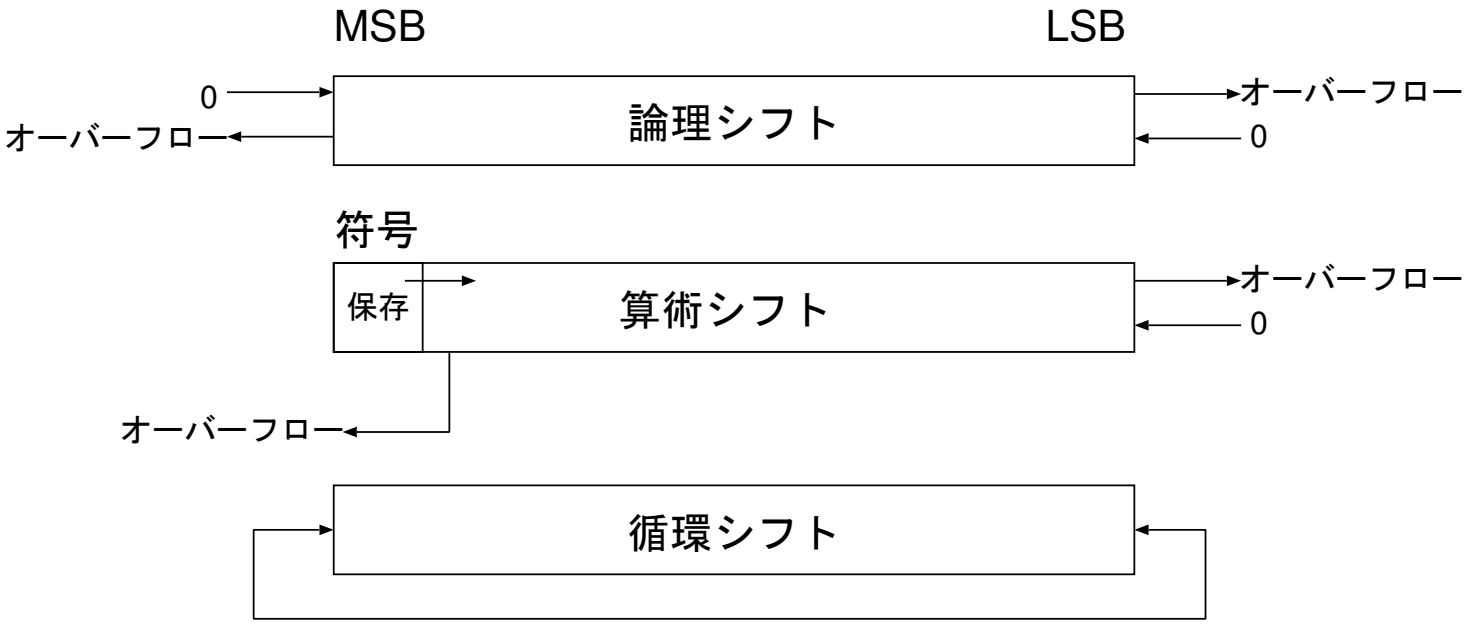
シフト命令

ビット列を左右にずらす.

論理シフト そのまま何も考えずに, 左右にシフトする. 左右からは0が入力される.

算術シフト 符号ビット(MSB)を保ったまま左右にシフトする. 左シフト=2倍, 右シフト=1/2倍. 右シフトの場合は, 符号ビットがそのままシフトする.

循環シフト MSBからあふれたビットはLSBに, LSBからあふれたビットはMSBに循環する.



教科書P61 図2.31より

シフトの実例(100, −100を右に3ビットシフト)

	2進数表記								符号付の10進数表記
元の数	0	1	1	0	0	1	0	0	100 (64+32+4)
算術シフト									
論理シフト									
循環シフト									

	2進数表記								符号付の10進数表記
元の数									−100
算術シフト									
論理シフト									
循環シフト									

計算機工学

第５章 制御アーキテクチャ

教科書 コンピュータアーキテクチャの基礎，柴山潔先生著(京都工芸繊維大学)

参考書 コンピュータの構成と設計，パターソン＆ヘネシー

ハードとソフト

ハードウエア (hardware) トランジスタなどからなる回路で構成されており，その機能が変更できないもの．

ソフトウエア (software) ハードウエアを制御するためのプログラム．

ファームウエア (firmware) ソフトウエアであるが，めったに書き換えないもの．hardwareより柔らかく，softwareより固い．

プロセッサに命令を実行させるための制御方式

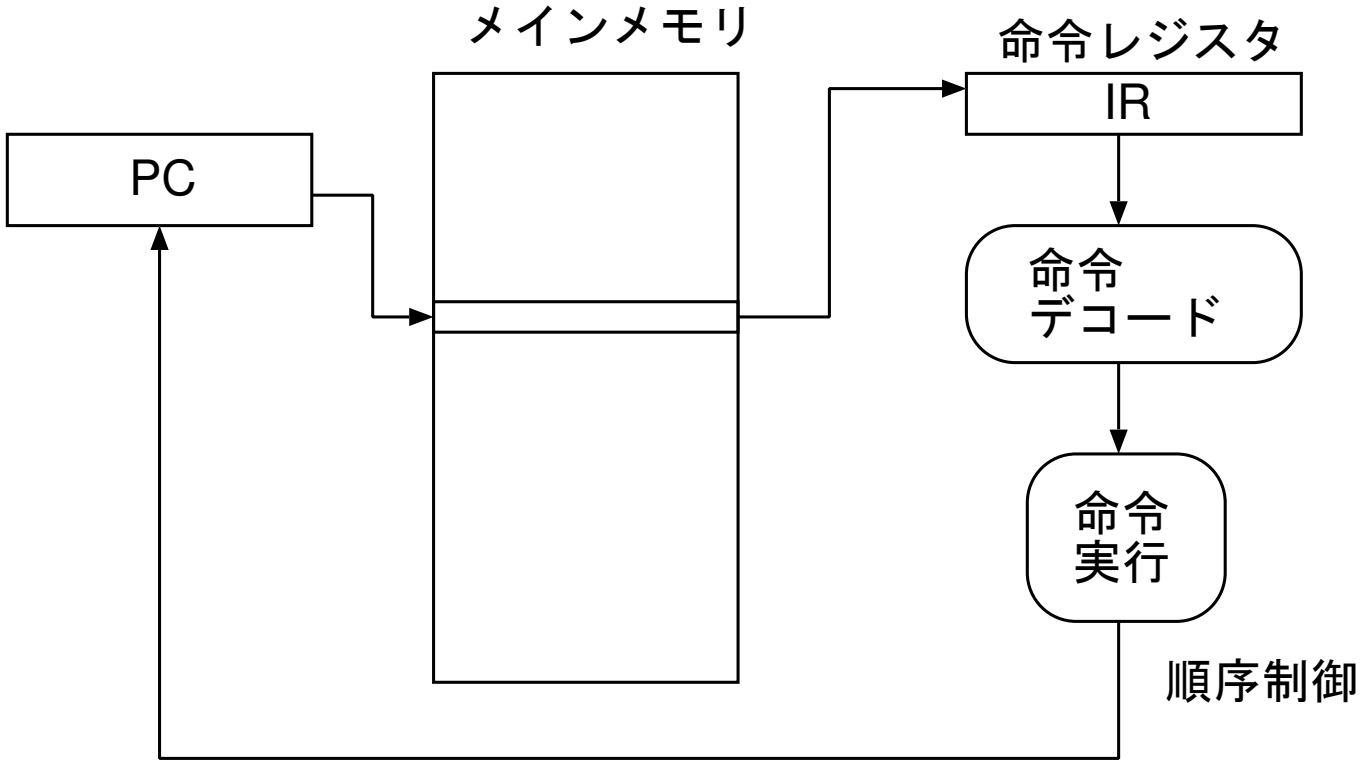
配線制御 論理回路によるハードウエアで制御を行なう．wired logic．布線論理．通常は1サイクルで行なう．RISC型の基本実行方式.

マイクロプログラム方式 1命令を複数のサブプログラム(マイクロプログラム)に分解して，複数サイクルにわたって実行する．CISC型の基本実行方式.

- 配線制御は，ハードウエアで実行，高速．ただし，ハードウエアコスト大．RISCは命令を限定してハードウエアのコストを抑える.
- マイクロプログラム制御は，ソフトウエア(プログラム)の量を少なくできる．少数の命令を組み合わせて複雑なことができる.

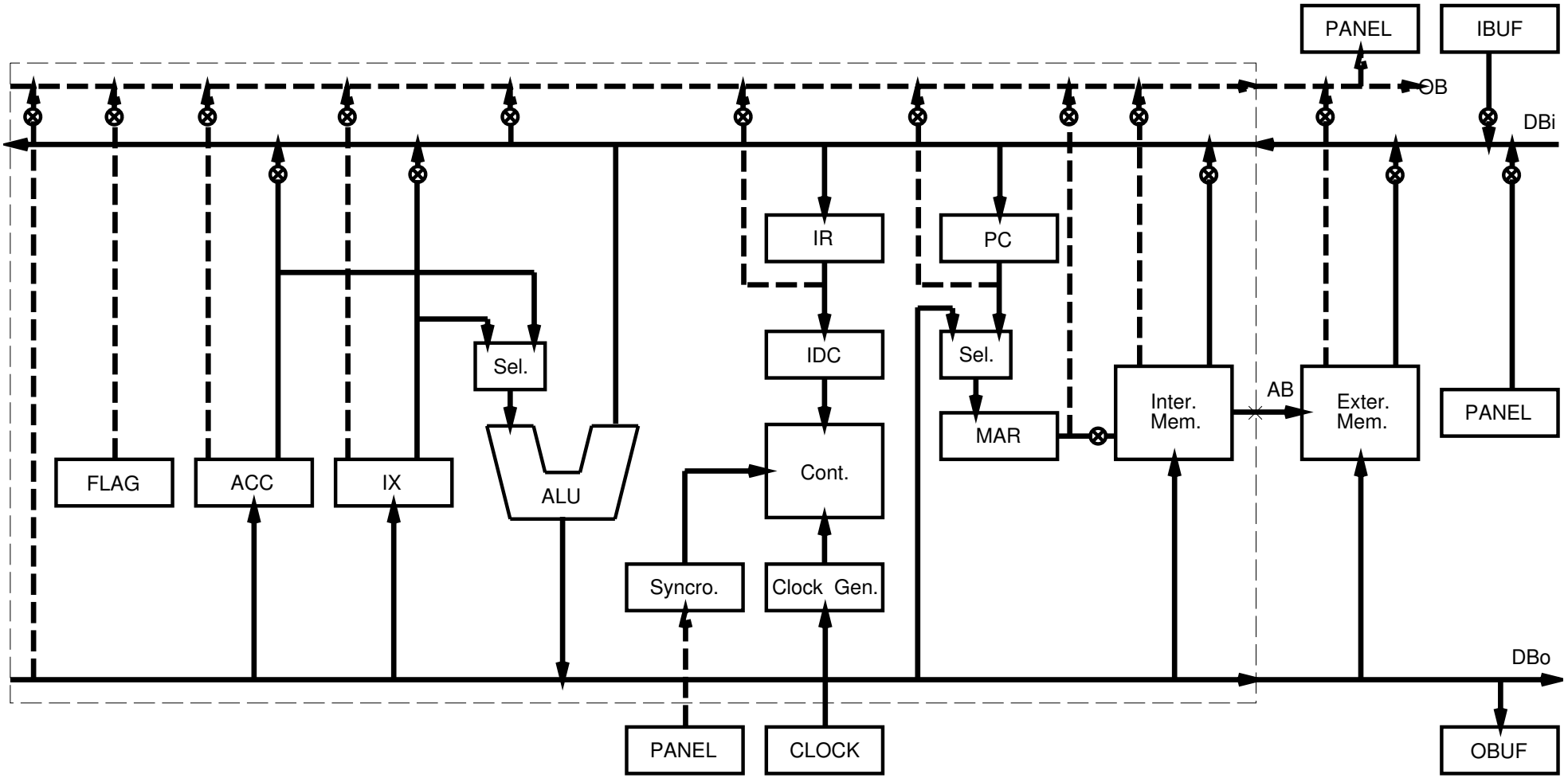
マシン命令実行サイクル

フェッチ 命令をメモリから取り出す
デコード 命令を解読(デコード)する.
実行 マシン命令機能を実行する
順序制御 次命令のアドレスを決定.



kuechip2のブロック図

８ビットのCISC型計算機



8ビットマイクロプロセッサ(Kuechip2)の命令コード表

略記号	命令コード(1語目)								B'	命令機能の概略	
NOP	0	0	0	0	0	-	-	-	×	No OPeration	
HLT	0	0	0	0	1	-	-	-	×	HaLT	停止
	0	1	0	1	-	-	-	-	×		未使用(HLT)
OUT	0	0	0	1	0	-	-	-	×	OUTput	$(ACC) \rightarrow OBUF$
IN	0	0	0	1	1	-	-	-	×	INput	$(IBUF) \rightarrow ACC$
RCF	0	0	1	0	0	-	-	-	×	Reset CF	$0 \rightarrow CF$
SCF	0	0	1	0	1	-	-	-	×	Set CF	$1 \rightarrow CF$
Bcc	0	0	1	1	cc				◎	Branch cc	条件が成立すれば$B' \rightarrow PC$
Ssm	0	1	0	0	A	0	sm		×	Shift sm	$(A) \rightarrow$ shift , rotate $\rightarrow A$
Rsm	0	1	0	0	A	1	sm		×	Rotate sm	はみ出したビット$\rightarrow CF$
LD	0	1	1	0	A		B		○	LoaD	$(B) \rightarrow A$
ST	0	1	1	1	A		B		◎	STore	$(A) \rightarrow B$
SBC	1	0	0	0	A		B		○	SuB with Carry	$(A) - (B) - CF \rightarrow A$
ADC	1	0	0	1	A		B		○	ADd with Carry	$(A) + (B) + CF \rightarrow A$
SUB	1	0	1	0	A		B		○	SUBtract	$(A) - (B) \rightarrow A$
ADD	1	0	1	1	A		B		○	ADD	$(A) + (B) \rightarrow A$
EOR	1	1	0	0	A		B		○	Exclusive OR	$(A) \oplus (B) \rightarrow A$
OR	1	1	0	1	A		B		○	OR	$(A) \vee (B) \rightarrow A$
AND	1	1	1	0	A		B		○	AND	$(A) \wedge (B) \rightarrow A$
CMP	1	1	1	1	A		B		○	CoMPare	$(A) - (B)$

- オペランドは2アドレス(DST兼SRC:A, SRC:B)

8ビットマイクロプロセッサ(Kuechip2)の命令コード表(2)

A ＝ 0 : ACC	B ＝ 000 : ACC (アキュムレータレジスタ)
A ＝ 1 : IX	B ＝ 001 : IX (インデックスレジスタ)
	B ＝ 01− : d(2語目) (即値)
B'(2語目)	B ＝ 100 : 絶対 (プログラム領域) (B' : アドレス)
× : 不用	B ＝ 101 : 絶対 (データ領域) (B' : アドレス)
○ : 不用 or 必要	B ＝ 110 : 修飾 (プログラム領域) (B'＋(IX) : アドレス)
◎ : 必要	B ＝ 111 : 修飾 (データ領域) (B'＋(IX) : アドレス)

- Kuechip2は, 学生実験(実験A, 3回生前期)で使用する教育用の8ビットマイクロプロセッサ.
- すべてのレジスタの値が外から見える.
- 「修飾」は相対アドレス
- CISCプロセッサなので, メモリ内のデータの演算が可能.

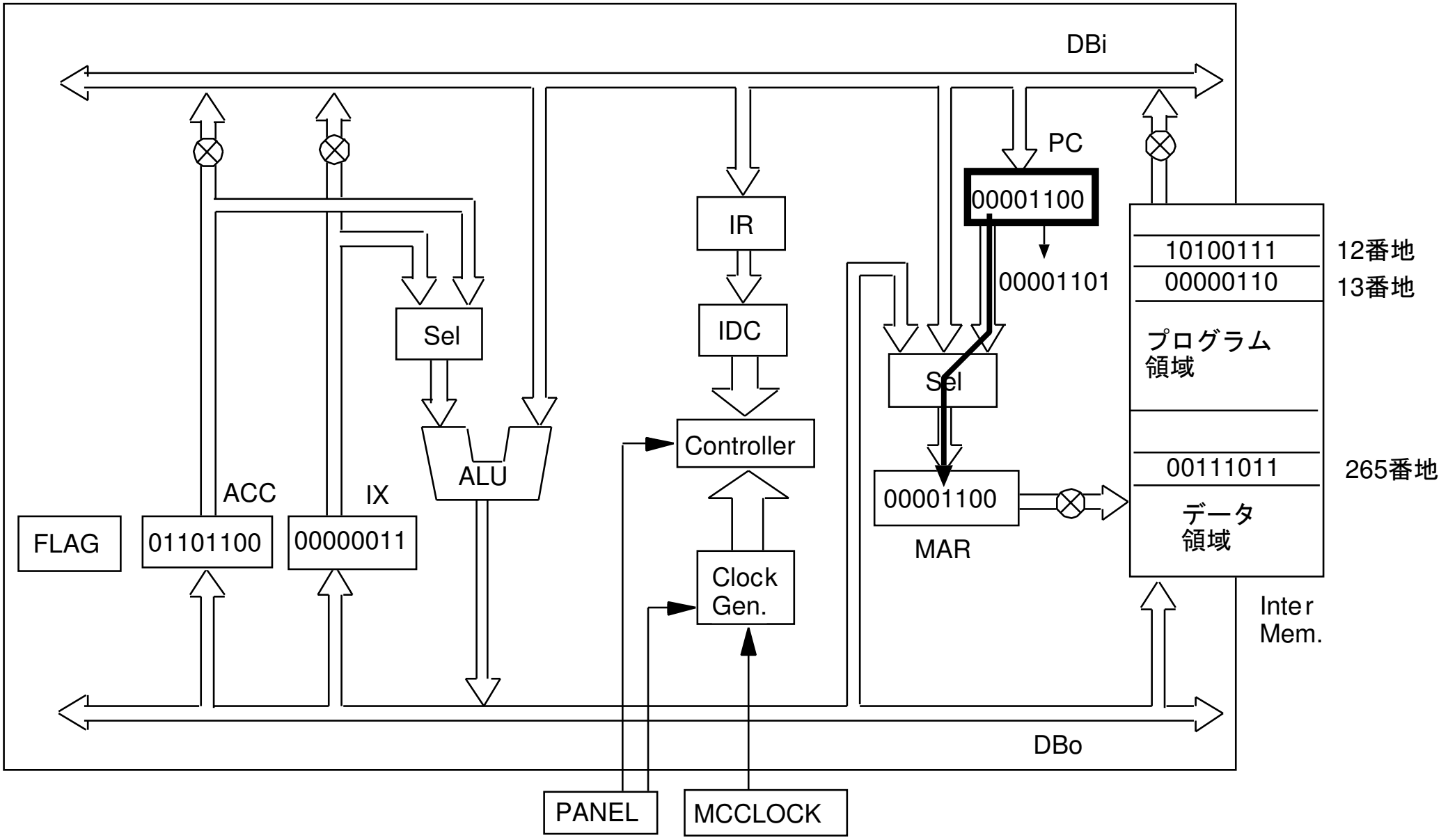
8ビットマイクロプロセッサ(Kuechip2)の命令コード表(3)

cc : Condition Code			
A	0 0 0 0	Always	常に成立
VF	1 0 0 0	on oVerFlow	桁あふれ　$VF = 1$
NZ	0 0 0 1	on Not Zero	$\neq 0$　$ZF = 0$
Z	1 0 0 1	on Zero	$= 0$　$ZF = 1$
ZP	0 0 1 0	on Zero or Positive	≥ 0　$NF = 0$
N	1 0 1 0	on Negative	< 0　$NF = 1$
P	0 0 1 1	on Positive	> 0　$(NF \vee ZF) = 0$
ZN	1 0 1 1	on Zero or Negative	≤ 0　$(NF \vee ZF) = 1$
NI	0 1 0 0	on No Input	$IBUF\_FLG\_IN = 0$
NO	1 1 0 0	on No Output	$OBUF\_FLG\_IN = 1$
NC	0 1 0 1	on Not Carry	$CF = 0$
C	1 1 0 1	on Carry	$CF = 1$
GE	0 1 1 0	on Greater than or Equal	≥ 0　$(VF \oplus NF) = 0$
LT	1 1 1 0	on Less Than	< 0　$(VF \oplus NF) = 1$
GT	0 1 1 1	on Greater Than	> 0　$((VF \oplus NF) \vee ZF) = 0$
LE	1 1 1 1	on Less than or Equal	≤ 0　$((VF \oplus NF) \vee ZF) = 1$

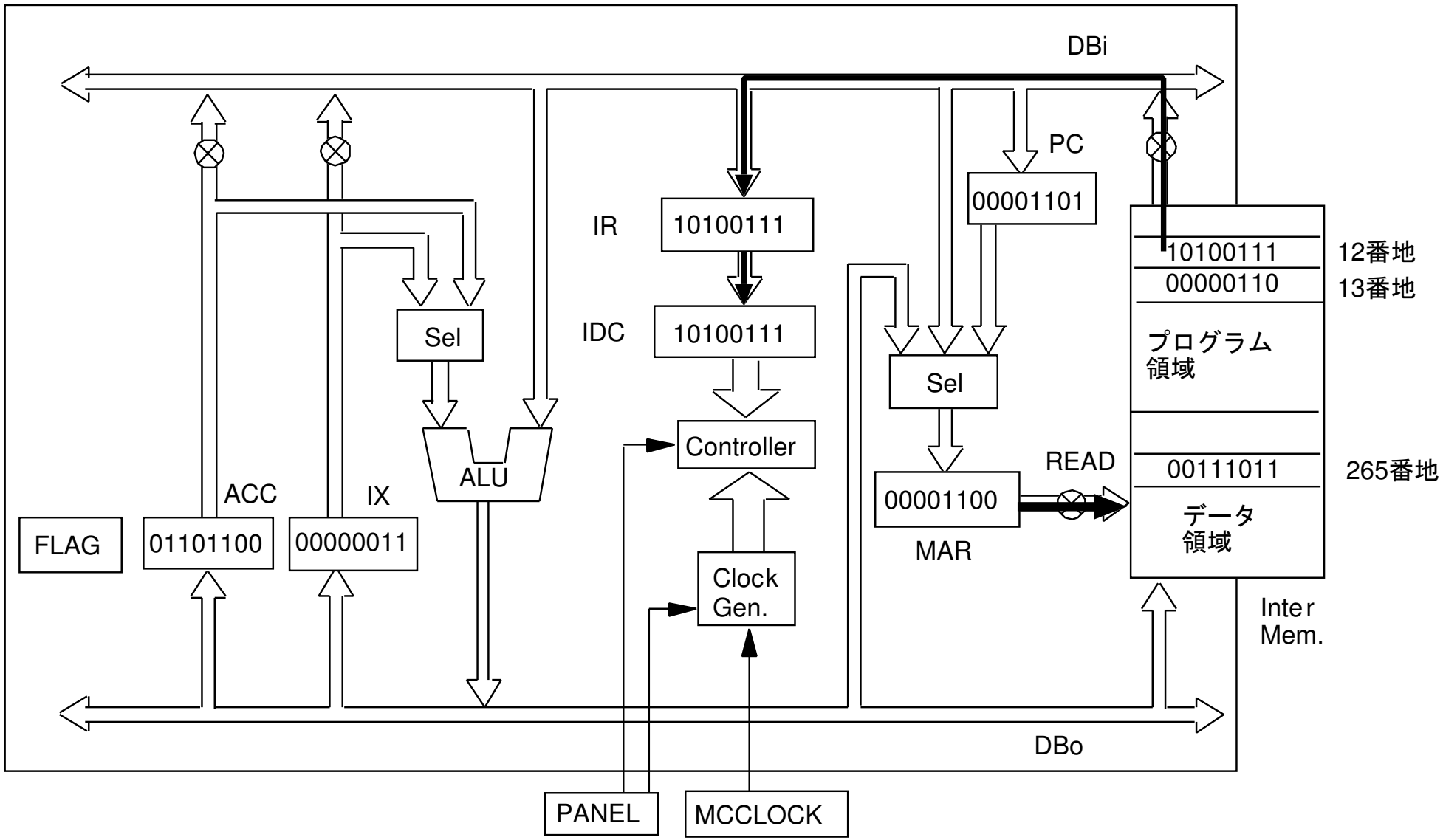
KUE-CHIP2の命令実行クロックフェーズ(配布資料)

instruction \ phase		P0	P1	P2	P3	P4
HLT		(PC) → MAR PC++	(Mem) → IR	HALT		
NOP				NO OPERATION		
OUT				(ACC) → OBUF	0 → OBUF_WE	
IN				(IBUF) → ACC 0 → IBUF_RE	0 → IBUF_FLG_CLR	
RCF				0 → CF		
SCF				1 → CF		
Bcc				(PC) → MAR PC++	STATUS CHECK (Mem) → PC (if condition satisfied)	
Ssm Rsm				TCF SET SHIFT A	NF,ZF,VF,CF SET	
LD	ACC IX			(B) → A		
	d			(PC) → MAR PC++	(Mem) → A	
	[d] (d)				(Mem) → MAR	(Mem) → A
	[IX+d] (IX+d)				(IX) (Mem) → ALU → MAR	
ST	[d] (d)			(PC) → MAR PC++	(Mem) → MAR	(A) → (Mem)
	[IX+d] (IX+d)				(IX) (Mem) → ALU → MAR	
SBC ADC SUB ADD EOR OR AND CMP	ACC IX			(A) (B) [(CF)] → ALU → A NF,ZF,VF [,CF] SET		
	d			(PC) → MAR PC++	(A) (B) [(CF)] → ALU → A NF,ZF,VF [,CF] SET	
	[d] (d)				(Mem) → MAR	(A) (Mem) [(CF)] → ALU → A NF,ZF,VF [,CF] SET
	[IX+d] (IX+d)				(IX) (Mem) → ALU → MAR	

動作フェーズP0(配布資料参照)

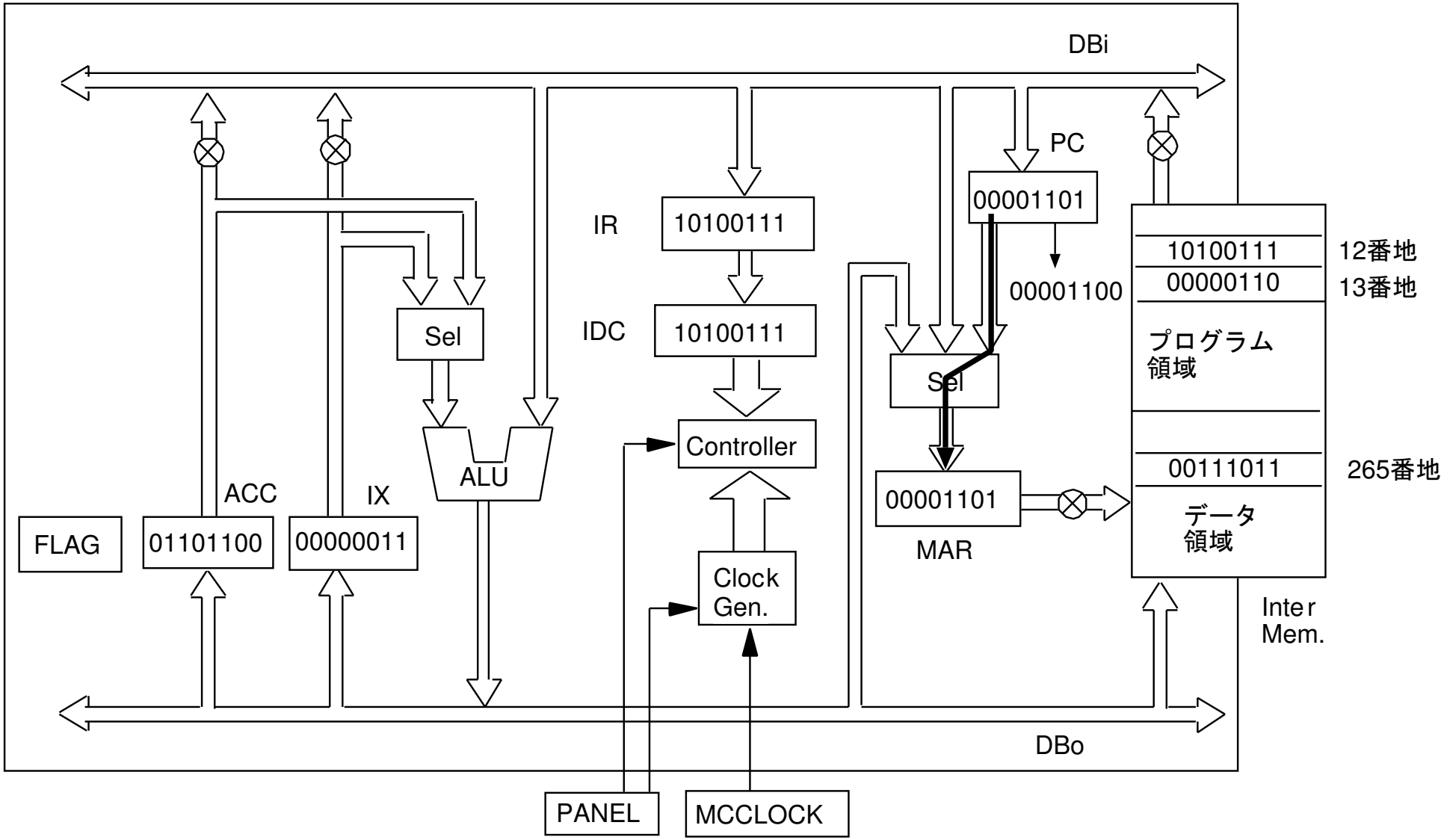


動作フェーズP1



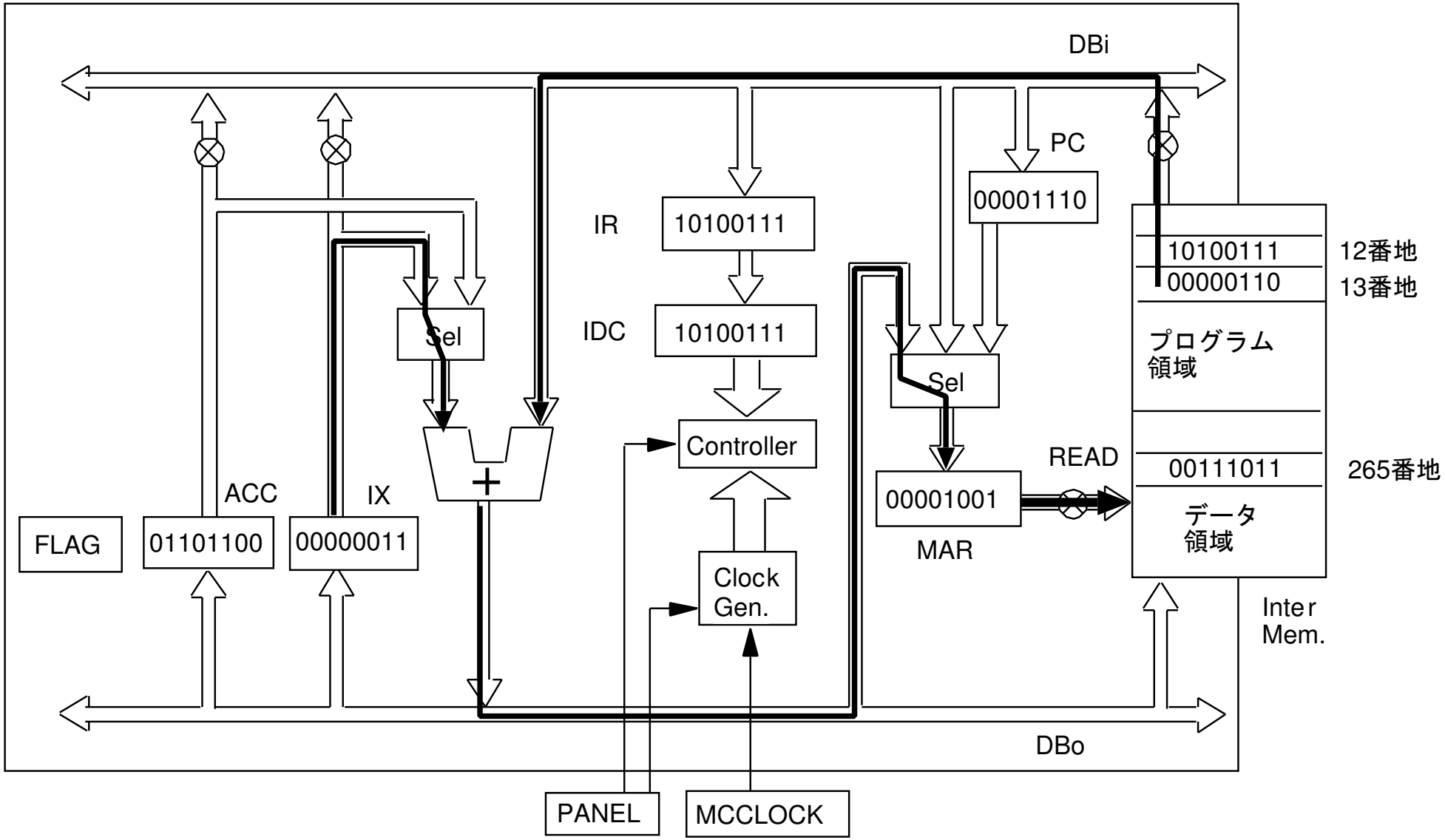
(b) フェーズP1

動作フェーズP2



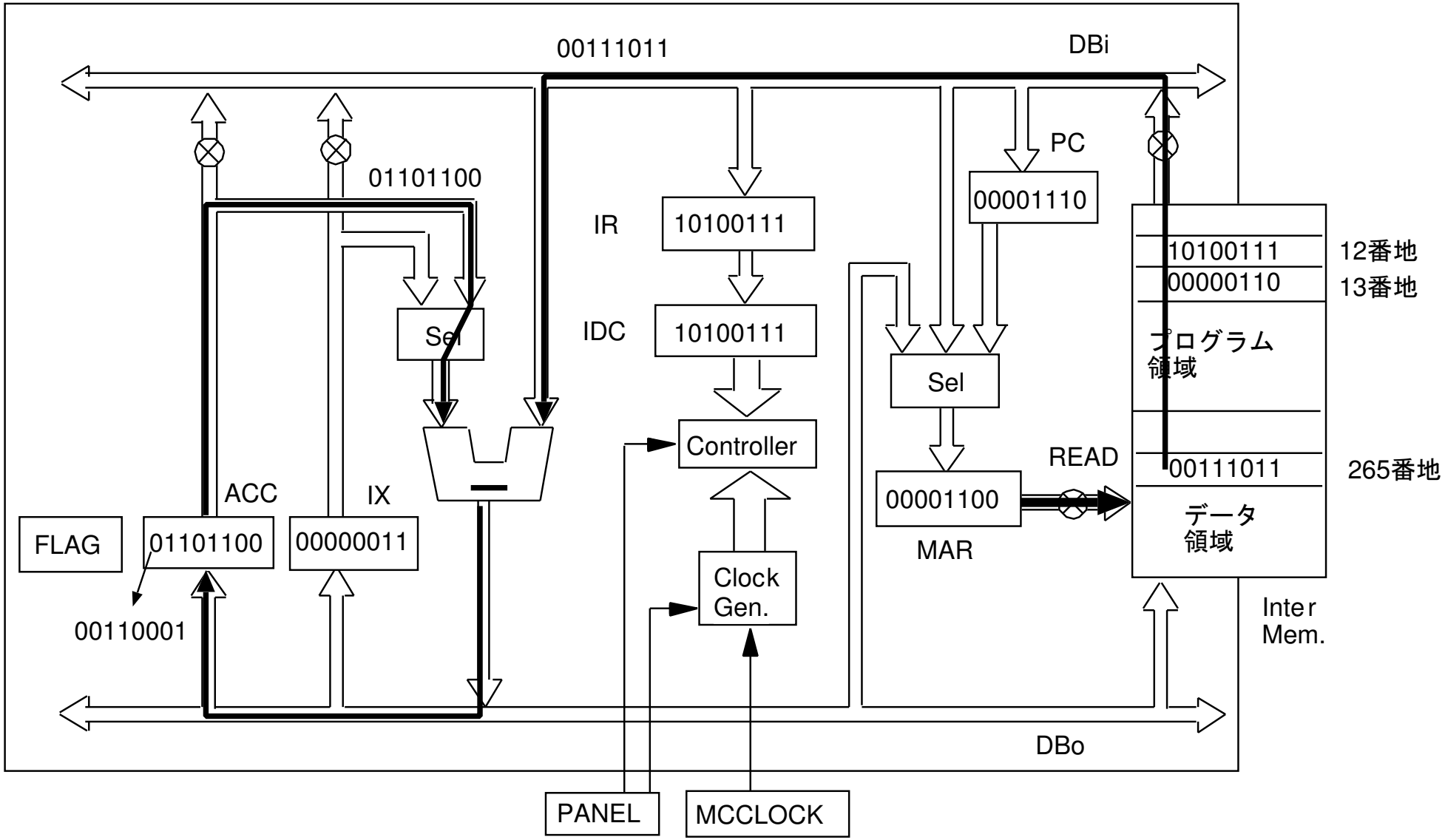
(a) フェーズP2

動作フェーズP3



(b) フェーズP3

動作フェーズP4



フェーズP4

命令デコードステージの制御順序

1. 命令形式の決定．Kuechip2では全ビット見ないと判断できない．

2. Kuechip2では，B'が必要な場合に，もう1ワードフェッチしてくる．

3. OPコードとオペランドを分離する．
   (a) 配線制御の場合は，命令をそのまま実行
   (b) マイクロプログラム制御の場合は，マイクロプログラムに分岐する．

命令実行順序制御

- 通常は，命令実行は，アドレスが増えていく方向．PCを命令の長さ分だけカウントアップする．
- 分岐命令では，分岐先にPCが変化する．
  - Kuechip2では，Bccが分岐命令．
  - 事前の演算によってセットされた，桁上げフラグCF，桁あふれフラグVF，ネガティブフラグNF，零フラグZFの値によって分岐するかしないかを決める．

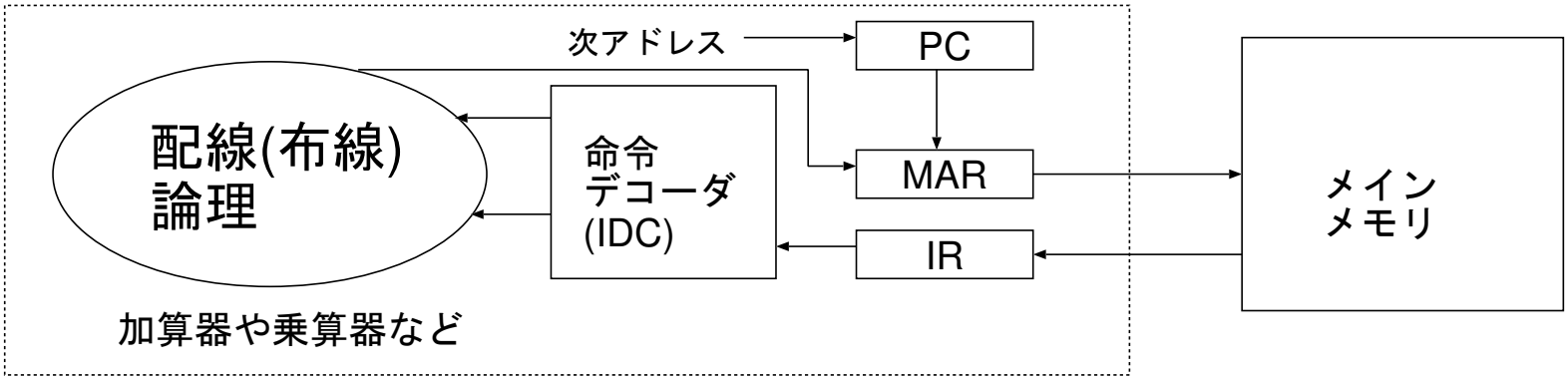
配線論理制御機構のハードウエア

プログラムカウンタ (PC) 実行中の命令アドレス．通常は命令が終了すると，＋1ワード分，値が変化する．

命令レジスタ (IR) 現在実行中の命令語を格納．PCの値に応じて，メインメモリから供給される．

命令デコーダ (Instruction Decoder: IDC) IRに保存されている命令語を解読して，ALUや各種レジスタへの制御信号を生成する．

メモリアドレスレジスタ (MAR) 現在アクセス中のメモリのアドレスを保存



教科書 P127 図5.3より

Kuechip2での命令コード例

IXに$(10)_{10}$=$(1010)_2$を代入(ロード)する

ACCにデータ領域のメモリの$(21)_{10}$=$(10101)_2$番地の値を代入する

ACCとIXを加算して，ACCに代入する.

ACCに，データ領域のメモリの$(10)_{10}$番地の値を加算する.

同期制御と非同期制御

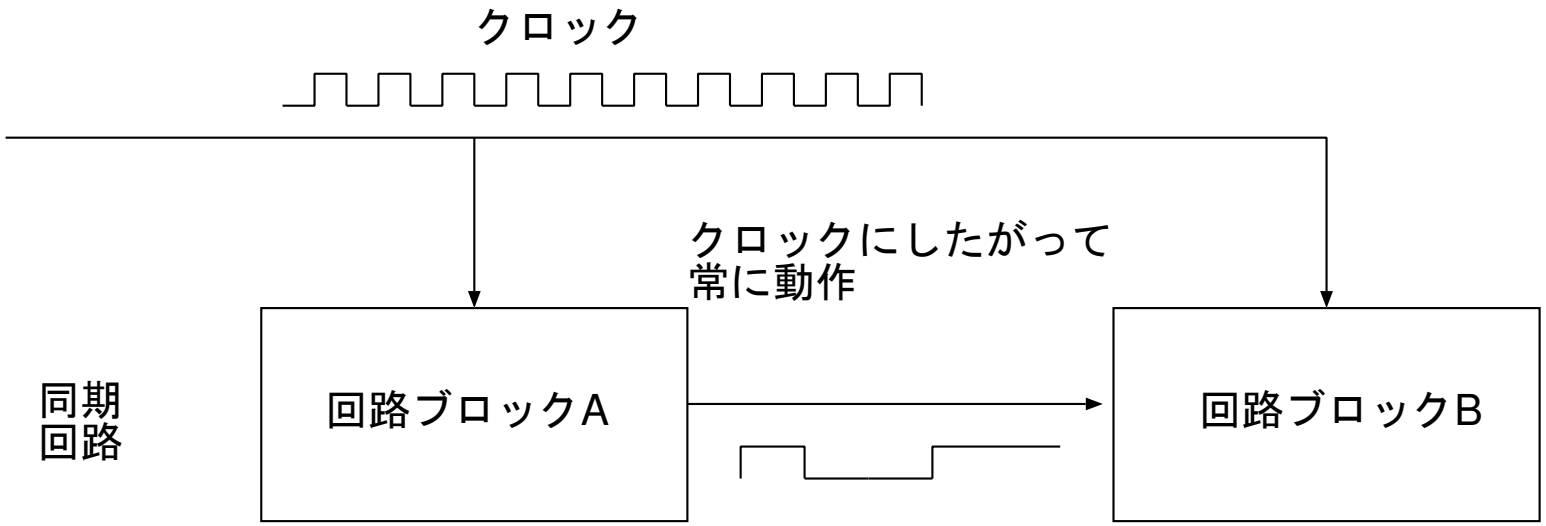
同期 (Synchronous) 制御 制御のタイミングをクロックに委ねる.

- クロックにより時間を量子化する.
- すべての制御/演算信号はクロックに同期する. (クロック信号: 一定の周期で発振している信号.) 「単一のクロック対制御/演算信号」という, 「1 対多数」のタイミングを見れば良いので, 設計が簡便. 自動設計に向いている.

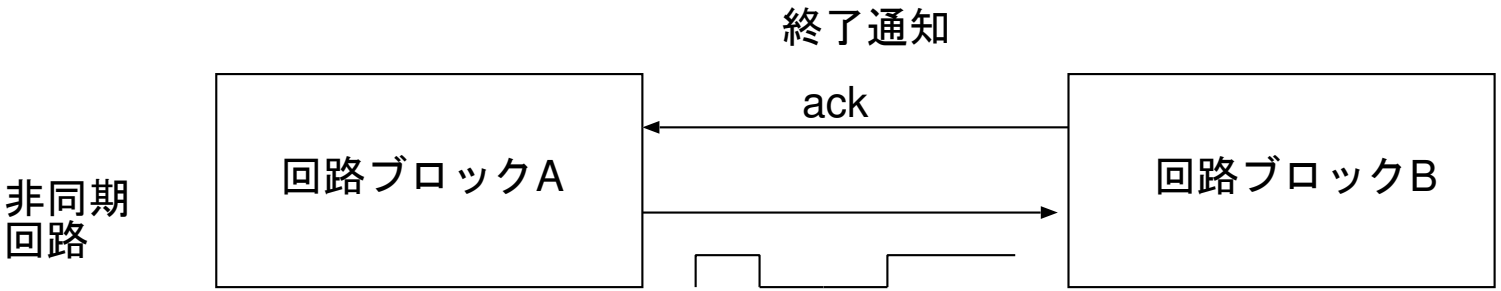
非同期 (Asyncronous) 制御 動いたら, 動作させる.

- 信号間のタイミング設計が難しい. 多対多
- 信号を受け取ったことを相手に伝える必要あり (ハンドシェイク). 頑張れば高速になるが, 設計が大変.

非同期回路と同期回路

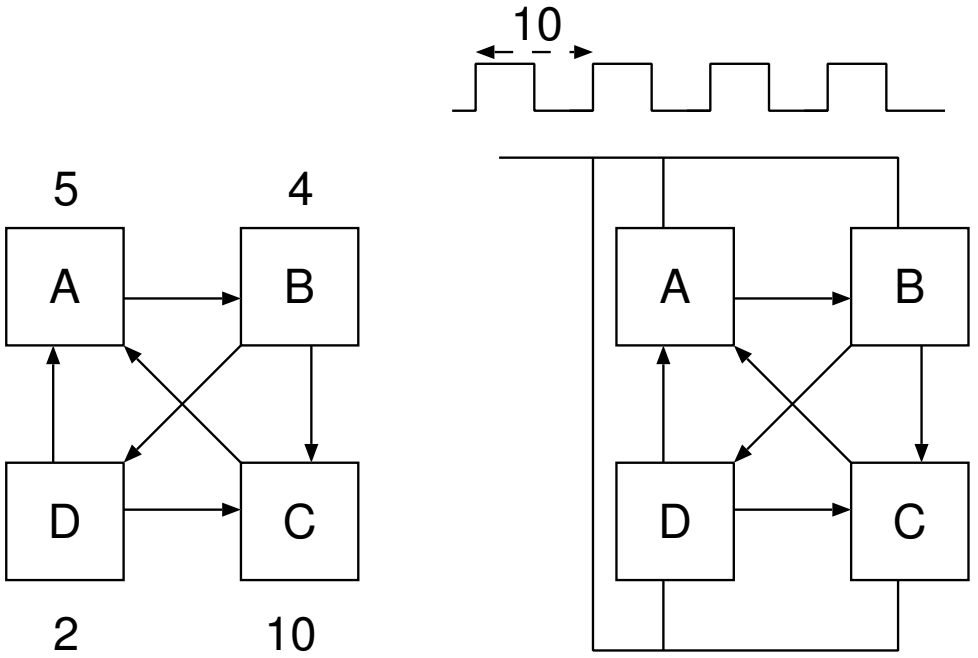


Aからクロックに同期して
Bに送るとBは必ず処理



BはAからの信号来ると
処理を開始

非同期回路と同期回路(2)



- 左は非同期回路．数字は動作速度．データは, いつやってくるか不明.
- 右は同期回路．もっとも遅いブロックにあわせてクロックを設定．データは必ず, クロックのあとに一定の遅延でやってくる.

マイクロプログラム制御

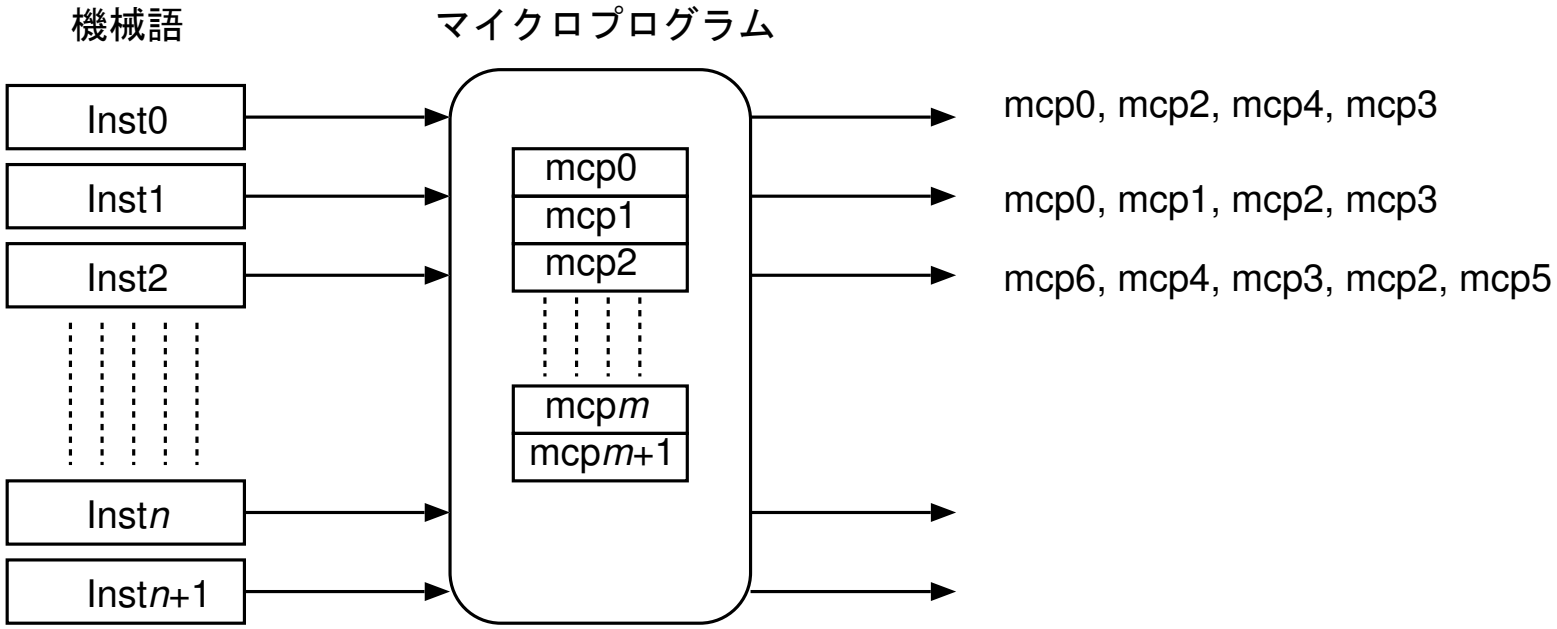
- 複雑な処理を行なう命令語を簡単な命令列に置き換える.
- 制御回路が簡単になる.

制御メモリ マイクロプログラムを格納するメモリ

マイクロ命令レジスタ マイクロ命令用の命令レジスタ

マイクロプログラムシーケンサ マイクロ命令の制御装置

マイクロプログラム制御



教科書 P131 図5.8より

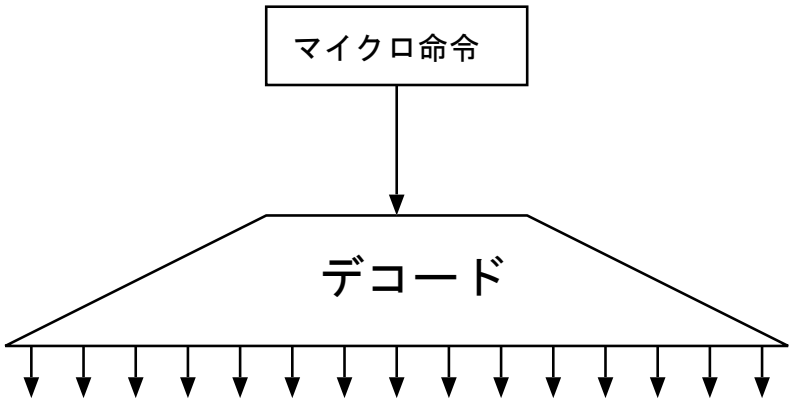
- マイクロプログラムを変更するだけで, プロセッサの性能が上がる (バイナリ互換).

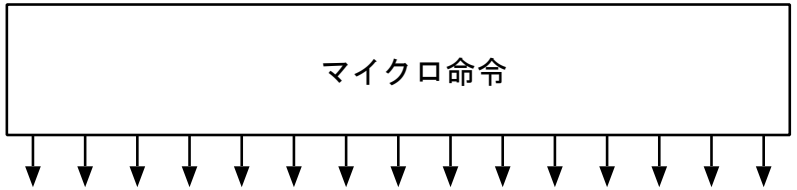
エミュレーション マイクロプログラムによるマシン命令セットの模擬実行 (狭義).

マイクロ命令形式

垂直型 マイクロ命令をデコードして，各ハードウエアの制御信号を作成.

水平型 マイクロ命令＝各ハードウエアの制御信号.





1ビットが1制御信号

教科書 P131 図5.8より

マイクロプログラム制御の活用

- マイクロプログラムを書き換え可能に．ROMではなくて, PROM (Programmable ROM) に.

ハードウエアのソフト化 マシン命令を複数のマイクロプログラムで.

ソフトウエアのハード化 複数のマシン命令をひとつのマシン命令に.

最新のエミュレーション技術

Intel社 Pentium 8086命令を内部で，RISC命令に変換.

Transmeta社 Crusoe, Efficeon 内部は，VLIW(命令長の長い並列プロセッサ).
8086命令をソフトウエアで，VLIW コードにリアルタイムに変換．小規模なハードウエアで8086互換プロセッサを実現.

http://journal.mycom.co.jp/news/2003/12/08/13.htmlより

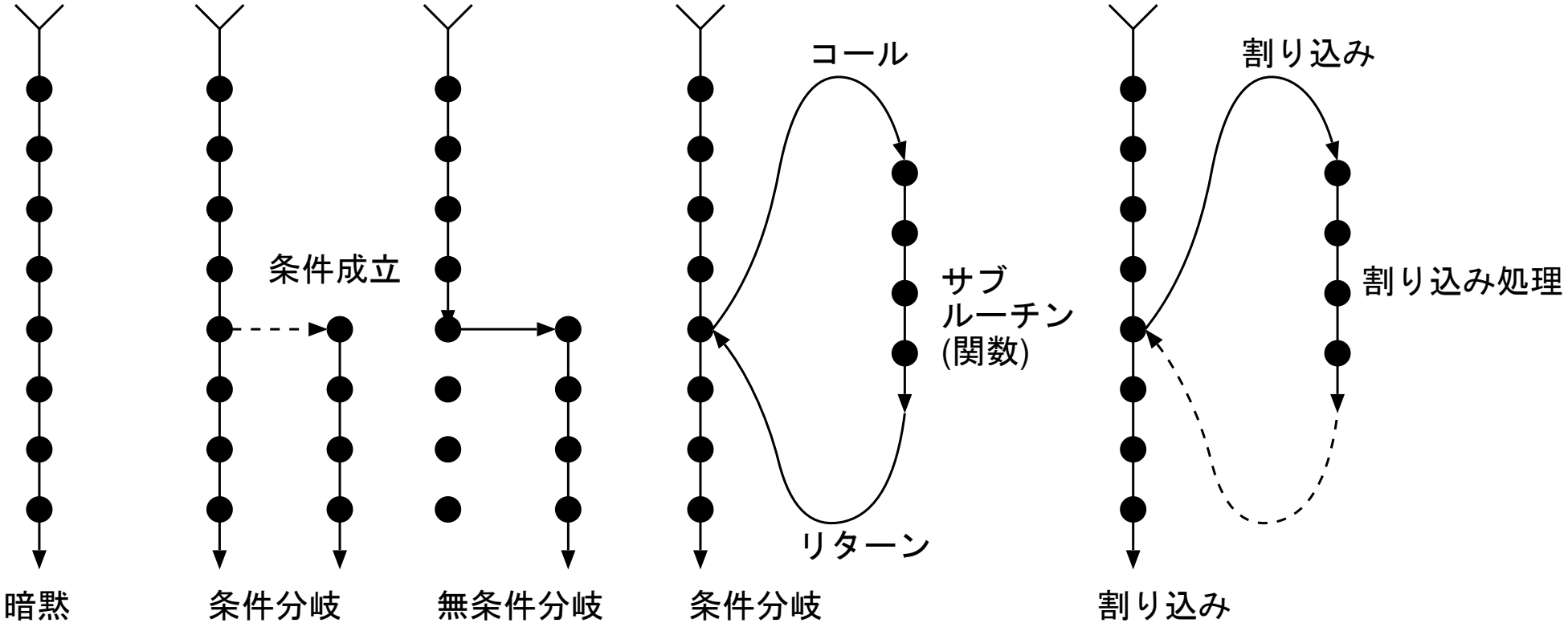
命令実行順序制御

- 命令はPCの値により決まる.
- 通常は, PCは現命令実行後, インクリメント(PC++)される. プログラムカウンタによる順序制御.
- 命令とデータが同じなら, プログラムの実行順序は不変.

シーケンサ(sequencer)（sequence: 順番に配列する）命令順の制御を行なうハードウエア群.

- PC, IR, 制御用レジスタ(ゼロレジスタ等), 割り込み処理用レジスタ, アドレス計算機構.

次アドレスの決定



教科書 P134 図5.9より

FIFOとスタック

FIFO: First-In First-Out 先入れ先出しメモリ.

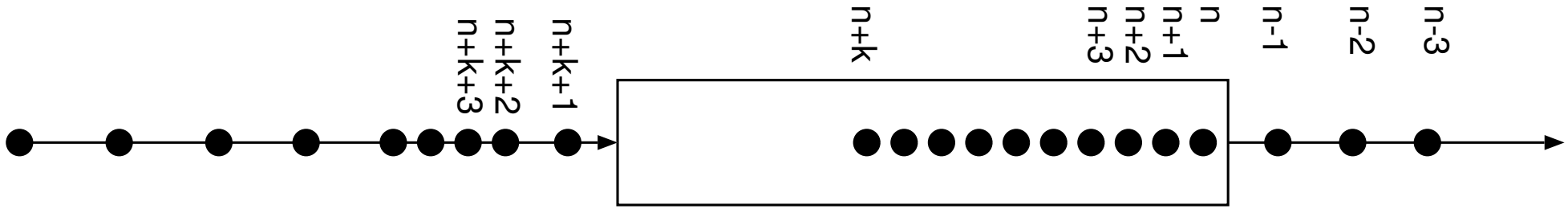
- 書き込んだ順に読み出される.
- 動画像や音声データを転送する時などのバッファとして利用される.

スタック: stack, Least-In First-Out 後入れ先出しメモリ.

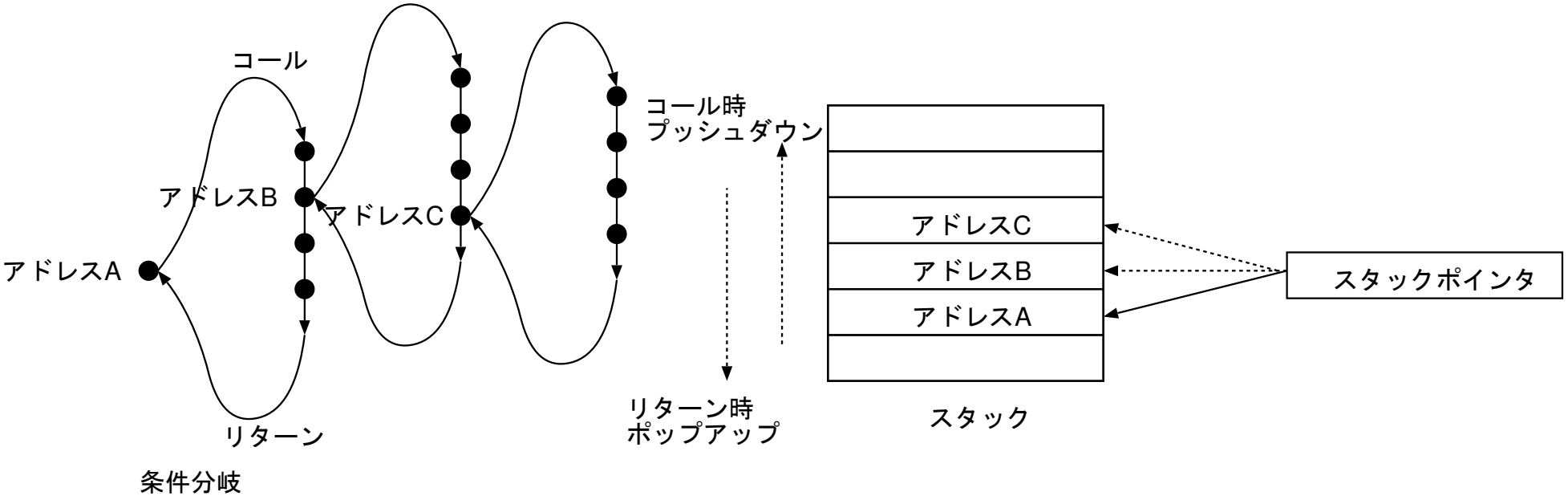
- 書き込まれた値は順に山に詰まれていき, 読出しはその山の上から行なわれる.

スタックポインタ: stack pointer 現在のスタックの場所を示すポインタ.

FIFOとスタック(2)



FIFO



スタック

教科書 P242 図7.9より

パイプライン処理

ベルトコンベアーによる流れ作業のように，一つの作業を手順毎に分けて，並列に行なうこと．

先行制御 次サイクルの命令を可能なら現サイクルで行なうこと．

命令先取り制御 命令フェッチ↑命令実行 という流れを作り，命令実行時に次命令をフェッチする．

命令パイプライン処理 1命令の実行を複数のステージに分け，ステージ毎に順に実行していく．

命令実行時間＝ステージ毎の処理時間×ステージ段数
単位時間あたりの命令実行数＝1/ステージ毎の処理時間

パイプライン処理

RISCプロセッサでは，通常は5段パイプライン．

命令フェッチ (Instruction Fetch, IF) メモリより命令取り出し

命令デコード，レジスタフェッチ (Instrunction Decode ID) 命令解析とレジスタファイルから値を読み出す

実行 (Execution, EX) 実行して，レジスタに値を格納．分岐の場合はPC書き換え

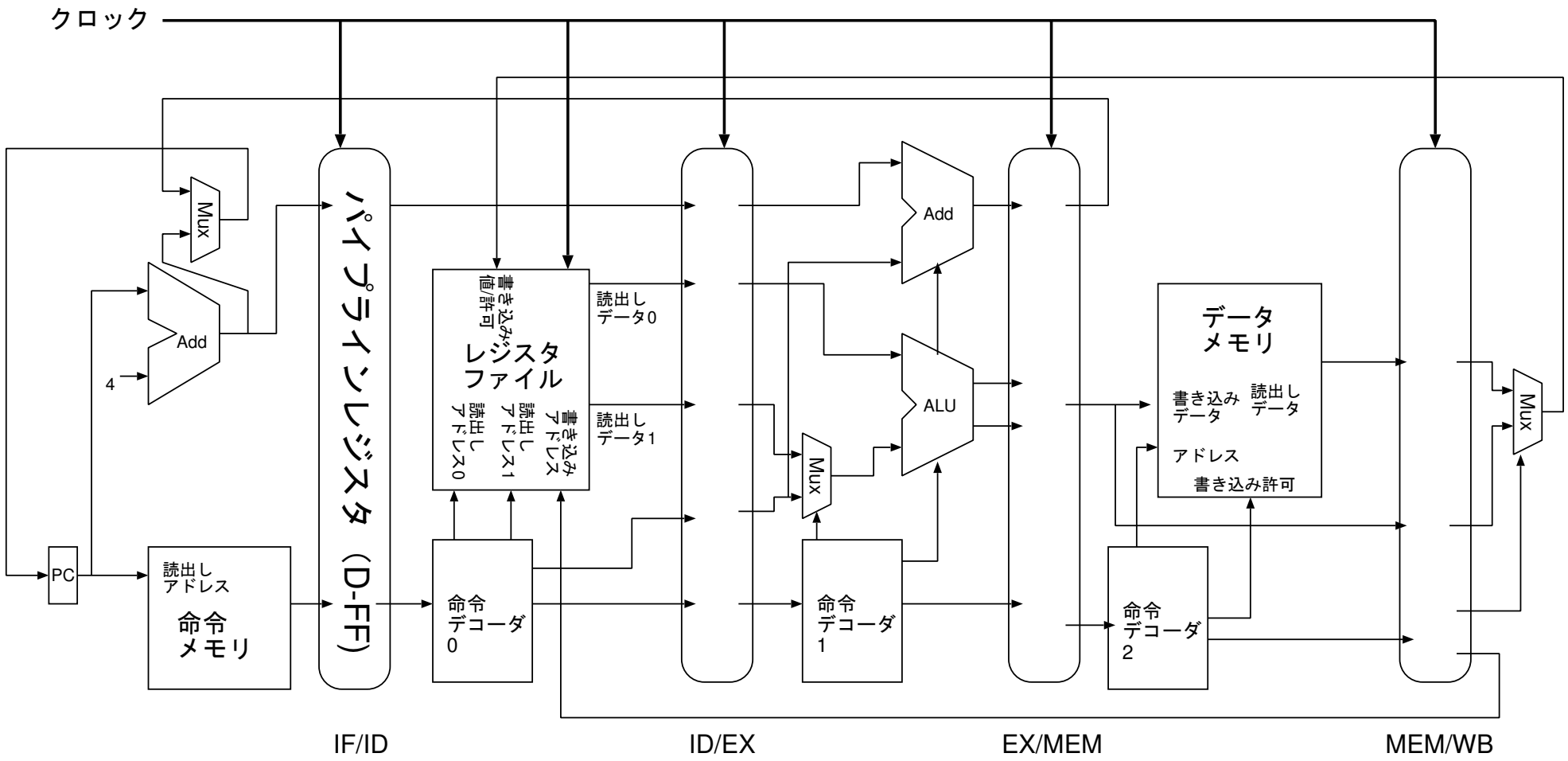
メモリアクセス (Memory access, MEM) メモリに書き込む，または読み出す．

書込 (WriteBack, WB) メモリからのデータをレジスタファイルに書き込む

パイプライン処理

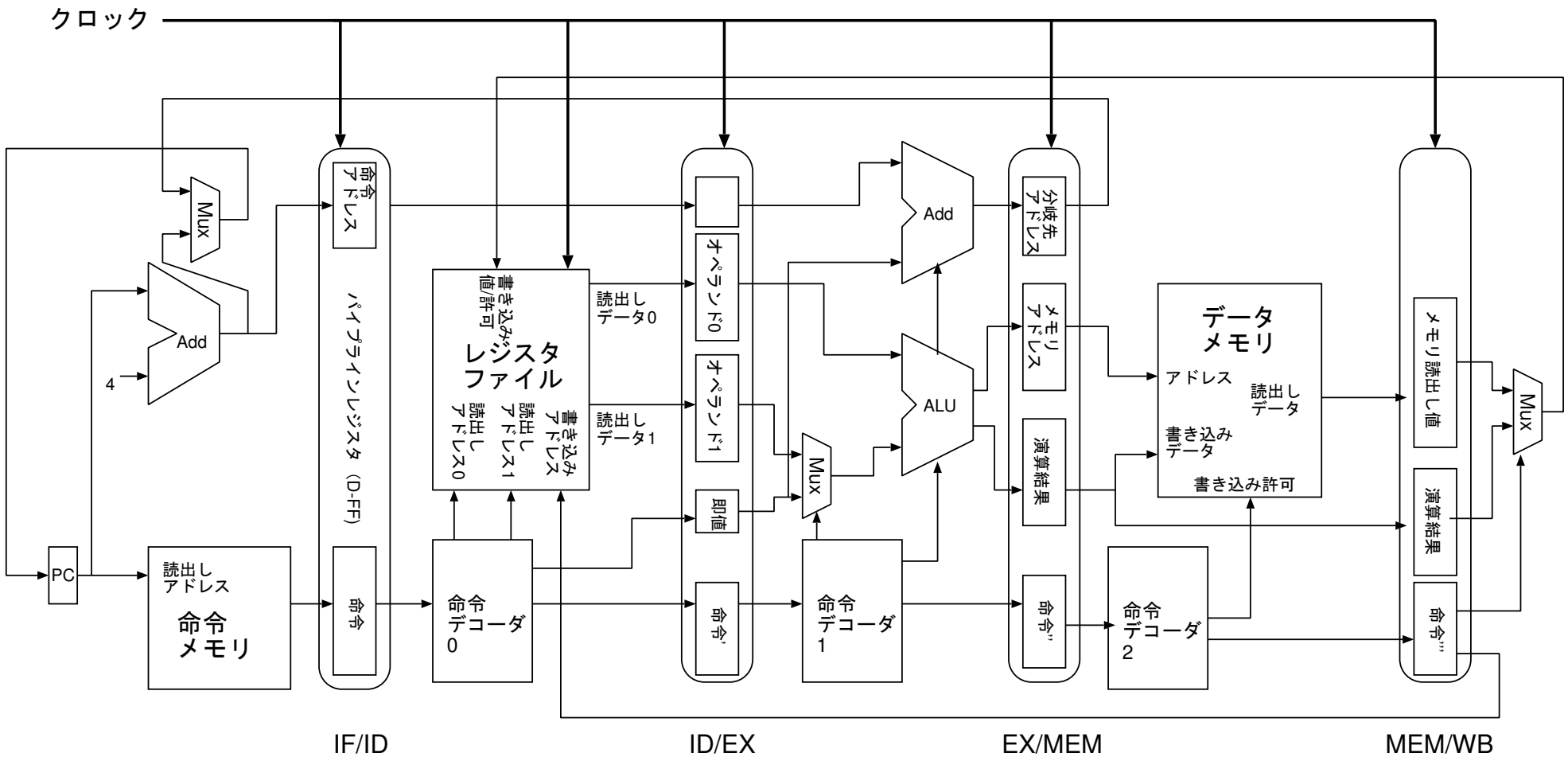

命令1
IF
ID
EX
MEM
WB
命令2
IF
ID
EX
MEM
WB
命令3
IF
ID
EX
MEM
WB
命令4
IF
ID
EX
MEM
WB
1クロックサイクル

5段パイプラインRISC計算機のブロック図



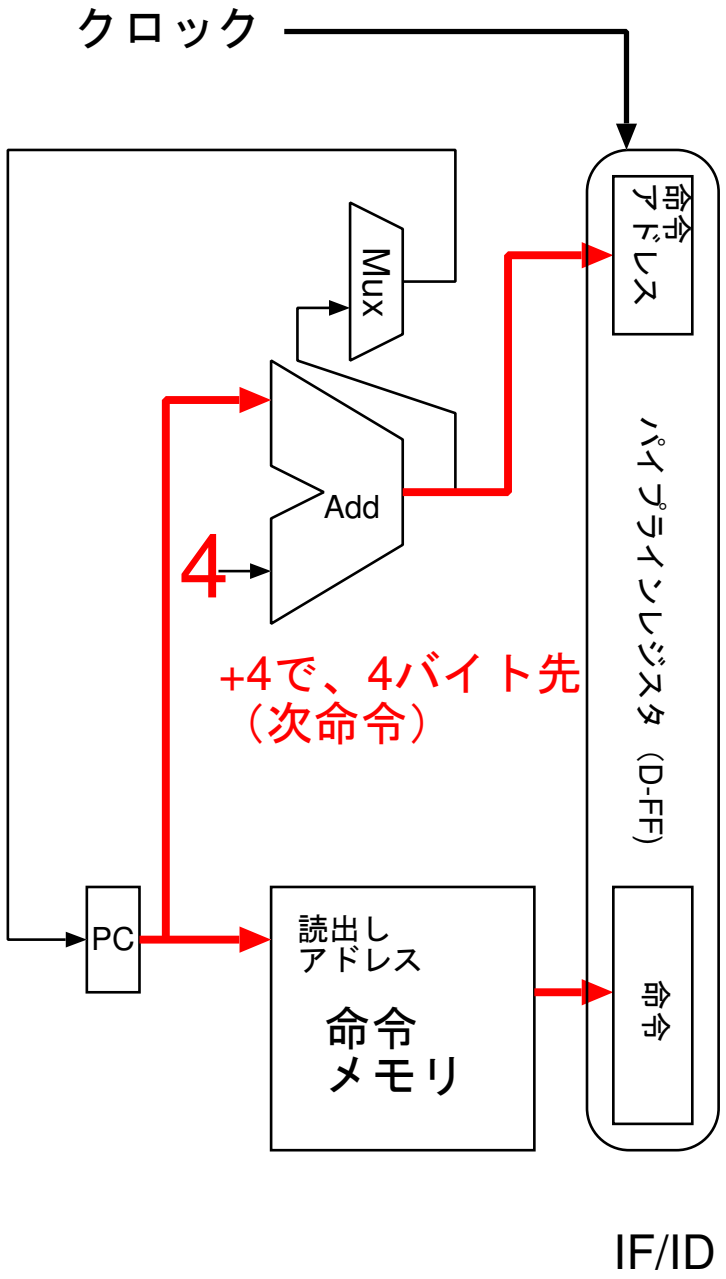
参考書 P352(96) 図6.11より

パイプラインレジスタの詳細化



参考書 P352(96) 図6.11より

IF: Instruction Fetch



IF/ID

- 命令形式は，３アドレス(DSTx1，SRCx2)と仮定.

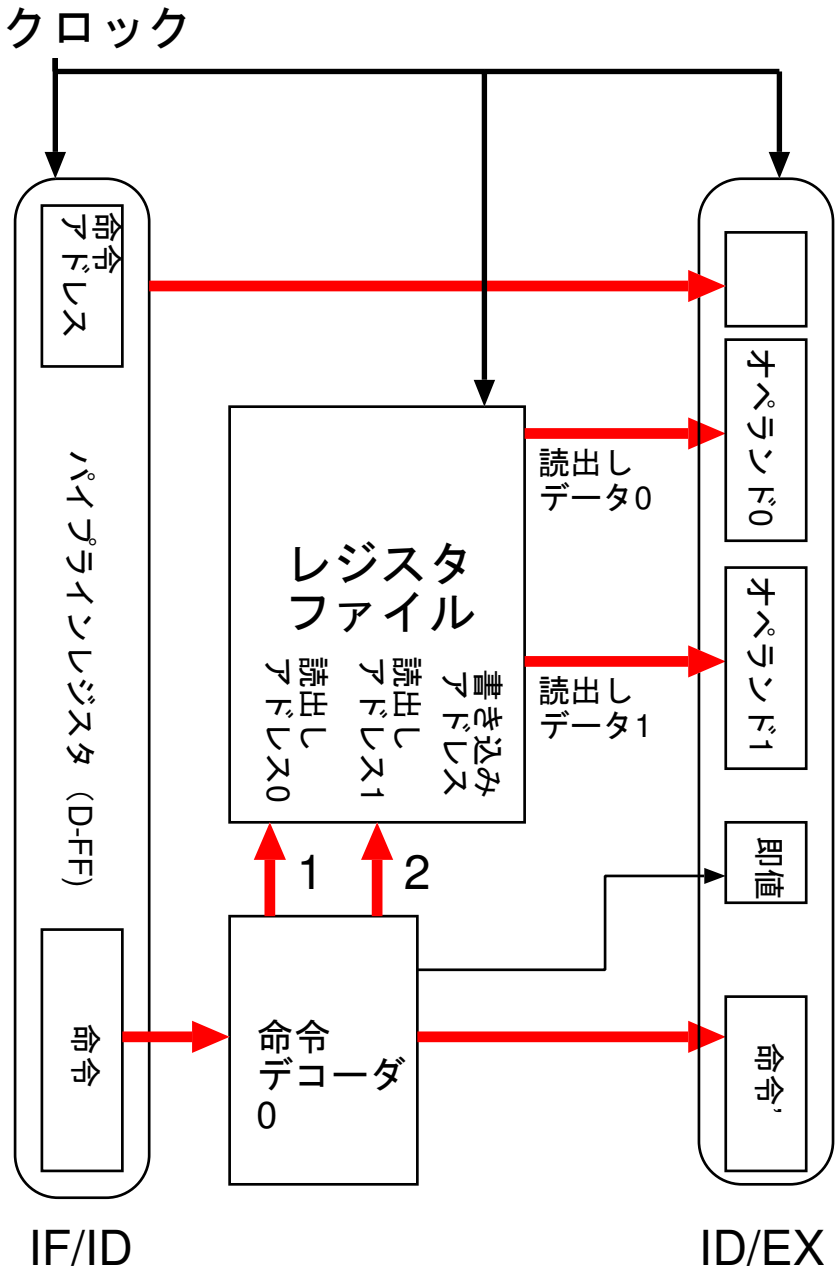
OPERATION	SRC0	SRC1	DST
ADD	REG[1]	REG[2]	REG[3]

REG[1] と REG[2] の値を加算して，REG[3]に保存する.

- PCの現在の値を用いて，命令メモリから命令が読み込まれ，パイプラインレジスタ内の命令レジスタに保存
- PCの値は，＋4(バイト＝32ビット分)されて，パイプラインレジスタ内の命令アドレスレジスタに保存.

参考書 P354(98) 図6.12より

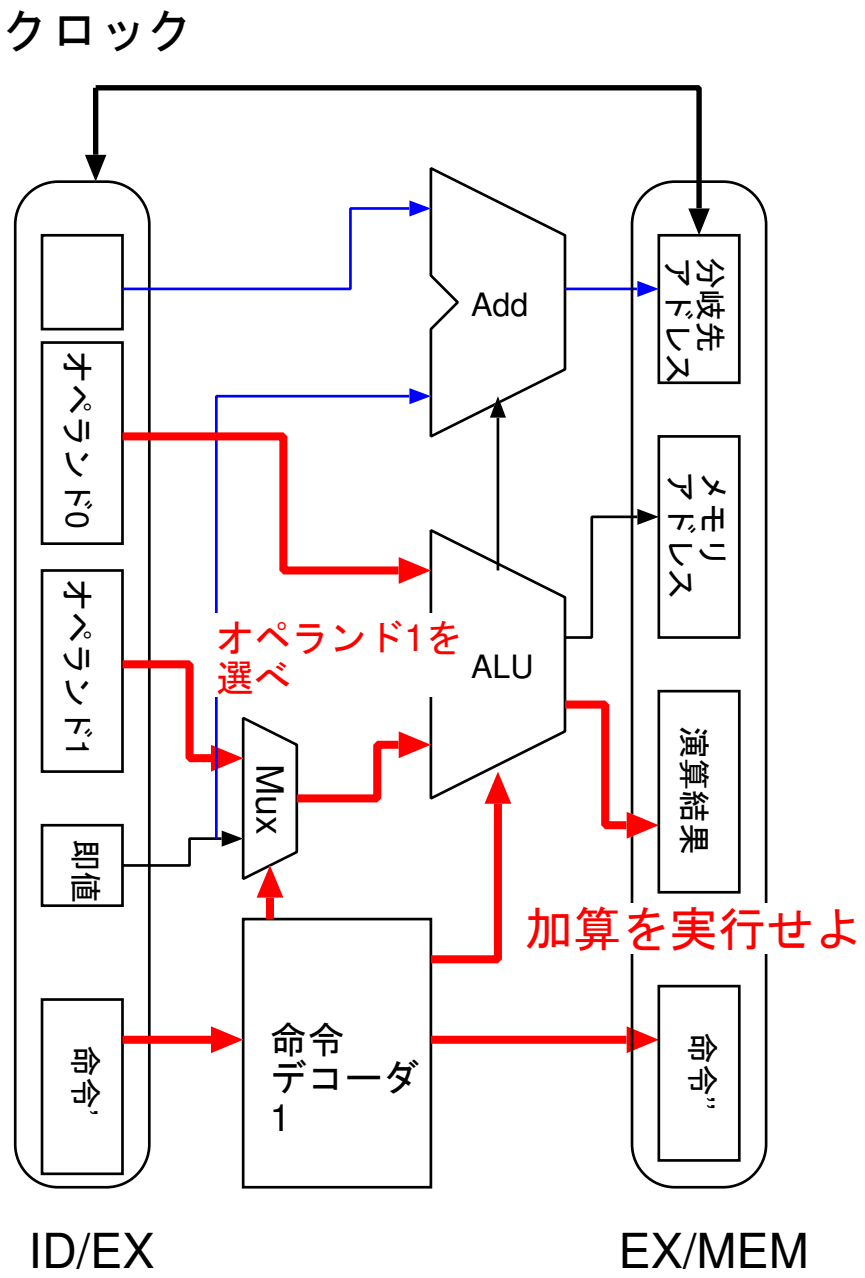
ID: Instruction Decode



- 命令レジスタの内容を解析して，オペランド/即値を取り出す．
- オペランドを，レジスタファイルより取り出し，パイプラインレジスタ内のオペランド0，オペランド1に格納する．
- 命令アドレスレジスタはそのまま次段に転送．

参考書 P354(98) 図6.12より

EX: Execution Stage



演算命令(加算/減算等)の場合

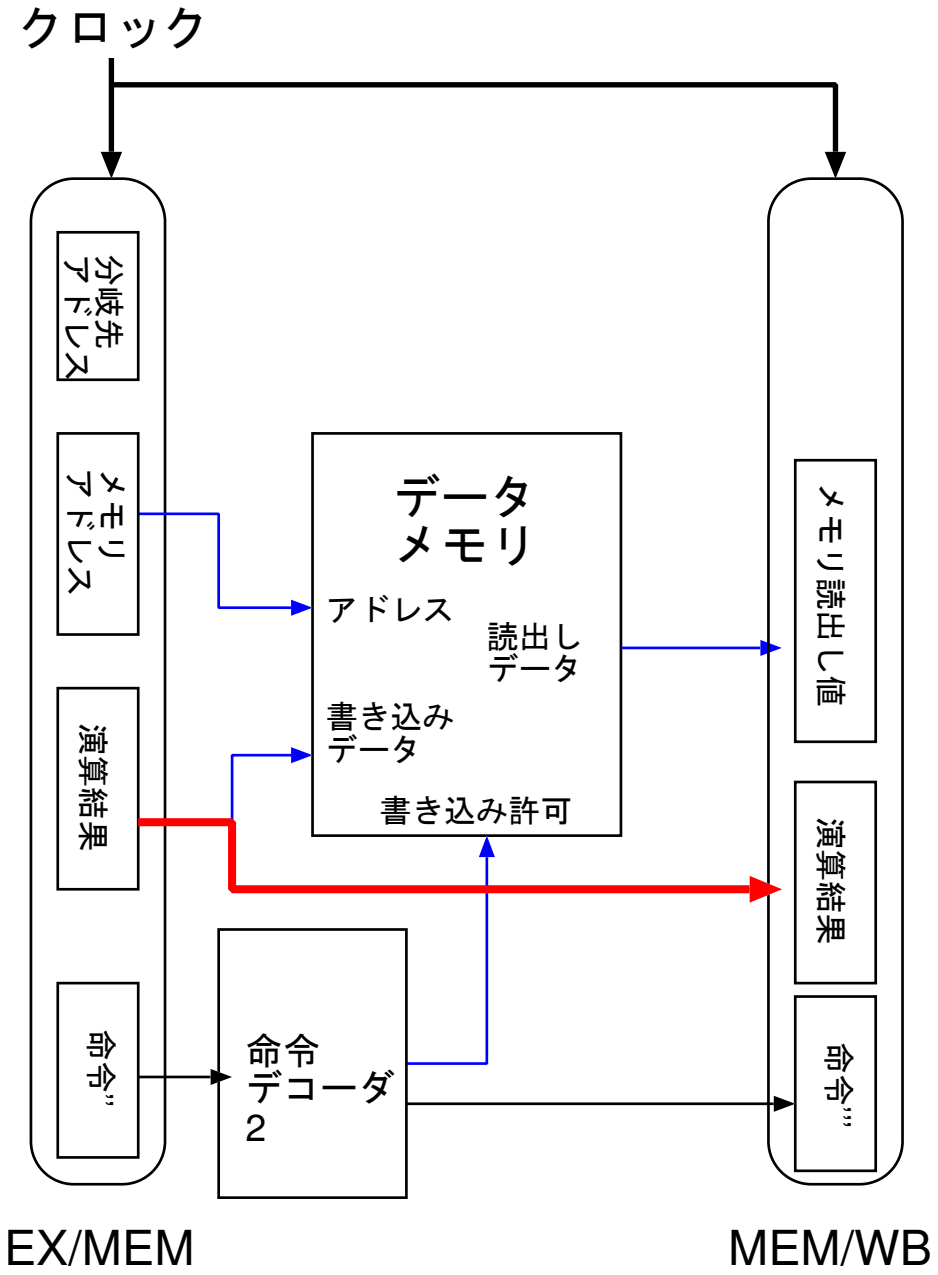
- オペランド/即値の値を用いて，演算を行ない，演算結果レジスタに格納

分岐命令の場合

- 分岐先のアドレスを生成して，分岐先アドレスレジスタに保存.

参考書 P355(99) 図6.13より

MEM: Memory Access Stage



演算命令(加算/減算等)の場合

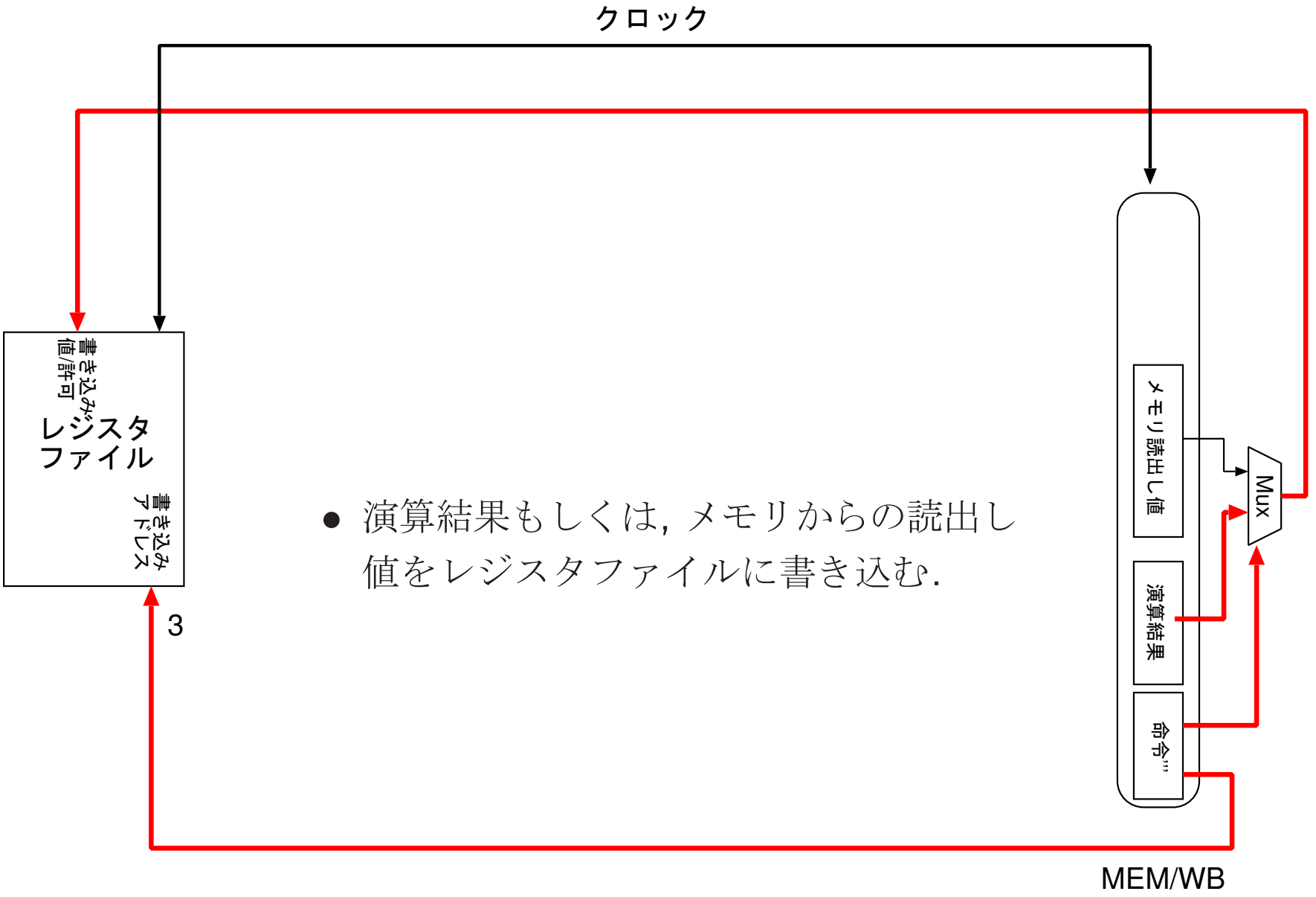
- メモリにはアクセスしない(RISC型プロセッサでは，演算結果はメモリに書き込まない)ので，単にデータを次段のパイプラインレジスタに転送するだけ．

メモリアクセス命令の場合

- メモリに書き込む． またはメモリから値を読み出して，パイプラインレジスタに格納する．

参考書 P356(100) 図6.14より

WB: Write Back

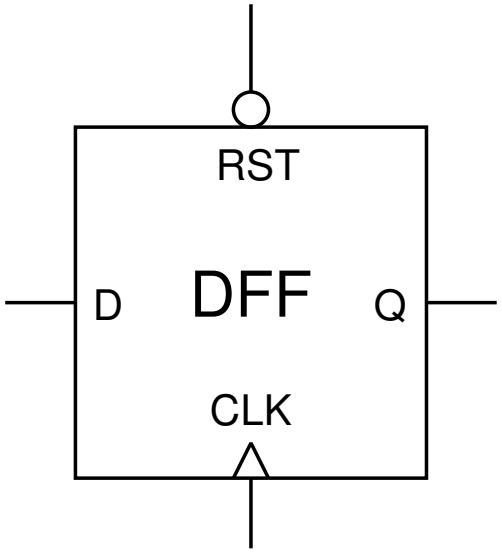


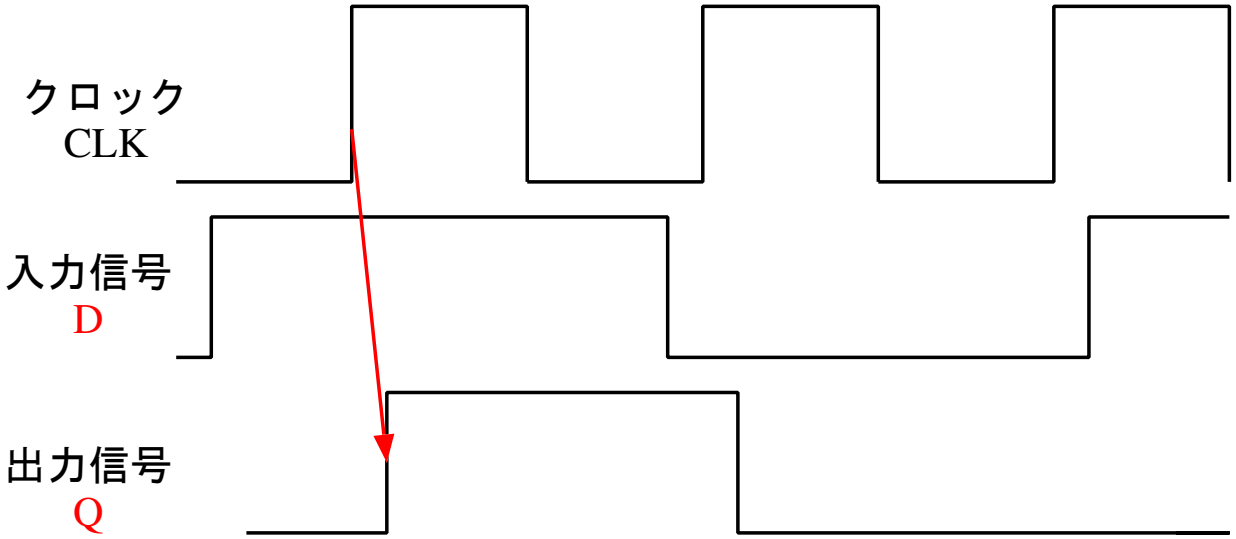
- 演算結果もしくは，メモリからの読出し値をレジスタファイルに書き込む．

参考書 P356(100) 図6.14より

D-FFによるレジスタ

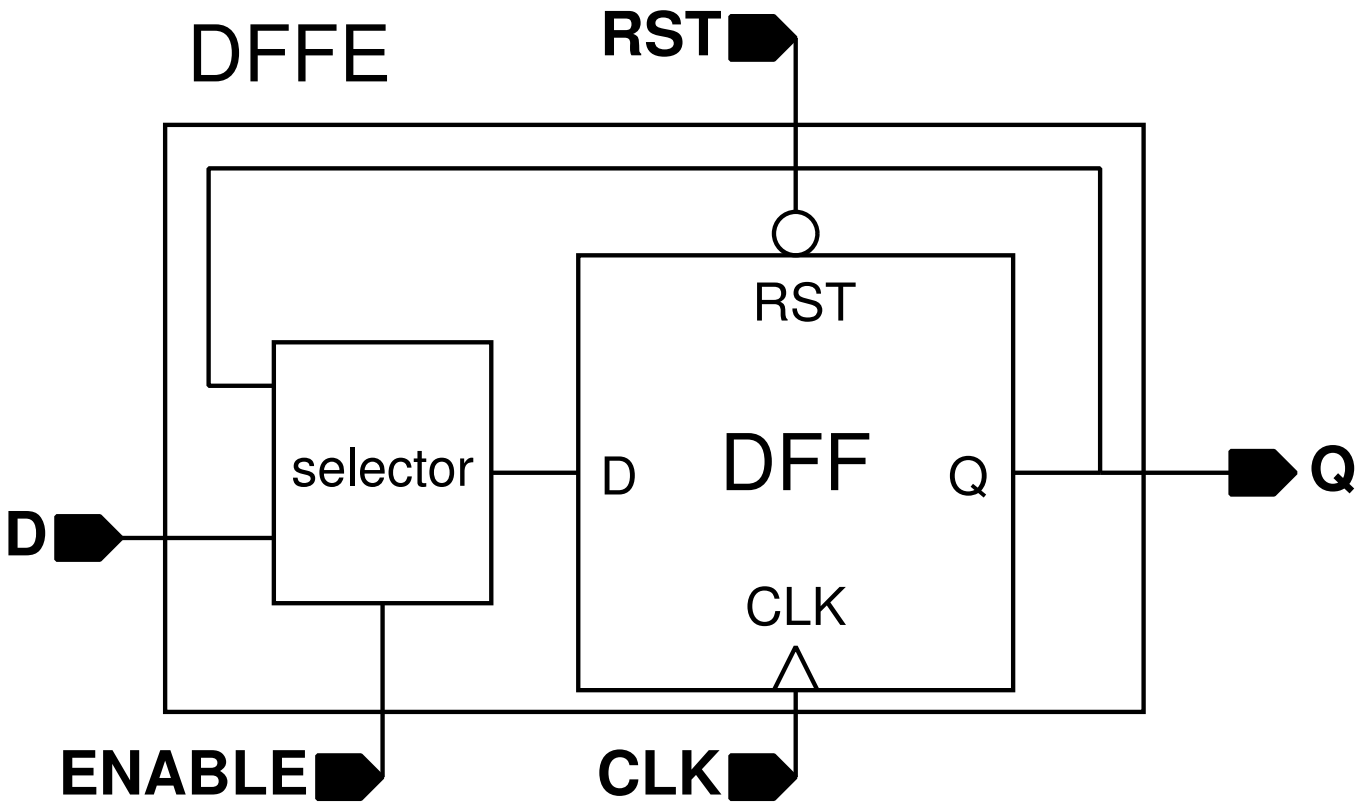
- レジスタは通常，D-FFにより実現される．
- D-FFでは単なる遅延素子．



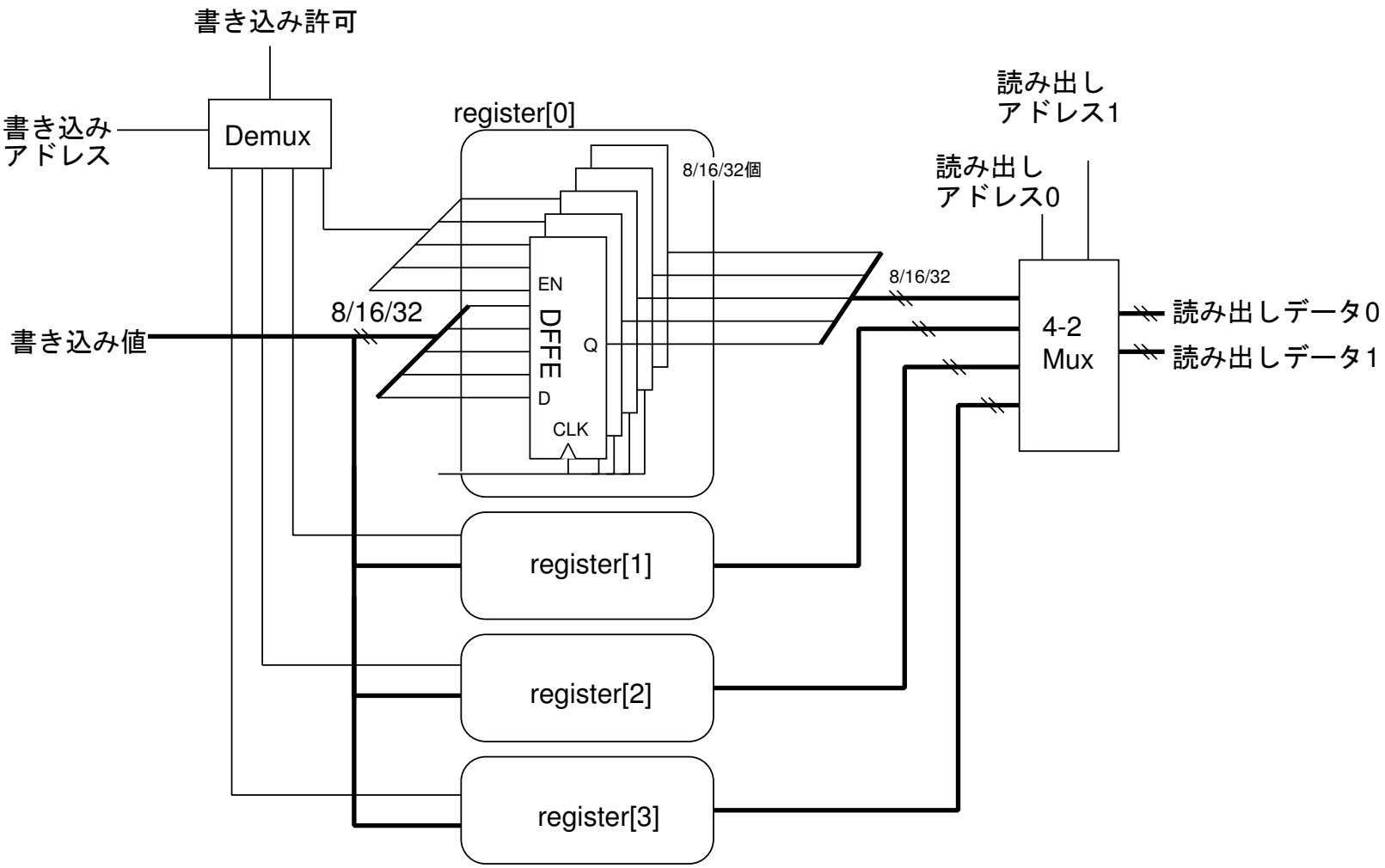


イネーブルつきD-FF

- ENABLEが0の時は前の値を保持する.



D-FFEによるレジスタファイルの実現（2読出1書込）



4ワードのレジスタファイルの回路構成

Mux: Multiplexer 切り替え器（入力数＞出力数）

Demux: demultiplexer 逆切り替え器（入力数＜出力数）

パイプラインプロセッサにおける工夫

分岐命令がなければ，パイプラインは滞りなく動く．

単に性能をあげるために

遅延分岐 分岐後に，パイプライン内に残っている命令を実行してから分岐する．命令列をあらかじめ，分岐後の数命令は実行されることを前提に並べる．

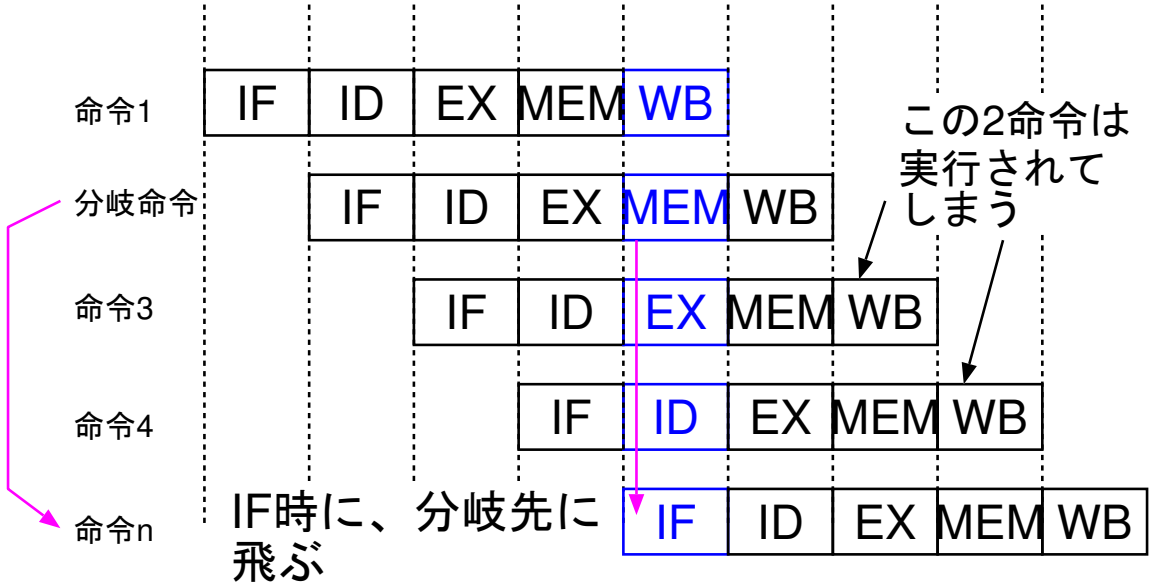
パイプラインを乱さないために

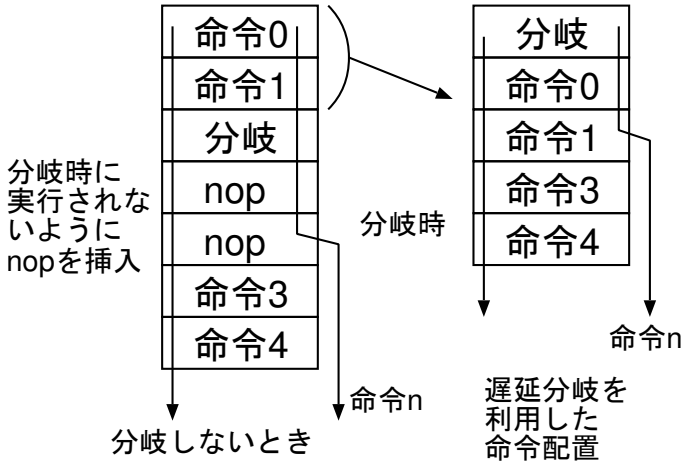
分岐予測 パイプラインの乱れは，性能の低下につながるので，分岐が起こるか起こらないかをあらかじめ予測して，予測した方の命令を実行しておく．予測が当たらない場合は，性能が低下．分岐予測のための付加ハードウエアが必要．

条件実行命令 条件によって，実行するしないを制御できる命令．分岐命令の数を減らすことができる．ARMプロセッサに実装されている．

遅延分岐

- MEMステージでのアドレスがIFステージで有効になるまでに２命令実行されている．
- 通常は，この２命令を捨てずにそのまま実行する．捨てるためには付加ハードウエアが必要．
- 実行されてしまうところに，NOPを入れると簡単
- 捨てずに実行する部分にNOPではなく，命令の前後を入れ替えることで，通常の命令を入れることも．(遅延分岐)





パイプライン計算機の性能

D段のパイプラインに，I個の命令を投入．ステップ数は?

パイプライン化　　$S_P = I + D - 1$

パイプラインなし　　$S_N = I \times D$

- Dステージを一気に実行して，終了後に次のステージを実行．
- パイプラインをつぶすと，パイプラインレジスタは不要．

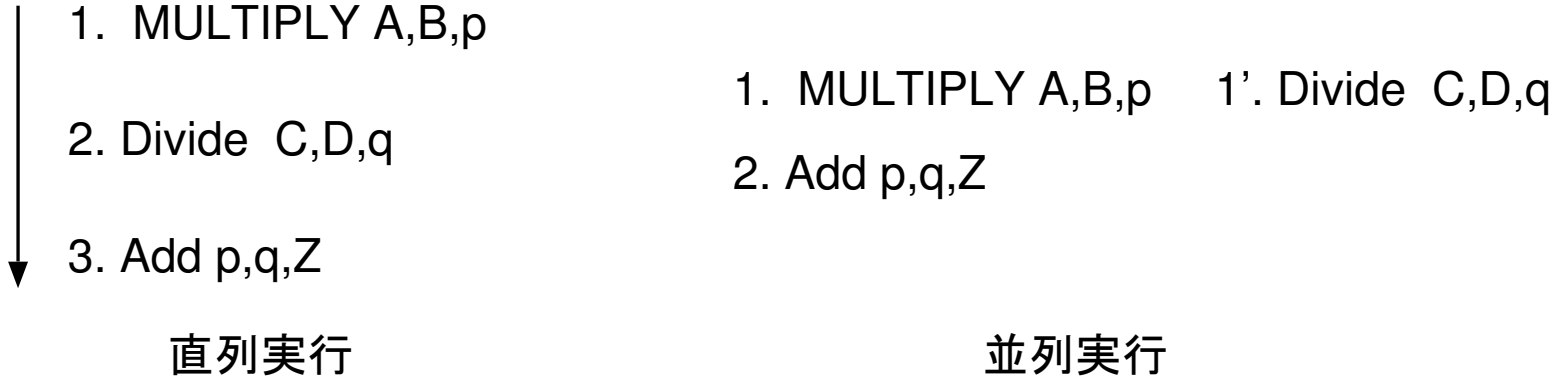
性能向上率　　$$P = \frac{S_N}{S_P} = \frac{I \cdot D}{I + D - 1}$$

$$P \simeq D \qquad (I >> D)$$

命令レベル並列処理: Instruction Level Parallelism

同時に実行可能な命令を並列に実行する.

$$A \times B + (C/D) \rightarrow Z$$



VLIW: Very Long Instruction Word 命令列をあらかじめ並列化しておく. コンパイラ (ソフトウエア) で並列化.

Super Scalar 並列化されていない命令列を実行時に, ハードウエアで並列化.

VLIW: Very Long Instruction Word

- 意味は，「非常に長い命令語長」．
- 実体は，複数の命令を並べて一つの命令にしている．
- コンパイラが，自動的に並列化可能な命令を一つの命令語にする．**コンパイラ(ソフトウエア)による並列化**
- ハードウエアの規模が抑えられる．(ソフトウエア重視)
- VLIWではない計算機と，命令語の互換性はない．
- Transmeta社のCrusoeは，x86命令をコンパイラにより，リアルタイムにVLIW命令に置き換えている．
- Intel社の，Itaniumは，VLIWプロセッサ(Pentiumとの命令互換性なし)．

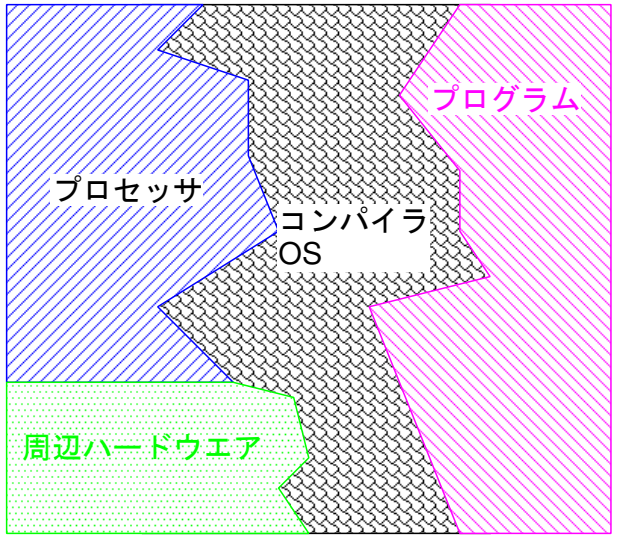
Super Scalar: スーパスカラ(スーパースケーラ)

- スカラ(直列)処理の限界である，1CPIを超える．
- すでにコンパイル済の並列ではない機械語より，ハードウエアによって，並列性を抽出し，並列実行する．
- 並列性を抽出するための，ハードウエアが必要．(ハードウエア重視)
- Super Scalarではない，計算機と命令語の互換性がある．
- Pentium以降の，x86プロセッサは，命令をRISC命令に置き換え，それをSuper Scalarで並列実行している．

制御機構とオペレーティングシステム

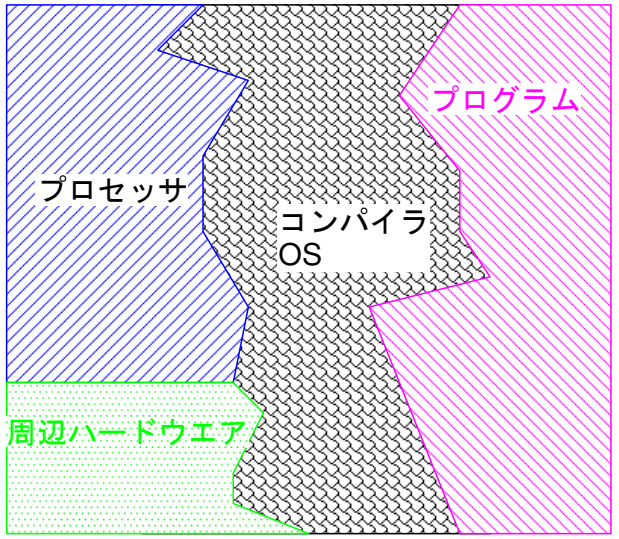
OS: Operating System ハードウエアと直接かかわり合うソフトウエア機構.

- **普通のプログラムは，ハードウエアと直接やり取りしない.**
- プログラム側は異なるハードウエアでも同じ手続きで制御できる.



プロセス (process)，タスク (task) ある機能を実現するために実行するプログラム.

- Windows なら，　タスクマネージャ(プロセス),Mac (OS-X)，Unix (Linux等) なら，ps (プロセス) コマンドで見えるのがプロセス



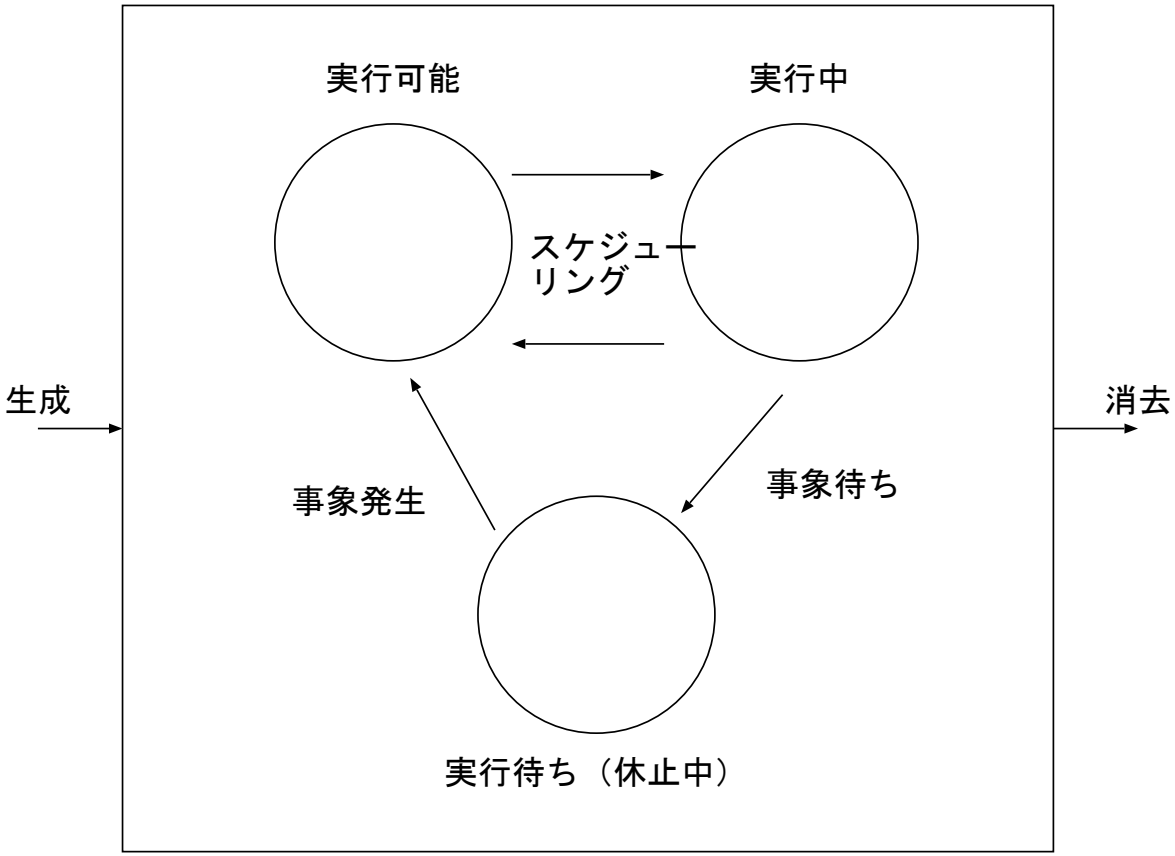
マルチタスキング (multi-tasking) 複数のプロセスを時分割 (Time Division) 制御で，順番に実行.

- タスク切り替えの周期は，おおよそ数m(ミリ)秒

OSによるプロセス管理

- プロセスの生成と消去(削除)
- 実行プロセスの切り替え(タスクスイッチ)
- 複数プロセスの同期
- プロセス実行時のスケジューリング
- プロセス(メモリ)の保護

プロセスの状態



- 実行しなければならないプロセスは，数ｍ秒単位で，実行可能状態と実行中状態を行き来する．
- 実行待ちとは，入力を待っていたり，時間を待っている状態．

教科書 P144 図5.17より

プロセススイッチ

- 実行可能状態のプロセスのうち，一つを実行中状態にする．

  プロセスコンテクスト ー プロセッサの状態
  - ー プログラムカウンタ
  - ー 汎用レジスタ
  - ー (メモリイメージ，メインメモリの管理情報，割り込み優先度)

- プロセススイッチでは，現在実行中状態のプロセスコンテクストをメインメモリに待避． 次に実行させるプロセスのプロセスコンテクストをメインメモリからプロセッサにコピー．

プロセスのスケジューリングアルゴリズム

横取り不可能 (non-preemptive) プロセス自体が終了, 消去, 実行待ちしない限り, そのプロセスの実行が止まらない.

- プログラムが暴走すると, 止められない.

横取り可能 (preemptive) OS が強制的に実行中のプロセスからプロセッサを横取りする.

横取り方法　　**優先度** 優先度が高い順に実行. 低い優先度のプロセスが実行されにくい.

ラウンドロビン 一定時間毎に順に切り替えていく. 実現が簡単. 優先度の高い処理 (動画のデコードなど) が遅くなる可能性あり.

マルチスレッド

- **スレッド:** プロセスから生成される命令実行列.
- プロセス間は, メモリ共有は不可能.
- スレッド間はメモリ共有可能.
- **マルチスレッドの例:** DVD 再生時の, 音声デコードと映像デコード

プロセッサの状態

ユーザ状態 (ユーザモード) OS 以外の普通のプロセスを実行している状態.

スーパバイザ (特権) モード OS そのものを実行している状態.

特権命令 OS しか実行できない命令. ハードウエアを直接触る命令.

メモリ管理, システム保護などを行なう.