

コンピュータシステム・アーキテクチャ特論

コンピュータアーキテクチャ I・II の復習
～プロセッサ動作原理と命令パイプライン～

井上こうじ

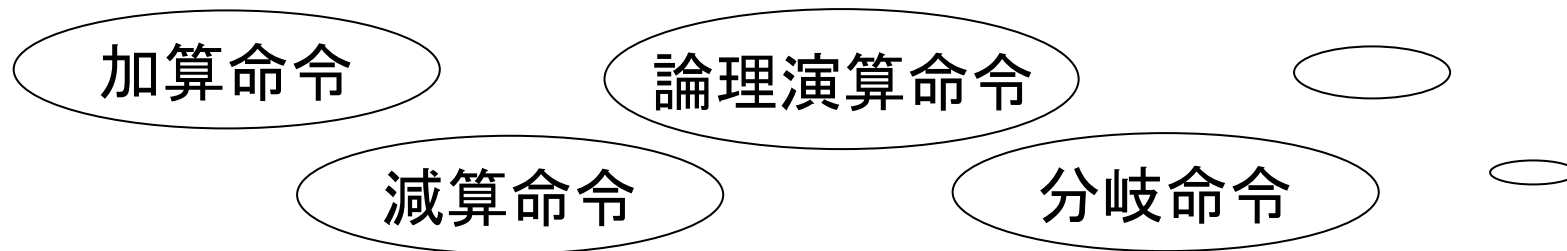
コンピュータアーキテクチャ I

「パターソン&ヘネシー コンピュータの構成と設計」第2.1～2.3節

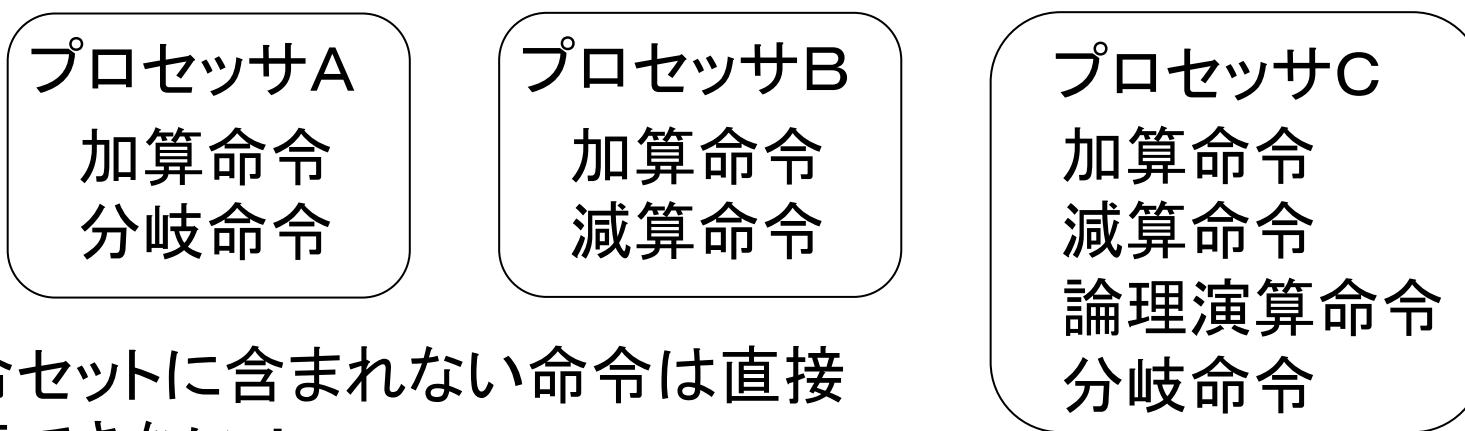
命令と命令表現

プロセッサの命令と命令セット

- **命令**: プロセッサへの指示 (プロセッサが実行可能な処理)



- **命令セット**: プロセッサが実行可能な命令の集合 (プログラマから見えるプロセッサの論理仕様)



命令セットに含まれない命令は直接実行できない！

プログラム(命令シーケンス)の実行

高水準プログラミング言語
(high-level programming language)

```
if (y == 1)
    c = a + b;
else
    c = a - b;
```

ソースプログラム

命令セット

コンパイラ
(compiler)

「命令シーケンス」として
プログラムを表現

アセンブリプログラム

Target:
lw \$4, 0(\$1)
lw \$5, 4(\$1)
add \$2, \$4, \$5

アセンブリ言語

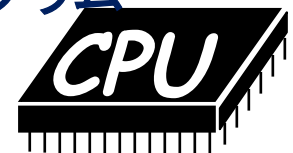
機械語

(CPUへの命令)

```
100101001010100
000001011011100
111001111010011
```

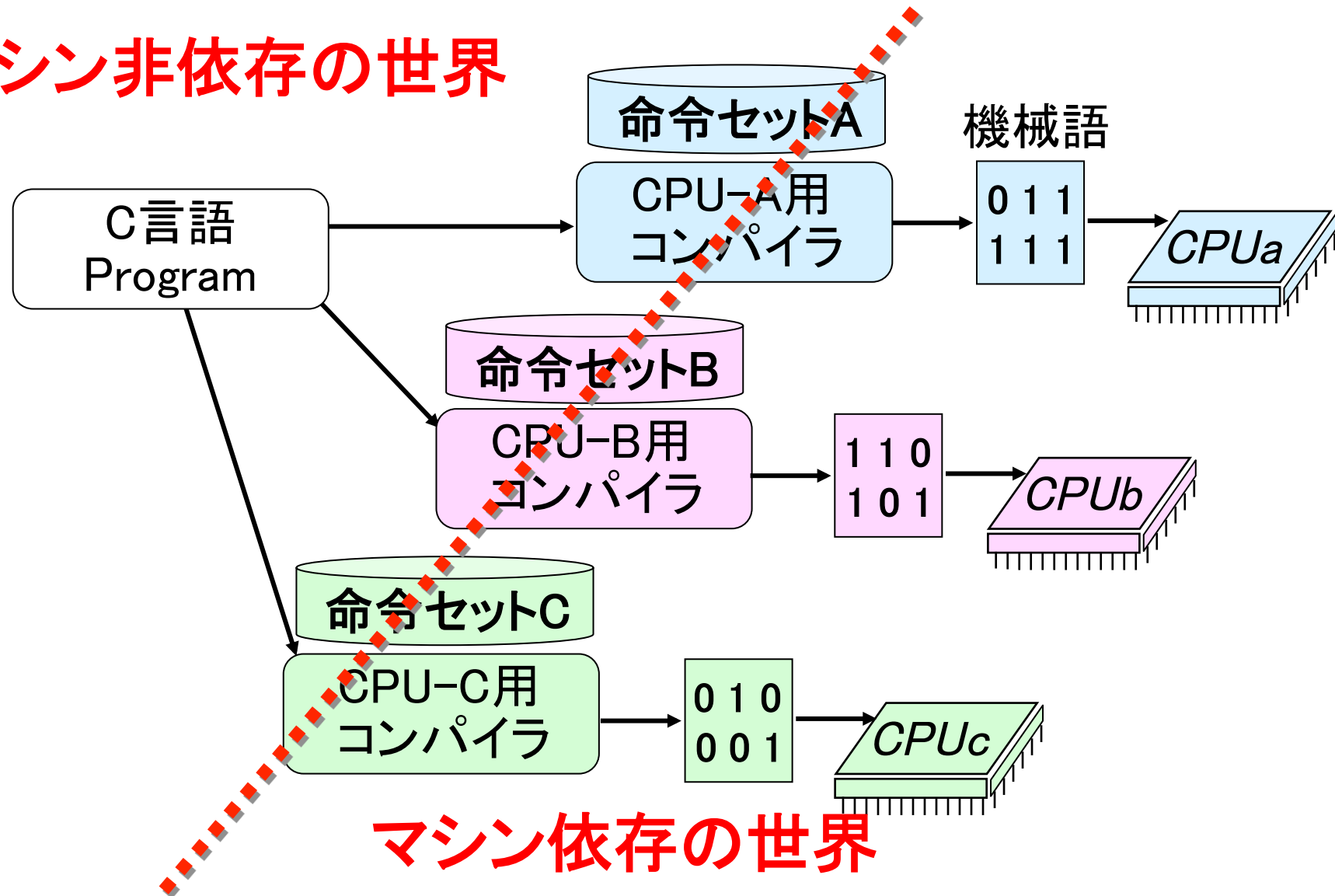
2進表現

オブジェクトプログラム



プログラムとCPUのインタフェース

マシン非依存の世界



MIPSとその命令セット

- この授業では MIPS の命令セットを例にする.
 - 基本の考え方はどのプロセッサでもあまり変わらない.
 - MIPS は PlayStation で使用されているプロセッサ.

MIPSの命令セット(R2000/3000) :

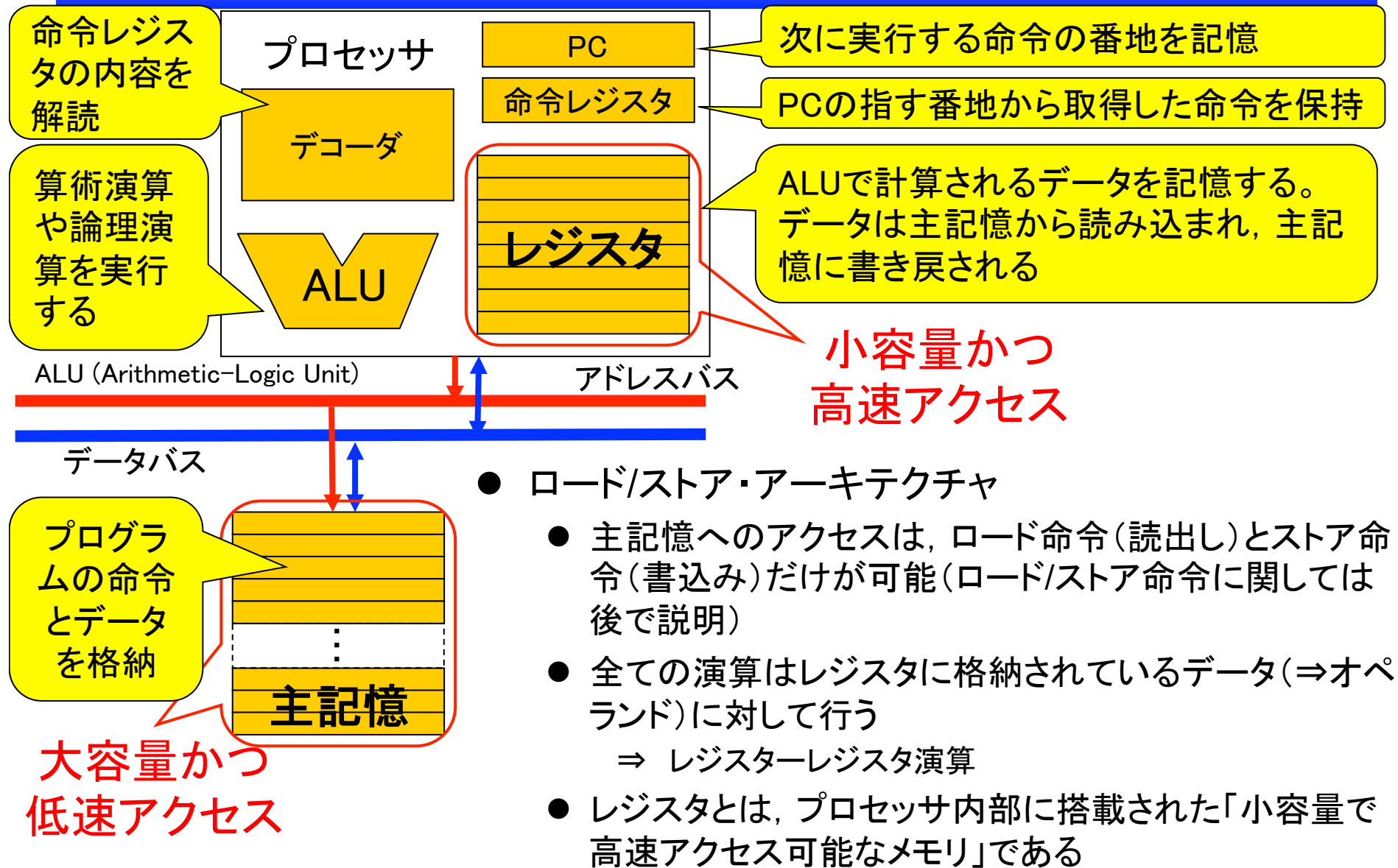
転送命令	LB, BLU, LH, LHU, LUI, LW, LWL, LWR, SB, SH, SW, SWL, SWR
算術演算命令	ADD, ADDI, ADDIU, ADDU, DIV, DIVU, MULT, MULTU, SLT, SLTI, SLTIU, SLTU, SUB, SUBU
論理演算命令	AND, ANDI, NOR, OR, ORI, SLL, SLLV, SRA, SRAV, SRL, SRLV, XOR, XORI
分岐命令	BEQ, BGEZ, BGEZAL, BGTZ, BLEZ, BLTZ, BLTZAL, BNE, J, JAL, JALR, JR
その他の命令	BCzF, BCzT, BREAK, CFCz, COPz, CTCz, LWCz, MFC0, MFCz, MFHI, MFLO, MTC0, MTCz, MTHI, MTL0, RFE, SWCz, SYSCALL, TLBP, TLBR, TLBWI, TLBWR

MIPSの命令(一部)

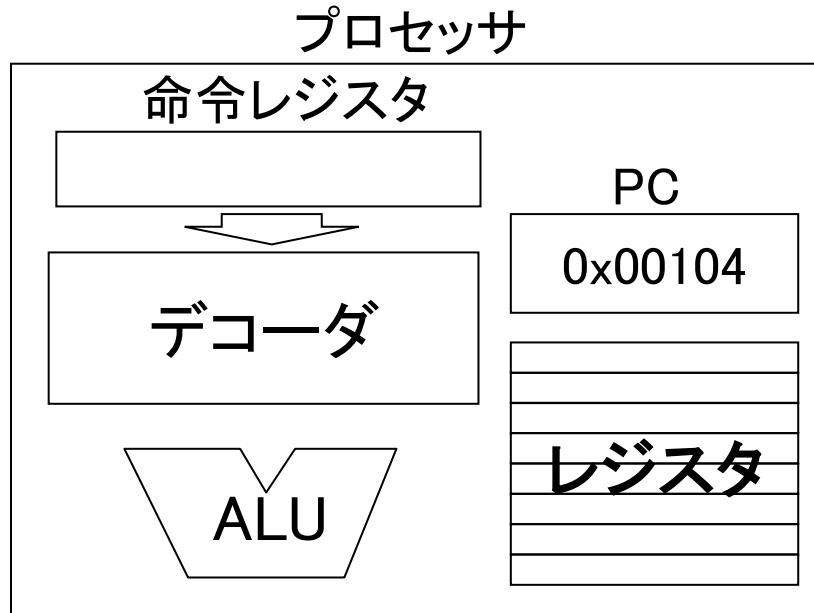
命令区分	命令	例	意味
算術演算	add	add \$s1, \$s2, \$s3	$\$s1 = \$s2 + \$s3$
	subtract	sub \$s1, \$s2, \$s3	$\$s1 = \$s2 - \$s3$
データ転送	load word	lw \$s1, 100(\$s2)	\$s1に, メモリの[\$s2+100]番地のワードデータ(4B)を読み込み
	store word	sw \$s1, 100(\$s2)	メモリの[\$s2+100]番地に, \$s1のワードデータ(4B)を書込み
条件分岐	branch on equal	beq \$s1, \$s2, L	もし、 $\$s1 == \$s2$ ならLへ分岐
	branch on not equal	bne \$s1, \$s2, L	もし、 $\$s1 \neq \$s2$ ならLへ分岐
	set on less than	slt \$s1, \$s2, \$s3	もし、 $\$s2 < \$s3$ なら $\$s1 = 1$, 以外なら $\$s1 = 0$
無条件ジャンプ	jump	j L	Lにジャンプ
	jump register	jr \$s1	\$s1の値が示すアドレスにジャンプ

\$s1～\$s3は汎用レジスタ

プロセッサでの命令実行(1)



命令の実行手順



アドレス	主記憶
	...
0x00104	add \$t0, \$s1, \$s2
0x00108	sub \$t1, \$t0, \$s3
0x0010c	lw \$t1, 320(\$s4)
	...

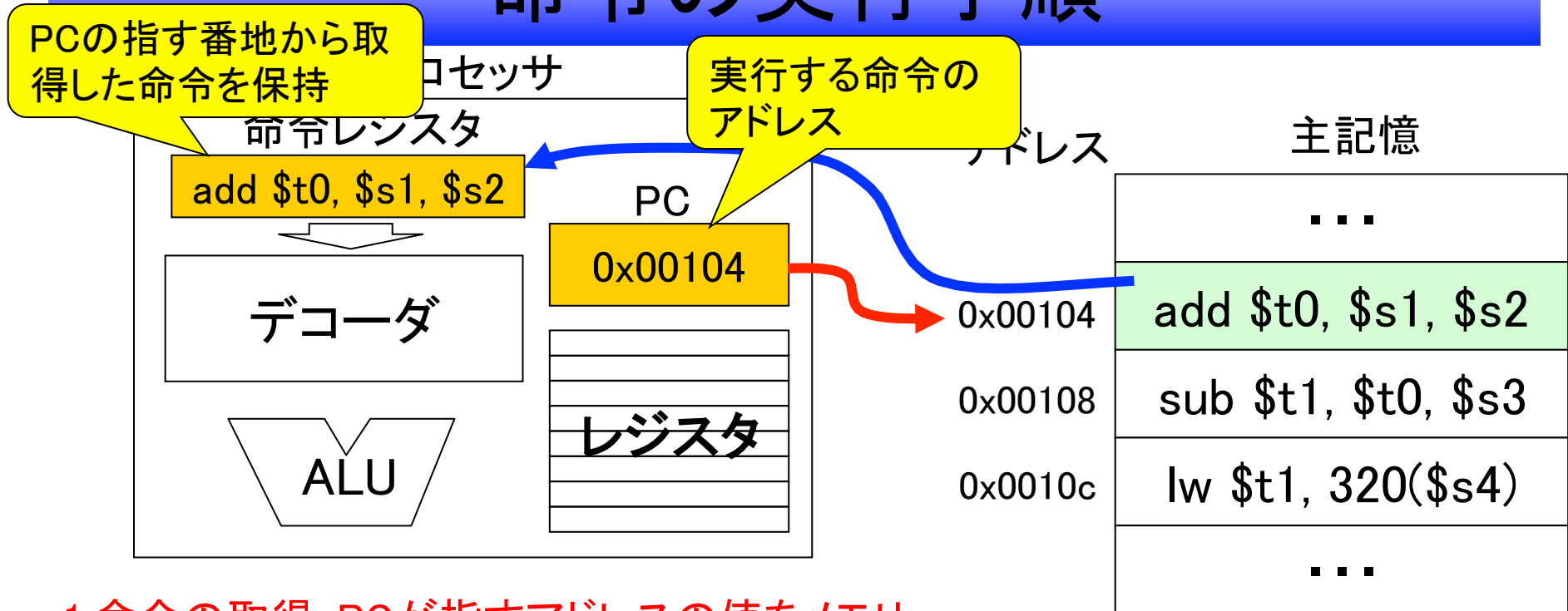
1. 命令の取得: PCが指すアドレスの値をメモリから命令レジスタへ取得. 同時に, 次命令の取得に備えてPCの値を更新(例えば+4)

2. 命令の解読: 命令レジスタの値をデコード

3. 命令の実行: 解読結果に従って実行

実際には, 各命令は2進表現でメモリに格納されている

命令の実行手順



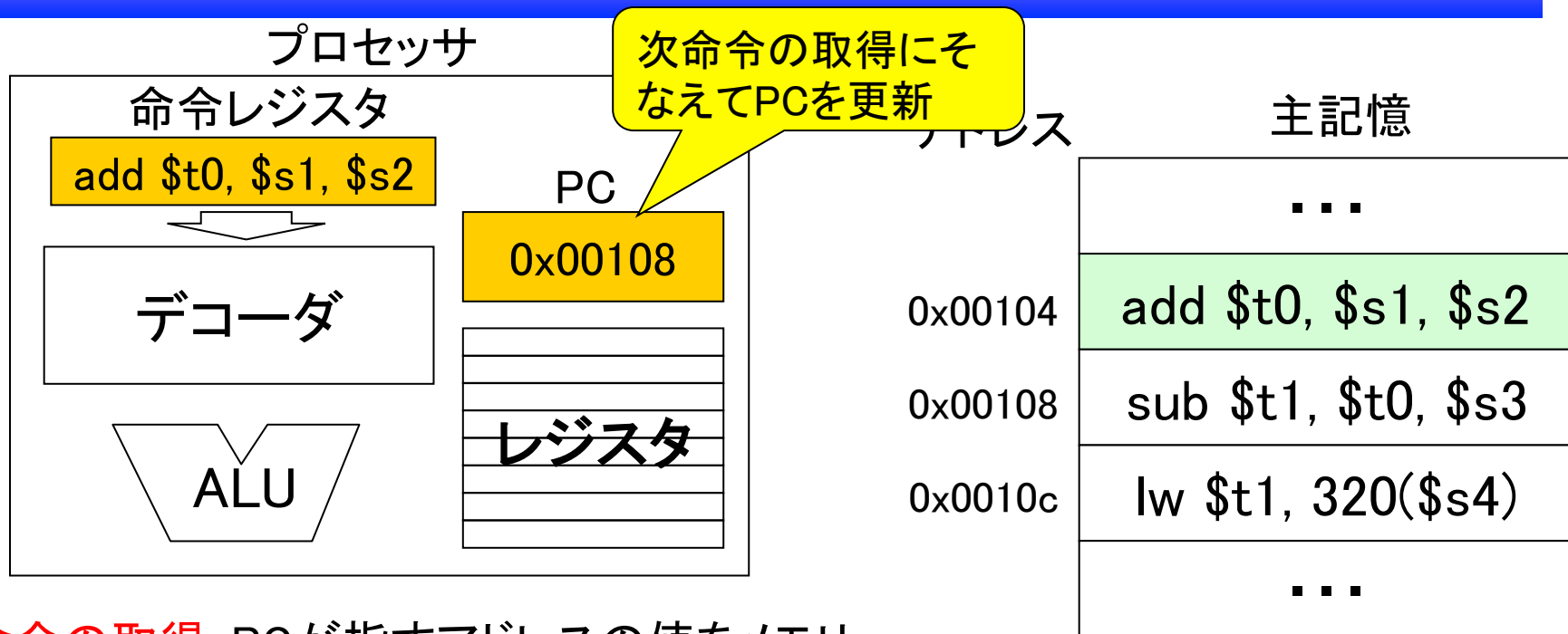
1. 命令の取得: PCが指すアドレスの値をメモリから命令レジスタへ取得. 同時に, 次命令の取得に備えてPCの値を更新(例えば+4)

2. 命令の解読: 命令レジスタの値をデコード

3. 命令の実行: 解読結果に従って実行

実際には, 各命令は2進表現でメモリに格納されている

命令の実行手順



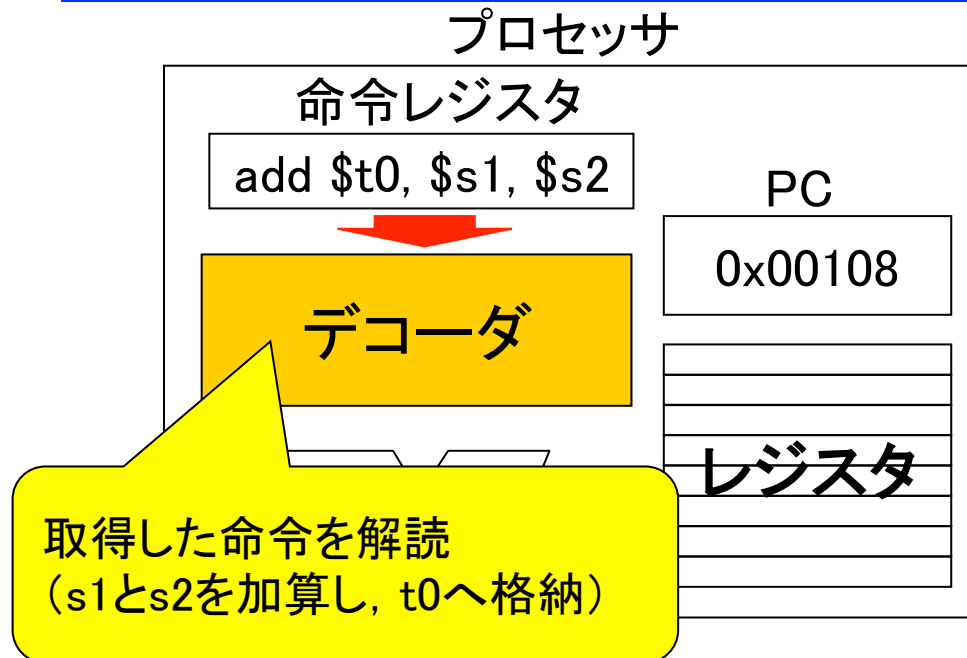
1. 命令の取得: PCが指すアドレスの値をメモリから命令レジスタへ取得. **同時に、次命令の取得に備えてPCの値を更新(例えば+4)**

2. 命令の解読: 命令レジスタの値をデコード

3. 命令の実行: 解読結果に従って実行

実際には、各命令は2進表現でメモリに格納されている

命令の実行手順



1. 命令の取得: PCが指すアドレスの値をメモリから命令レジスタへ取得. 同時に, 次命令の取得に備えてPCの値を更新(例えば+4)

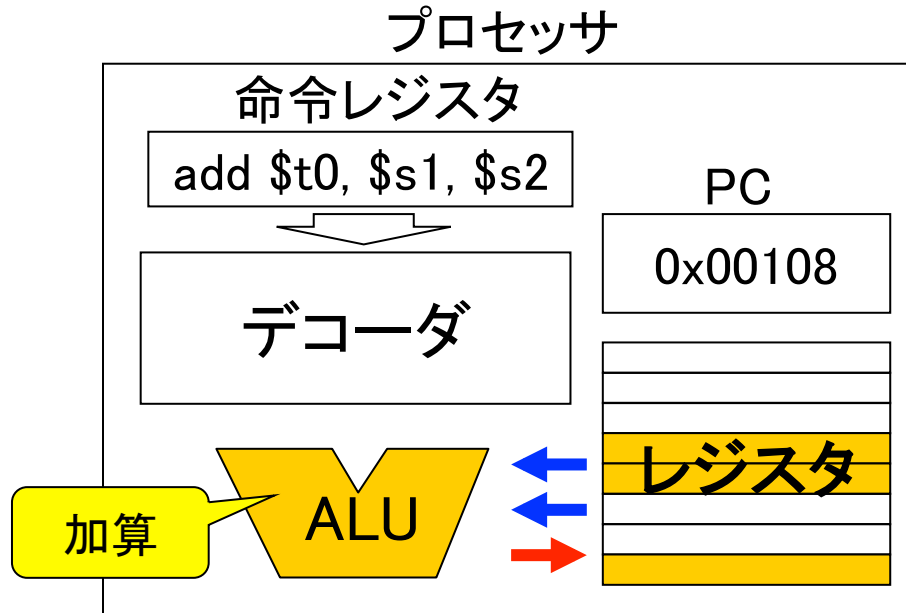
2. 命令の解読: 命令レジスタの値をデコード

3. 命令の実行: 解読結果に従って実行

アドレス	主記憶
	...
0x00104	add \$t0, \$s1, \$s2
0x00108	sub \$t1, \$t0, \$s3
0x0010c	lw \$t1, 320(\$s4)
	...

実際には, 各命令は2進表現でメモリに格納されている

命令の実行手順



アドレス	主記憶
	...
0x00104	add \$t0, \$s1, \$s2
0x00108	sub \$t1, \$t0, \$s3
0x0010c	lw \$t1, 320(\$s4)
	...

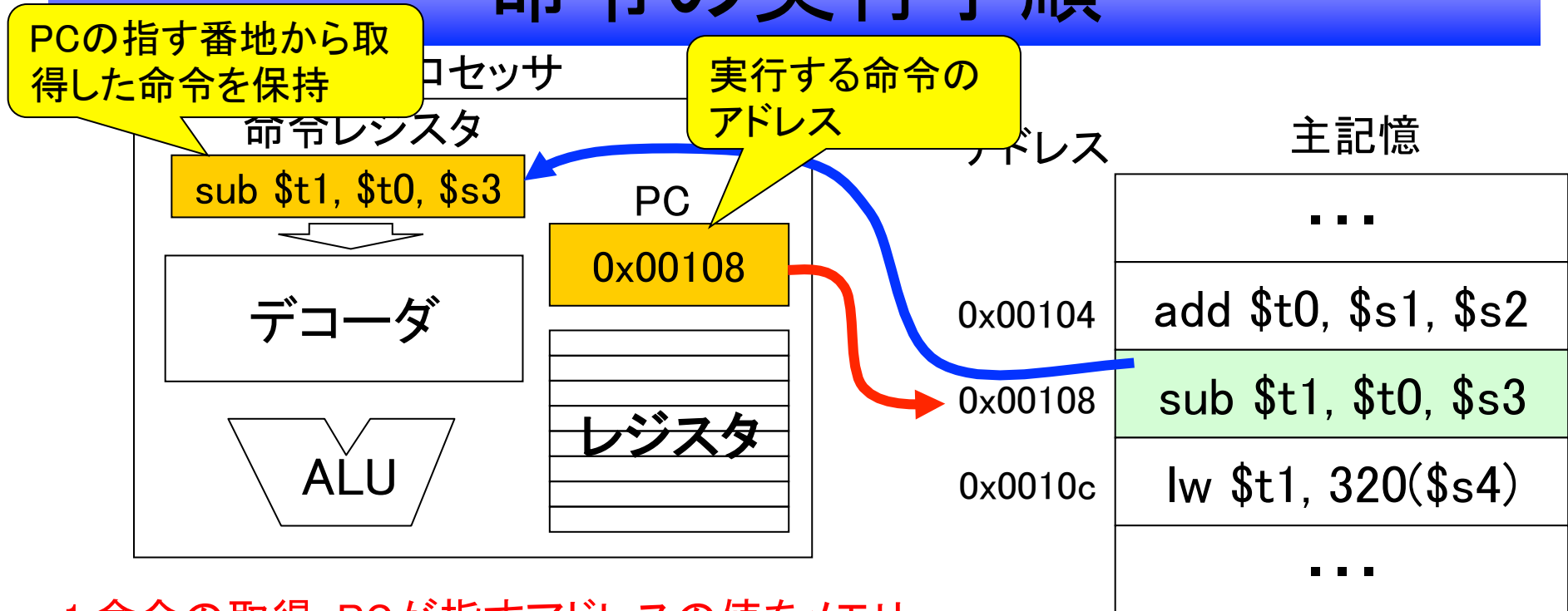
1. 命令の取得: PCが指すアドレスの値をメモリから命令レジスタへ取得. 同時に, 次命令の取得に備えてPCの値を更新(例えば+4)

2. 命令の解読: 命令レジスタの値をデコード

3. 命令の実行: 解読結果に従って実行

実際には, 各命令は2進表現でメモリに格納されている

命令の実行手順



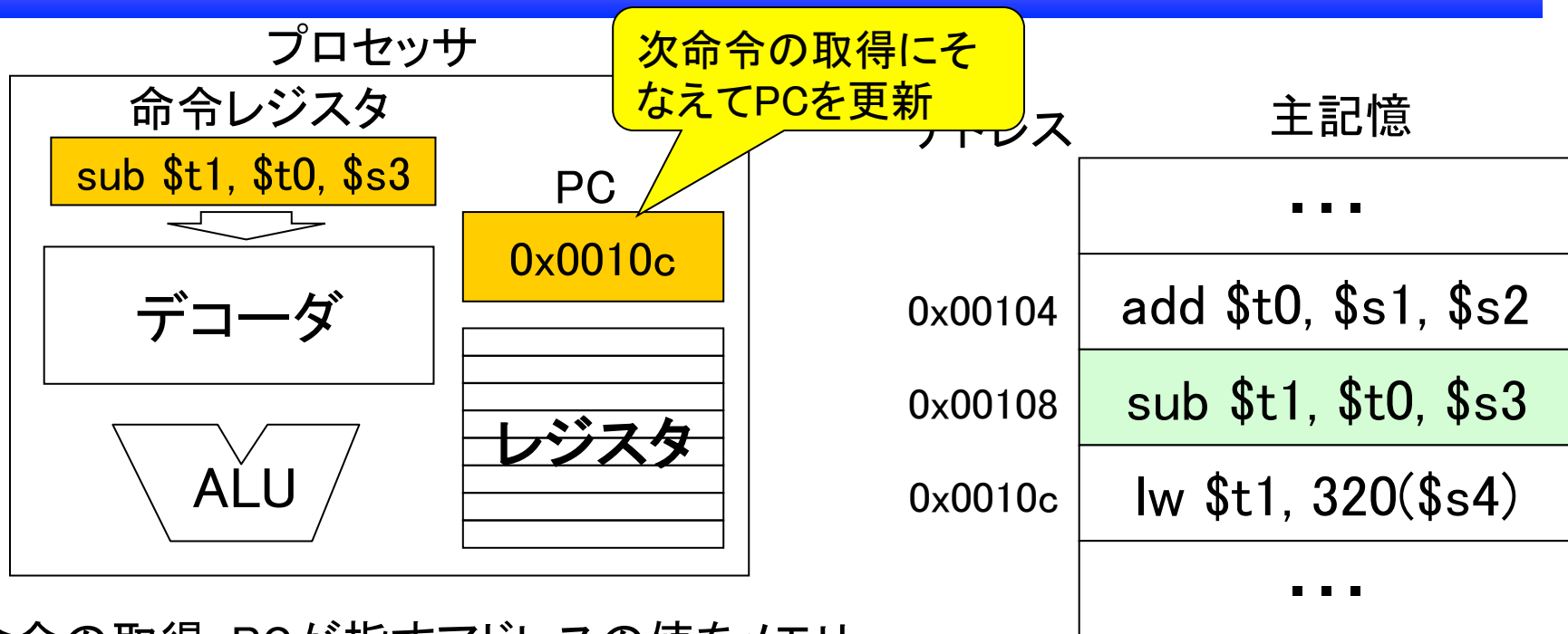
1. 命令の取得: PCが指すアドレスの値をメモリから命令レジスタへ取得. 同時に, 次命令の取得に備えてPCの値を更新(例えば+4)

2. 命令の解読: 命令レジスタの値をデコード

3. 命令の実行: 解読結果に従って実行

実際には, 各命令は2進表現でメモリに格納されている

命令の実行手順



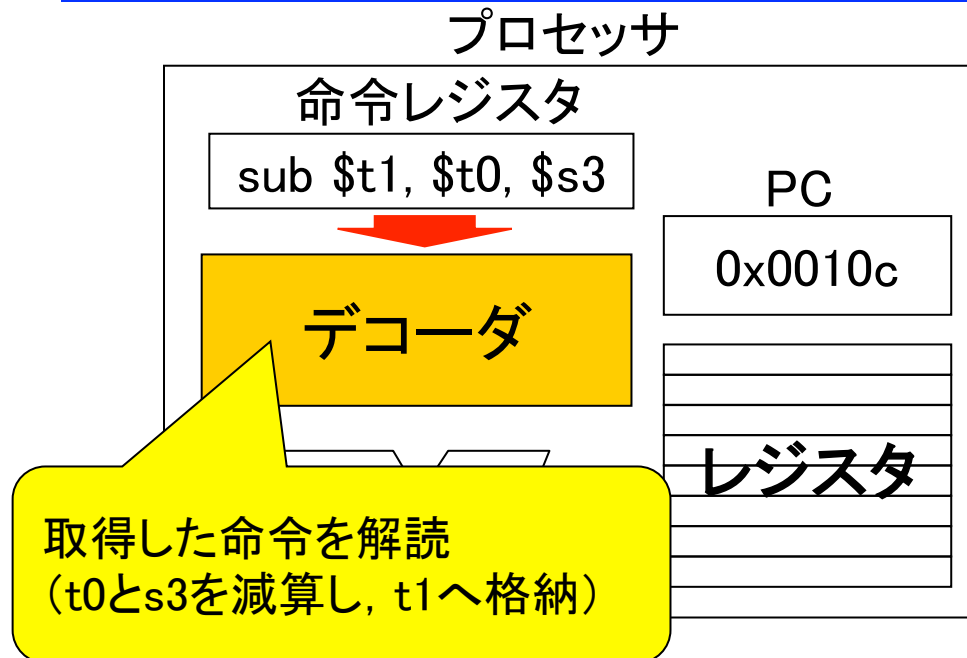
1. 命令の取得: PCが指すアドレスの値をメモリから命令レジスタへ取得. **同時に, 次命令の取得に備えてPCの値を更新(例えば+4)**

2. 命令の解読: 命令レジスタの値をデコード

3. 命令の実行: 解読結果に従って実行

実際には, 各命令は2進表現でメモリに格納されている

命令の実行手順



1. 命令の取得: PCが指すアドレスの値をメモリから命令レジスタへ取得. 同時に, 次命令の取得に備えてPCの値を更新(例えば+4)

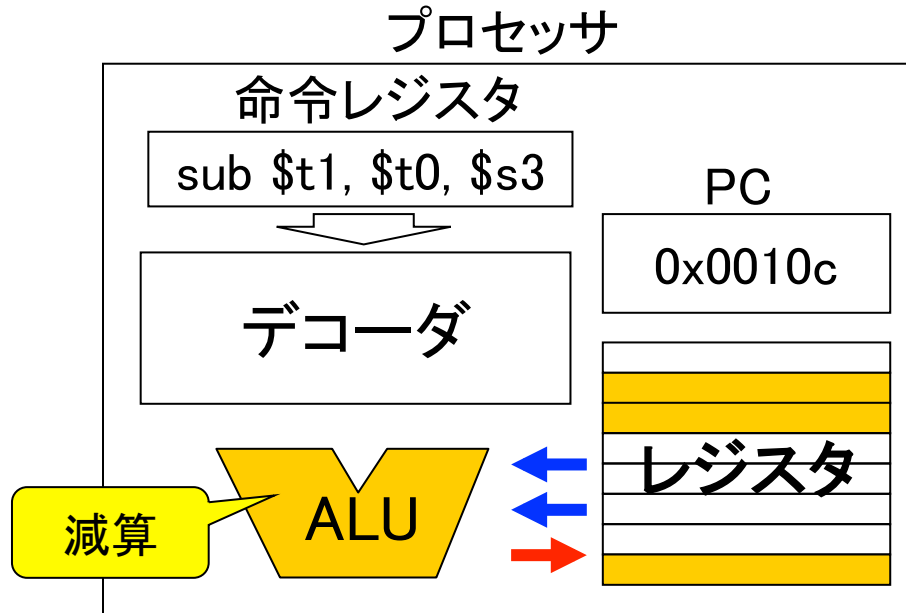
2. 命令の解読: 命令レジスタの値をデコード

3. 命令の実行: 解読結果に従って実行

アドレス	主記憶
	...
0x00104	add \$t0, \$s1, \$s2
0x00108	sub \$t1, \$t0, \$s3
0x0010c	lw \$t1, 320(\$s4)
	...

実際には, 各命令は2進表現でメモリに格納されている

命令の実行手順



アドレス	主記憶
	...
0x00104	add \$t0, \$s1, \$s2
0x00108	sub \$t1, \$t0, \$s3
0x0010c	lw \$t1, 320(\$s4)
	...

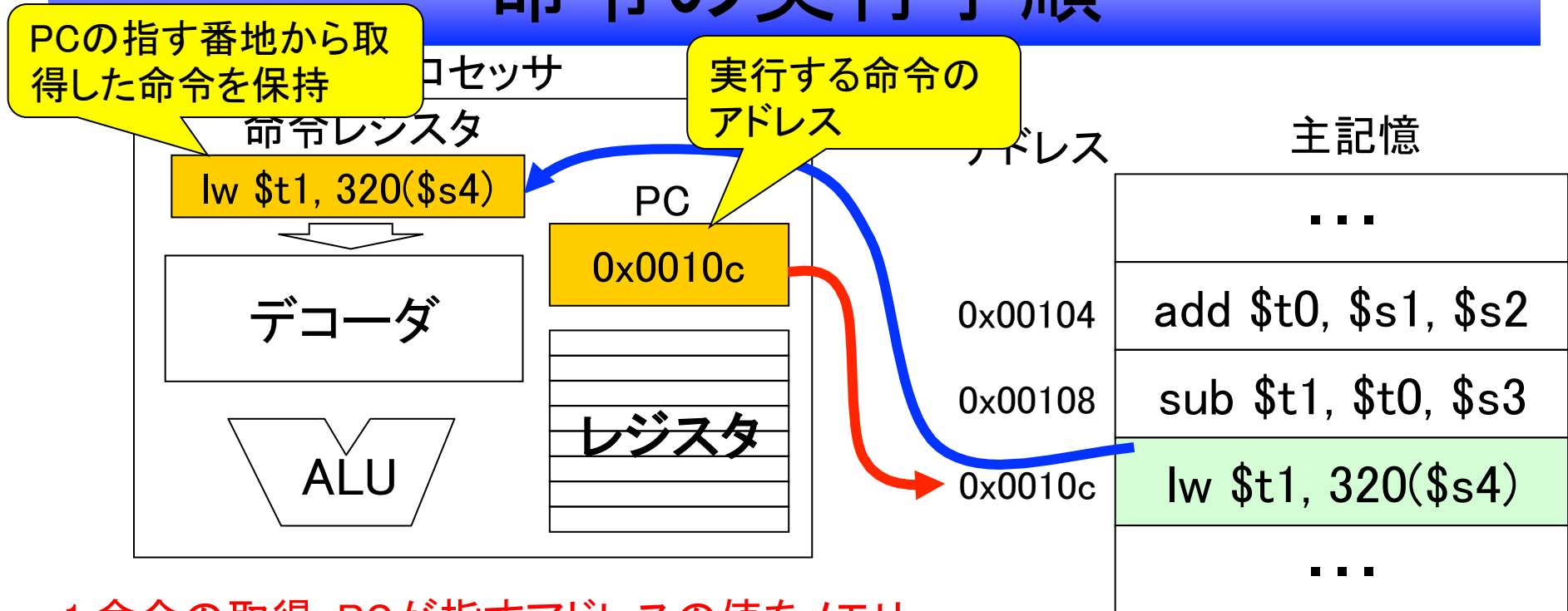
1. 命令の取得: PCが指すアドレスの値をメモリから命令レジスタへ取得. 同時に, 次命令の取得に備えてPCの値を更新(例えば+4)

2. 命令の解読: 命令レジスタの値をデコード

3. 命令の実行: 解読結果に従って実行

実際には, 各命令は2進表現でメモリに格納されている

命令の実行手順



1. 命令の取得: PCが指すアドレスの値をメモリから命令レジスタへ取得。同時に、次命令の取得に備えてPCの値を更新(例えば+4)

2. 命令の解読: 命令レジスタの値をデコード

3. 命令の実行: 解読結果に従って実行

以降、繰り返し...

実際には、各命令は2進表現でメモリに格納されている

算術演算：加算命令/減算命令

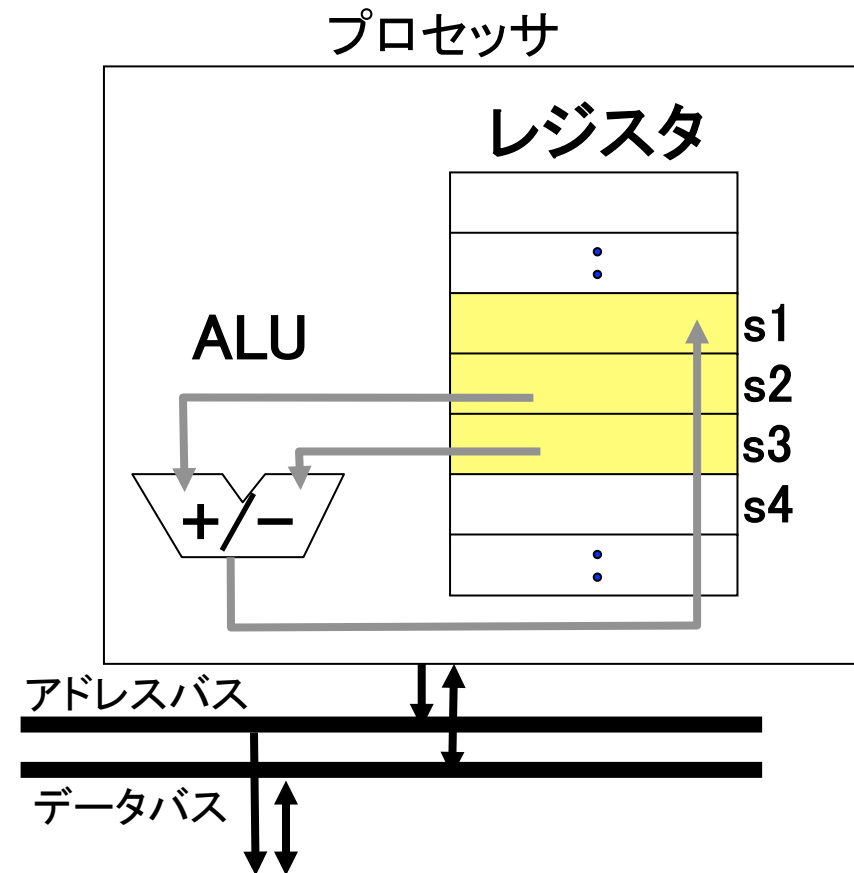
例)

add \$s1, \$s2, \$s3

レジスタ\$s2の値と、レジスタ
\$s3の値を加算して、レジスタ
\$s1に格納する

sub \$s1, \$s2, \$s3

レジスタ\$s2の値からレジスタ
\$s3の値を減算して、レジスタ
\$s1に格納する



データ転送：ロード/ストア命令

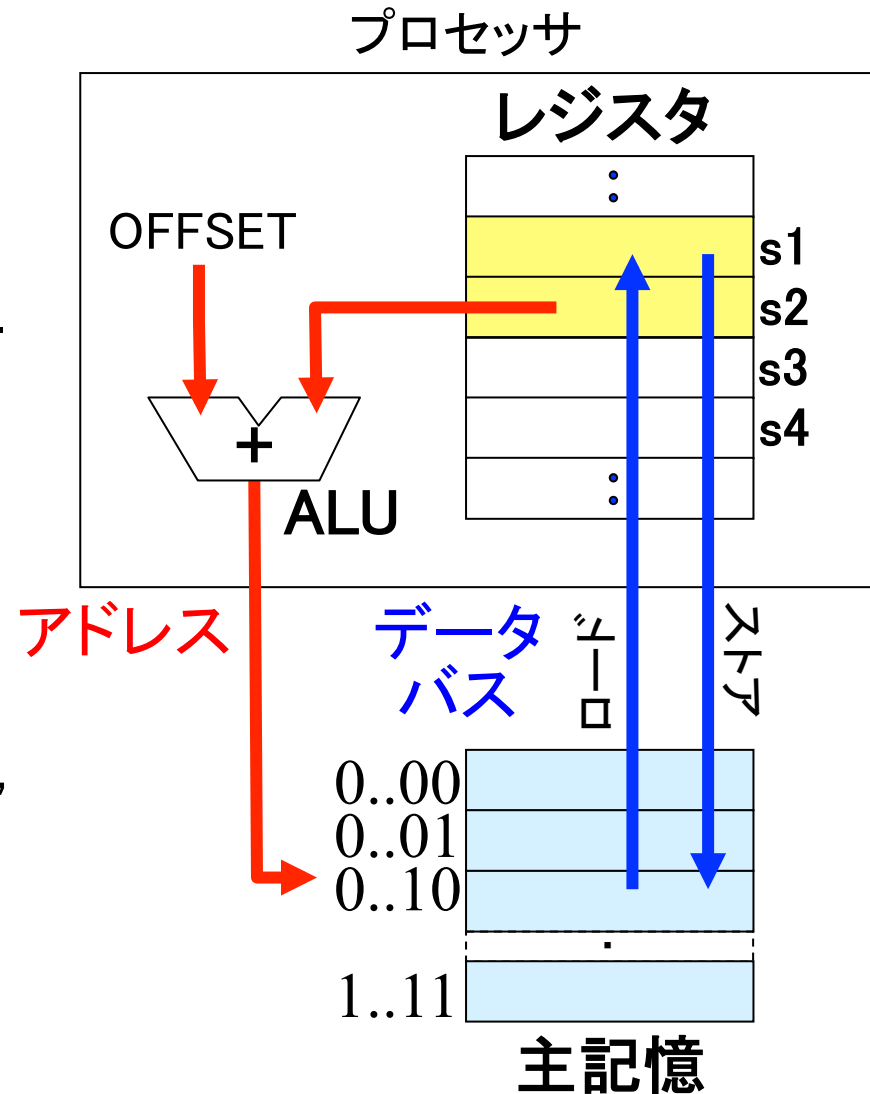
例)

lw \$s1, OFFSET(\$s2)

レジスタ\$s2の値にOFFSETを加えてメモリ・アクセス用アドレスを得る。このアドレスで指定されるメモリ内ワードデータ(4B)を読み出し、レジスタ\$s1に格納する(ロード)

sw \$s1, OFFSET(\$s2)


レジスタ\$s2の値にOFFSETを加えて、メモリ・アクセス用アドレスを得る。このアドレスで指定されるメモリ領域に対しレジスタ\$s1の値(4B)を書き込む(ストア)



逐次制御だけで十分なのか？

- 命令実行に関する基本動作
 - 主記憶に格納された命令を1個ずつ、順番に実行する(逐次制御)
- 全ての場合に対応できるのだろうか？

```
A[80] = (g + A[2]) - (h + A[i]);
```



```
lw $t0, 8($s4)
add $t0, $t0, $s1
add $t1, $s3, $s3
add $t1, $t1, $t1
add $t1, $t1, $s4
lw $t1, 0($t1)
add $t1, $t1, $s2
sub $t1, $t0, $t1
sw $t1, 320($s4)
```

逐次制御で問題なし！

```
if (i == j)
    f = g + h;
else
    f = g - h;
```


???

逐次制御だけでは対応できない！
条件に応じて分岐する必要がある！

条件分岐命令(1)


条件分岐命令がない場合

① add \$s1, \$s2, \$s3
② add \$t0, \$s1, \$s4
③ lw \$t1, 8(\$s5)
④ sub \$t1, \$t1, \$t0
⋮



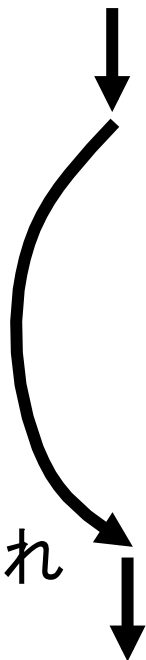
条件分岐命令がある場合

① add \$s1, \$s2, \$s3
② beq \$s1, \$s6, GoTo
add \$t0, \$s2, \$s4
sw \$t0, 8(\$s5)
⋮



命令実行の流れ

GoTo:
③ add \$s4, \$t4, \$t3
④ sw \$s4, 8(\$s5)



②で分岐条件が成立
した場合

分岐条件が不成立
の場合

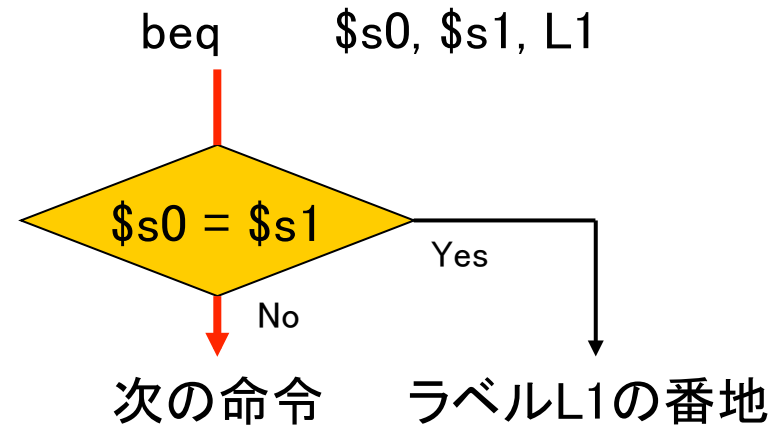
条件分岐命令(2)

例)

beq \$s0, \$s1, L1

レジスタ\$s0の値と\$s1の値を比較して、同じであればラベルL1へ分岐(PCの値を設定)

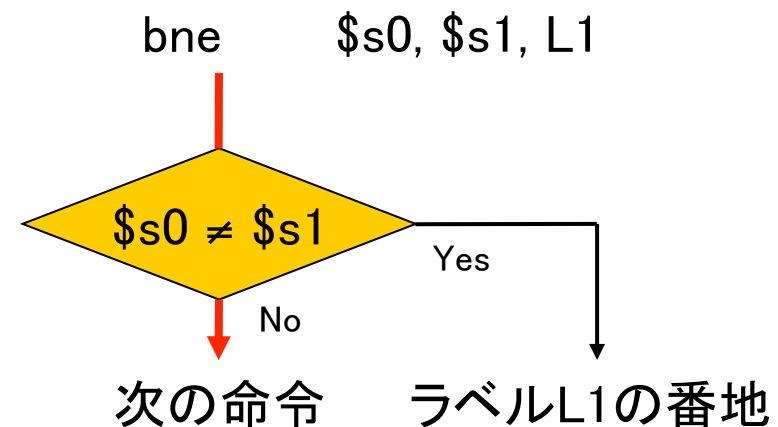
beq: Branch on EQual



bne \$s0, \$s1, L1

レジスタ\$s0の値と\$s1の値を比較して、同じでなければラベルL1へ分岐(PCの値を設定)

bne: Branch on Not Equal

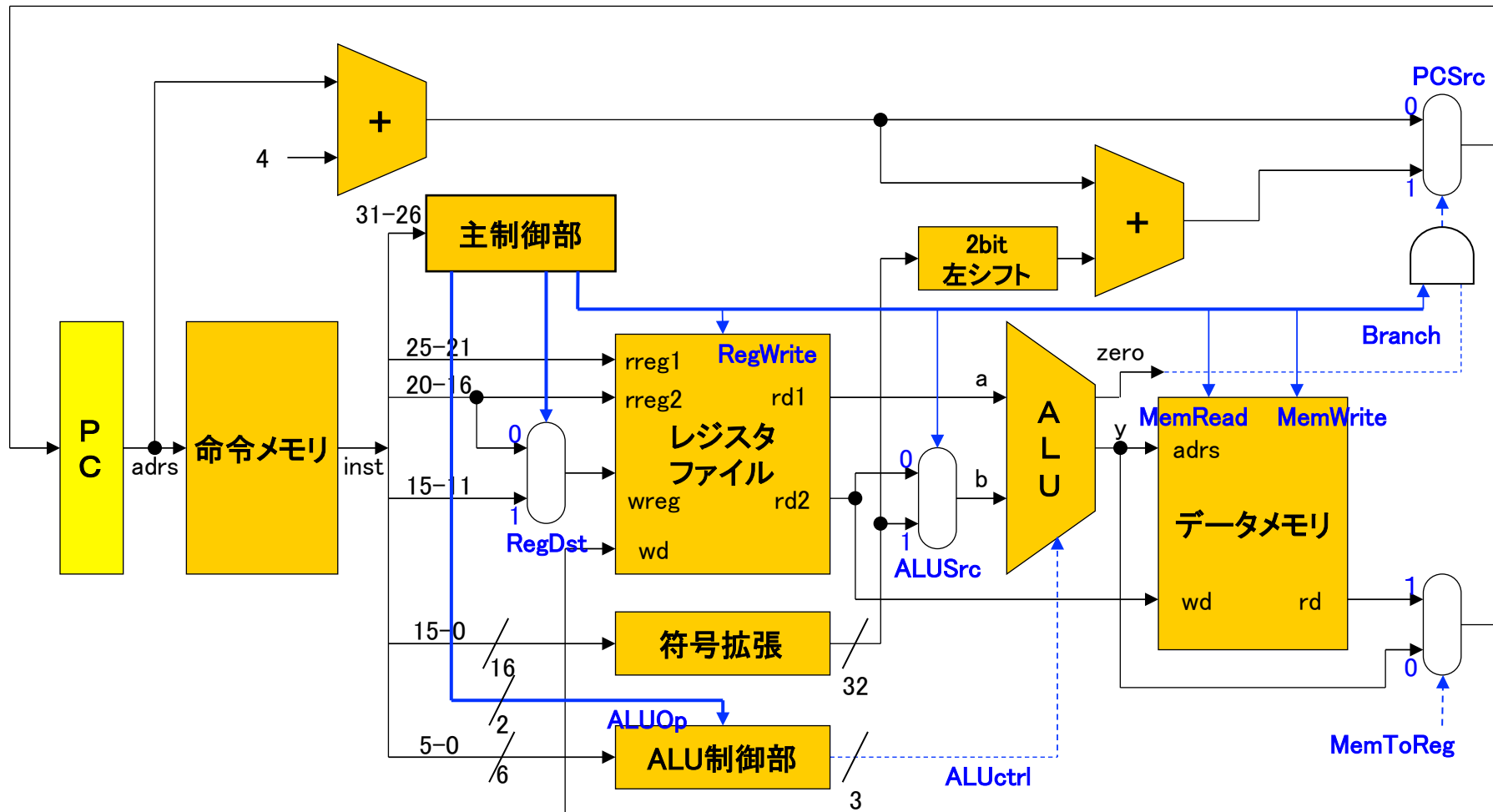


コンピュータ・アーキテクチャⅡ

「パターソン&ヘネシー コンピュータの構成と設計」第6.1～6.3節

命令パイプライン処理

シングルサイクル・データパス



命令実行のスループット

- プログラム実行時間

= 時間 / プログラム

$$= \frac{\text{実行命令数}}{\text{プログラム}} \times \frac{\text{クロック・サイクル数}}{\text{命令}} \times \frac{\text{時間}}{\text{クロックサイクル}}$$

= 実行命令数 × CPI × クロックサイクル時間

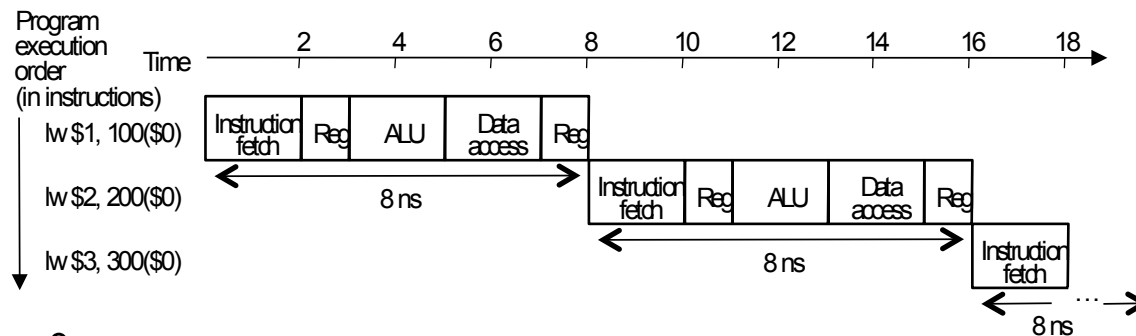
- CPI (clock-cycles per instruction) : 1 命令当りの平均所要クロックサイクル数

- 命令実行スループット

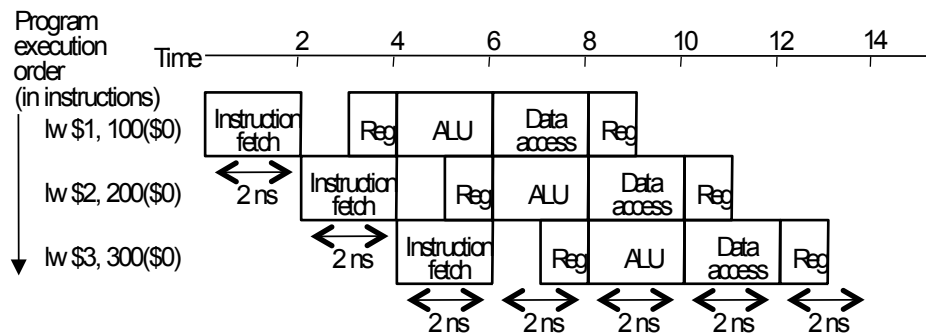
- IPC (instructions per clock-cycle) : 1 クロックサイクル当りに実行可能な命令数
- CPI の逆数

命令パイプライン処理による 命令実行スループット向上

- 非命令パイプライン処理 (シングルサイクルデータパス)
 - スループット: 1命令/8ns



- 命令パイプライン処理
 - スループット: 1命令/2ns



命令パイプライン処理：概念（１）

- 命令の実行過程を複数のステージに分割。たとえば，マルチサイクル・データパスの実行過程（次スライド参照）に倣って、以下の5ステージに分割
 1. 命令フェッチ（IF）
 2. 命令デコード，レジスタ読出し（ID）
 3. 命令実行（EX）
 4. メモリ・アクセス（MEM）
 5. レジスタ書込み（WB）
- 異なる命令の異なるステージを同時に処理することで，命令実行のスループットを向上

マルチサイクル・データパスの 命令実行過程

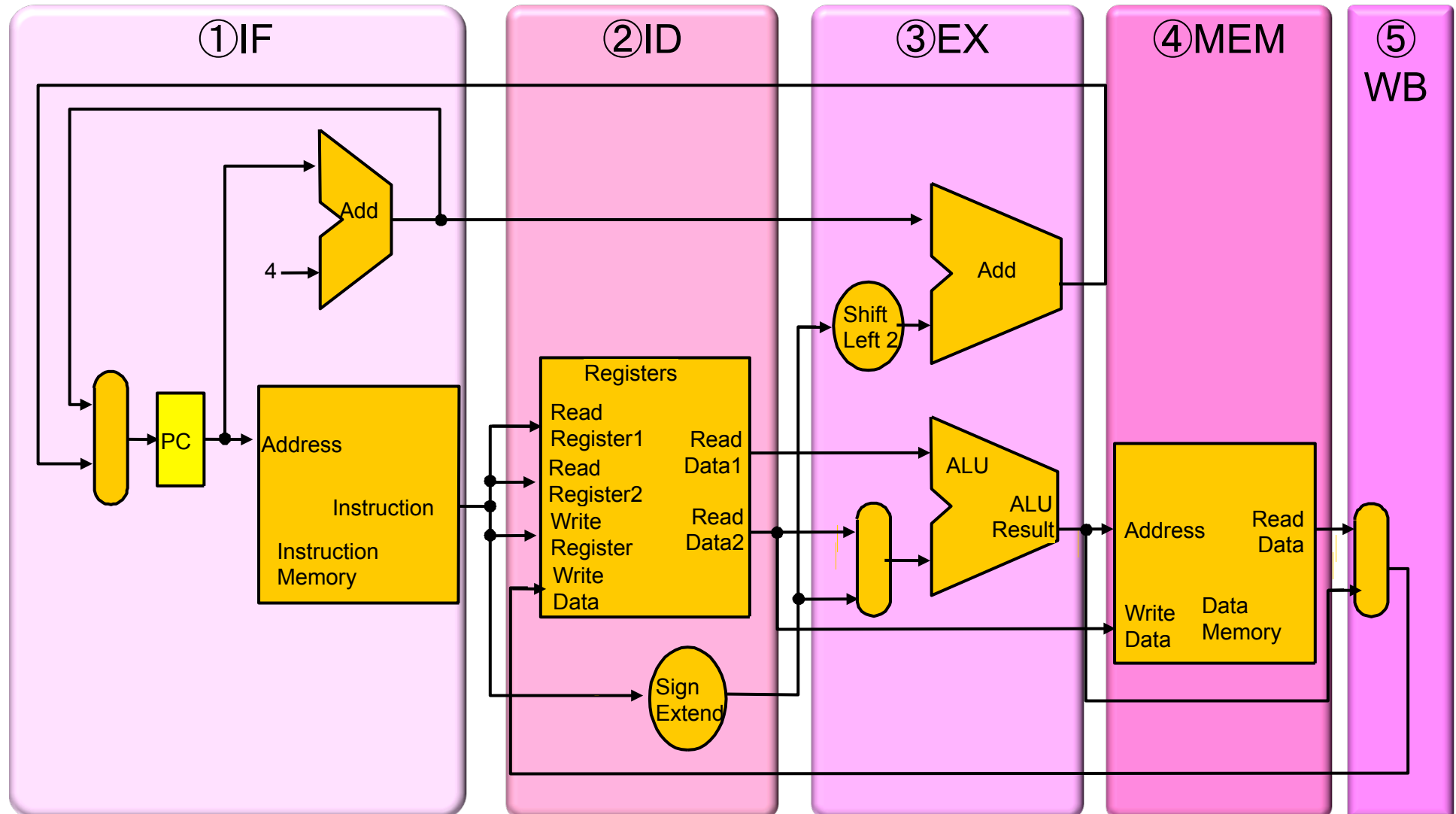
実行ステップ	算術論理 演算命令	ロード/ストア命令	分岐命令	ジャンプ命令
1. 命令フェッチ	$IR = \text{Memory}[PC]$ $PC = PC + 4$			
2. 命令デコード, オペランド読出し	$A = \text{Reg}[IR[25-21]]$ $B = \text{Reg}[IR[20-16]]$ $ALUOut = PC + (\text{sign-extend}(IR[15-0]) \ll 2)$			
3. 命令実行	$ALUOut = A \text{ op } B$	$ALUOut$ $= A + \text{sign-extend}(IR[15-0])$	if (A==B) then $PC = ALUOut$	PC $= PC[31:28] $ $(IR[15-0]) \ll 2$
4. メモリアクセス	$\text{Reg}[IR[15-11]]$ $= ALUOut$	$MDR = \text{Memory}[ALUOut]$		
		$\text{Memory}[ALUOut] = B$		
5. レジスタ書込み		$\text{Reg}[IR[20-16]] = MDR$		

命令パイプライン処理：概念（2）

- 命令の実行過程を複数のステージに分割。たとえば、マルチサイクル・データパスの実行過程（前スライド参照）に倣って、以下の5ステージに分割
 1. 命令フェッチ（IF）
 2. 命令デコード，レジスタ読出し（ID）
 3. 命令実行（EX）
 4. メモリ・アクセス（MEM）
 5. レジスタ書込み（WB）
- 異なる命令の異なるステージを同時に処理することで，命令実行のスループットを向上

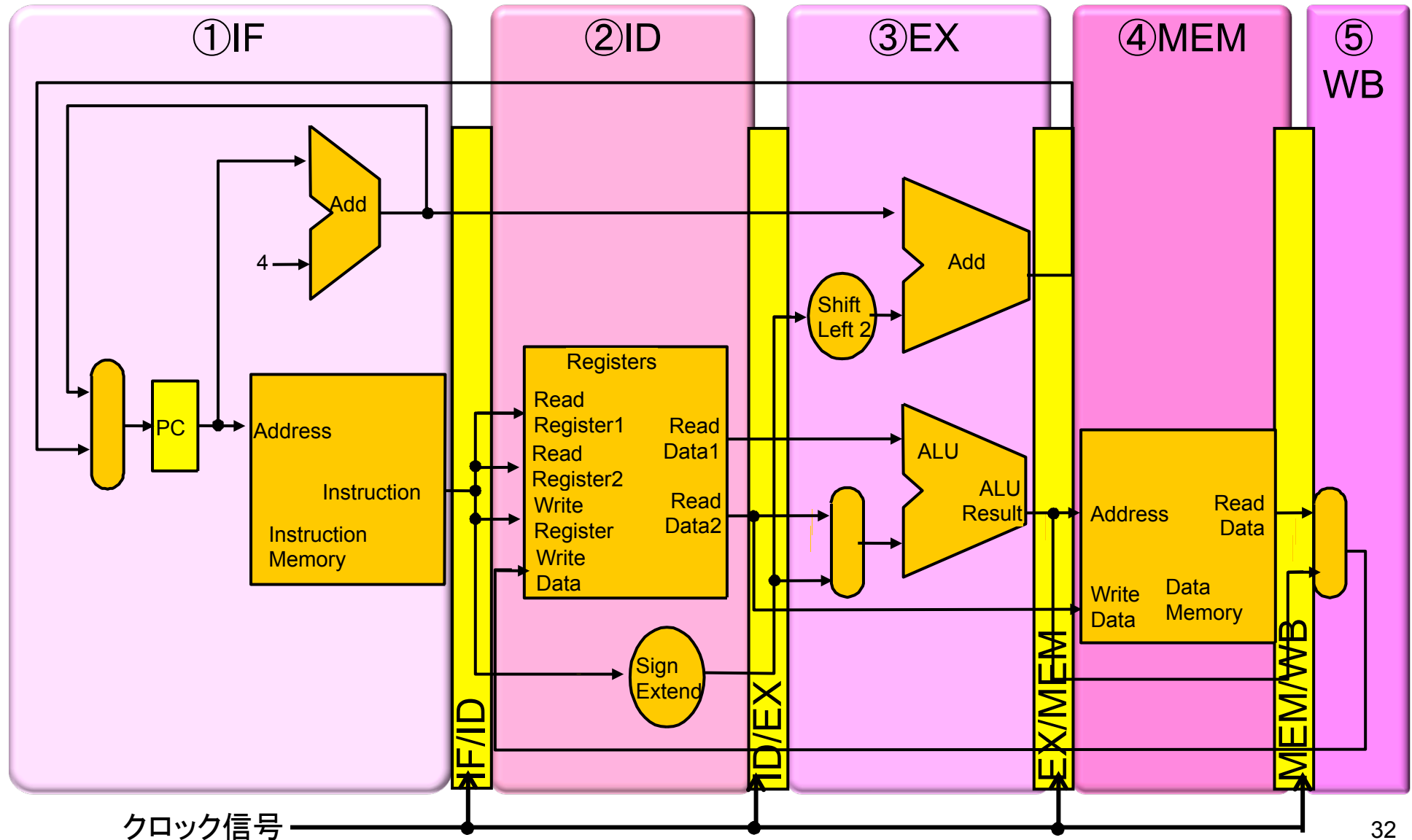
データパスのパイプライン化(1)

ーシングルサイクル・データパスを複数のステージに分割ー



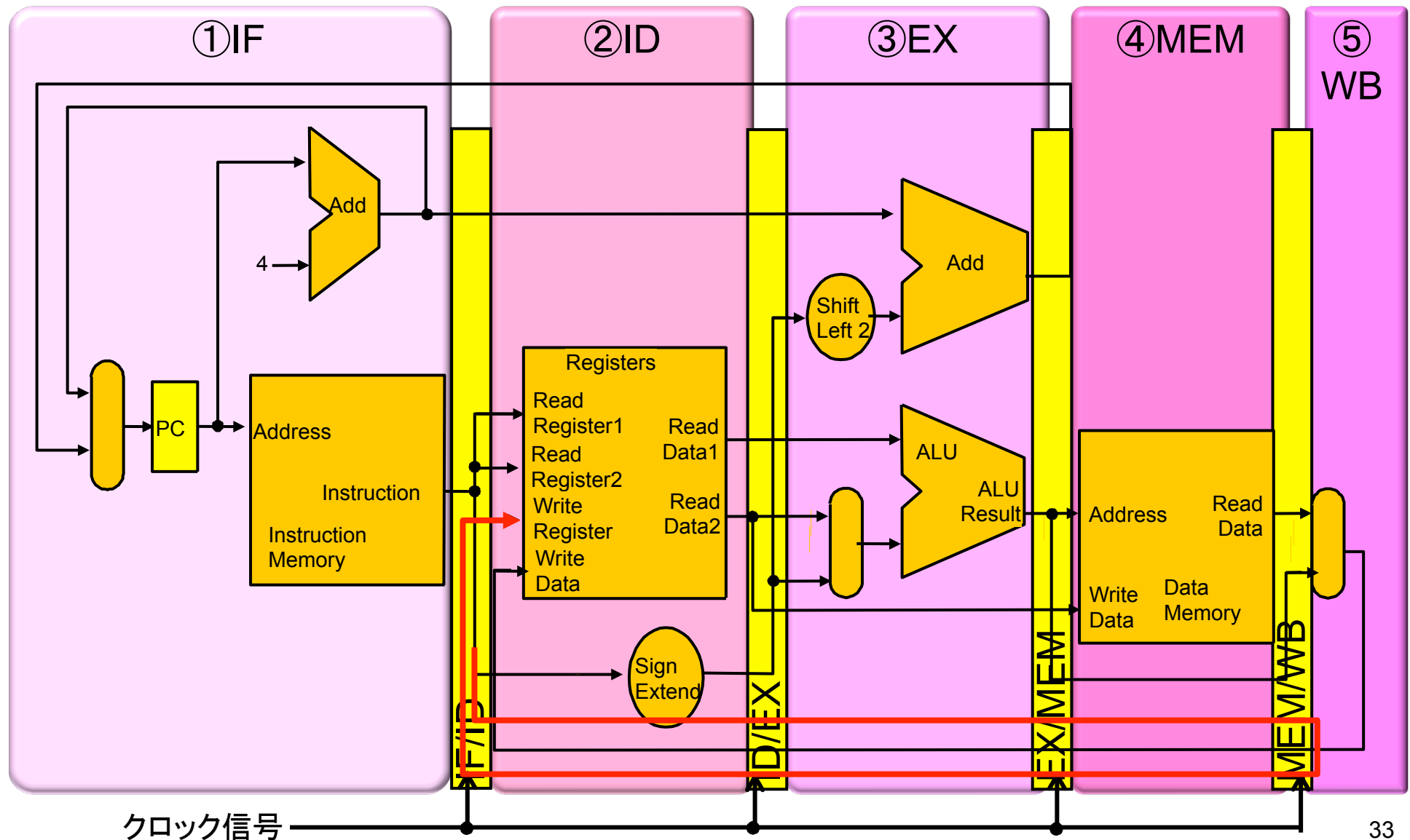
データパスのパイプライン化(2)

ーパイプラインレジスタをステージ間に挿入ー



データパスのパイプライン化(3)

—WBステージで必要となる情報を伝搬—



パイプライン・レジスタ

ーまとめー

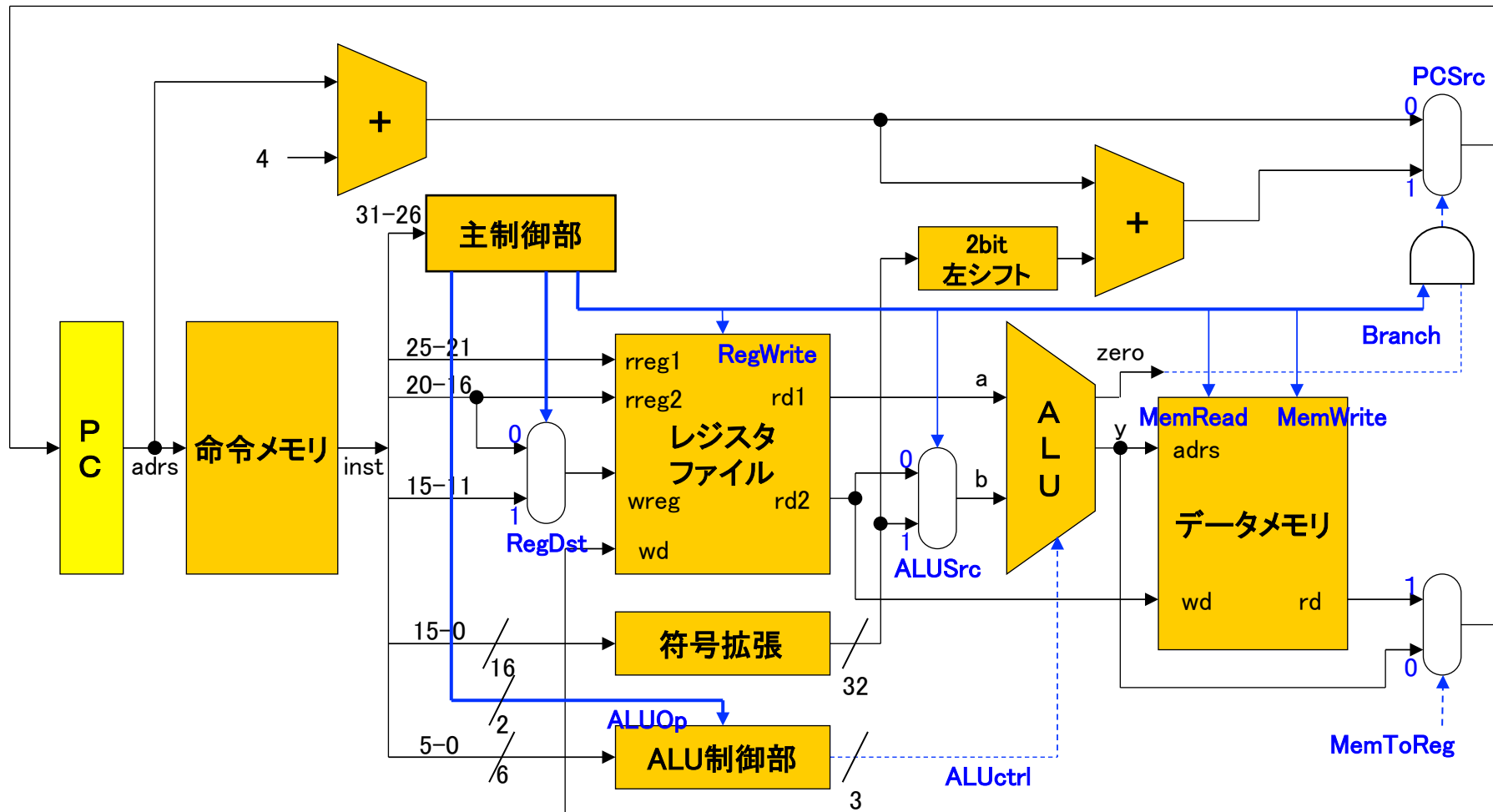
パイプライン・レジスタ	フィールド	意味	マルチサイクル・データパスのハードウェア・レジスタとの対応
IF/ID	IR	IFステージでフェッチされた命令32ビットを格納する命令レジスタ。	IR
	PC4	IFステージでフェッチされた命令のPCに4を加えた値。当該命令が分岐命令の場合、EXステージにおける分岐先命令アドレス生成に使用。	—
ID/EX	A	IRに格納されている命令のrsフィールドで指定されるレジスタの値を読み出した値。	A
	B	IRに格納されている命令のrtフィールドで指定されるレジスタの値を読み出した値。	B
	SE	IRに格納されている命令のビット15-0を32ビットに符号拡張した結果。	—
	WR1	IRに格納されている命令のrdフィールド(R形式の場合)で指定されるレジスタ番号。	—
	WR2	IRに格納されている命令のrtフィールド(I形式の場合)で指定されるレジスタ番号。	—
	PC4*	IF/ID(PC4)を伝搬。	—
EX/MEM	ALUOut	EXステージでALUを用いて演算した結果。	ALUOut
	BA	EXステージで計算した分岐先アドレス。	—
	Zero	EXステージでALUを用いて演算した結果が0か否か？	—
	B*	ID/EX(B)を伝搬。	—
	WR	命令形式に従って、ID/EX(WR1)かID/EX(WR2)の一方を選択して格納。	—
MEM/WB	MDR	ロード命令がMEMステージで読み出した値。	MDR
	ALUout*	EX/MEM(ALUOut)を伝搬。	—
	WR*	EX/MEM(WR)を伝搬。	—

命令パイプライン処理における命令実行過程

パイプライン・ ステージ	算術論理演算命令	ロード/ストア命令	分岐命令
1. 命令フェッチ(IF)	$IF/ID(IR) = \text{InstructionMemory}[PC]$ $IF/ID(PC4) = PC + 4 \quad PC = PC + 4$		
2. 命令デコード, オペランド読出し (ID)	$ID/EX(A) = \text{Reg}[IR[25-21]] \quad ID/EX(B) = \text{Reg}[IR[20-16]]$ $ID/EX(SE) = \text{sign-extend}(IR[15-0])$ $ID/EX(PC4^*) = PC4$ $ID/EX(WR1) = IR[15-10] \quad ID/EX(WR2) = IR[15-10]$		
3. 命令実行(EX)	$EX/MEM(ALUOut) = A \text{ op } B$ $EX/MEM(WR) = WR1 \text{ or } WR2$	$EX/MEM(ALUOut) = A + SE$ $EX/MEM(B^*) = B$	$EX/MEM(BA) = PC4^* + (SE \ll 2)$ $EX/MEM(Zero) = 1 \text{ if } (A == B)$
4. メモリアクセス (MEM)	$MEM/WB(ALUout^*) = ALUout$ $MEM/WB(WR^*) = WR$	$MEM/WB(MDR) = \text{DataMemory}[ALUOut]$ $MEM/WB(WR^*) = WR$	$\text{if } (Zero == 1) \text{ then } PC = BA$
		$\text{DataMemory}[ALUOut] = B^*$	
5. レジスタ書込み (WB)	$\text{Reg}[WR^*] = ALUout^*$	$\text{Reg}[WR^*] = MDR$	

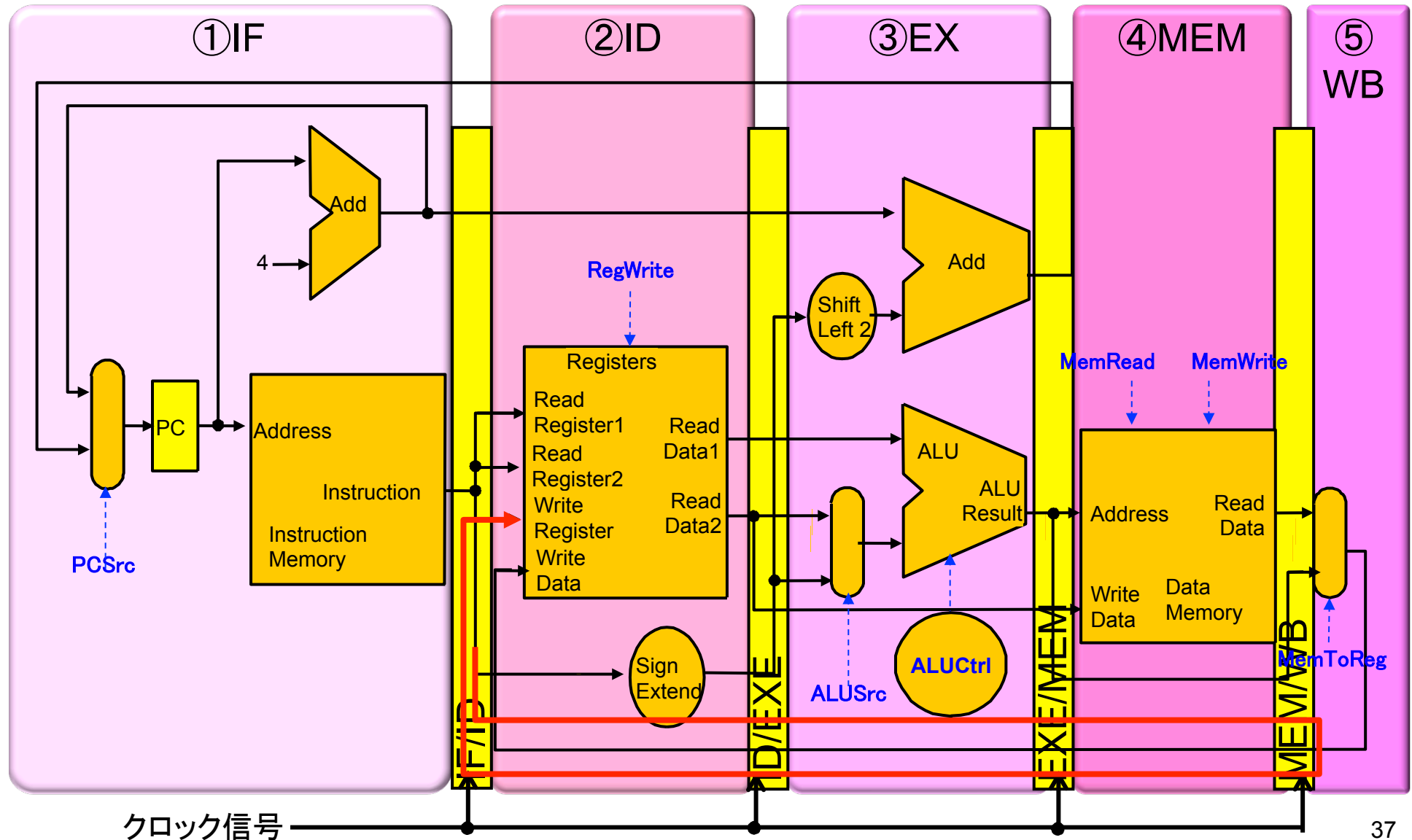
命令パイプラインの制御(1)

一元のシングルサイクル・データパスの制御



命令パイプラインの制御(2)

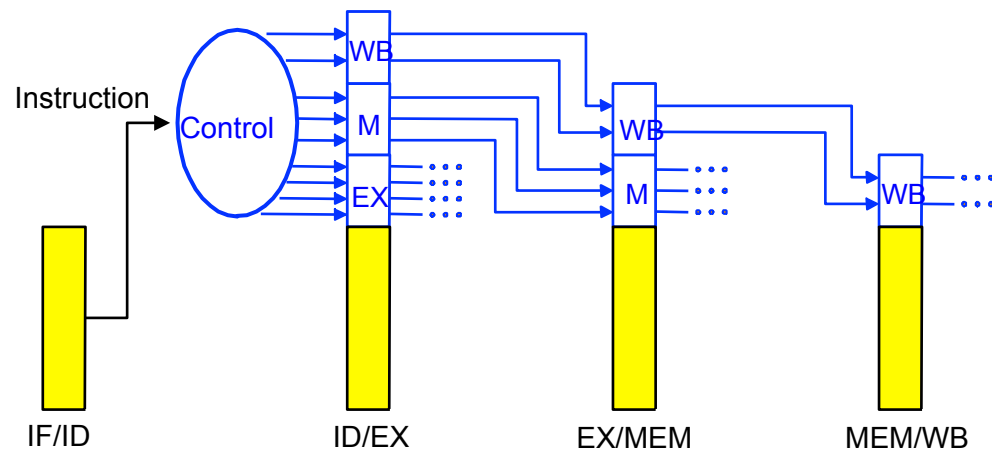
－同様に制御信号を挿入－



命令パイプラインの制御(3)

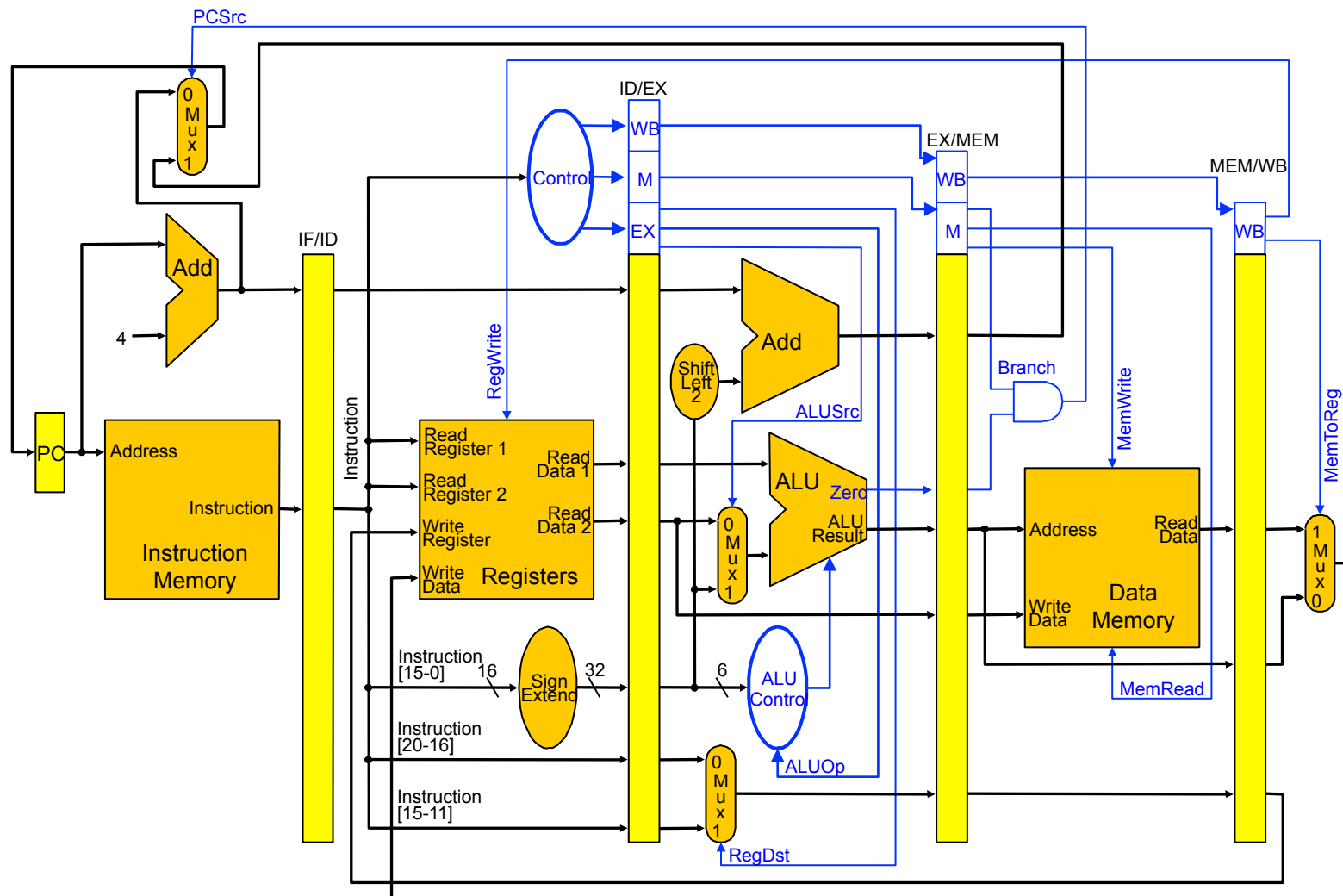
ー制御信号もデータと一緒にステージを移動ー

命令	EXステージへの制御信号				MEMステージへの制御信号			WBステージへの制御信号	
	Reg Dst	ALU Op1	ALU Op0	ALU Src	Branch	Mem Read	Mem Write	Reg Write	MemTo Reg
R形式	1	1	0	0	0	0	0	1	0
lw	0	0	0	1	0	1	0	1	1
sw	x	0	0	1	0	0	1	0	x
beq	x	0	1	0	1	0	0	0	x



命令パイプラインの制御(4)

－完成版－



命令パイプラインが満たすべき条件

- 命令の実行過程を複数のステージに分割
- 各ステージは、前段のパイプライン・レジスタの内容に従って、当該ステージに割り当てられたハードウェア資源を用いて処理を行い、その結果を後段のパイプライン・レジスタに格納
 - 上記を基本的には1クロックサイクルで実施
- 各ハードウェア資源は高々1つのステージに割り当てられ、当該ステージでのみ使用(次スライド参照)
 - 例外: PC(プログラムカウンタ)、レジスタファイル(汎用レジスタ、浮動小数点レジスタ)
- PC(プログラムカウンタ)は毎クロックサイクル更新され、毎クロックサイクル、新しい命令がフェッチされる
- [努力目標]各ステージの処理負荷を均等化して、命令実行スループットを向上！

ハードウェア資源とステージとの関係

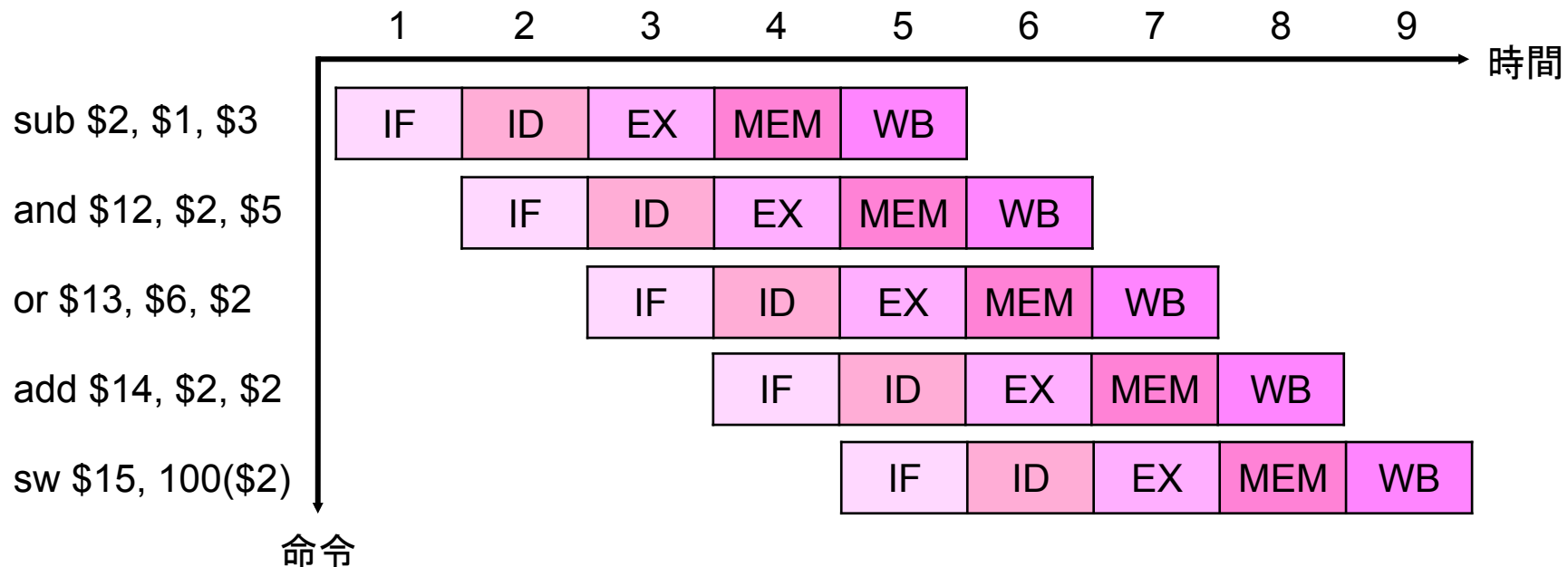
ハードウェア 資源	パイプライン・ステージ				
	IF	ID	EX	MEM	WB
PC	✓			✓	
命令メモリ	✓				
加算器 (PC+4)	✓				
レジスタファ イル		✓			✓
符号拡張		✓			
ALU			✓		
加算器 (分岐先アド レス計算)			✓		
データメモリ				✓	

例題(1)

ー命令の実行過程をシミュレーションー

- 以下の命令列の実行過程をクロックサイクル1～5の5クロックサイクル分に関してシミュレーション(模擬)する。

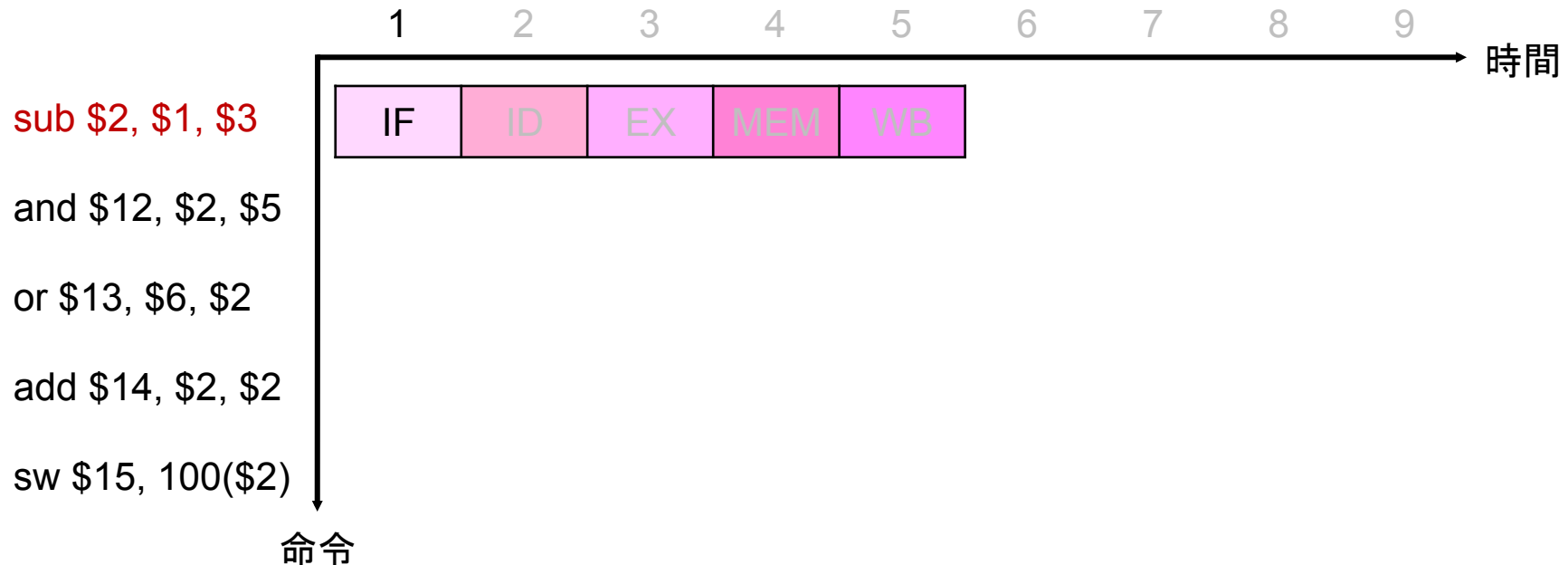
100₁₆: sub \$2, \$1, \$3 # \$2 = \$1 - \$3
104₁₆: and \$12, \$2, \$5 # \$12 = \$2 AND \$5
108₁₆: or \$13, \$6, \$2 # \$13 = \$6 OR \$2
10C₁₆: add \$14, \$2, \$2 # \$14 = \$2 + \$2
110₁₆: sw \$15, 100(\$2) # Mem[\$2+100] = \$15



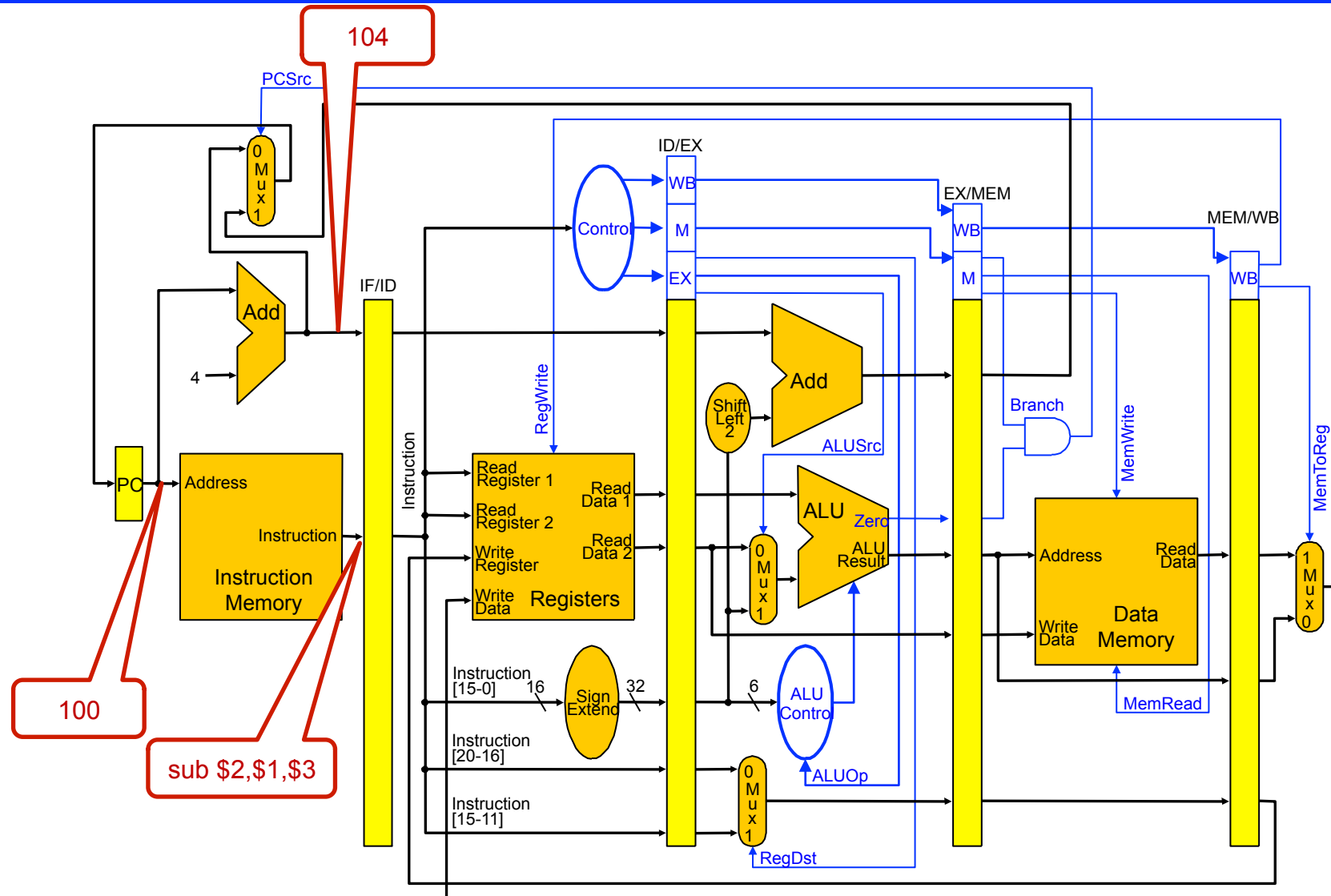
例題(2)：第1クロックサイクル

- 以下の命令列の実行過程をクロックサイクル1～5の5クロックサイクル分に関して示す。

100 ₁₆ :	sub \$2, \$1, \$3	# \$2 = \$1 - \$3
104 ₁₆ :	and \$12, \$2, \$5	# \$12 = \$2 AND \$5
108 ₁₆ :	or \$13, \$6, \$2	# \$13 = \$6 OR \$2
10C ₁₆ :	add \$14, \$2, \$2	# \$14 = \$2 + \$2
110 ₁₆ :	sw \$15, 100(\$2)	# Mem[\$2+100] = \$15



例題(3): 第1クロックサイクル



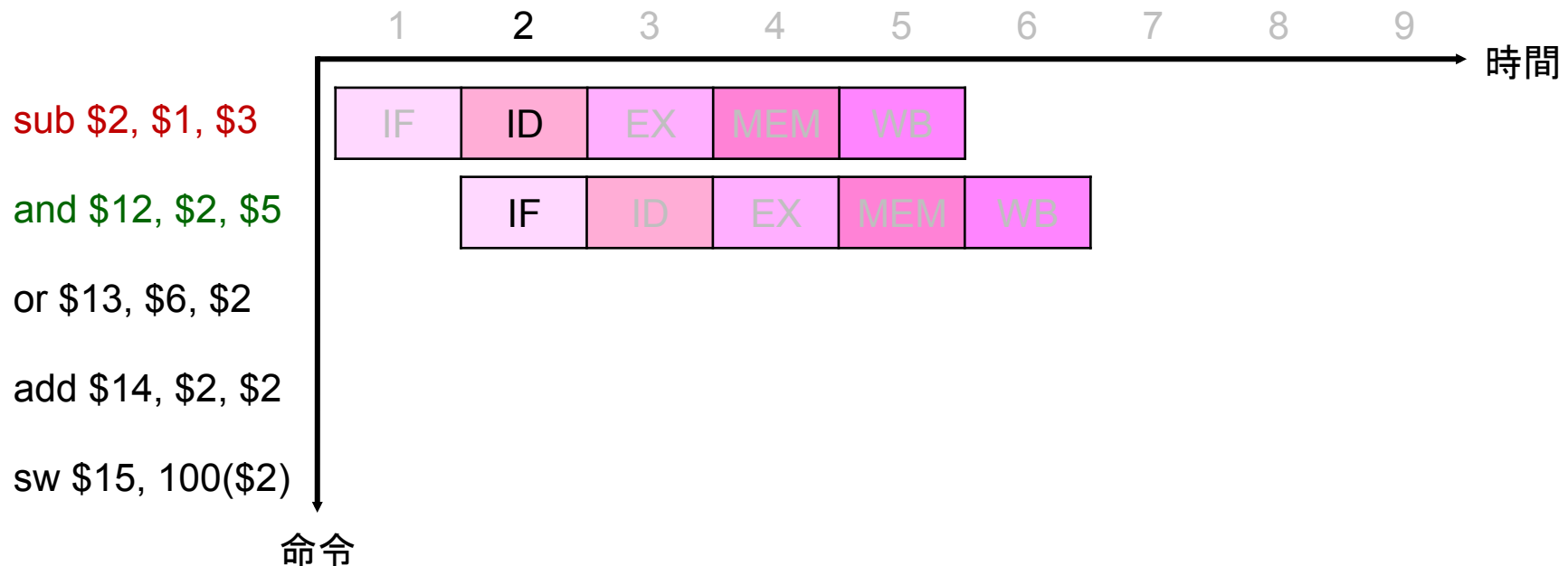
例題(4): 第1クロックサイクル

PC, パイプライン・レジスタ	フィールド	値	制御信号	値
PC	—	100		
IF/ID	IR			
	PC4			
ID/EX	A		ALUOp	
	B		RegDst	
	SE		ALUSrc	
	WR1		Branch	
	WR2		MemRead	
	PC4*		MemWrite	
			RegWrite	
			MemToReg	
EX/MEM	ALUOut		Branch	
	BA		MemRead	
	Zero		MemWrite	
	B*		RegWrite	
	WR		MemToReg	
MEM/WB	MDR		RegWrite	
	ALUout*		MemToReg	
	WR*			

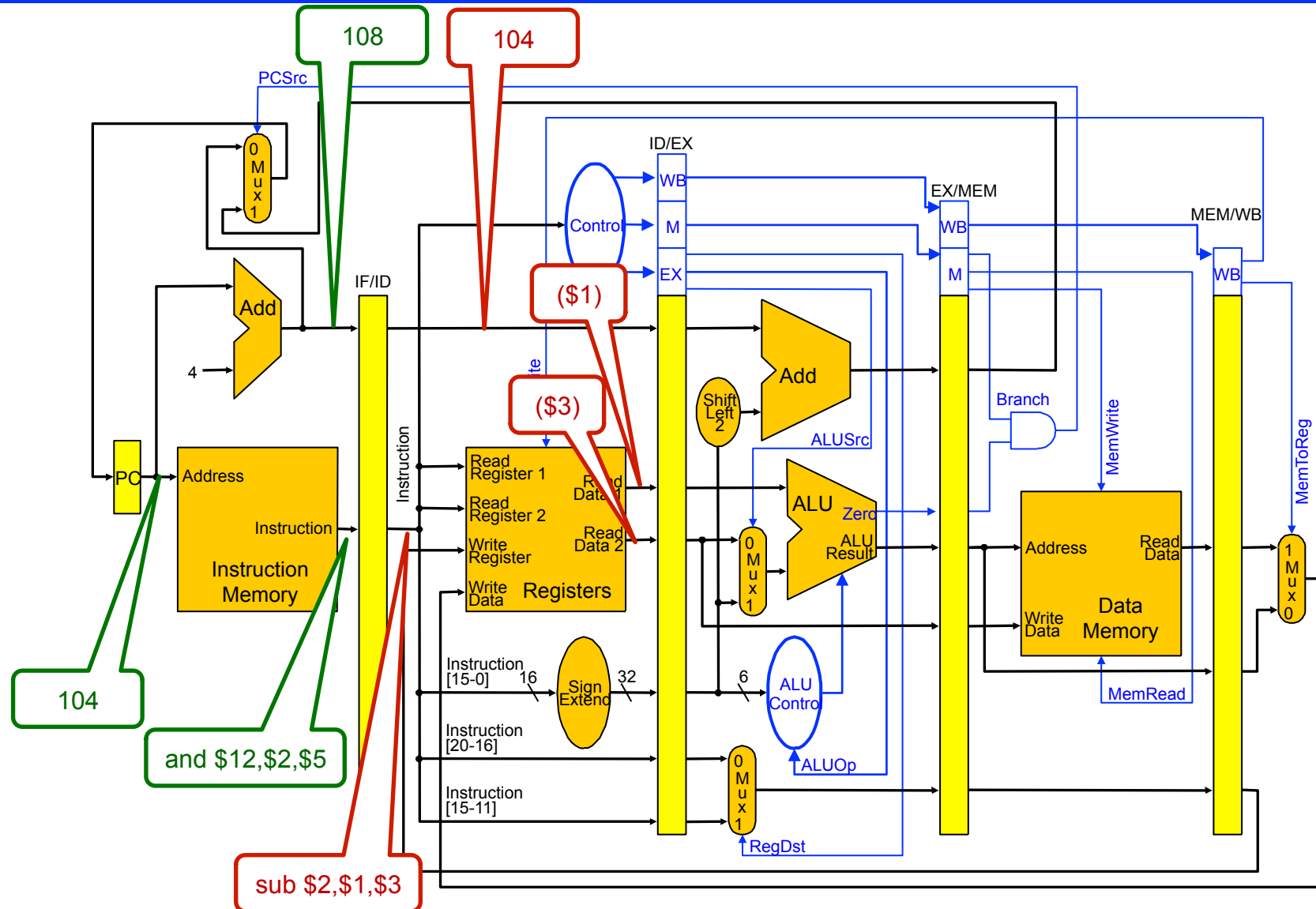
例題(5)：第2クロックサイクル

- 以下の命令列の実行過程をクロックサイクル1～5の5クロックサイクル分に関して示す。

100 ₁₆ :	sub \$2, \$1, \$3	# \$2 = \$1 - \$3
104 ₁₆ :	and \$12, \$2, \$5	# \$12 = \$2 AND \$5
108 ₁₆ :	or \$13, \$6, \$2	# \$13 = \$6 OR \$2
10C ₁₆ :	add \$14, \$2, \$2	# \$14 = \$2 + \$2
110 ₁₆ :	sw \$15, 100(\$2)	# Mem[\$2+100] = \$15



例題(6): 第2クロックサイクル



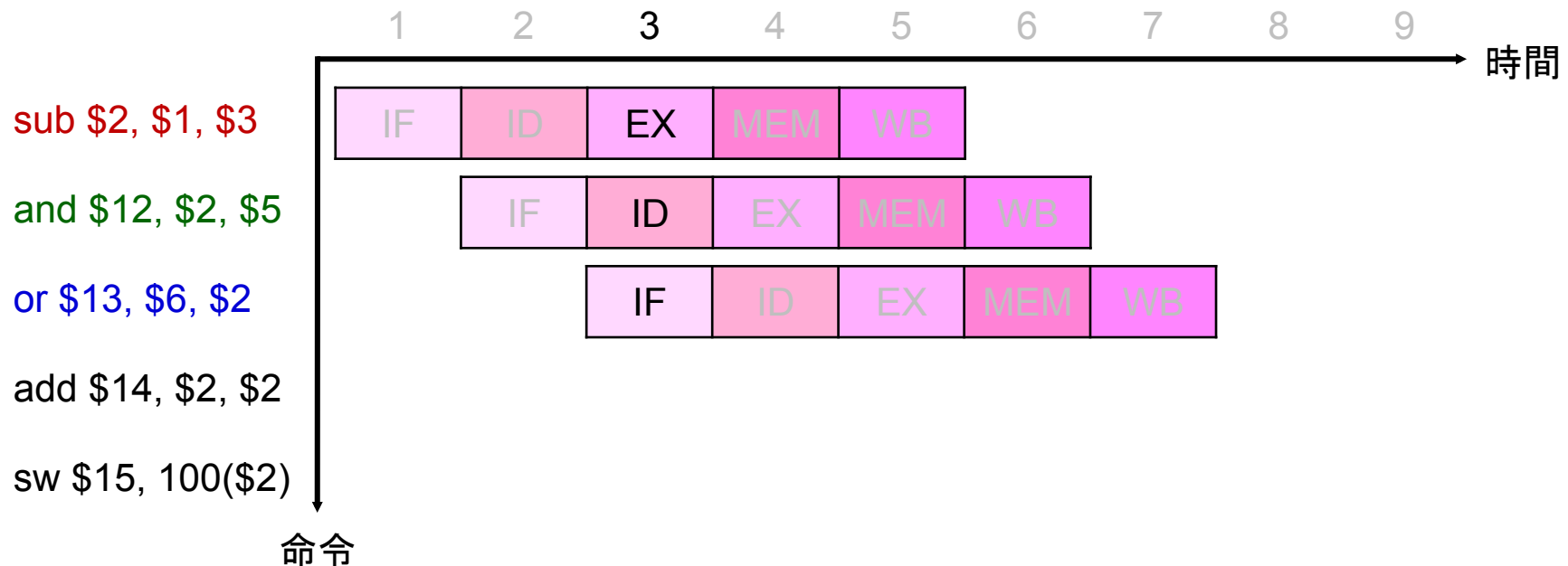
例題(7): 第2クロックサイクル

PC, パイプライン・レジスタ	フィールド	値	制御信号	値
PC	—	104		
IF/ID	IR	sub \$2,\$1,\$3		
	PC4	104		
ID/EX	A		ALUOp	
	B		RegDst	
	SE		ALUSrc	
	WR1		Branch	
	WR2		MemRead	
	PC4*		MemWrite	
			RegWrite	
			MemToReg	
EX/MEM	ALUOut		Branch	
	BA		MemRead	
	Zero		MemWrite	
	B*		RegWrite	
	WR		MemToReg	
MEM/WB	MDR		RegWrite	
	ALUout*		MemToReg	
	WR*			

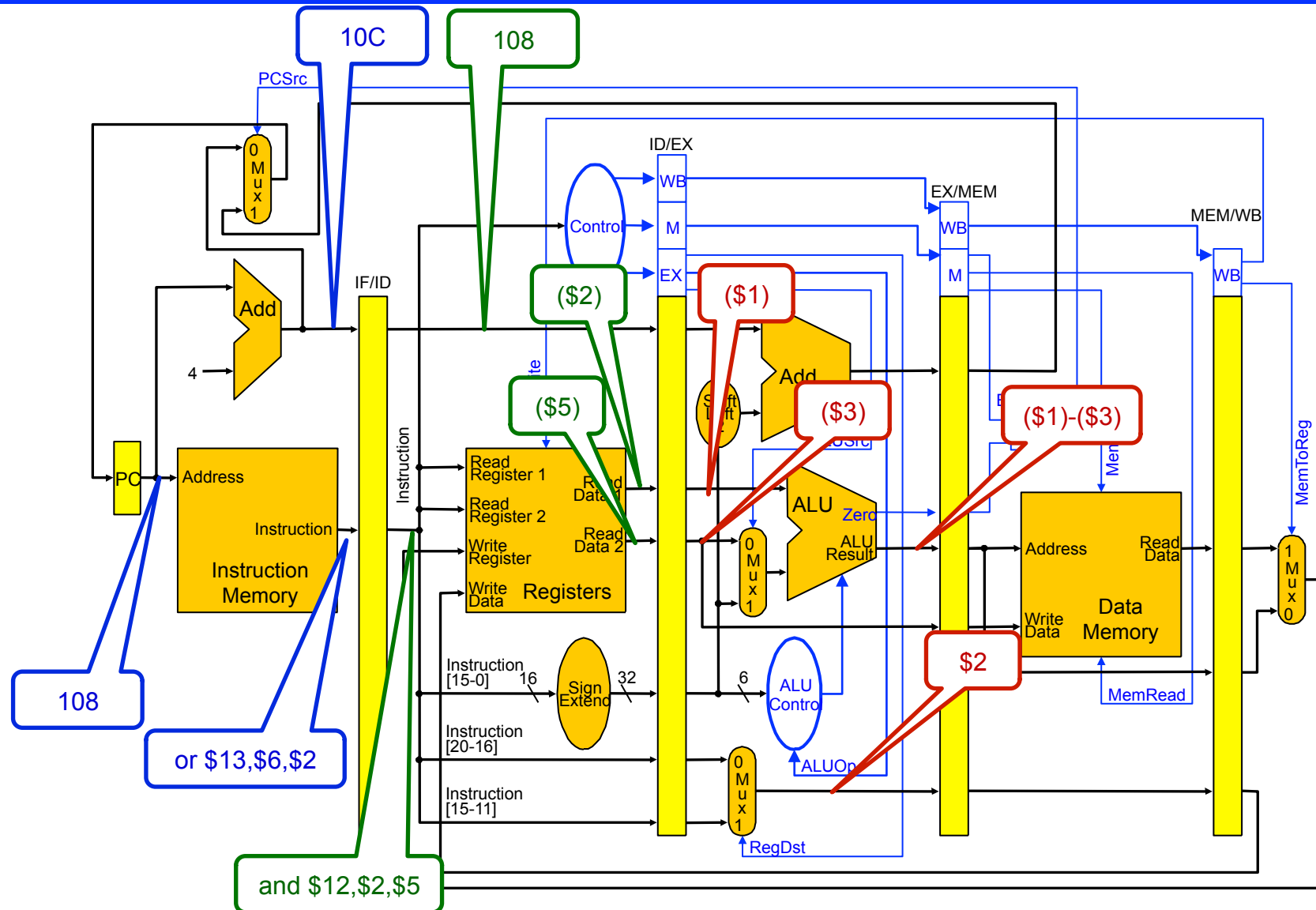
例題(8)：第3クロックサイクル

- 以下の命令列の実行過程をクロックサイクル1～5の5クロックサイクル分に関して示す。

100 ₁₆ :	sub \$2, \$1, \$3	# \$2 = \$1 - \$3
104 ₁₆ :	and \$12, \$2, \$5	# \$12 = \$2 AND \$5
108 ₁₆ :	or \$13, \$6, \$2	# \$13 = \$6 OR \$2
10C ₁₆ :	add \$14, \$2, \$2	# \$14 = \$2 + \$2
110 ₁₆ :	sw \$15, 100(\$2)	# Mem[\$2+100] = \$15



例題(9) : 第3クロックサイクル

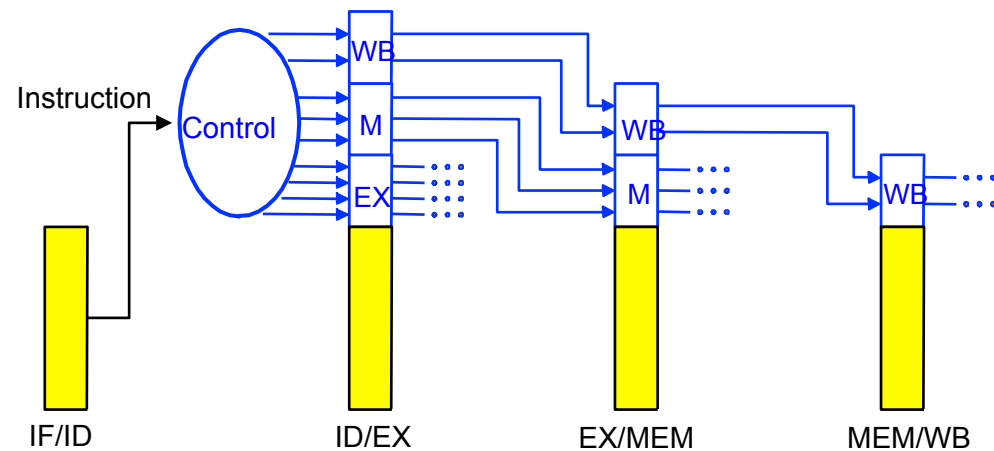


例題(10): 第3クロックサイクル

PC, パイプライン・レジスタ	フィールド	値	制御信号	値
PC	—	108		
IF/ID	IR	and \$12,\$2,\$5		
	PC4	108		
ID/EX	A	(\$1)	ALUOp	
	B	(\$3)	RegDst	
	SE	-	ALUSrc	
	WR1	\$2	Branch	
	WR2	\$3	MemRead	
	PC4*	104	MemWrite	
			RegWrite	
			MemToReg	
EX/MEM	ALUOut		Branch	
	BA		MemRead	
	Zero		MemWrite	
	B*		RegWrite	
	WR		MemToReg	
MEM/WB	MDR		RegWrite	
	ALUout*		MemToReg	
	WR*			

例題(11): 命令パイプラインの制御

命令	EXステージへの制御信号				MEMステージへの制御信号			WBステージへの制御信号	
	Reg Dst	ALU Op1	ALU Op0	ALU Src	Branch	Mem Read	Mem Write	Reg Write	MemTo Reg
R形式	1	1	0	0	0	0	0	1	0
lw	0	0	0	1	0	1	0	1	1
sw	x	0	0	1	0	0	1	0	x
beq	x	0	1	0	1	0	0	0	x



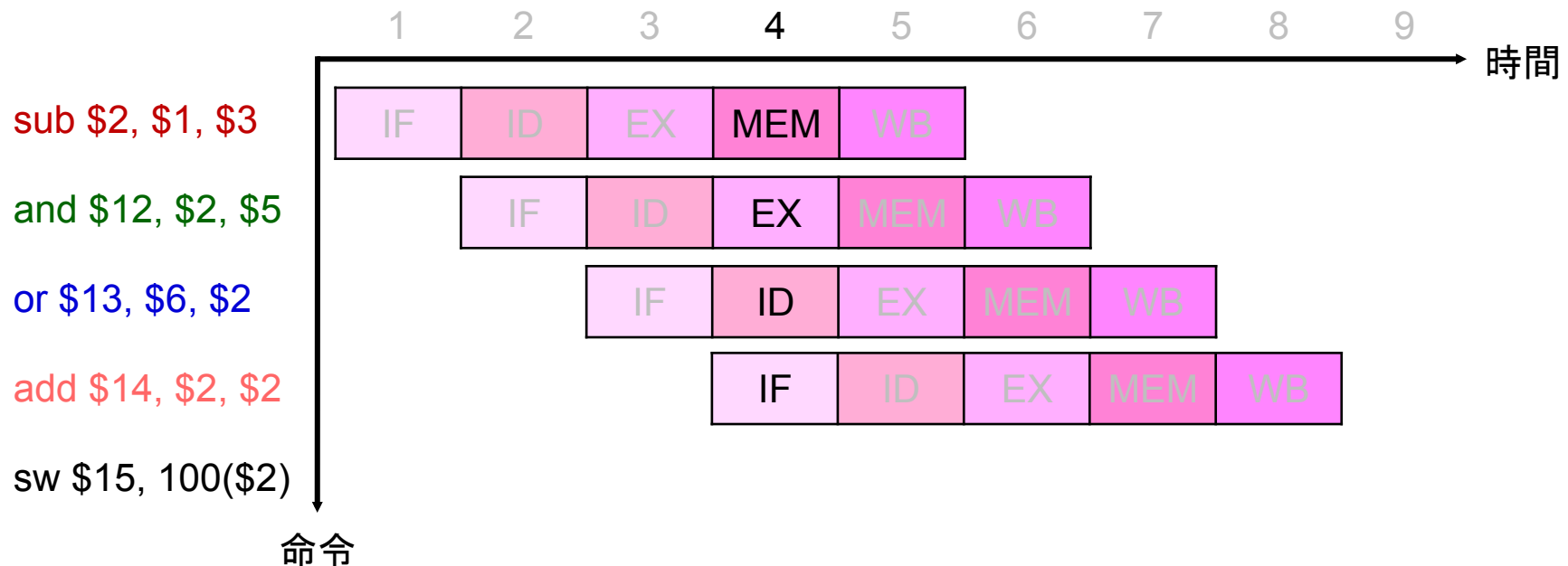
例題(12): 第3クロックサイクル

PC, パイプライン・レジスタ	フィールド	値	制御信号	値
PC	—	108		
IF/ID	IR	and \$12,\$2,\$5		
	PC4	108		
ID/EX	A	(\$1)	ALUOp	10
	B	(\$3)	RegDst	1
	SE	-	ALUSrc	0
	WR1	\$2	Branch	0
	WR2	\$3	MemRead	0
	PC4*	104	MemWrite	0
			RegWrite	1
			MemToReg	0
EX/MEM	ALUOut		Branch	
	BA		MemRead	
	Zero		MemWrite	
	B*		RegWrite	
	WR		MemToReg	
MEM/WB	MDR		RegWrite	
	ALUout*		MemToReg	
	WR*			

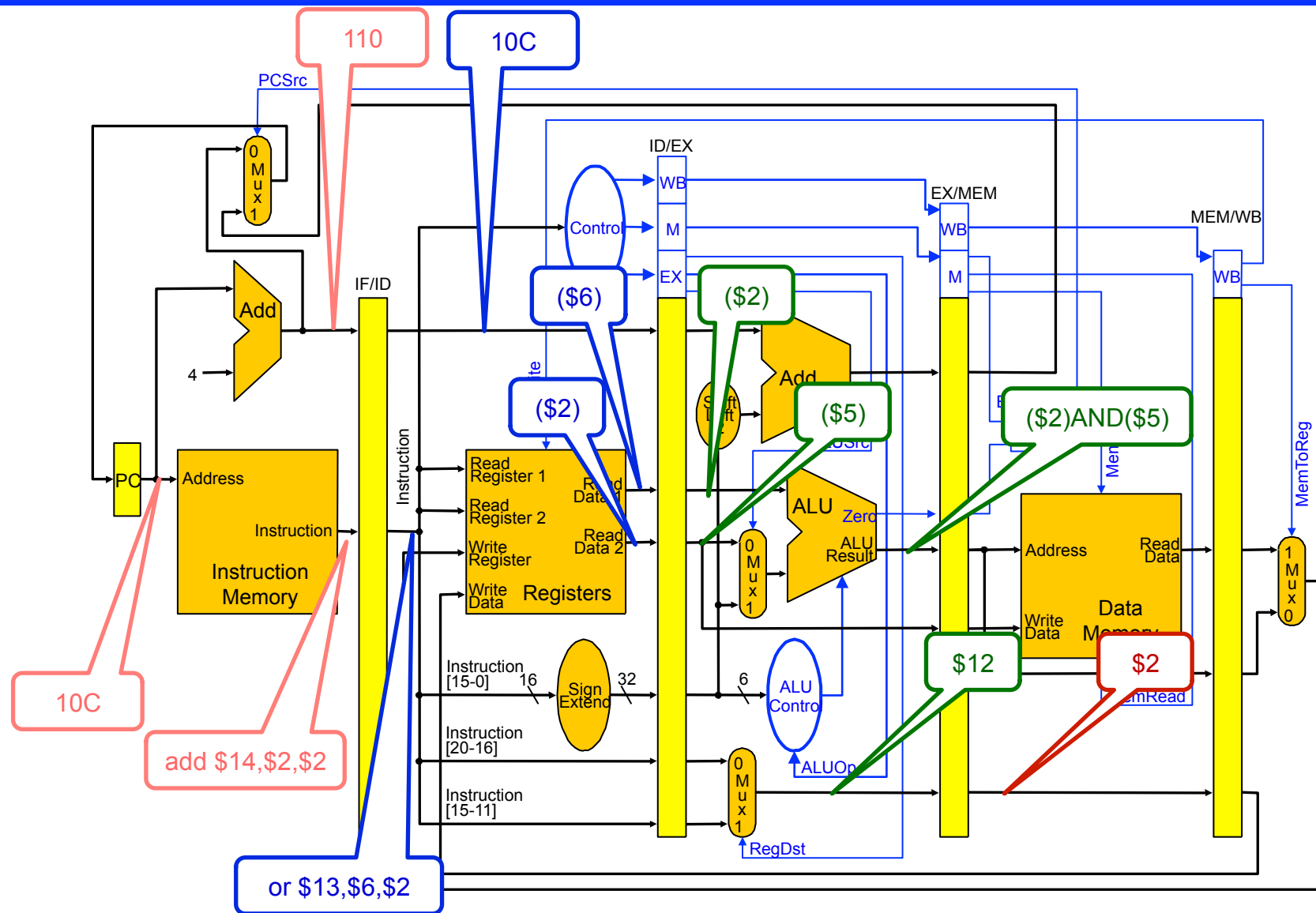
例題(13)：第4クロックサイクル

- 以下の命令列の実行過程をクロックサイクル1～5の5クロックサイクル分に関して示す。

100 ₁₆ :	sub \$2, \$1, \$3	# \$2 = \$1 - \$3
104 ₁₆ :	and \$12, \$2, \$5	# \$12 = \$2 AND \$5
108 ₁₆ :	or \$13, \$6, \$2	# \$13 = \$6 OR \$2
10C ₁₆ :	add \$14, \$2, \$2	# \$14 = \$2 + \$2
110 ₁₆ :	sw \$15, 100(\$2)	# Mem[\$2+100] = \$15



例題(14): 第4クロックサイクル



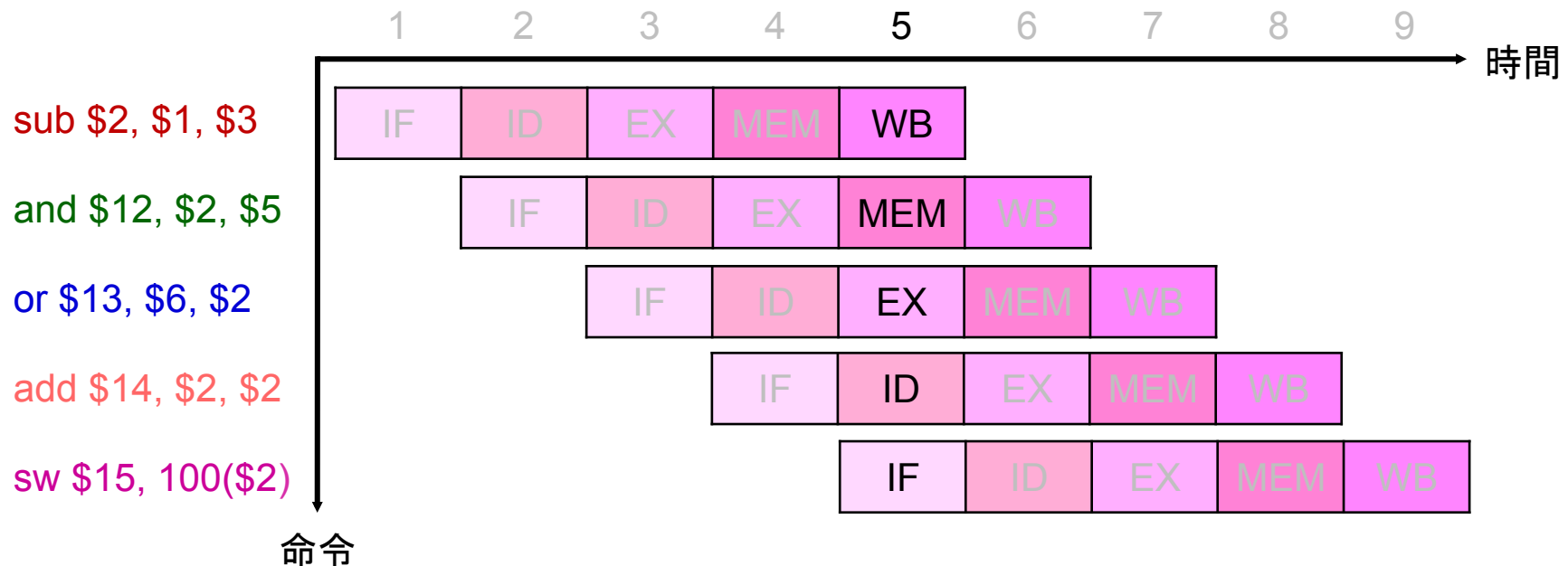
例題(15): 第4クロックサイクル

PC, パイプライン・レジスタ	フィールド	値	制御信号	値
PC	—	10C		
IF/ID	IR	or \$13,\$6,\$2		
	PC4	10C		
ID/EX	A	(\$2)	ALUOp	10
	B	(\$5)	RegDst	1
	SE	-	ALUSrc	0
	WR1	\$12	Branch	0
	WR2	\$5	MemRead	0
	PC4*	108	MemWrite	0
			RegWrite	1
			MemToReg	0
EX/MEM	ALUOut	(\$1)-(\$3)	Branch	0
	BA	-	MemRead	0
	Zero	?	MemWrite	0
	B*	(\$3)	RegWrite	1
	WR	\$2	MemToReg	0
MEM/WB	MDR		RegWrite	
	ALUout*		MemToReg	
	WR*			

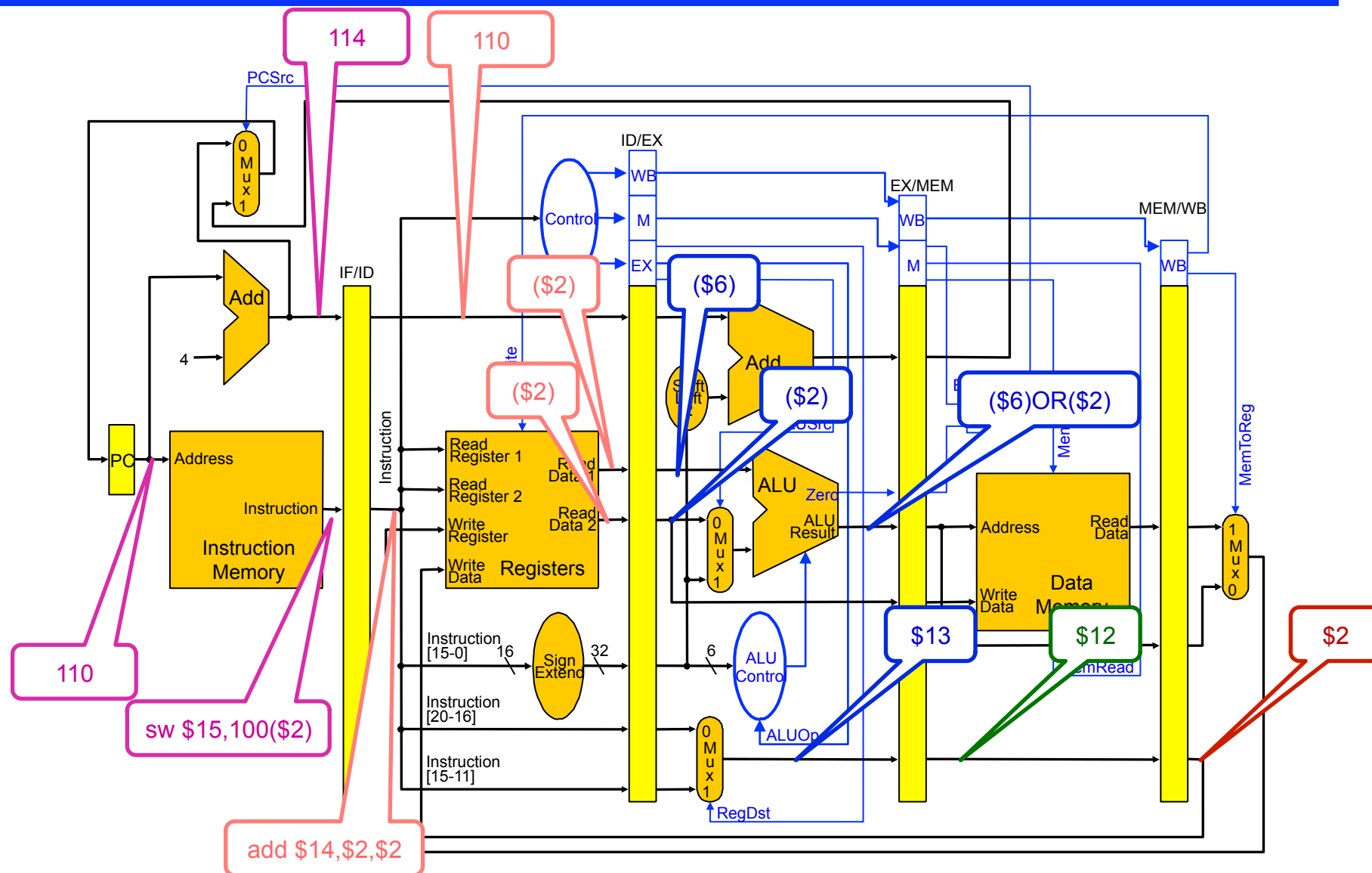
例題(16): 第5クロックサイクル

- 以下の命令列の実行過程をクロックサイクル1~5の5クロックサイクル分に関して示す。

100 ₁₆ :	sub \$2, \$1, \$3	# \$2 = \$1 - \$3
104 ₁₆ :	and \$12, \$2, \$5	# \$12 = \$2 AND \$5
108 ₁₆ :	or \$13, \$6, \$2	# \$13 = \$6 OR \$2
10C ₁₆ :	add \$14, \$2, \$2	# \$14 = \$2 + \$2
110 ₁₆ :	sw \$15, 100(\$2)	# Mem[\$2+100] = \$15



例題(17): 第5クロックサイクル



例題(18): 第5クロックサイクル

PC, パイプライン・レジスタ	フィールド	値	制御信号	値
PC	—	110		
IF/ID	IR	add \$14,\$2,\$2		
	PC4	110		
ID/EX	A	(\$6)	ALUOp	10
	B	(\$2)	RegDst	1
	SE	-	ALUSrc	0
	WR1	\$13	Branch	0
	WR2	\$2	MemRead	0
	PC4*	10C	MemWrite	0
			RegWrite	1
			MemToReg	0
EX/MEM	ALUOut	(\$2) AND (\$5)	Branch	0
	BA	-	MemRead	0
	Zero	?	MemWrite	0
	B*	(\$5)	RegWrite	1
	WR	\$12	MemToReg	0
MEM/WB	MDR	-	RegWrite	1
	ALUout*	(\$1)-(\$3)	MemToReg	0
	WR*	\$2		