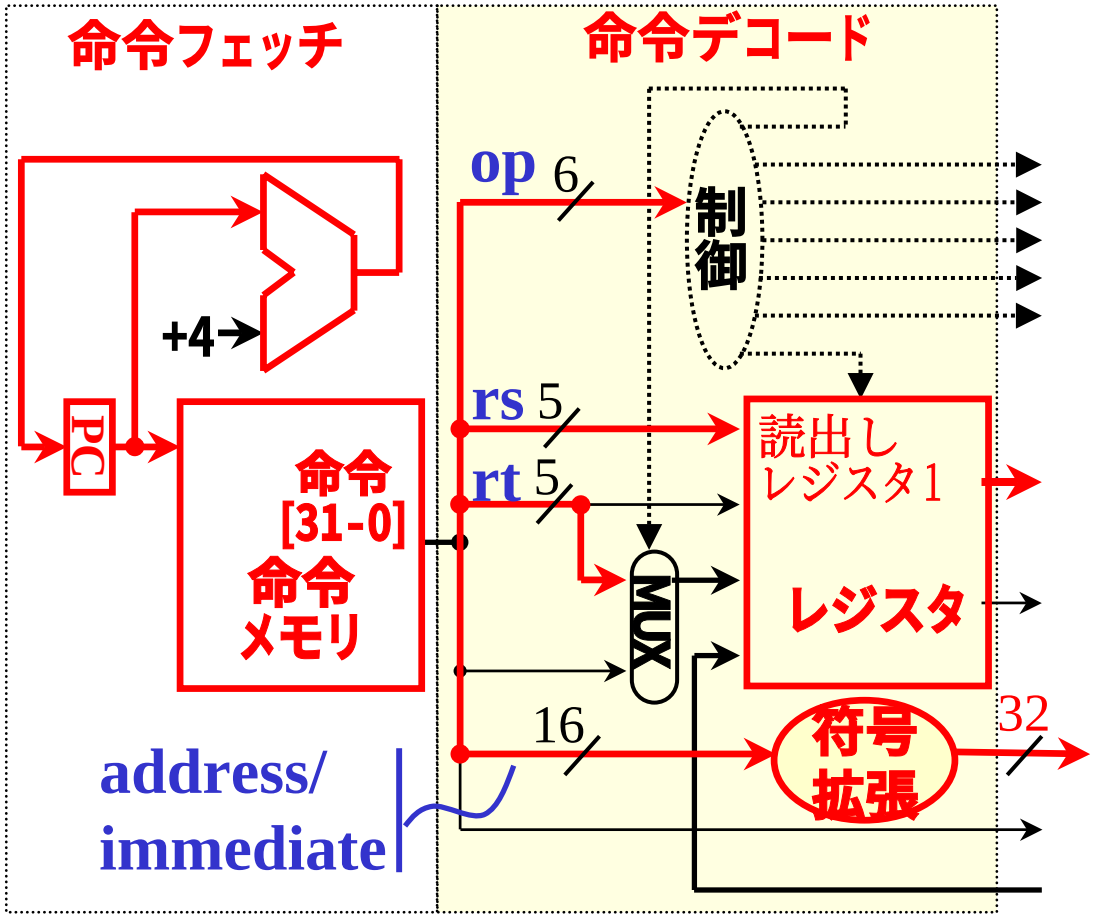
R形式命令のデータパス：レジスタ書込み

	31	25	20	15	10	5	0	
R形式命令	op	rs	rt	rd	shamt	funct		意　味
add	0	$被加数	$加数	$和	0	32		$和＝$被加数＋$加数
subtract	0	$被減数	$減数	$差	0	34		$差＝$被減数－$減数



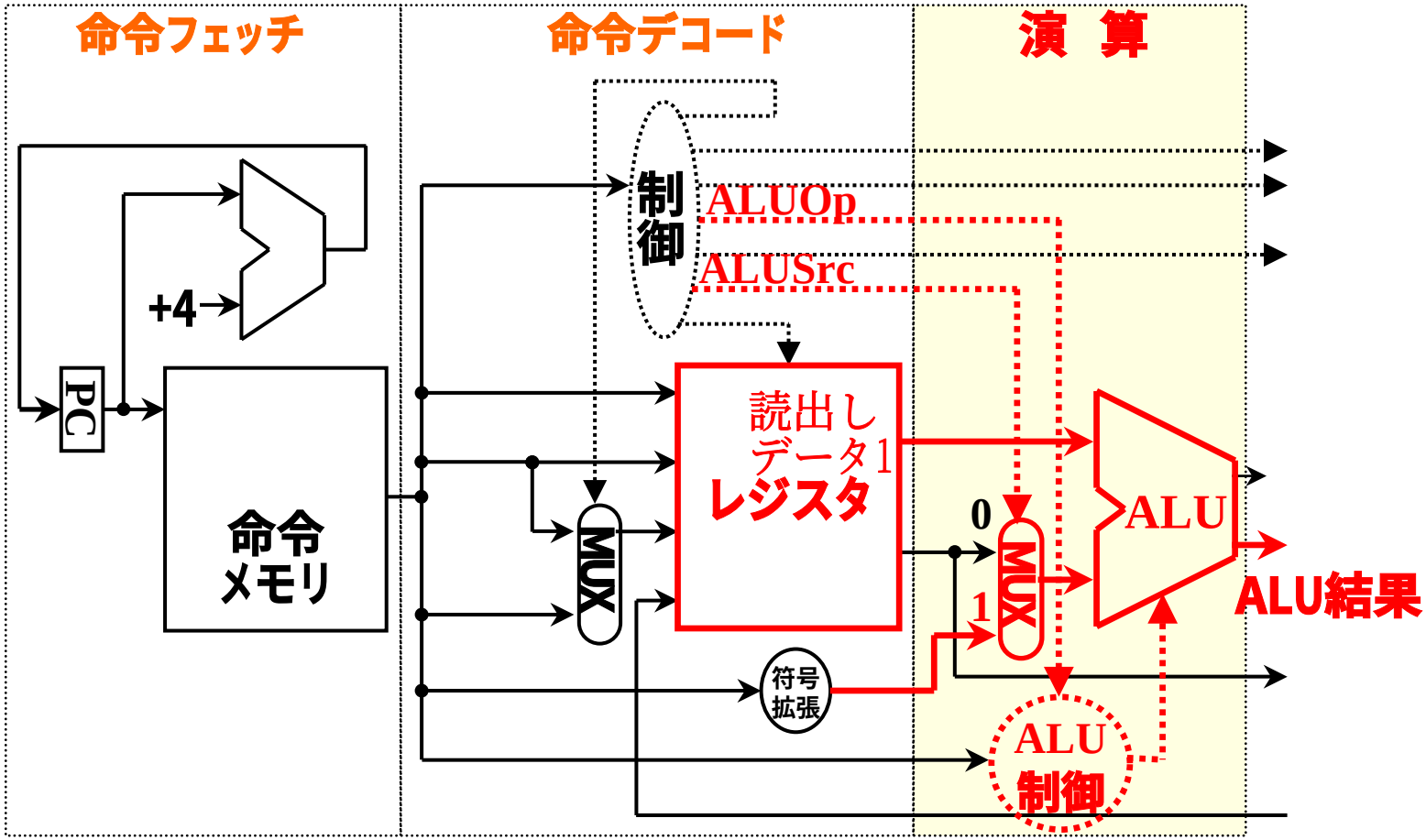
I 形式 load命令のデータパス：命令デコード

	31	25	20	15	10	5	0
I形式命令	op	rs	rt	address/immediate			意　味
load word	35	$index	$転送先	転送元相対ｱﾄﾞﾚｽ			$転送先＝ﾒﾓﾘ [$index+転送元相対ｱﾄﾞﾚｽ]



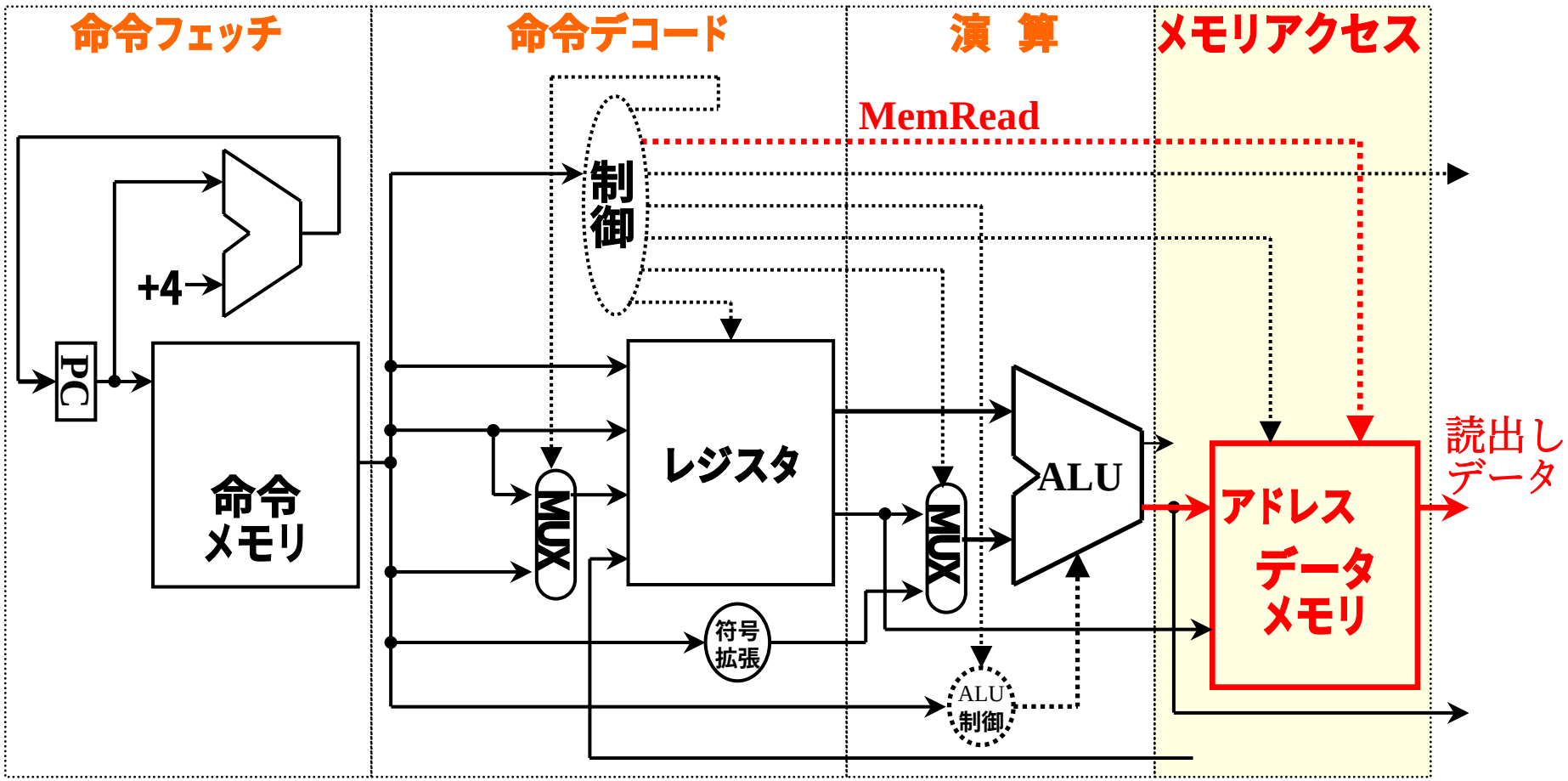
I形式 load命令のデータパス：演　算

	31	25	20	15	10	5	0
I形式命令	op	rs	rt	address/immediate			意　味
load word	35	$index	$転送先	転送元相対ｱﾄﾞﾚｽ			$転送先＝ﾒﾓﾘ［$index+転送元相対ｱﾄﾞﾚｽ］



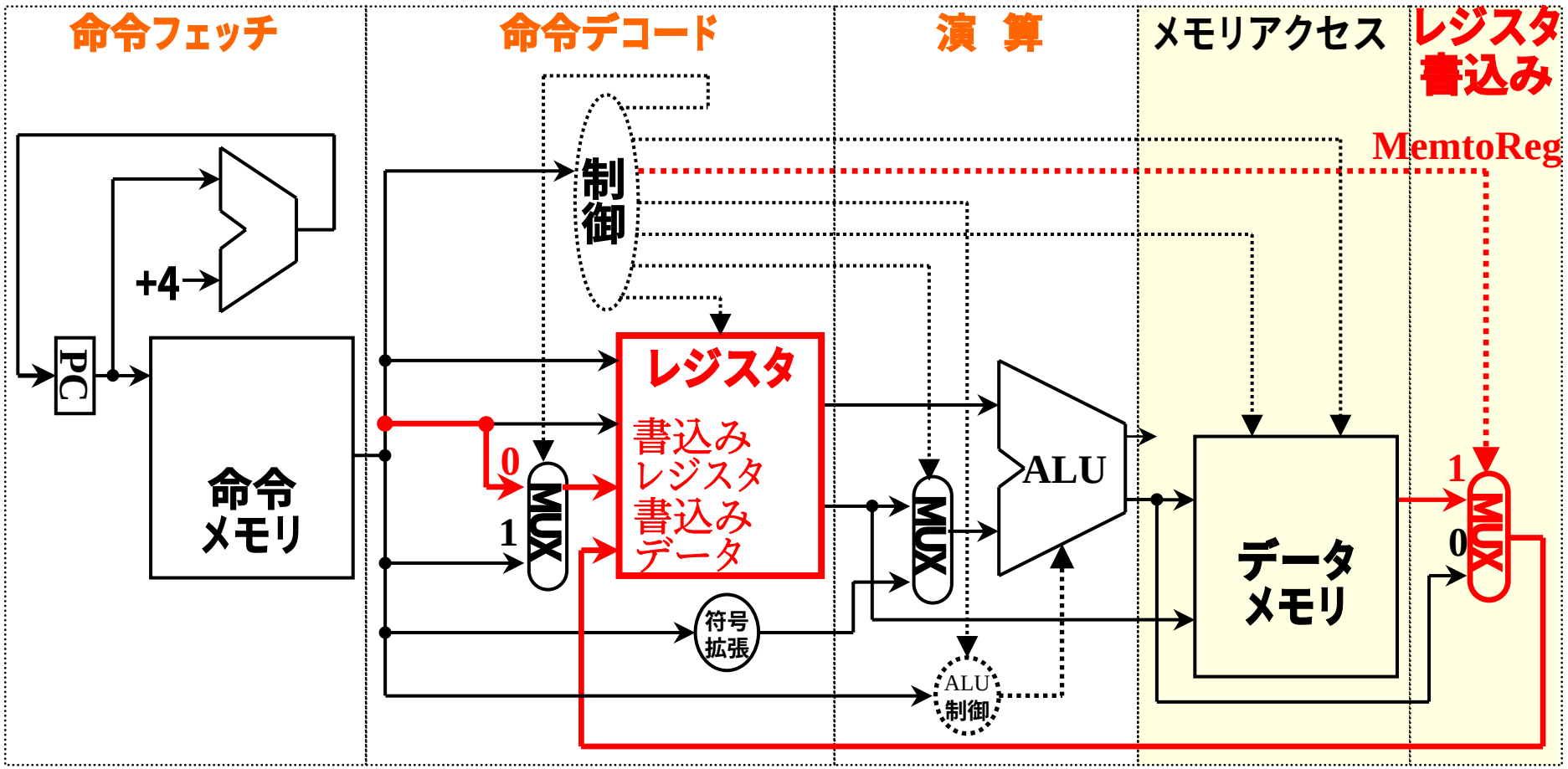
I 形式 load命令のデータパス：メモリアクセス

	31	25	20	15	10	5	0
I形式命令	op	rs	rt	address/immediate			意　味
load word	35	$index	$転送先	転送元相対ｱﾄﾞﾚｽ			$転送先＝ﾒﾓﾘ [$index+転送元相対ｱﾄﾞﾚｽ]



I 形式 load命令のデータパス：レジスタ書込み

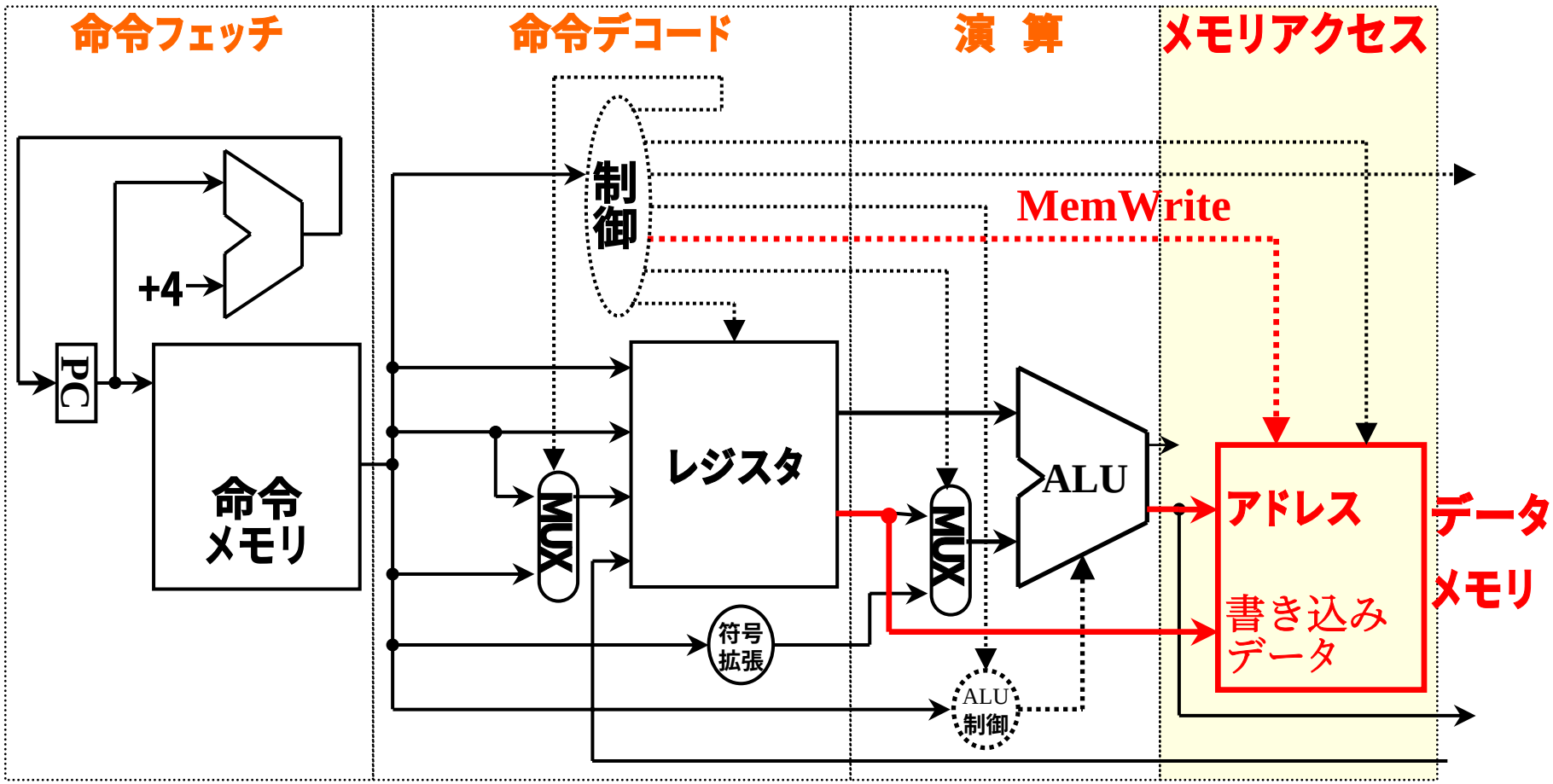
	31	25	20	15	10	5	0	
I形式命令	op	rs	rt	address/immediate				意　味
load word	35	$index	$転送先	転送元相対ｱﾄﾞﾚｽ				$転送先＝ﾒﾓﾘ [$index+転送元相対ｱﾄﾞﾚｽ]



I 形式 storeの命令データパス：メモリアクセス

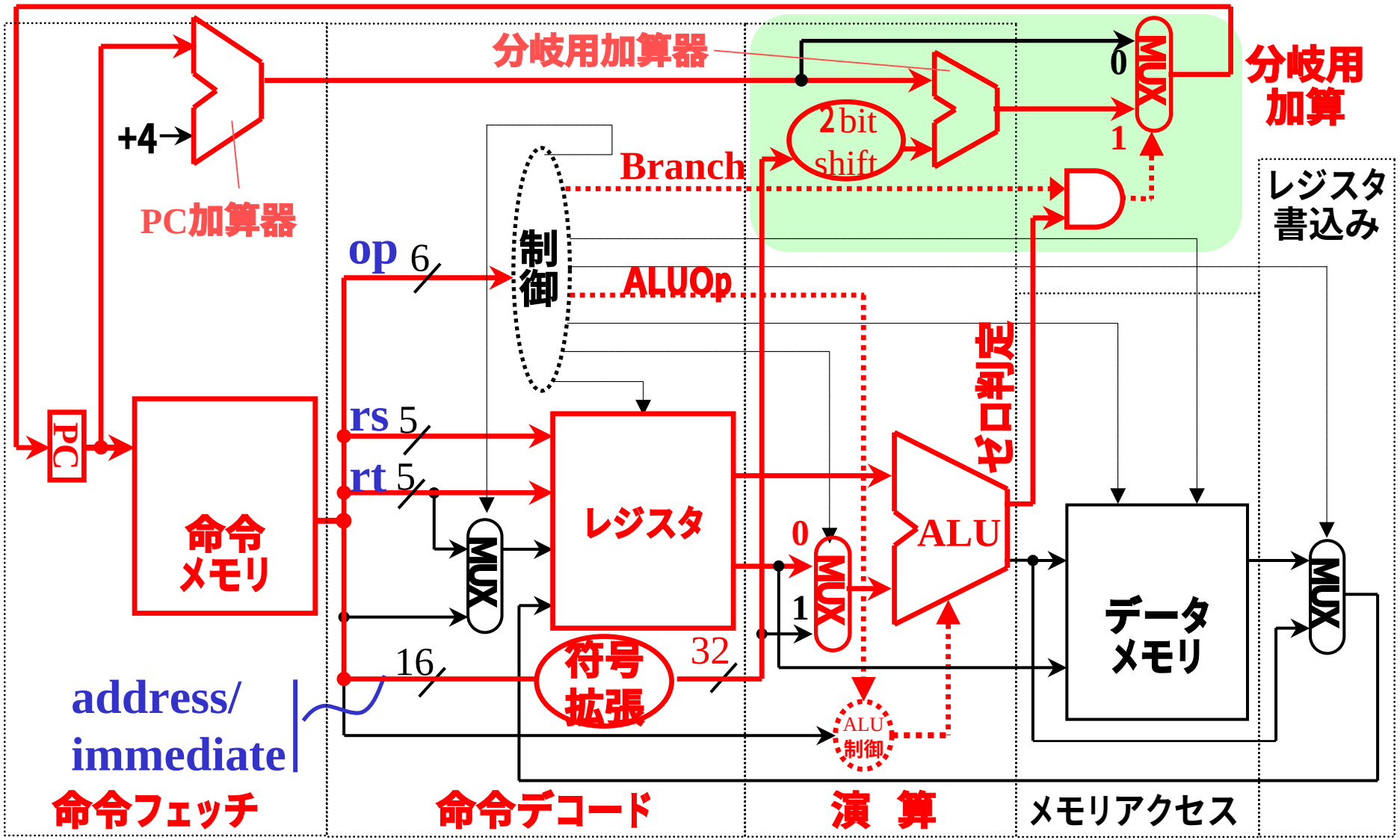
31	25	20	15	10	5	0

I形式命令	op	rs	rt	address/immediate	意　味
store word	43	$index	$転送元	転送先相対ｱﾄﾞﾚｽ	ﾒﾓﾘ［$index+転送先相対ｱﾄﾞﾚｽ］＝$転送元

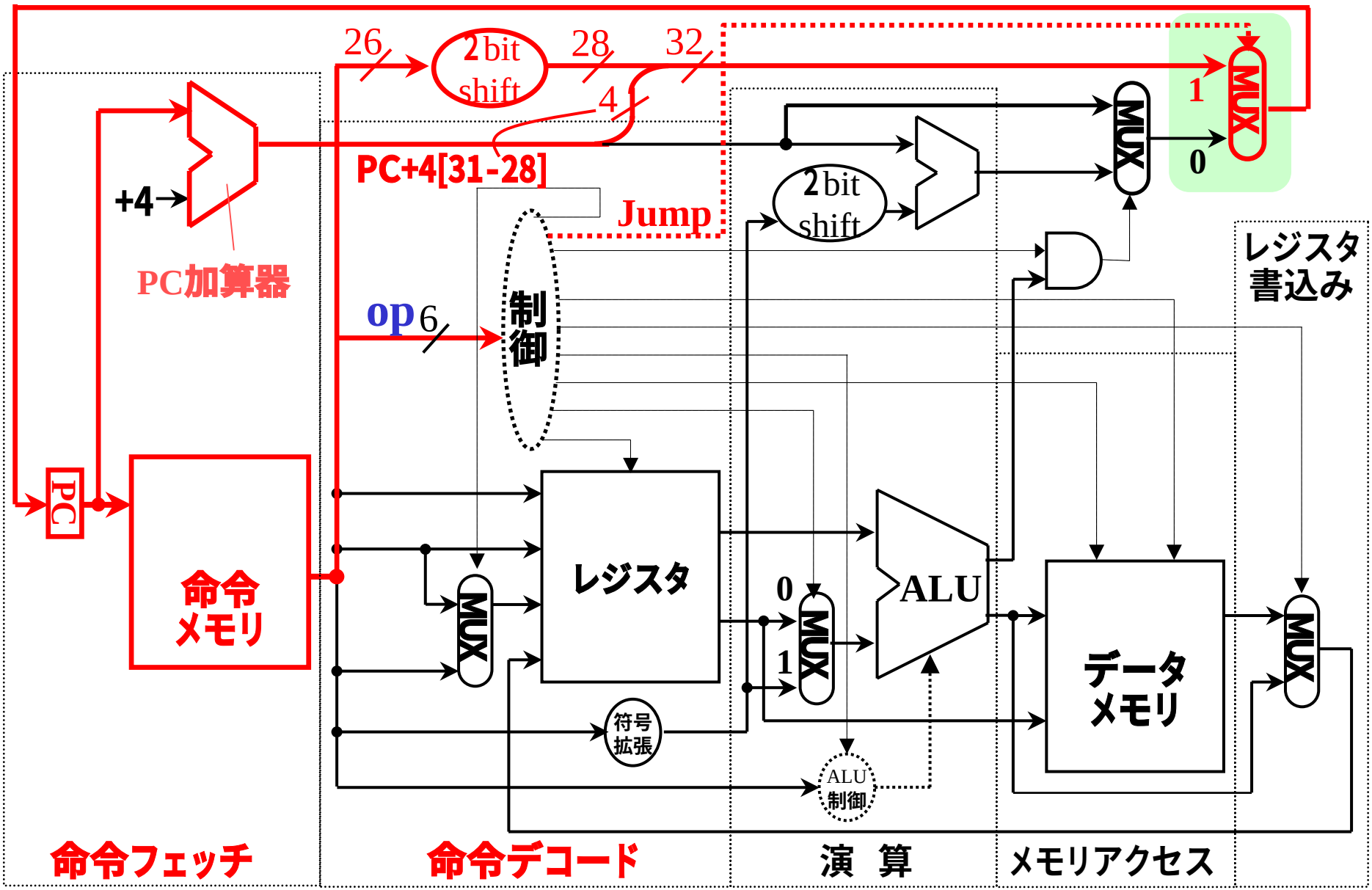


I 形式 branch命令のデータパス

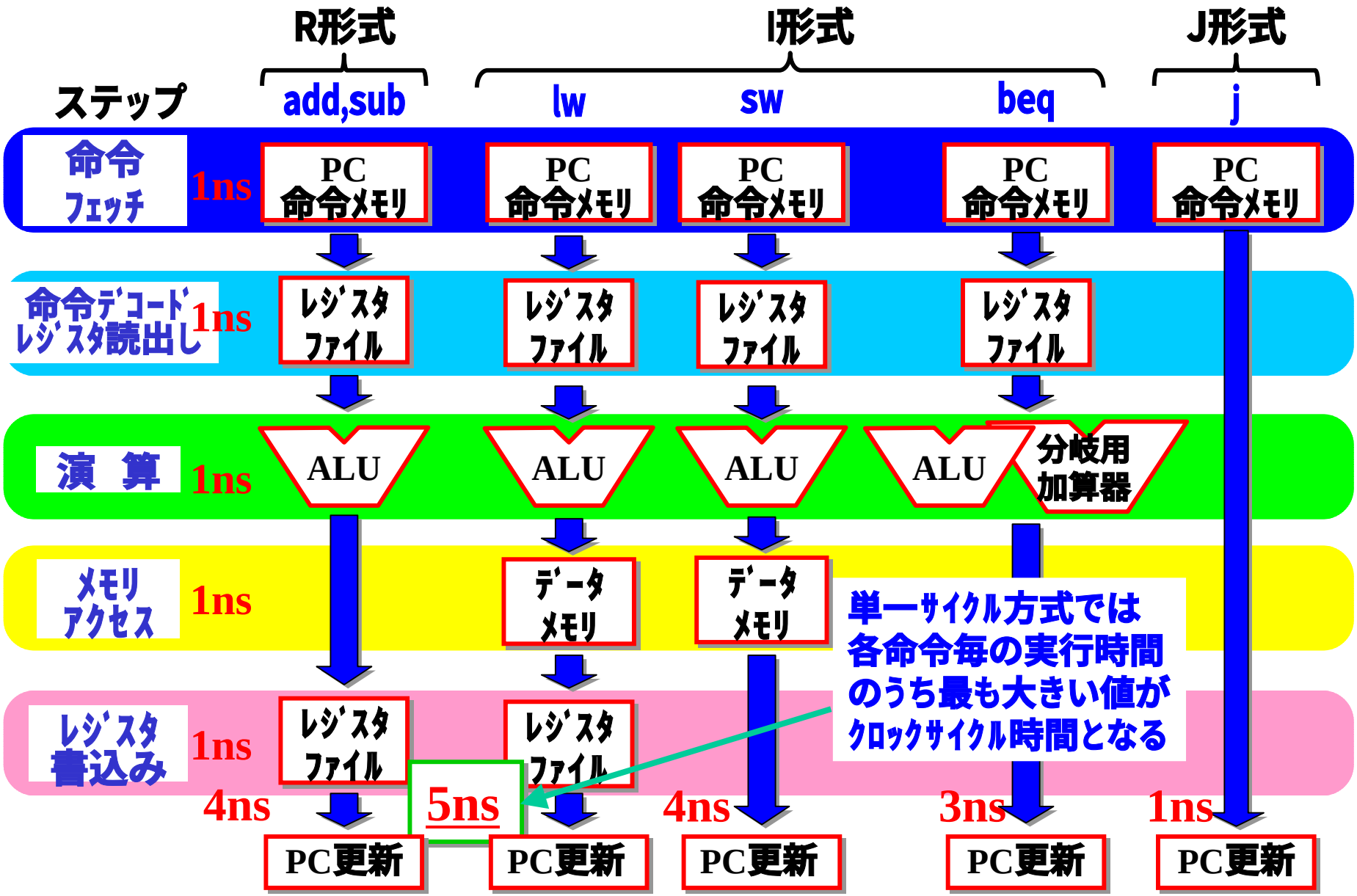
branch on equal	4	$A	$B	ｼﾞｬﾝﾌﾟ先Offset	$A＝$B→go to ｼﾞｬﾝﾌﾟ先



J 形式命令のデータパス



各命令毎の実行時間とサイクル時間



単一サイクル・データパスとサイクル時間

単一サイクル・データパスでは実行時間が最も大きい lw命令でサイクル時間が決まる