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

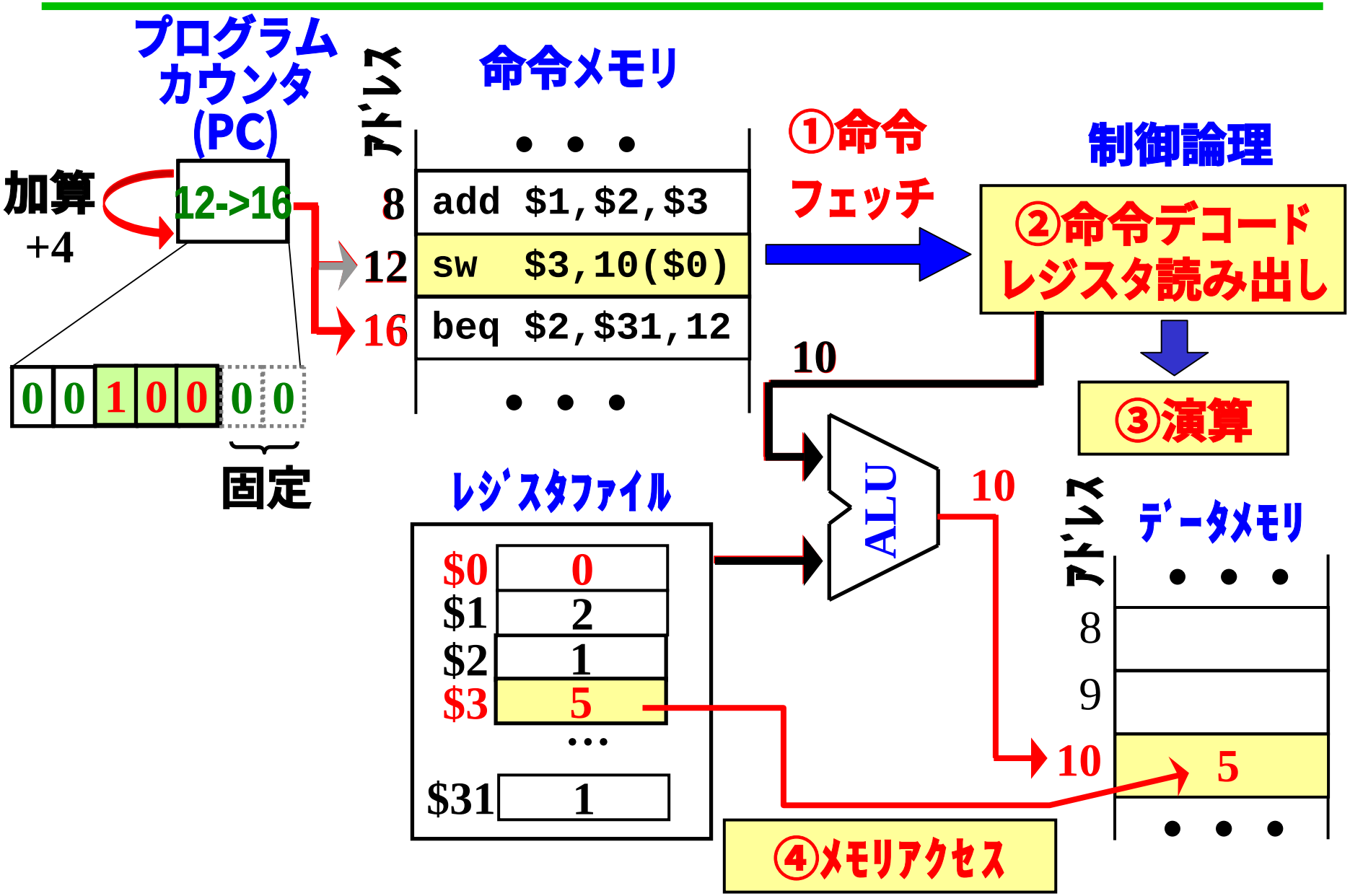
- 方式比較
- データパス/制御部のパイプライン化
- パイプラインハザード
- 更なる高速化手法

実現する命令セット一覧

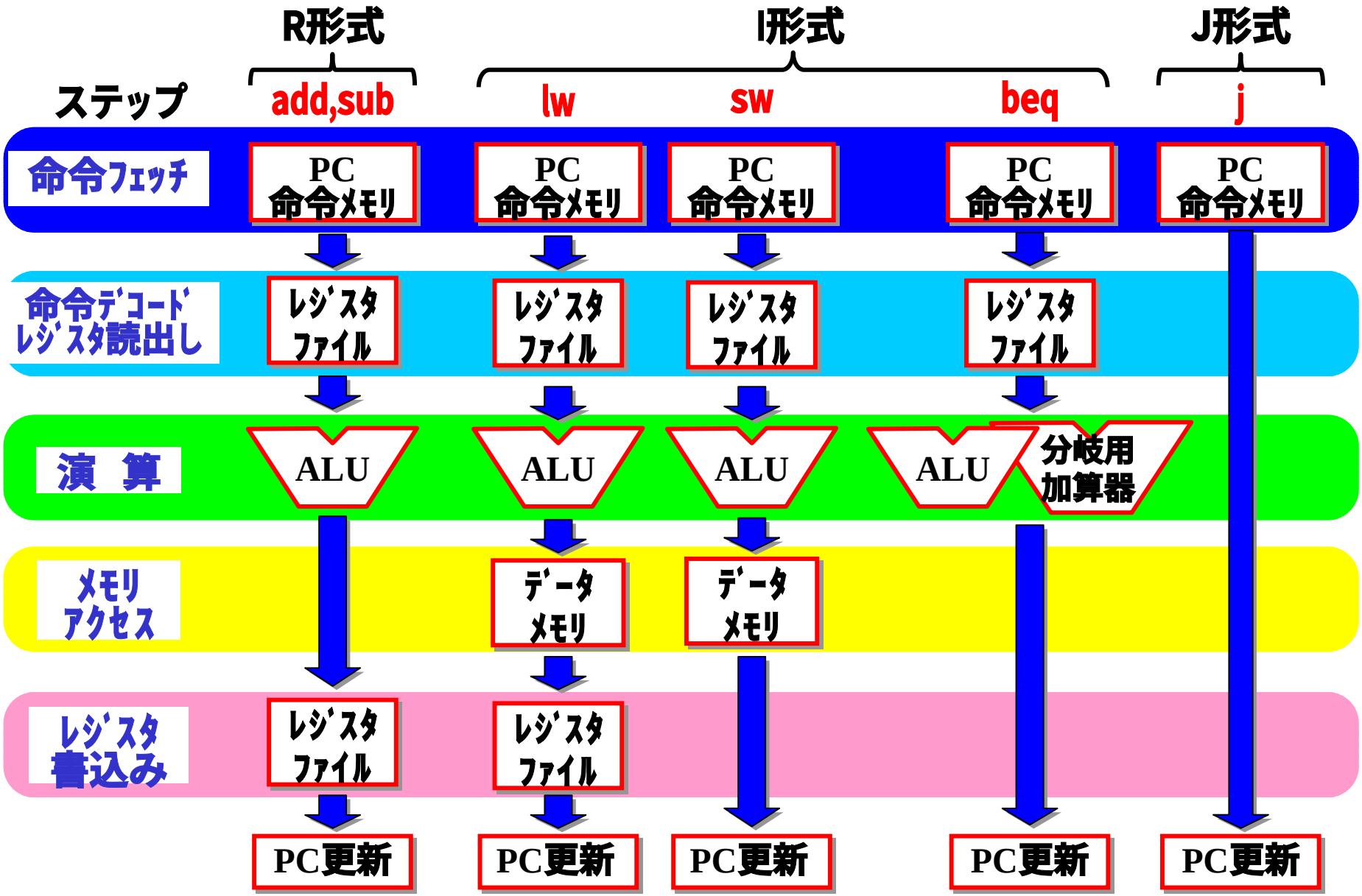
		31	25	20	15	10	5	0	
命令		op	rs	rt	rd	shamt	funct		意　味
R形式	add	0	$被加数	$加数	$和	0	32		$和=$被加数+$加数
	sub	0	$被減数	$減数	$差	0	34		$差=$被減数-$減数
I形式命令	lw	35	$index	$転送先	転送元相対ｱﾄﾞﾚｽ				$転送先＝ﾒﾓﾘ [$index+転送元相対ｱﾄﾞﾚｽ]
	sw	43	$index	$転送元	転送先相対ｱﾄﾞﾚｽ				ﾒﾓﾘ [$index+ 転送元相対ｱﾄﾞﾚｽ] ＝$転送元
	beq	4	$A	$B	ｼﾞｬﾝﾌﾟ先Offset				$A=$B → go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

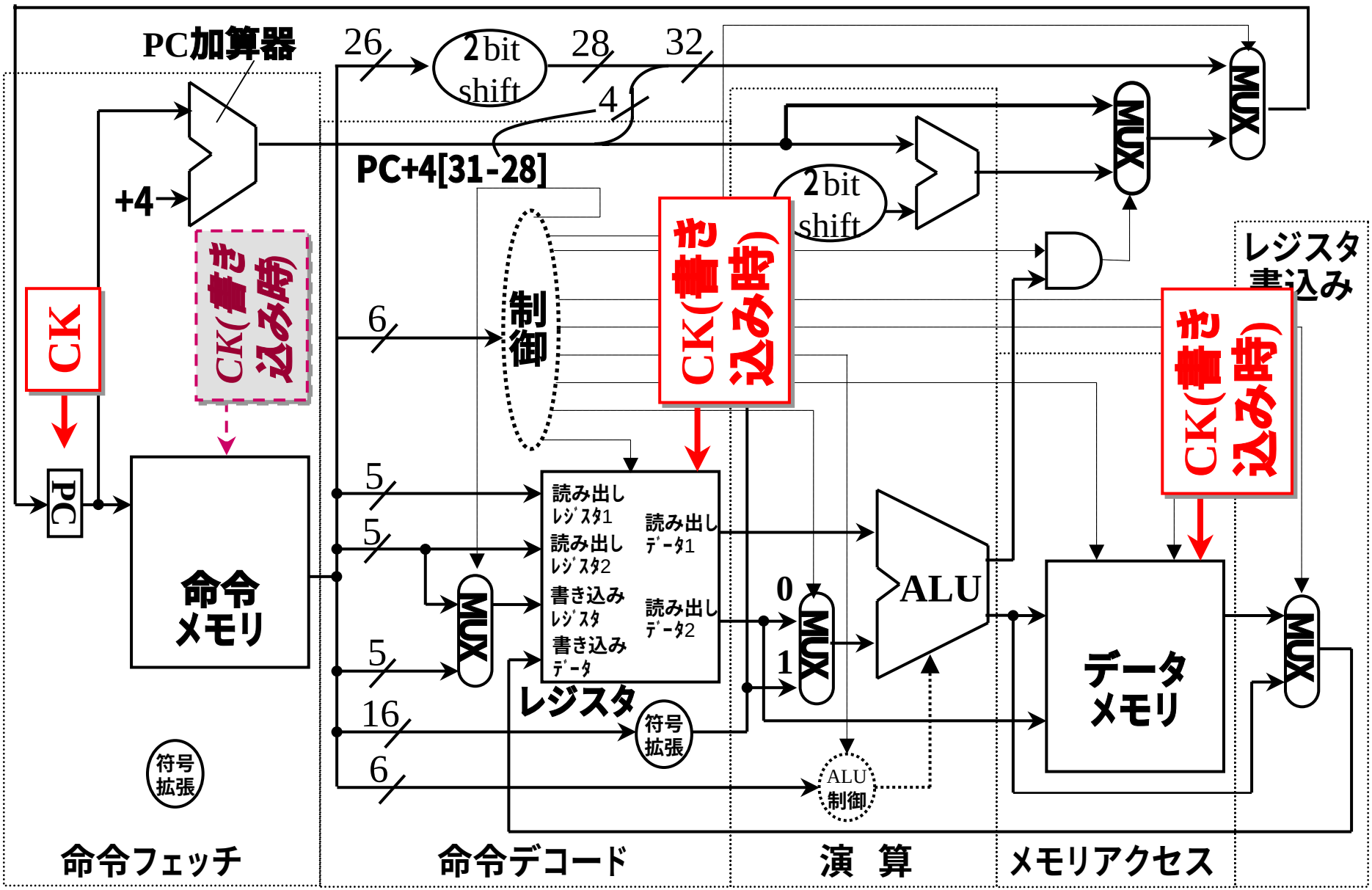
復習：プロセッサの処理イメージ2



復習：実行ステップと各命令毎の部品使用順序



復習：単一サイクル・データパス



クイズ

問2．本日学んだ単一サイクルデータパスにおいて、各命令の実行時に使用される部品に○をつけよ。
なお例として、add命令には既に○がつけてある。

	PC	PC加算器	命令ﾒﾓﾘ	ﾚｼﾞｽﾀ	ALU	分岐用加算器	ﾃﾞｰﾀﾒﾓﾘ
add	○	○	○	○	○		
sub	○	○	○	○	○		
lw	○	○	○	○	○		○
sw	○	○	○	○	○		○
beq	○	○	○	○	○	○	
j	○		○				

問3．本日学んだ単一サイクルデータパスにおいて、各部品の実行時間が下表で与えられる場合、

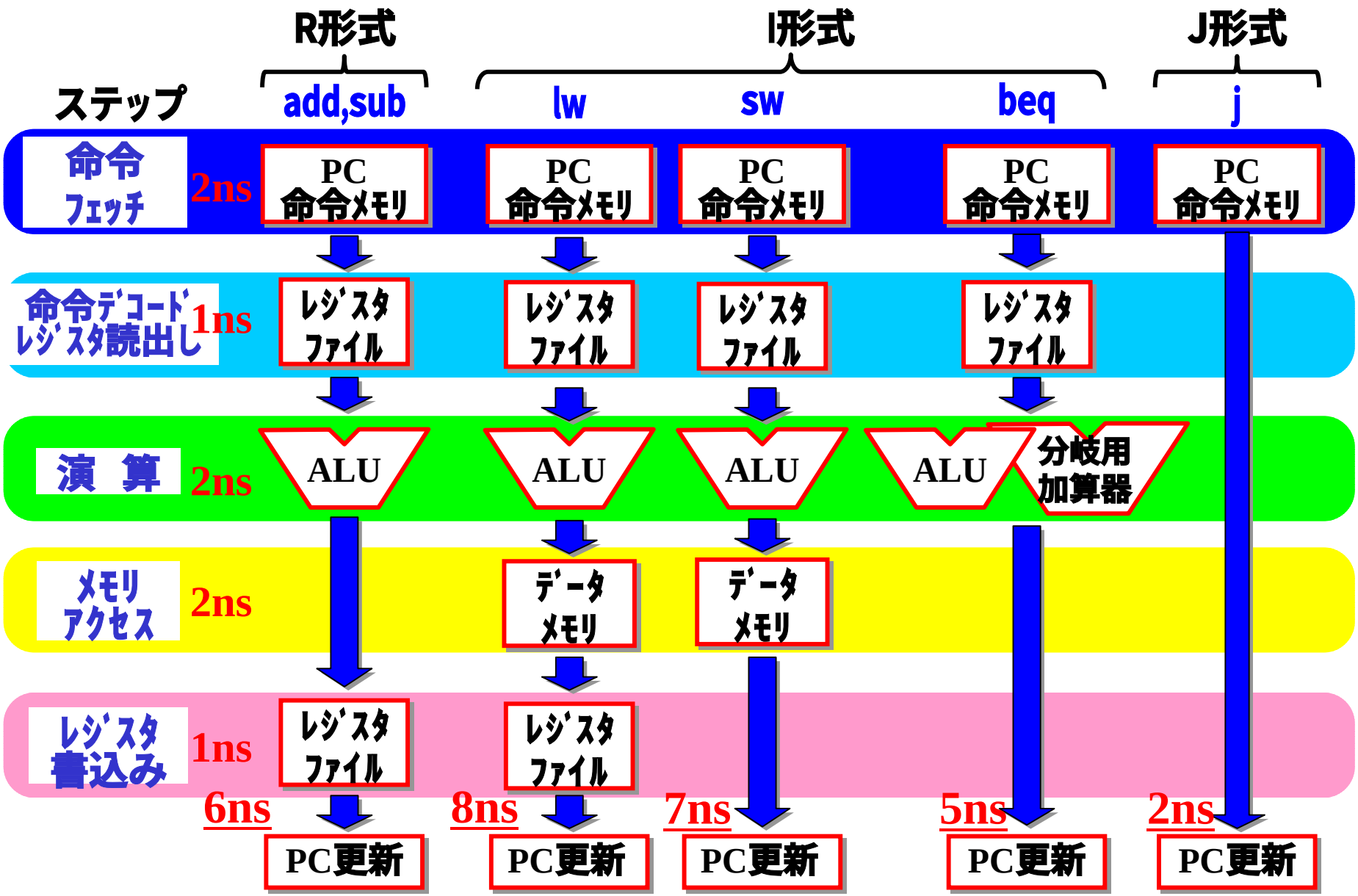
6種の命令(add,sub,lw,sw,beq,j)のうちサイクル時間を決める命令と、その命令のサイクル時間を答えよ。ただし下記以外の部品の実行時間は無視できるものとする。

部品	命令 ﾒﾓﾘ	ﾚｼﾞｽﾀ ﾌｧｲﾙ 読み出し	ALU	ﾒﾓﾘ ｱｸｾｽ	ﾚｼﾞｽﾀ ﾌｧｲﾙ 書込み	分岐用 加算器	PC 加算器
実行時間	2ns	1ns	2ns	2ns	1ns	2ns	2ns

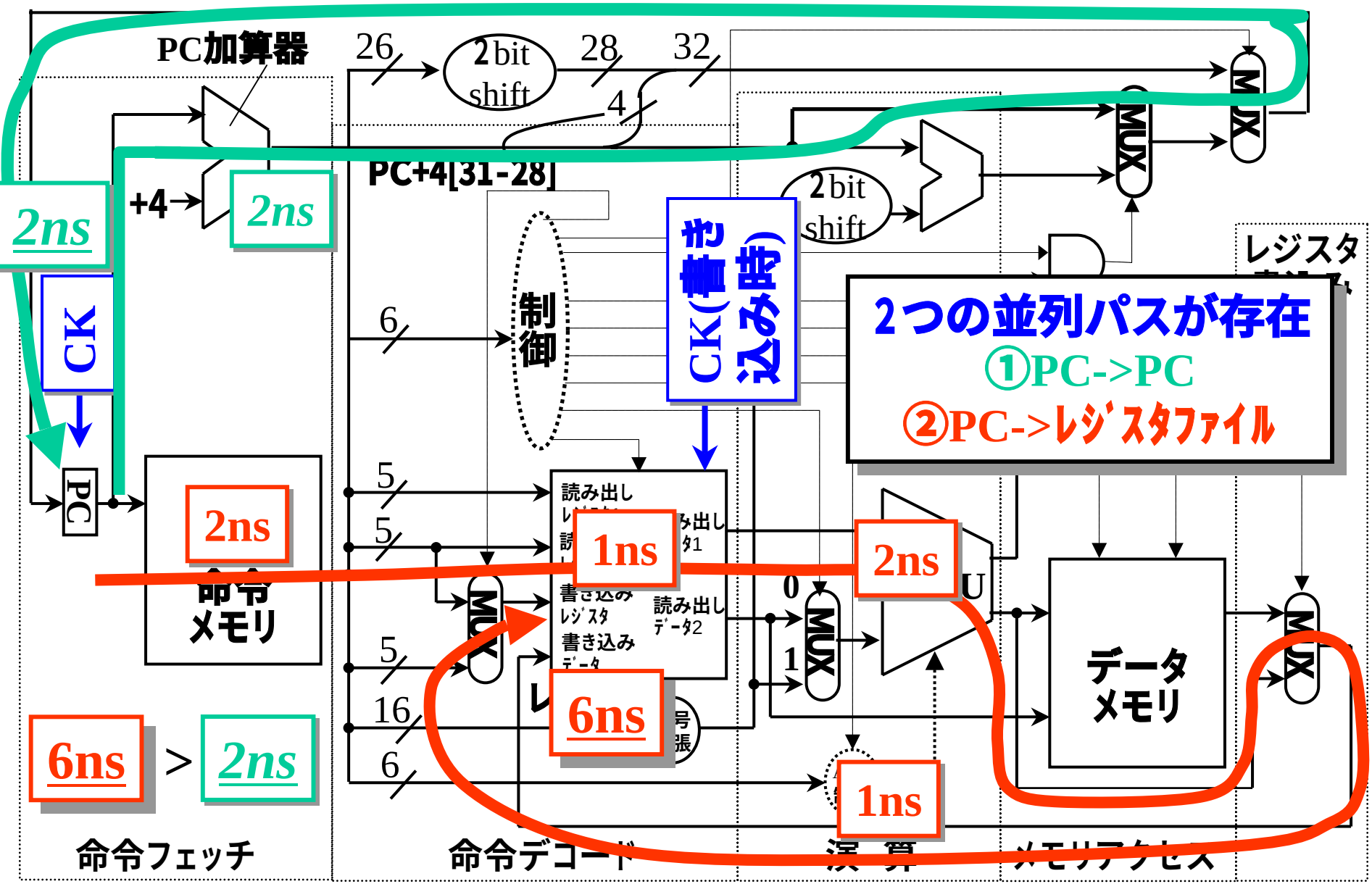
命令：lw

サイクル時間：8ns

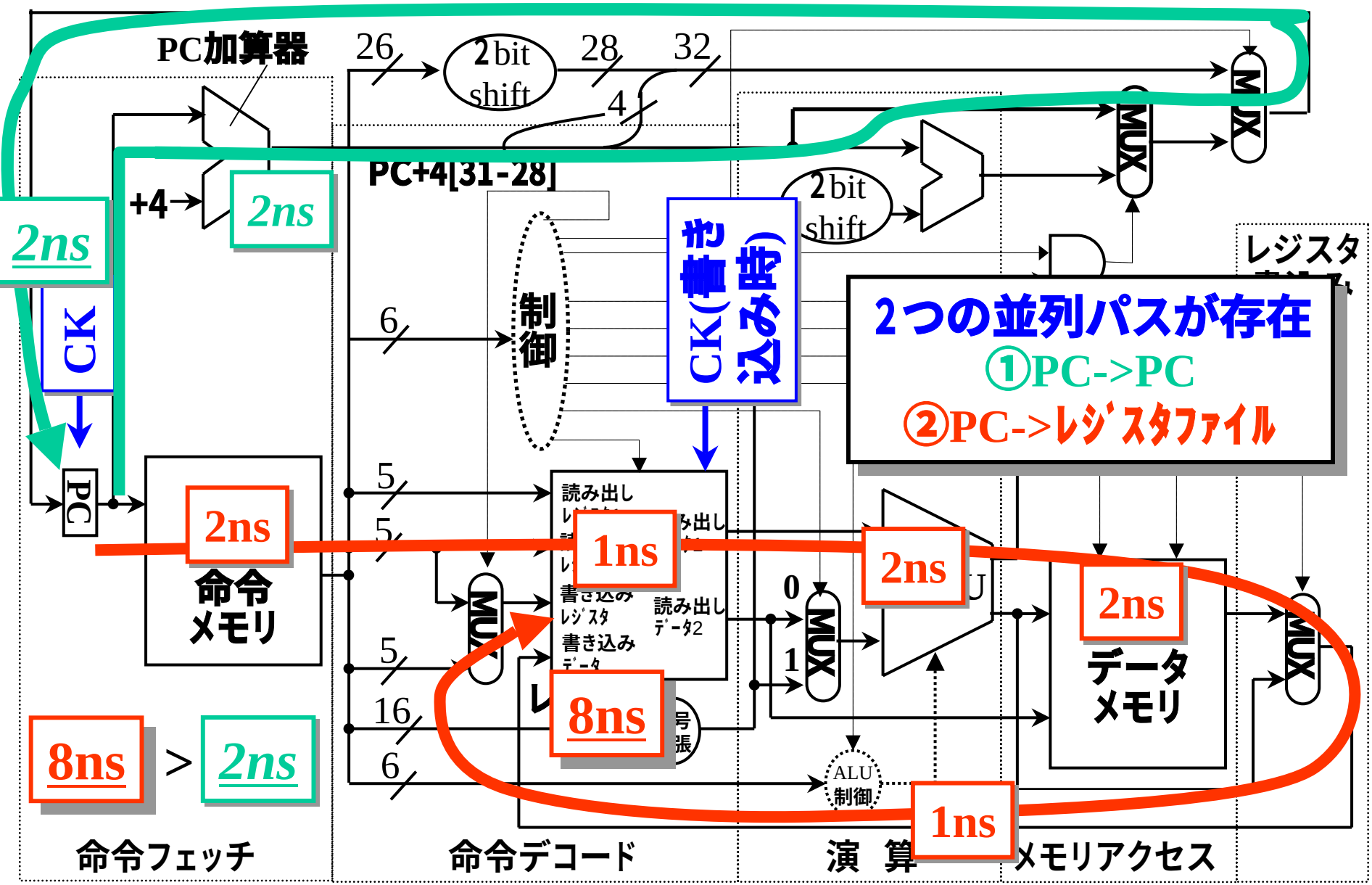
【補足】 何故“PC更新”時間は足さないの？



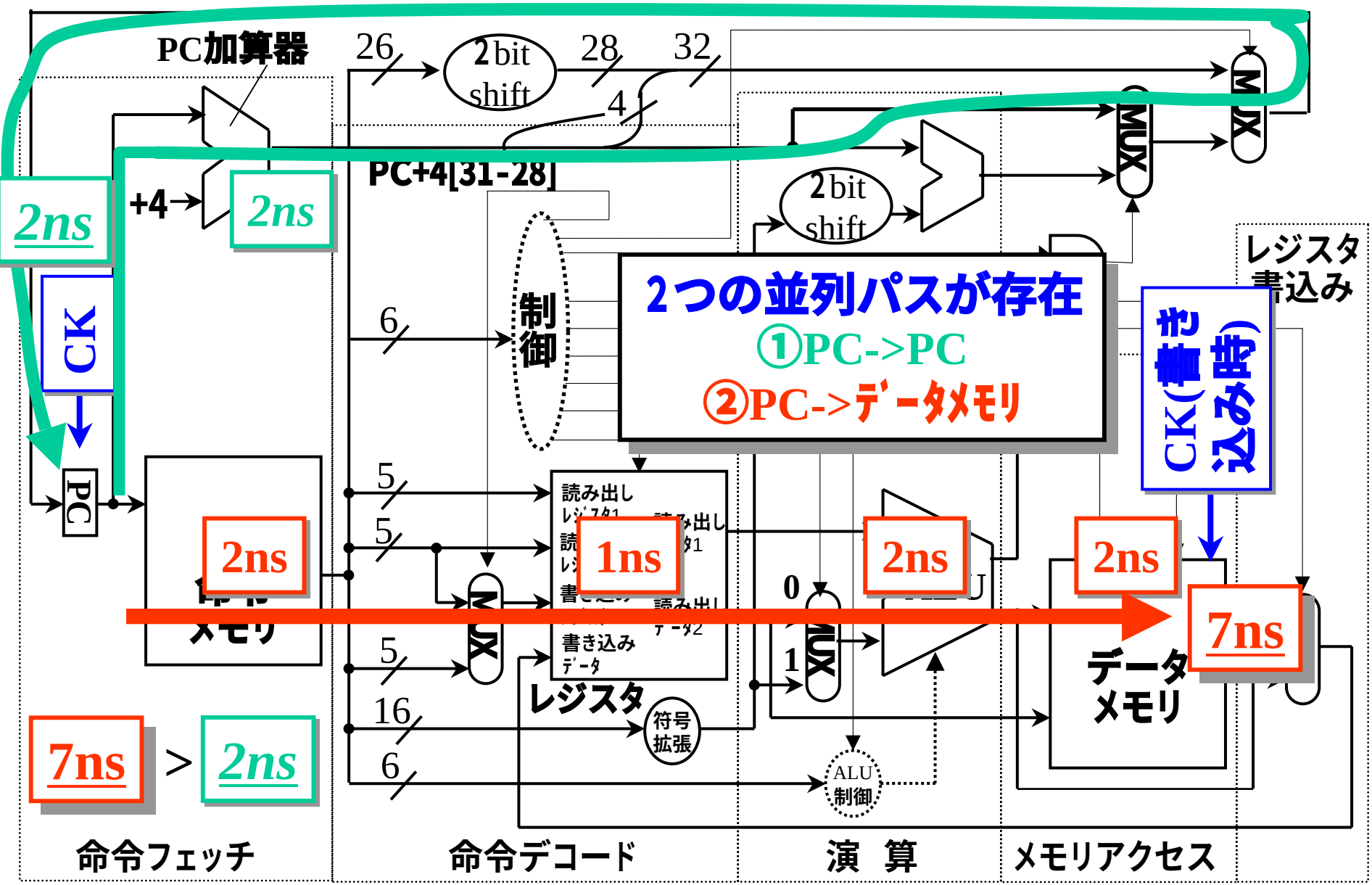
単一サイクル・データパス(add/sub命令)



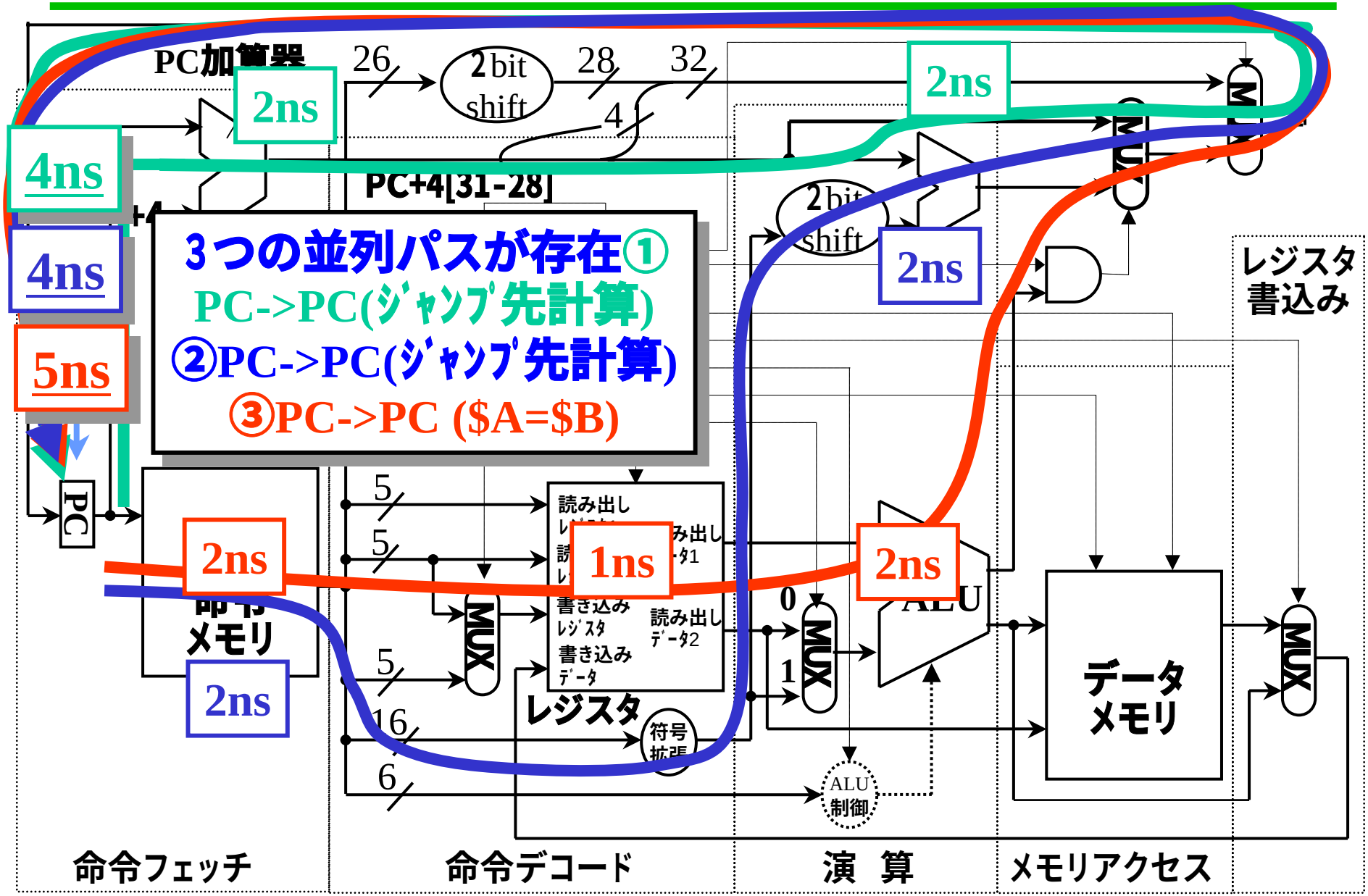
単一サイクル・データパス(lw命令)



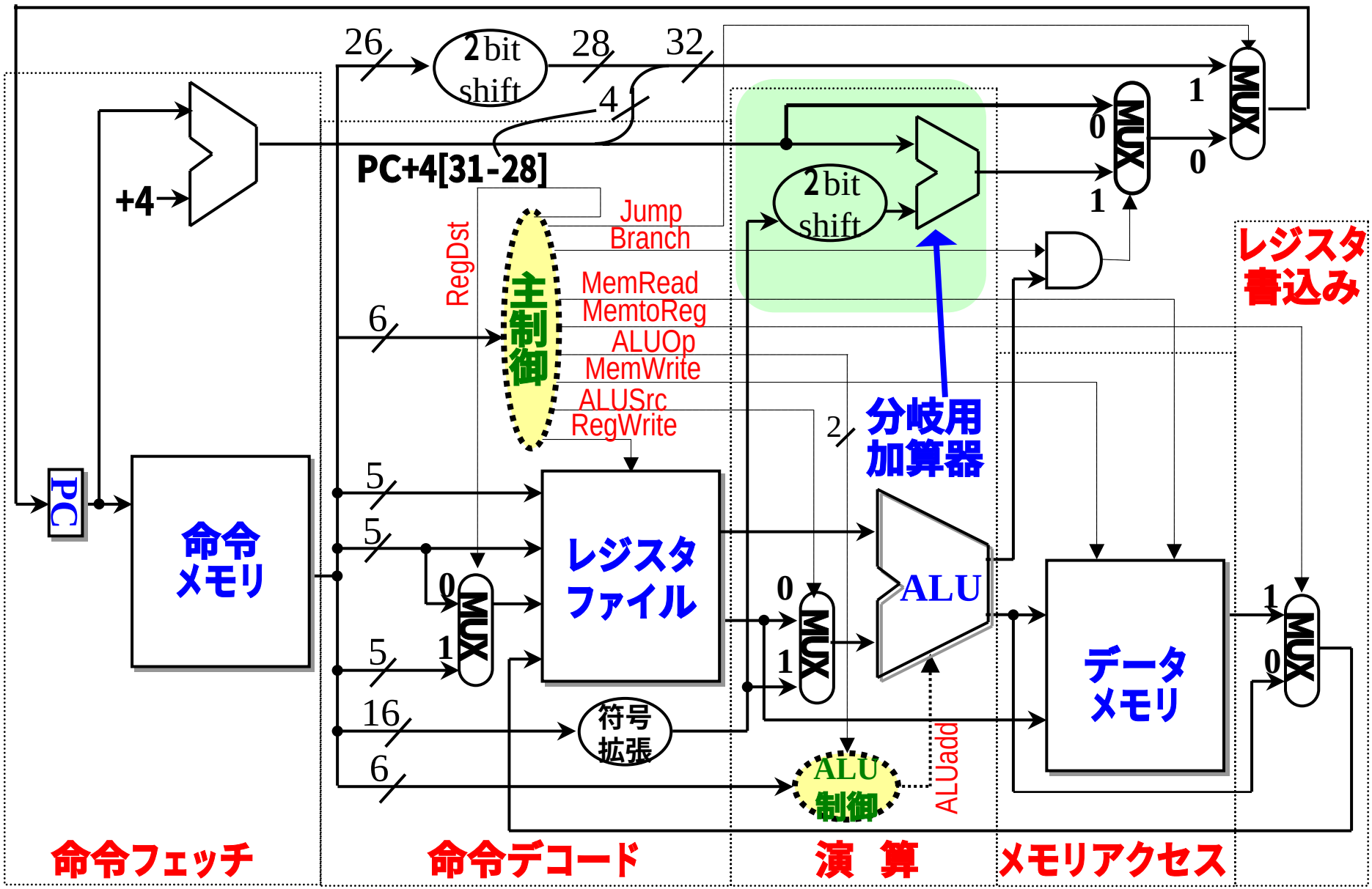
単一サイクル・データパス(st命令)



単一サイクル・データパス(beq命令)



制御論理の設計


26
2bit shift
28
32
4
PC+4[31-28]
+4
2bit shift
MUX
MUX
MUX
RegDst
主制御
Jump
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
6
2
分岐用加算器
レジスタ書込み
PC
命令メモリ
5
5
5
16
6
レジスタファイル
符号拡張
ALU
ALU制御
ALUadd
データメモリ
命令フェッチ
命令デコード
演　算
メモリアクセス

命令一覧

		31 26	25 21	20 16	15 11	10 6	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	$転送先	転送先相対ｱﾄﾞﾚｽ			ﾒﾓﾘ［$index+転送元相対ｱﾄﾞﾚｽ］＝$転送先
	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）

opｺｰﾄﾞの6bit=>主制御へ入力

functｺｰﾄﾞの6bit=>ALU制御に入力

Opコードとfunctフィールドの2進表現

命令	op	funct
add	000000	100000
sub	000000	100010
lw	100011	
sw	101011	-
beq	000100	-
j	000010	-

論理関数の記述方法

(1) 真理値表

入力			出力		
A	B	C	D	E	F
0	0	0	0	0	0
0	0	1	1	0	0
0	1	0	1	0	0
0	1	1	1	1	0
1	0	0	1	0	0
1	0	1	1	1	0
1	1	0	1	1	0
1	1	1	1	0	1

$\overline{A}\cdot B\cdot C$

$A\cdot\overline{B}\cdot C$

$A\cdot B\cdot\overline{C}$

真理値表=>ブール式への変換

①出力が'1'の各行に対し、
②各入力の論理積を求め、
③それら論理積の論理和を取る。

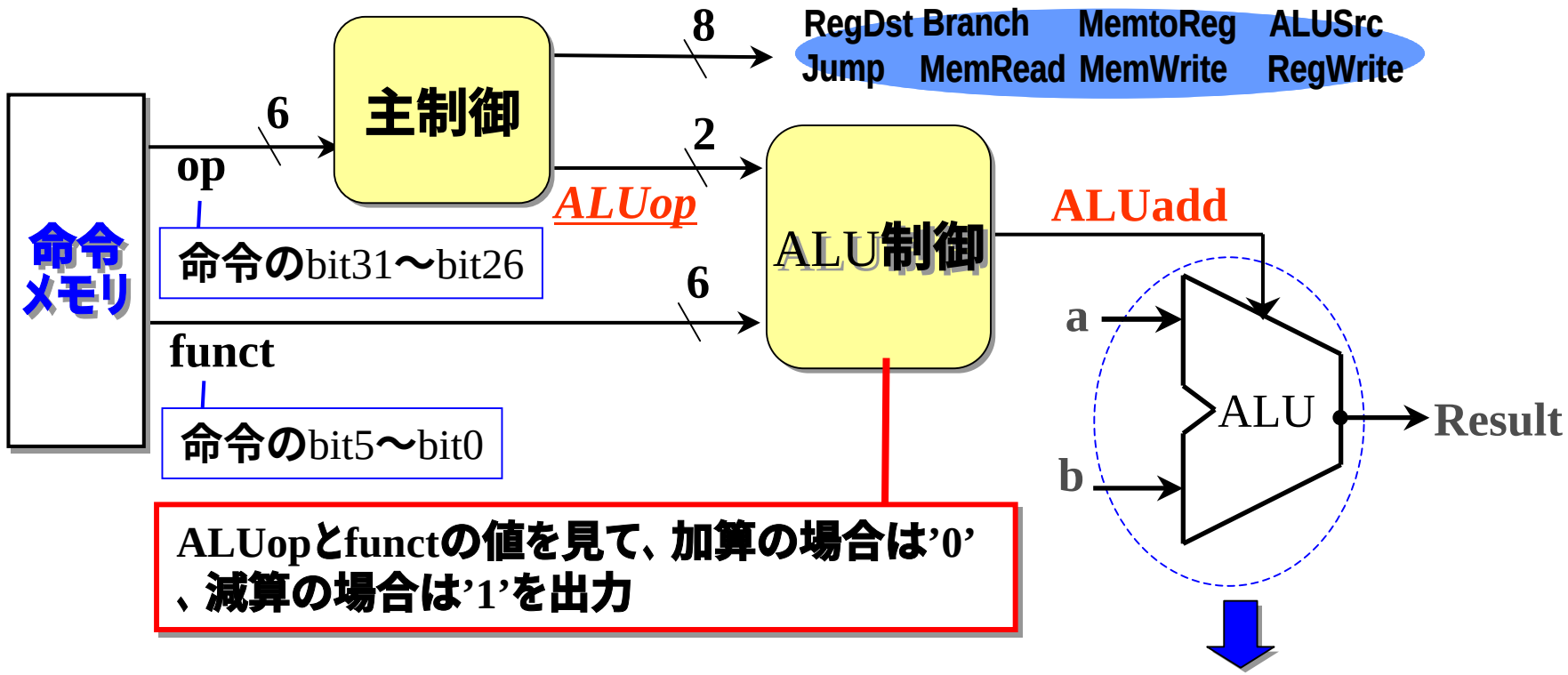
$$E=(\overline{A}\cdot B\cdot C)+(A\cdot\overline{B}\cdot C)+(A\cdot B\cdot\overline{C})$$

(2) ブール代数

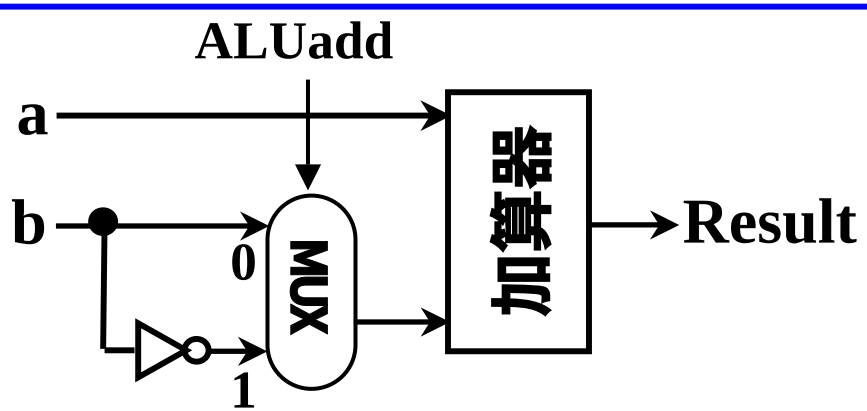
$$D=A+B+C$$
$$E=(\overline{A}\cdot B\cdot C)+(A\cdot\overline{B}\cdot C)+(A\cdot B\cdot\overline{C})$$
$$F=A\cdot B\cdot C$$

+：論理和
・：論理積
ー：否定

制御論理の接続関係



ALUadd	Result	機能
0	a+b	加算
1	a-b	減算



ALU制御論理の設計~ALUOpの定義

命令	実行する演算	入力								出力
		funct						ALUOp		
		f5	f4	f3	f2	f1	f0	ALUOp1	ALUOp0	ALUAdd
lw	加算	X	X	X	X	X	X	0	0	0
sw	加算	X	X	X	X	X	X	0	0	0
beq	減算	X	X	X	X	X	X	0	1	1
add	加算	1	0	0	0	0	0	1	0	0
sub	減算	1	0	0	0	1	0	1	0	1

X:任意値
(don't care)

addとsubの区別が可能

このままでは
加算(lw、sw)か
減算(beq)か
決められない

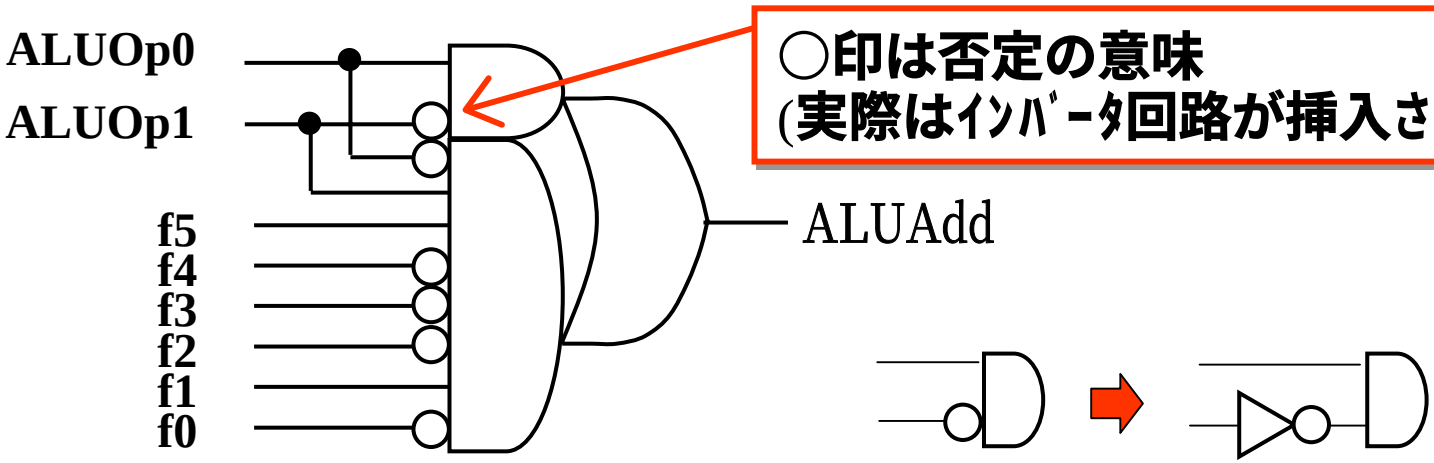
lw,sw/beq/add,subの3つの区別ため新たに定義

ALU制御ユニットの設計 =太枠の 真理値表の動作を行う論理回路の実現

ＡＬＵ制御論理の設計~論理ｹﾞｰﾄへのﾏｯﾋﾟﾝｸﾞ

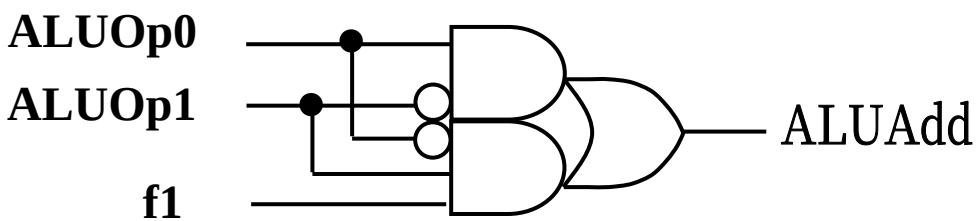
◆解1

$$ALUSrc = ALUOp0 \cdot \overline{ALUOp1} + \overline{ALUOp0} \cdot ALUOp1 \cdot f5 \cdot \overline{f4} \cdot \overline{f3} \cdot \overline{f2} \cdot f1 \cdot \overline{f0}$$



◆解2

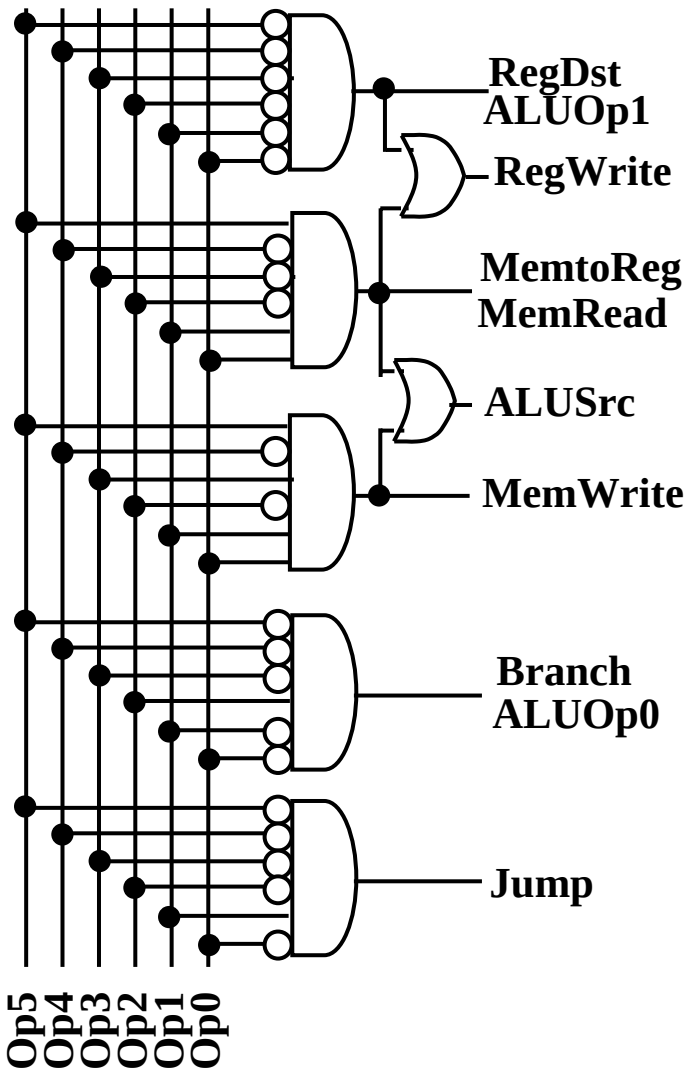
$$ALUSrc = ALUOp0 \cdot \overline{ALUOp1} + \overline{ALUOp0} \cdot ALUOp1 \cdot f1$$



=> add/subの区別だけなら、f1の値だけで可能なことに着目

主制御論理の設計

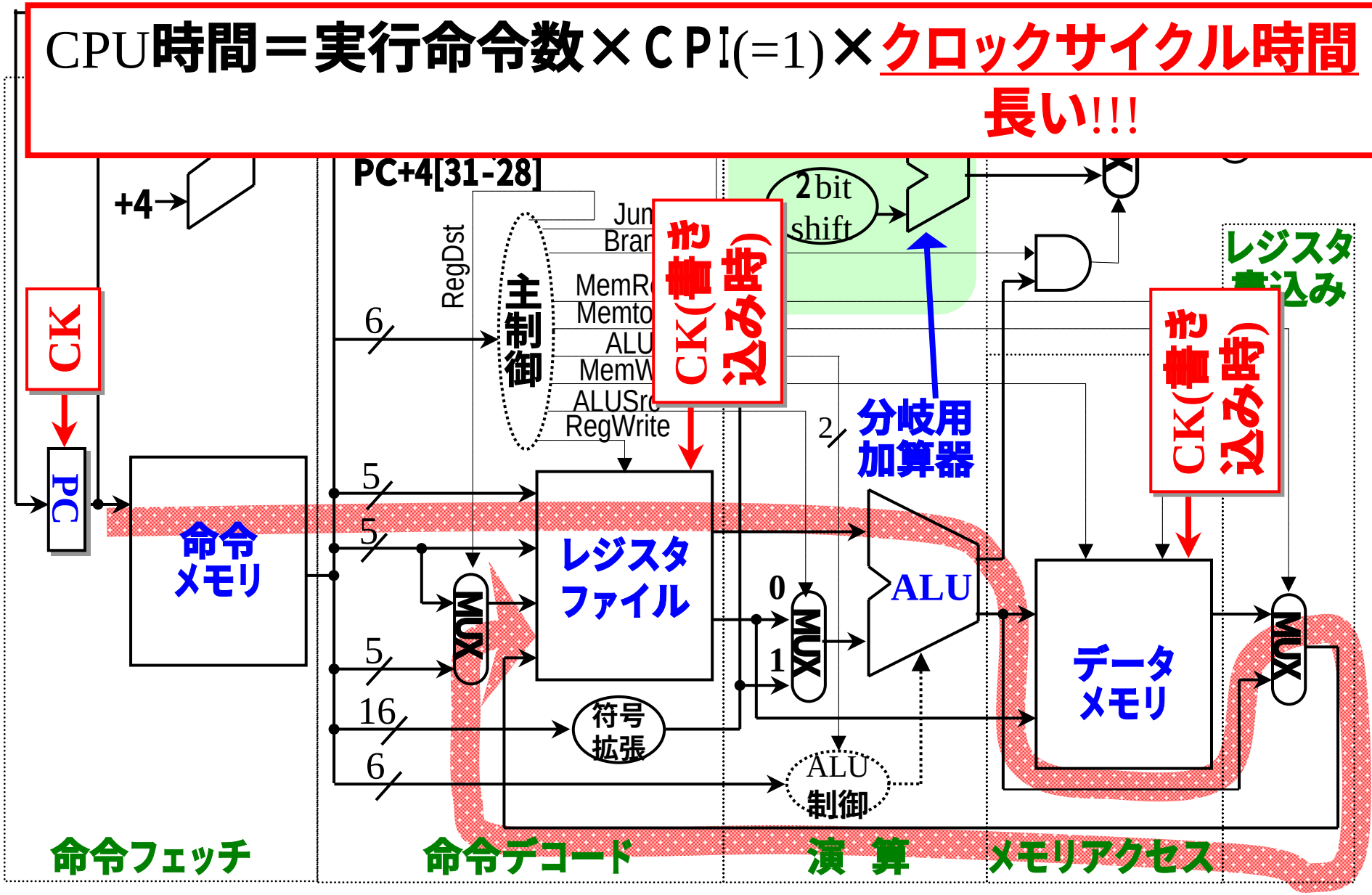
制御	信号名	add,sub	lw	sw	beq	j
入力	Op5	0	1	1	0	0
	Op4	0	0	0	0	0
	Op3	0	0	1	0	0
	Op2	0	0	0	1	0
	Op1	0	1	1	0	1
	Op0	0	1	1	0	0
出力	RegDst	1	0	X	X	X
	ALUSrc	0	1	1	0	X
	MemtoReg	0	1	X	X	X
	RegWrite	1	1	0	0	0
	MemRead	0	1	0	0	X
	MemWrite	0	0	1	0	0
	Branch	0	0	0	1	X
	Jump	0	0	0	0	1
	ALUOp1	1	0	0	0	X
	ALUOp0	0	0	0	1	X



X:**任意値**(don’t care)

単一サイクル方式の欠点

CPU時間＝実行命令数×ＣＰＩ(=1)×クロックサイクル時間長い!!!



単一サイクル方式の問題点と他の方式

単一サイクル方式

1命令当りクロック・サイクル数＝CPI＝1 ← 全命令で等しい

1クロック・サイクル時間＝MAX（使用機能ユニット実行時間合計）

命　令	Step1：命令フェッチ	Step2：命令デコード	Step3：演　算	Step4：メモリアクセス	Step5：レジスタ書込み	命令実行時間	クロックサイクル時間
R形式命令	2ns	1ns	2ns		1ns	6ns	8ns
load word	2ns	1ns	2ns	2ns	1ns	8ns	
store word	2ns	1ns	2ns	2ns		7ns	
分岐命令	2ns	1ns	2ns			5ns	
ジャンプ命令	2ns					2ns	

可変長クロックサイクル方式

命令種ごとに1クロック・サイクルの長さを変えて実行する方式

CPI=1　（ハードウェアでの実現は難しい。）

マルチサイクル方式

1クロック・サイクルを短くし、各命令毎にCPIを変える方式

＜例えば、上表の場合であれば、1クロック・サイクル時間＝2ns とする＞

各方式の性能比較

CPU時間＝実行命令数×ＣＰＩ×クロックサイクル時間

命　令	命令出現回数	単一サイクル方式			可変長ｸﾛｯｸｻｲｸﾙ			マルチサイクル方式		
		ＣＰＩ	ｸﾛｯｸｻｲｸﾙ	命令実行時間	ＣＰＩ	ｸﾛｯｸｻｲｸﾙ	命令実行時間	ＣＰＩ	ｸﾛｯｸｻｲｸﾙ	命令実行時間
R形式命令	50	1	8ns	400ns	1	6ns	300ns	4	2ns	400ns
load word	20	1	8ns	160ns	1	8ns	160ns	5	2ns	200ns
store word	10	1	8ns	80ns	1	7ns	70ns	4	2ns	80ns
分岐命令	10	1	8ns	80ns	1	5ns	50ns	3	2ns	60ns
ジャンプ命令	10	1	8ns	80ns	1	2ns	20ns	1	2ns	20ns
合計	100			800ns			600ns			760ns

最近のコンピュータでは上記方式は採用されていない!!!

⇩

パイプライン方式（来週説明）

マルチサイクルの実現

マルチサイクル方式

① 1つの命令中に1つの機能ユニットを2回以上使用可能

➡ 機能ユニット共有化によるハードウェア量削減が可能

② 主要な機能ユニットの後ろにレジスタを追加

➡ 該当ユニットの値を後続のクロックサイクルで必要になるまで保持

マルチサイクル･データパス