計算機科学実験及演習３ ハードウェア
SIMPLEアーキテクチャのプロセッサの実装

京都大学 工学部情報学科 計算機科学コース

計算機科学実験及演習３

ハードウェア担当

Ver.2026/04

実験３ハードウェアの内容と目的

- 内容
  - SIMPLEに準拠した独自のマイクロプロセッサの設計（方式設計、論理設計）とFPGAでの実装

目的

  - プロセッサの動作原理を理解する
  - 回路設計、最適化、動作テストの方法を習得する
  - プロセッサの種々の拡張方式や最適化技術を実践的に学ぶ
  - チームによるパッケージ製品開発の体験

- 参考文献
  - 富田眞治、中島浩： コンピュータハードウェア
  - D.A.パターソン、J.L.ヘネシー著、成田光彰訳：コンピュータの構成と設計(上),(下)　　など，，，

SIMPLEアーキテクチャの仕様

アーキテクチャ

- 命令セットアーキテクチャ
  - ISA: Instruction Set Architecture
  - プロセッサが理解して実行できる命令やレジスタの構造、アドレス指定方法などの定義
  - 例：MIPS, x86, SIMPLE
- マイクロアーキテクチャ
  - ISAを実際にハードウェアとしてどう実装するかの定義。モジュール構成、パイプライン、ALU、レジスタファイルなどの構成の詳細
  - 同じISAを実現するにもいくつものマイクロアーキテクチャがある
  - 例：AlderLake（Intel Core), Zen4 (AMD), SIMPLE/B

概要

SIxteen-bit MicroProcessor for Laboratory Experiment

- 😐簡単な命令セット
- ☺基本機能は1通り備えられている

- 特徴
  - 16bit固定長命令
  - 16bit×64K語の主記憶
  - ８本の汎用レジスタ
  - 2オペランド形式の命令セット(Rd op Rs -> Rd)

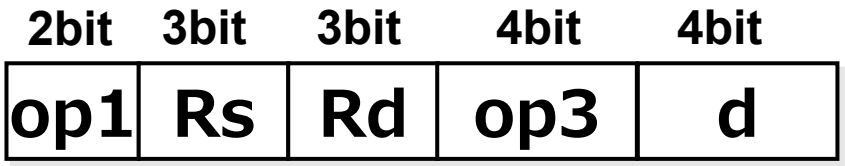
メモリ・レジスタの仕様

- メインメモリ
  - 16bit x 64k語
    （FPGAボードの上限にかかるため、実装は32k語）
- プログラムカウンタ
  - 16bit
- 汎用レジスタ
  - 16bit x 8本
- 条件コード
  - S サイン
  - Z ゼロ
  - C キャリー
  - V オーバーフロー

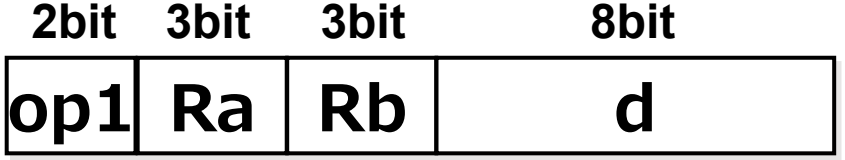
命令セットの仕様

命令はすべて16bit固定長。4種の命令形式がある。

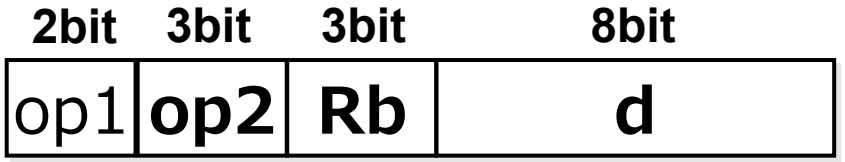
1. 演算／入出力命令形式



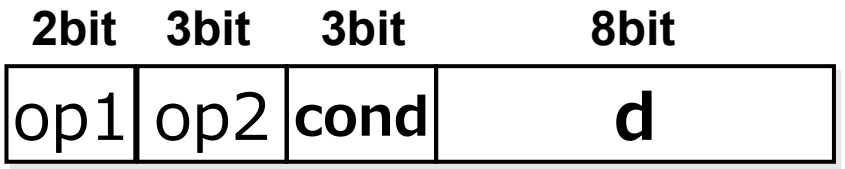
2. ロード／ストア命令形式



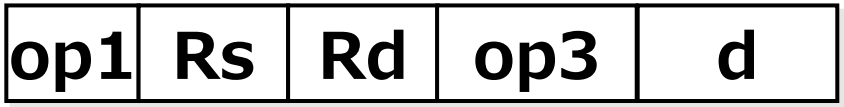
3. 即値ロード／無条件分岐命令形式



4. 条件分岐命令形式



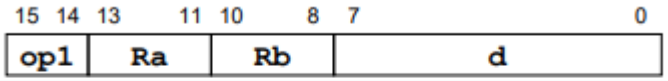
1.演算/入出力命令形式



mnemonic	op3	function
ADD Rd,Rs	0000	r[Rd] = r[Rd] + r[Rs]
SUB Rd,Rs	0001	r[Rd] = r[Rd] - r[Rs]
AND Rd,Rs	0010	r[Rd] = r[Rd] & r[Rs]
OR Rd,Rs	0011	r[Rd] = r[Rd] | r[Rs]
XOR Rd,Rs	0100	r[Rd] = r[Rd] ^ r[Rs]
CMP Rd,Rs	0101	r[Rd] - r[Rs]
MOV Rd,Rs	0110	r[Rd] = r[Rs]
(reserved)	0111	
SLL Rd,d	1000	r[Rd] = shift_left_logical(r[Rd],d)
SLR Rd,d	1001	r[Rd] = shift_left_rotate(r[Rd],d)
SRL Rd,d	1010	r[Rd] = shift_right_logical(r[Rd],d)
SRA Rd,d	1011	r[Rd] = shift_right_arithmetic(r[Rd],d)
IN Rd	1100	r[Rd] = input
OUT Rs	1101	output = r[Rs]
(reserved)	1110	
HLT	1111	halt()

- op1: 11
- Rs：ソースレジスタ
- Rd：演算結果を格納するレジスタ
- op3：演算の種類
- d：シフト桁数（シフト演算時のみ）

2.ロード／ストア



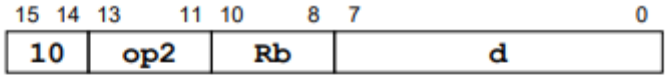
mnemonic	op1	function
LD Ra,d(Rb)	00	r[Ra] = *(r[Rb] + sign_ext(d))
ST Ra,d(Rb)	01	*(r[Rb] + sign_ext(d)) = r[Ra]

- op1: 00/01(ロード/ストア)
- Ra
  - LD: 値を格納するレジスタ
  - ST: ソースレジスタ
- Rb: ベースレジスタ
- d: 変位

ベースアドレス指定方式

メモリのアドレスを指定する際に、ベースレジスタに格納されている値と変位の値を加算した値を、アドレスとして用いる方式

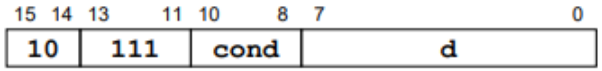
3.即値ロード/無条件分岐



mnemonic	op2	function
LI Rb,d	000	r[Rb] = sign_ext(d)
(reserved)	001	
(reserved)	010	
(reserved)	011	
B d	100	PC = PC + 1 + sign_ext(d)
(reserved)	101	
(reserved)	110	
(条件分岐命令)	111	表 4 参照

- op1: 10
- op2: 000/100 (LI/B)
- Rb:
  - LI: 値を格納するレジスタ
  - B: 未使用
- D:
  - LI: 即値
  - B: 変位

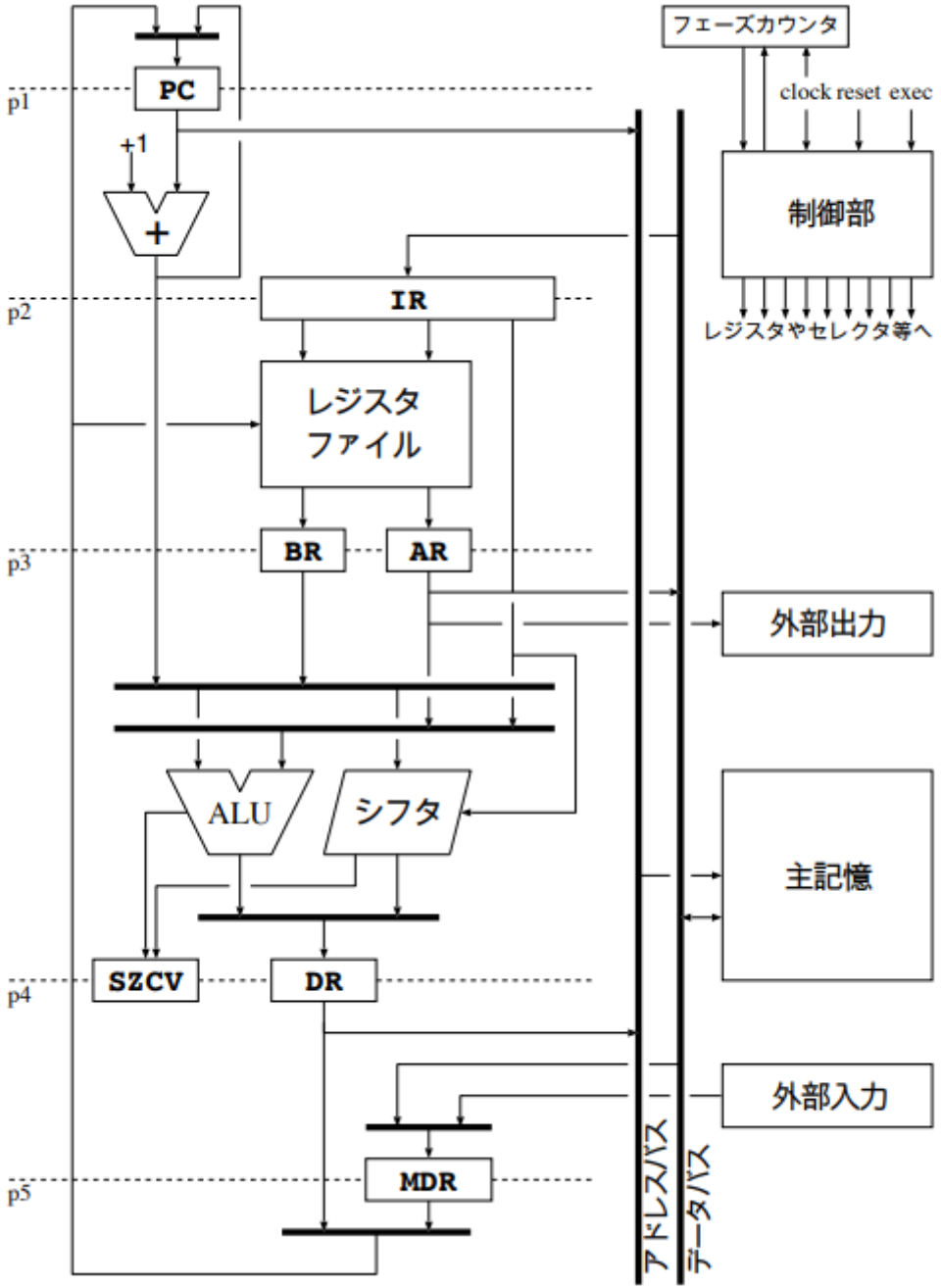
4.条件分岐命令



mnemonic	cond	function
BE d	000	if (Z) PC = PC + 1 + sign_ext(d)
BLT d	001	if (S ^ V) PC = PC + 1 + sign_ext(d)
BLE d	010	if (Z || (S ^ V)) PC = PC + 1 + sign_ext(d)
BNE d	011	if (!Z) PC = PC + 1 + sign_ext(d)
(reserved)	100	
(reserved)	101	
(reserved)	110	
(reserved)	111	

- op1: 10
- op2: 111
- cond:分岐条件
- d: 変位

マイクロアーキテクチャと動作の例
SIMPLE/B

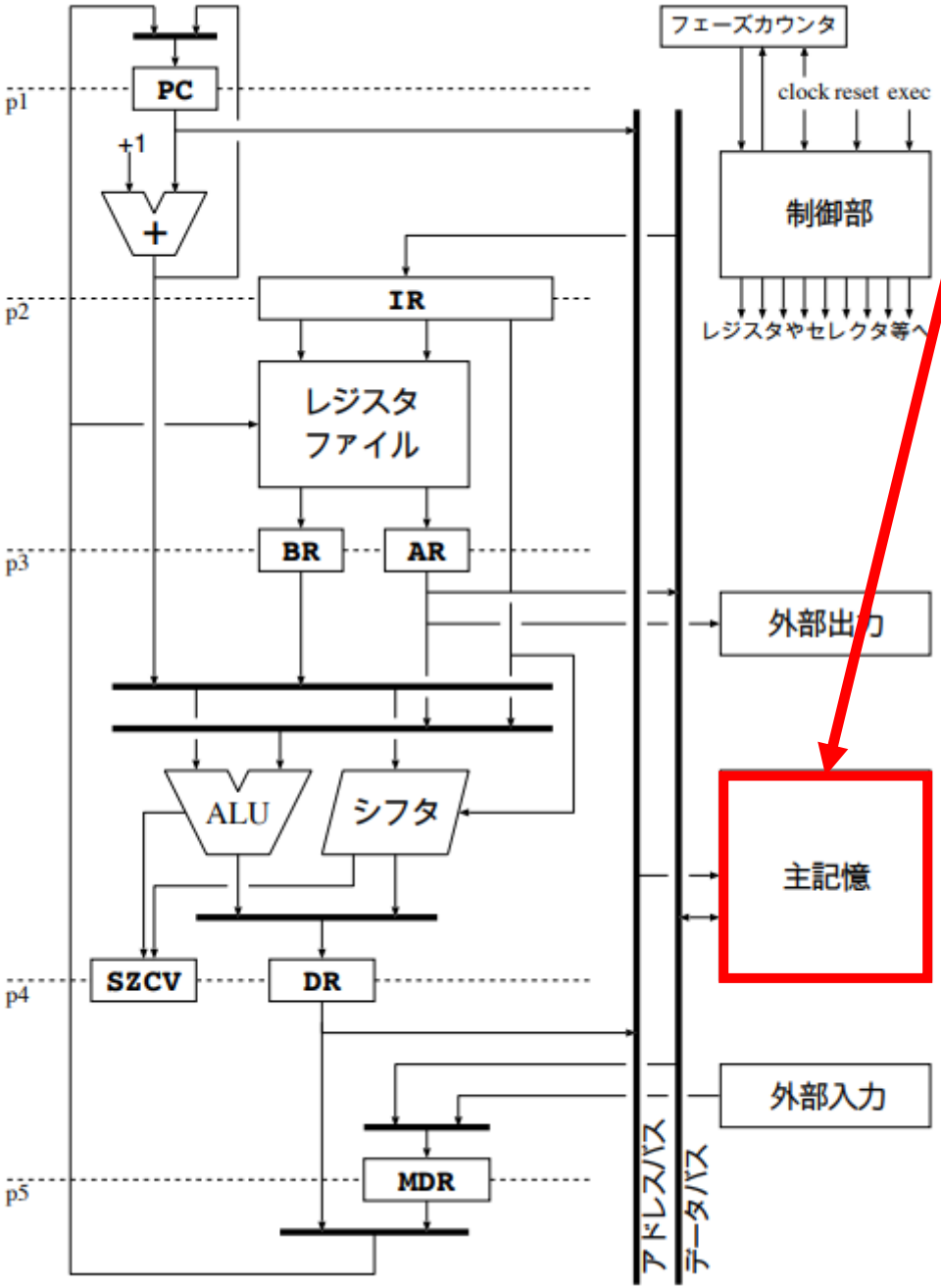


モジュール構成

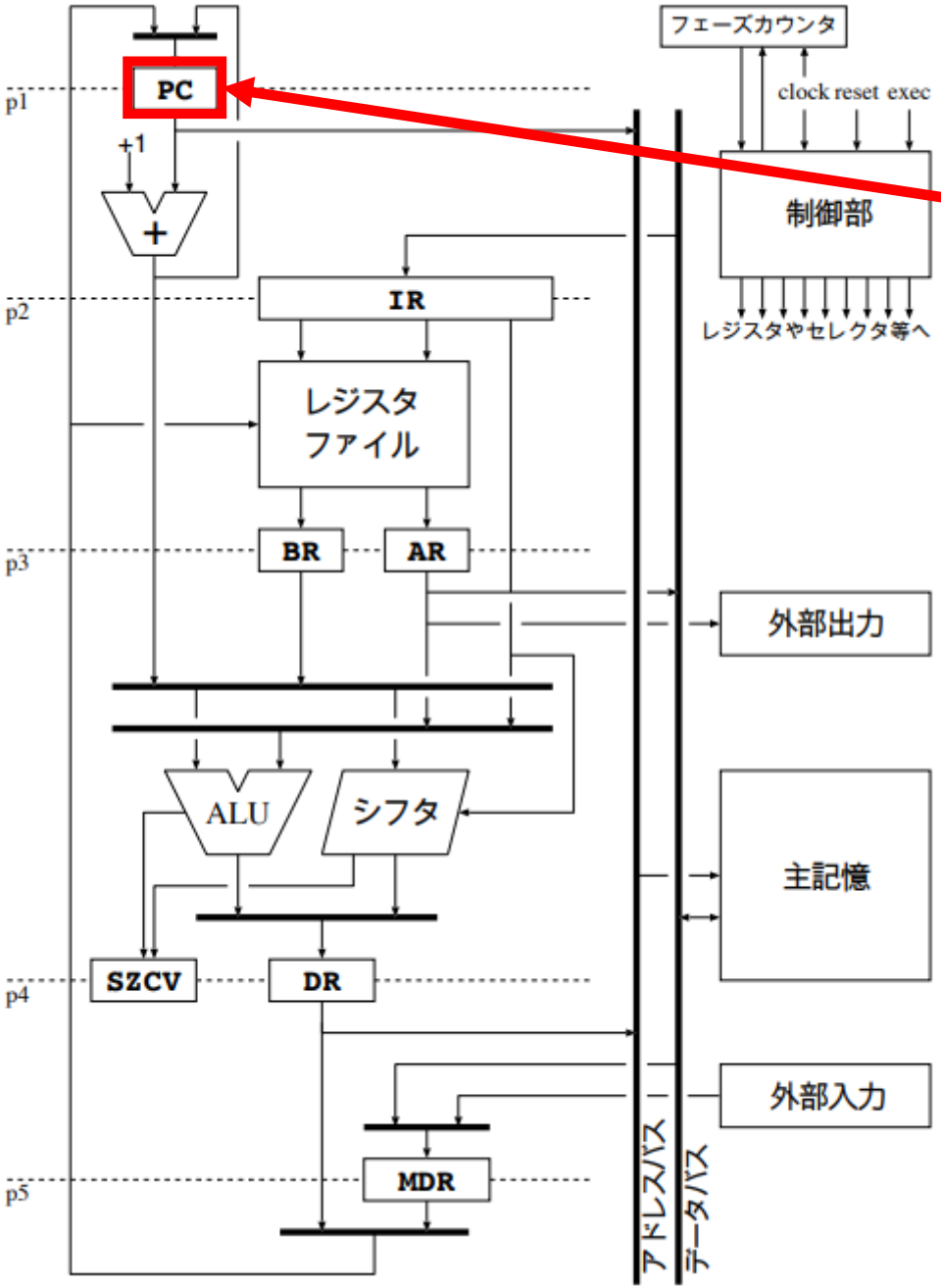
- メインメモリ
- プログラムカウンタ
- 汎用レジスタ
- ALU
- レジスタ

  （IR、AR・BR、DR、MDR）

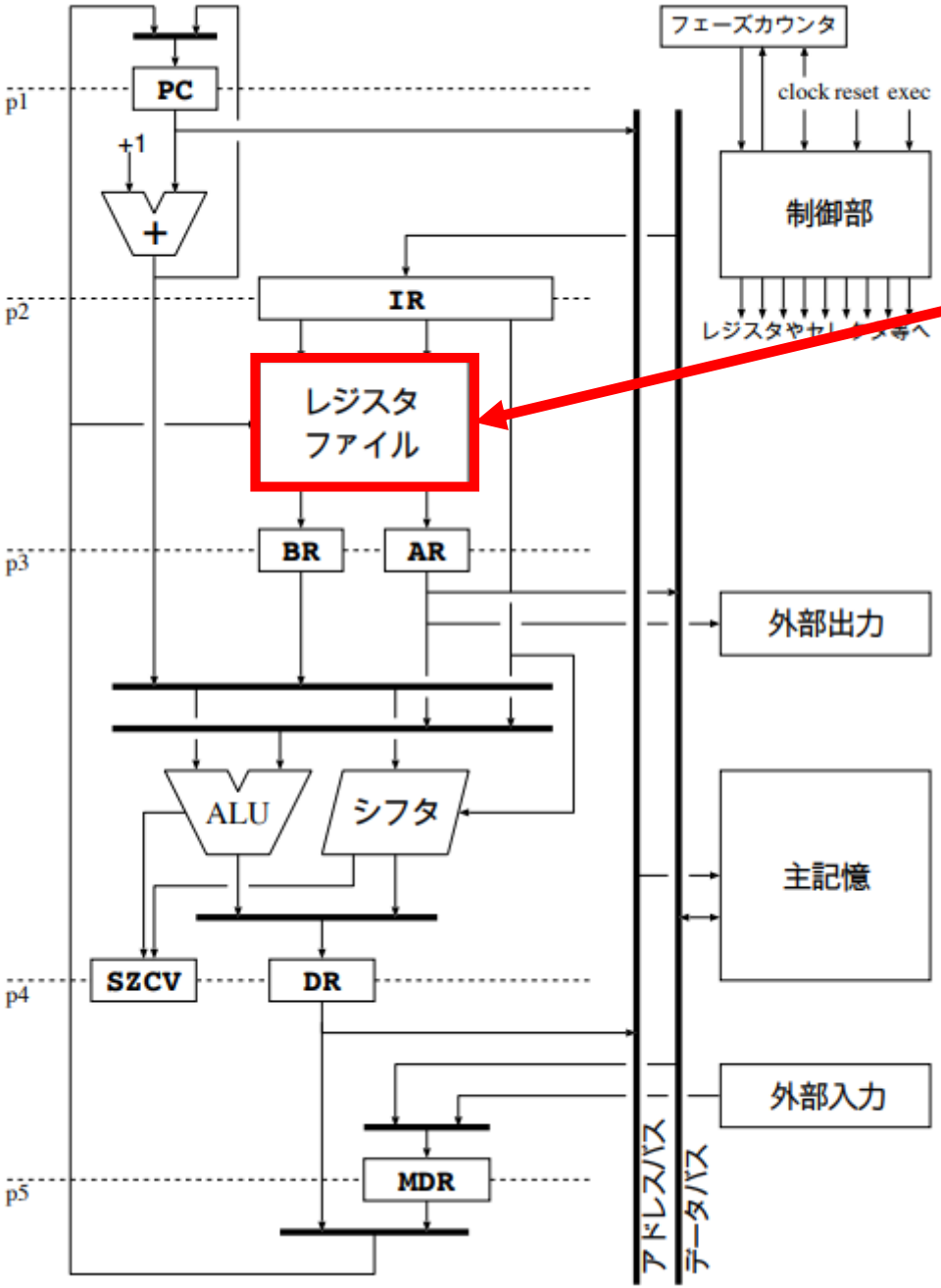
  各フェーズでの値を保持
- フェーズカウンタ
  - フェーズの活性化信号を生成
- 制御回路
  - マルチプレクサを切り替える
  - レジスタのenable信号
  - ALUの機能の選択
  - などなど



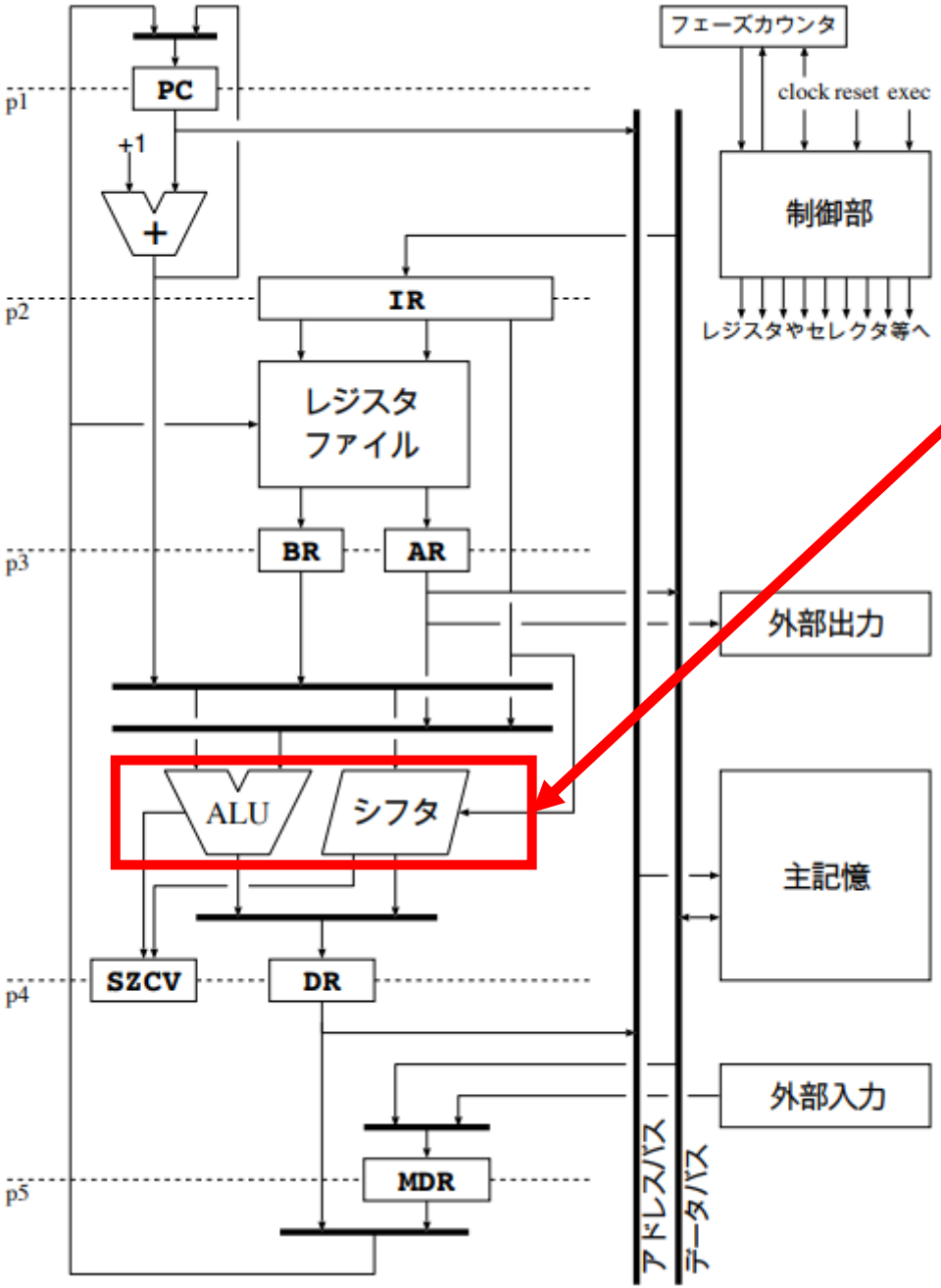
- メインメモリ
- プログラムカウンタ
- 汎用レジスタ
- ALU
- レジスタ
  （IR、AR・BR、DR、MDR）
  各フェーズでの値を保持
- フェーズカウンタ
  - フェーズの活性化信号を生成
- 制御回路
  - マルチプレクサを切り替える
  - レジスタのenable信号
  - ALUの機能の選択
  - などなど



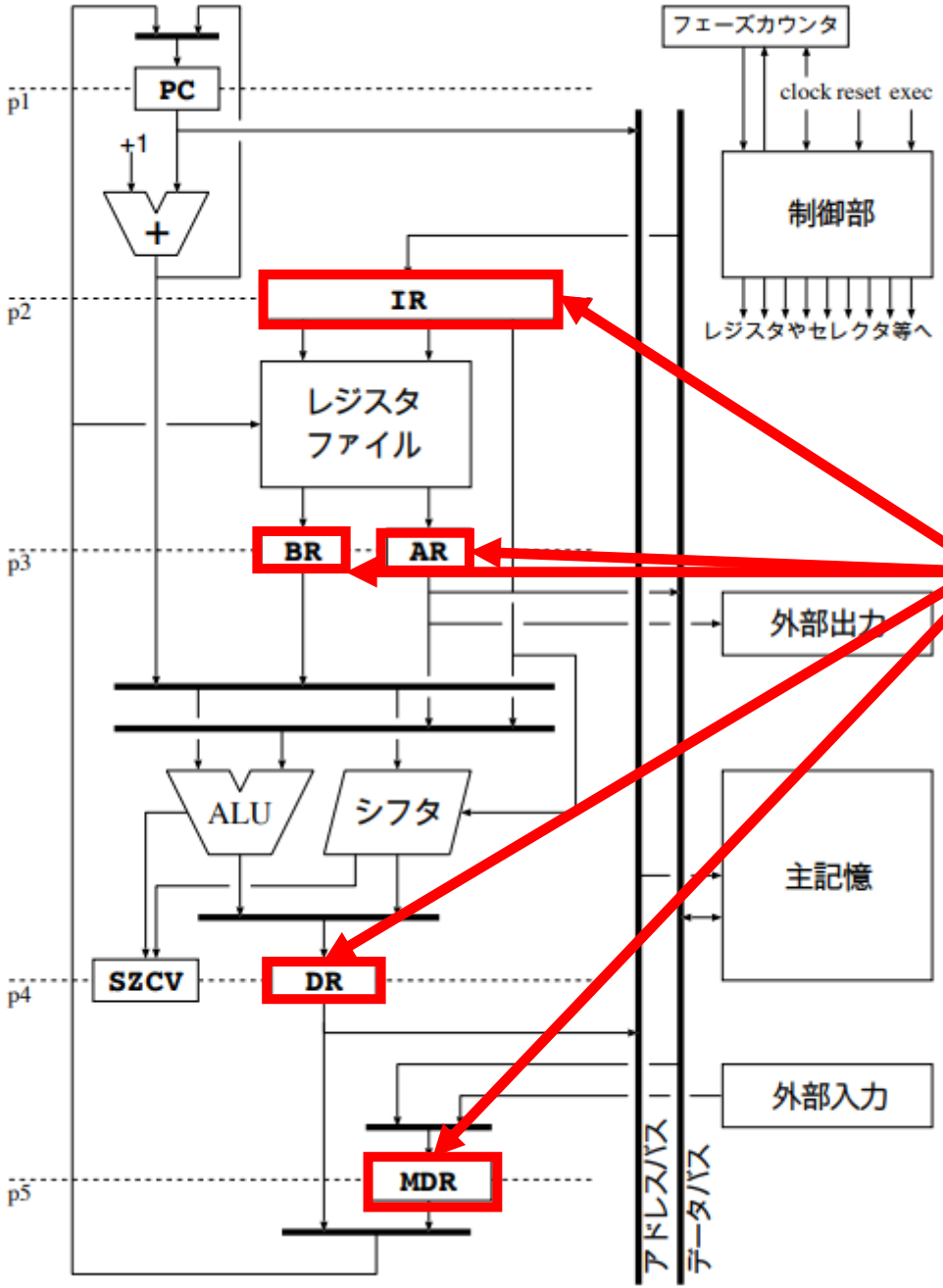
- メインメモリ
- プログラムカウンタ
- 汎用レジスタ
- ALU
- レジスタ
  （IR、AR・BR、DR、MDR）
  各フェーズでの値を保持
- フェーズカウンタ
  - フェーズの活性化信号を生成
- 制御回路
  - マルチプレクサを切り替える
  - レジスタのenable信号
  - ALUの機能の選択
  - などなど



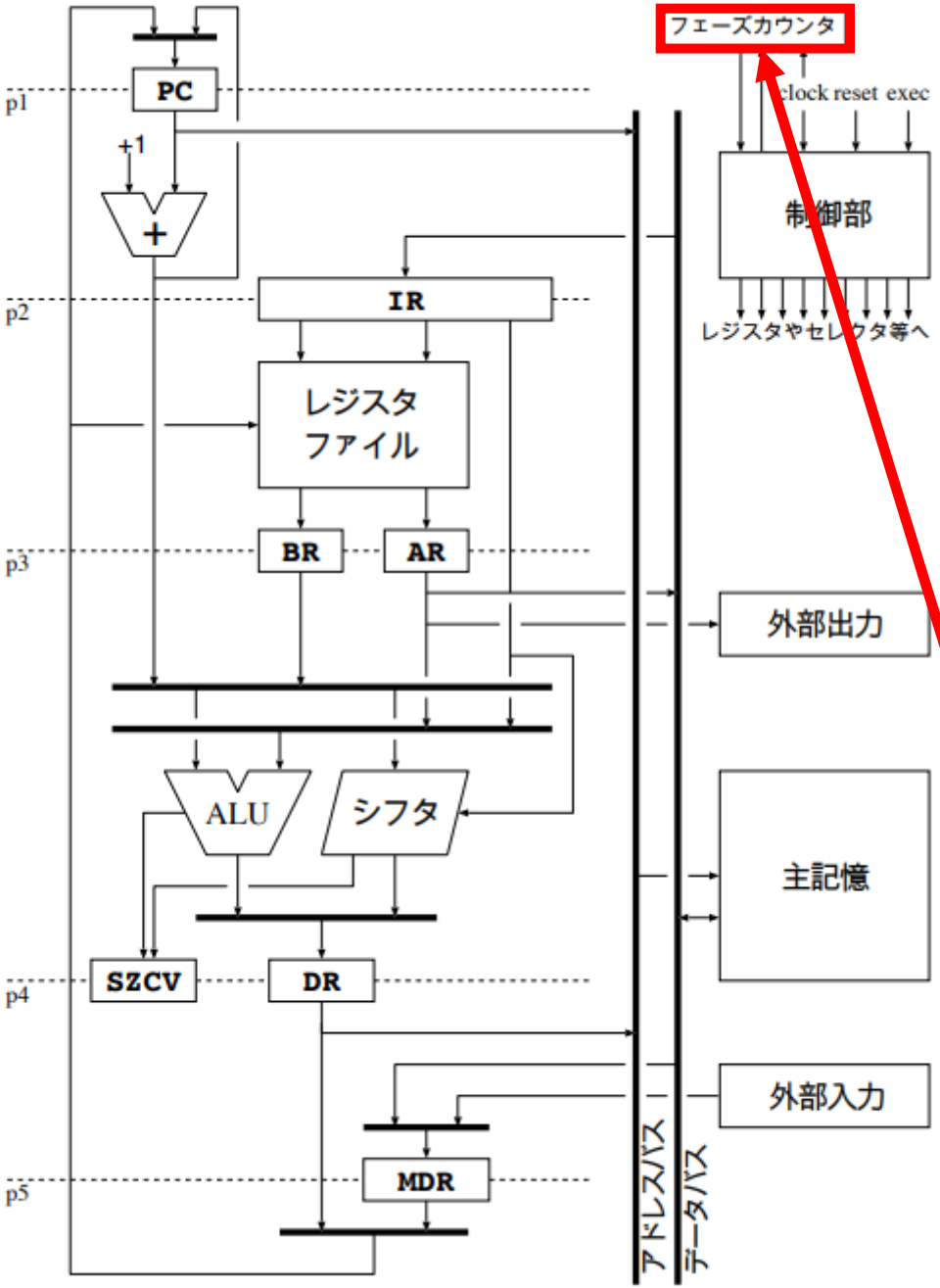
- メインメモリ
- プログラムカウンタ
- **レジスタファイル**
- ALU
- レジスタ
  （IR、AR・BR、DR、MDR）
  各フェーズでの値を保持
- フェーズカウンタ
  - フェーズの活性化信号を生成
- 制御回路
  - マルチプレクサを切り替える
  - レジスタのenable信号
  - ALUの機能の選択
  - などなど



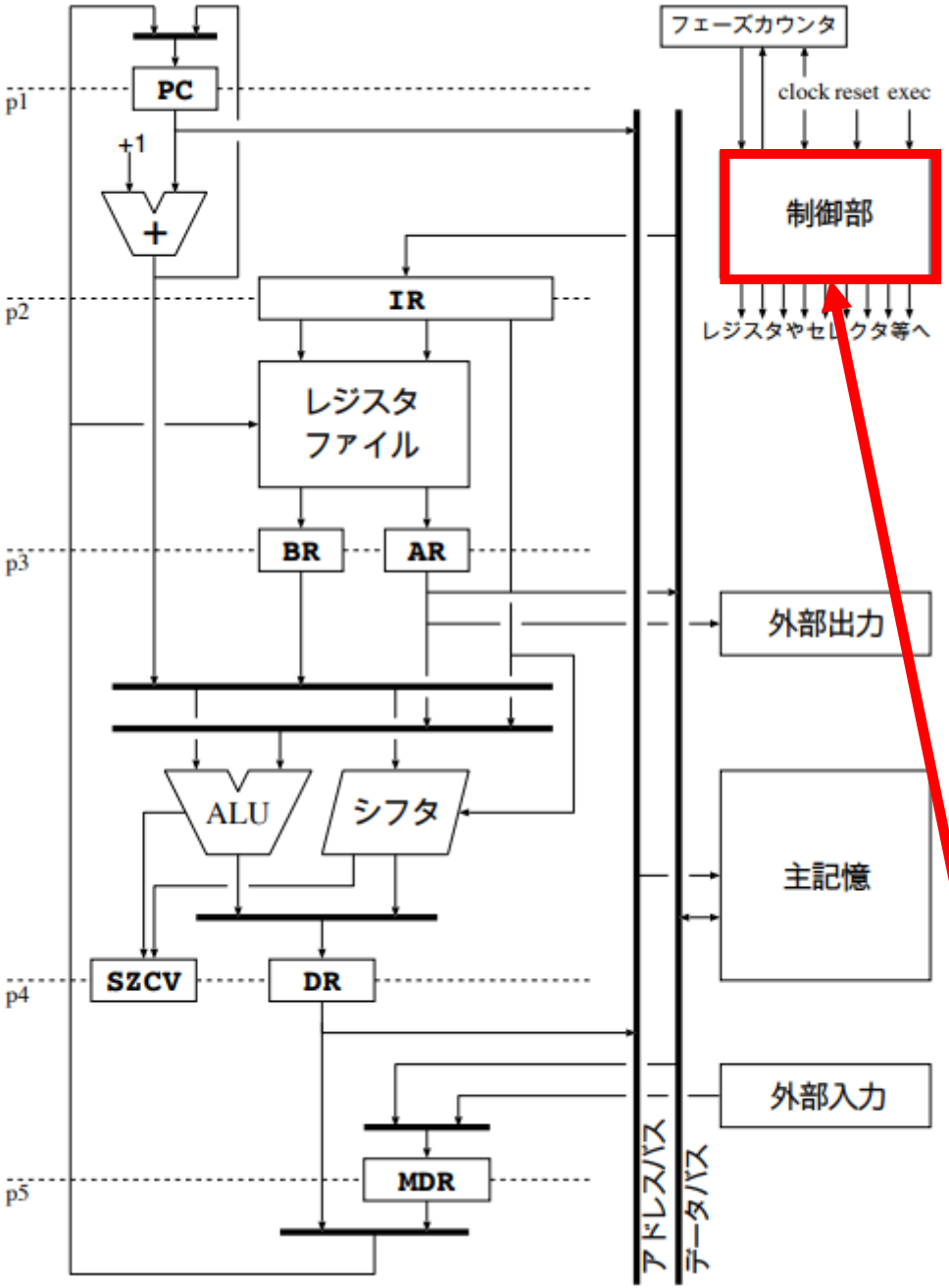
- メインメモリ
- プログラムカウンタ
- レジスタファイル
- ALU シフタ
- レジスタ
  （IR、AR・BR、DR、MDR）
  各フェーズでの値を保持
- フェーズカウンタ
  - フェーズの活性化信号を生成
- 制御回路
  - マルチプレクサを切り替える
  - レジスタのenable信号
  - ALUの機能の選択
  - などなど



- メインメモリ
- プログラムカウンタ
- レジスタファイル
- ALU
- **レジスタ**
  （IR、AR・BR、DR、MDR）
  各フェーズでの値を保持
- フェーズカウンタ
  - フェーズの活性化信号を生成
- 制御回路
  - マルチプレクサを切り替える
  - レジスタのenable信号
  - ALUの機能の選択
  - などなど

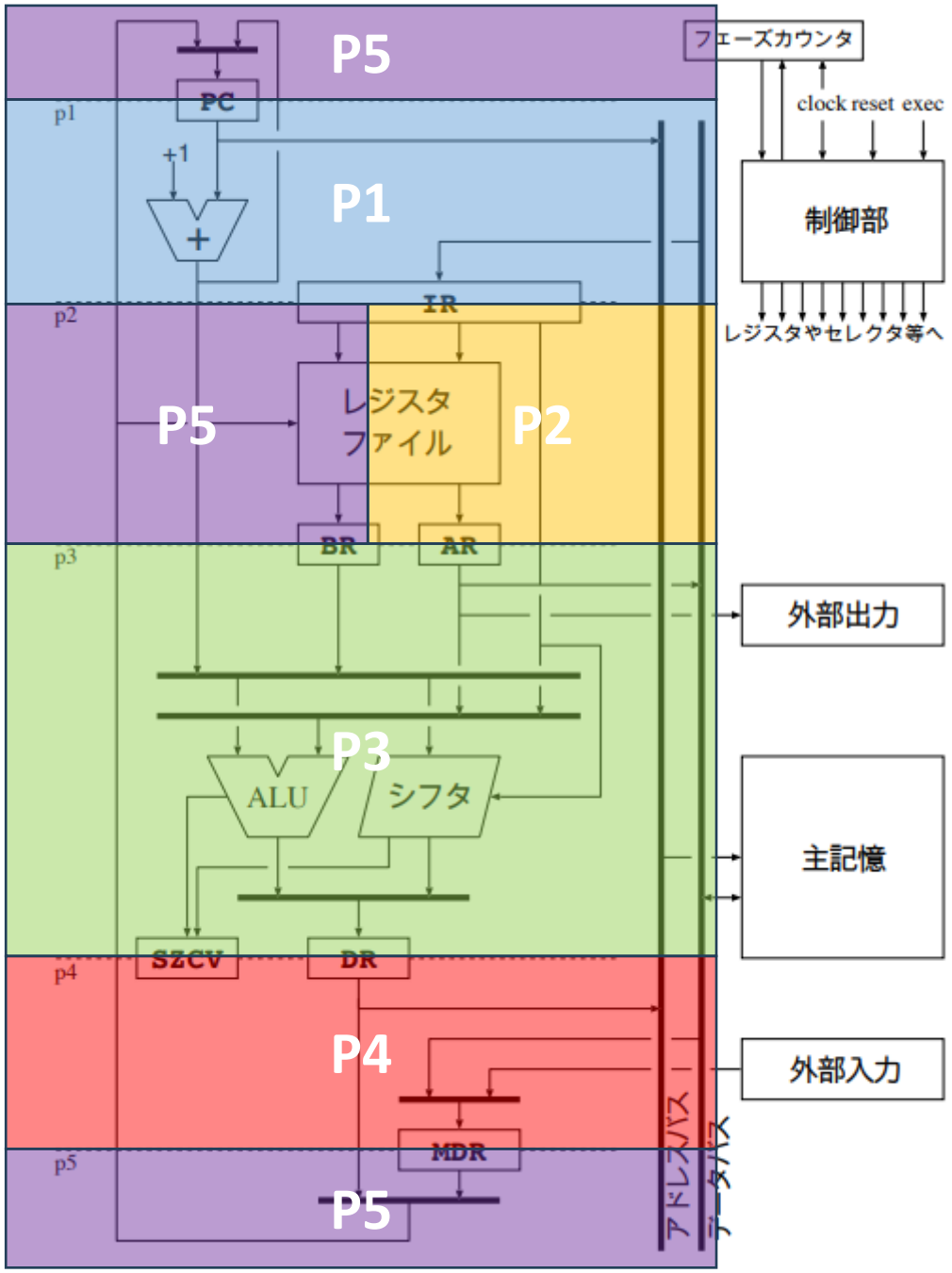


- メインメモリ
- プログラムカウンタ
- レジスタファイル
- ALU
- レジスタ
  （IR、AR・BR、DR、MDR）
  各フェーズでの値を保持
- **フェーズカウンタ**
  - **フェーズの活性化信号を生成**
- 制御回路
  - マルチプレクサを切り替える
  - レジスタのenable信号
  - ALUの機能の選択
  - などなど



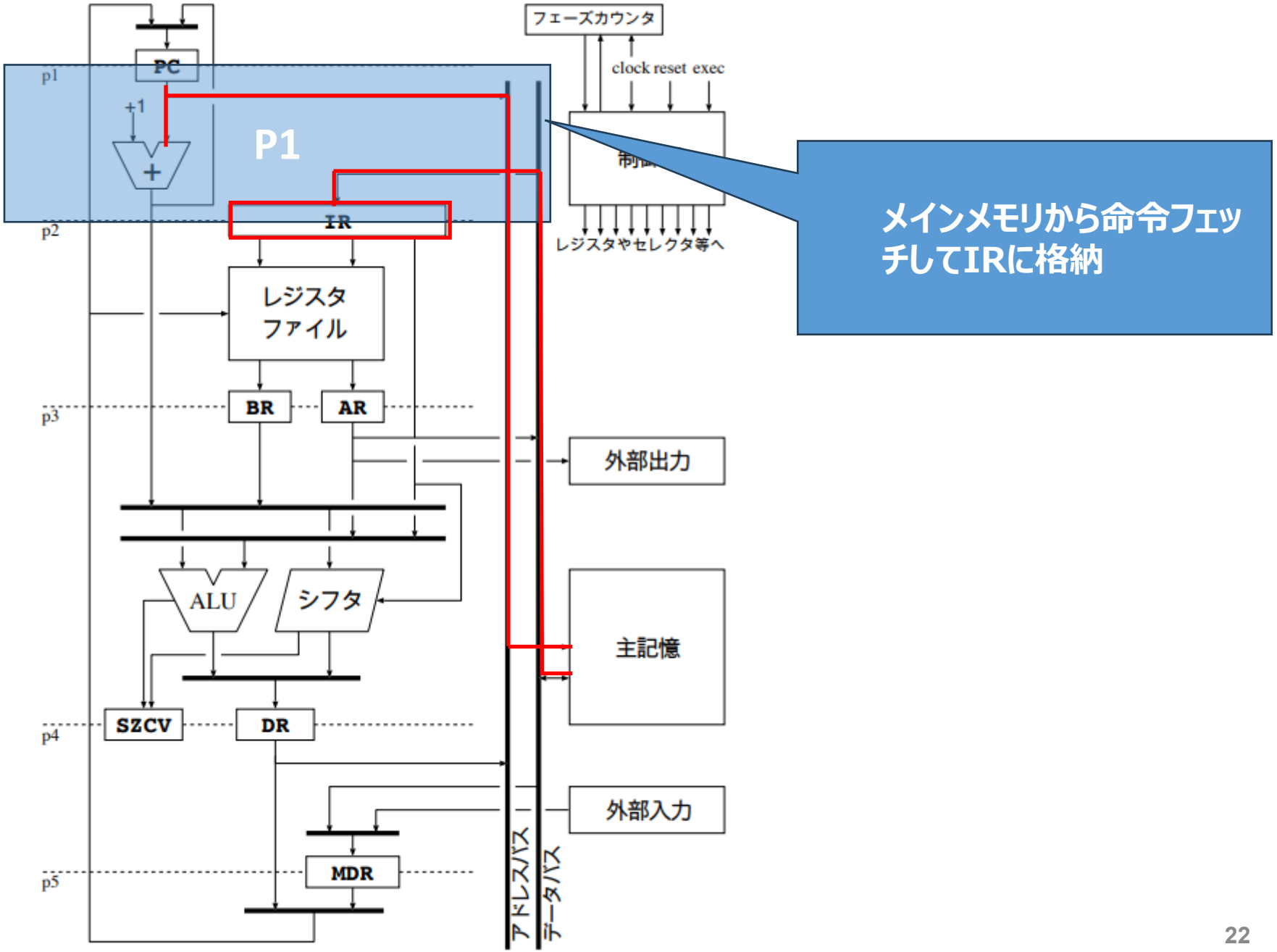
- メインメモリ
- プログラムカウンタ
- レジスタファイル
- ALU
- レジスタ
  （IR、AR・BR、DR、MDR）
  各フェーズでの値を保持
- フェーズカウンタ
  - フェーズの活性化信号を生成

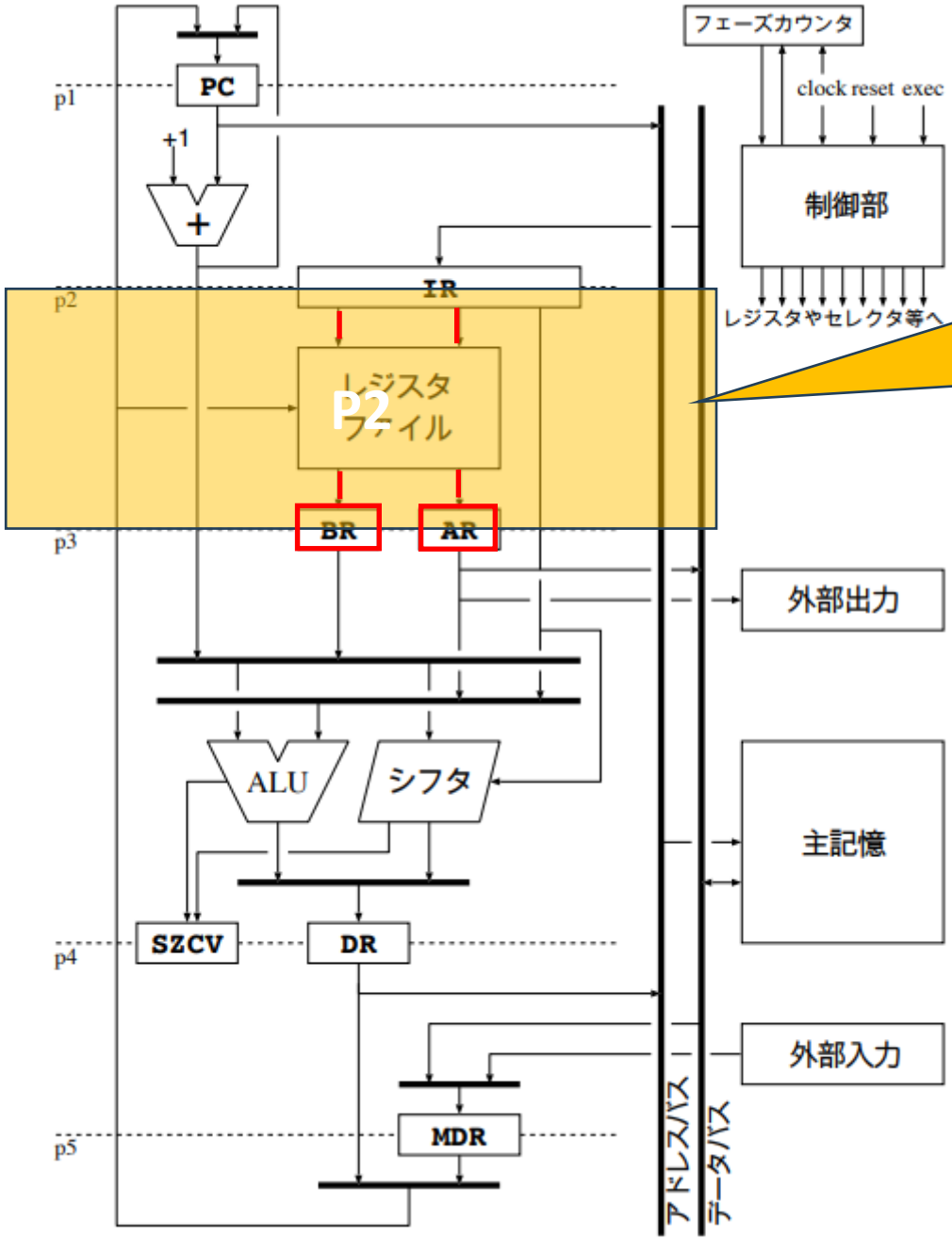
- 制御回路
  - マルチプレクサを切り替える
  - レジスタのenable信号
  - ALUの機能の選択
  - などなど



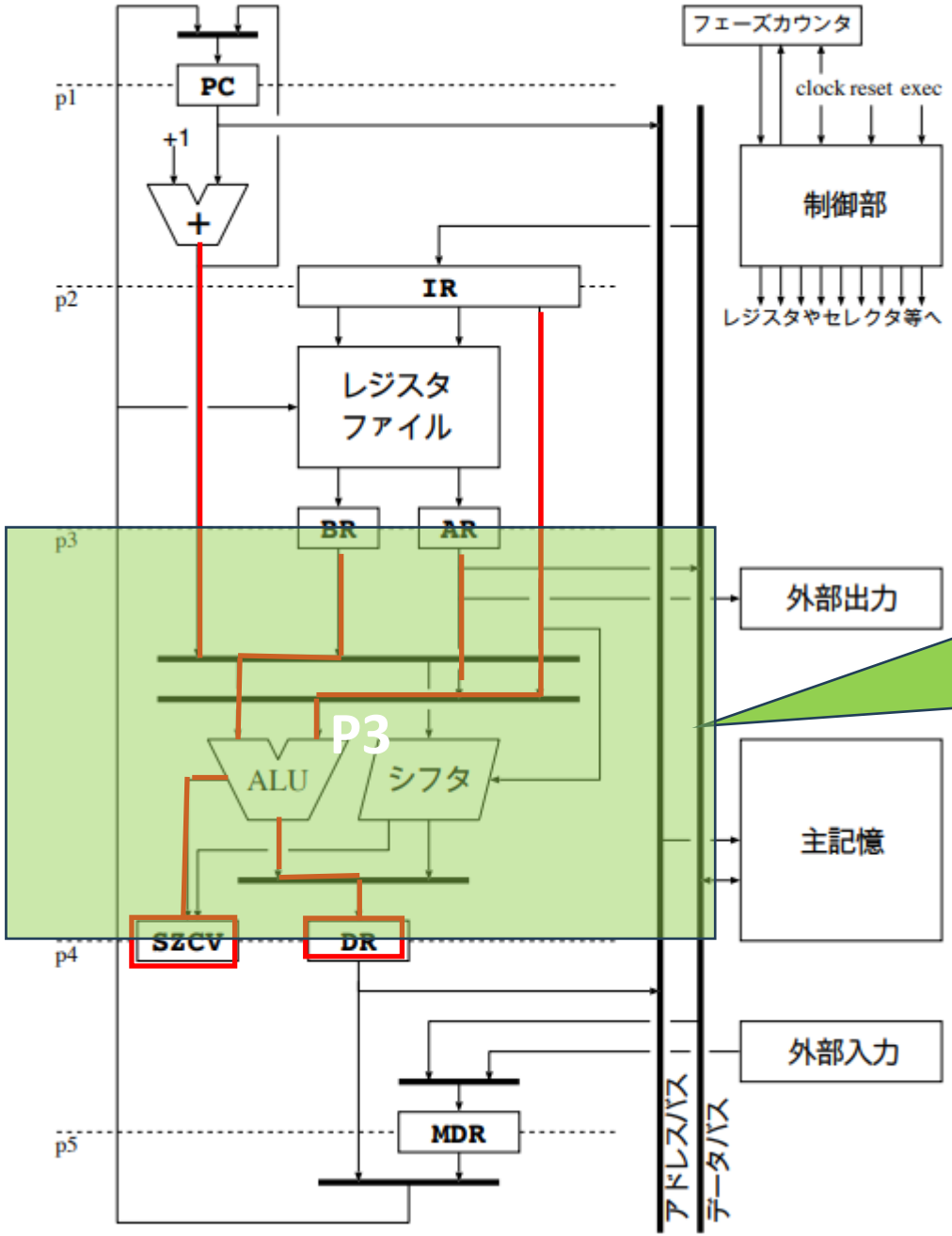
命令実行の過程を5つのフェーズに分割。逐次活性化

- P1:メインメモリから命令フェッチしてIRに格納
- P2:制御回路による命令デコード。レジスタからデータを読み出しAR, BRに格納
- P3:演算に必要なデータを選択しALUにより演算。結果をDRに格納
- P4:LD、STの場合はメインメモリにアクセス。値をMRDに格納
- P5:DR/MDRから必要なデータを選択しレジスタに書き込み/PC更新

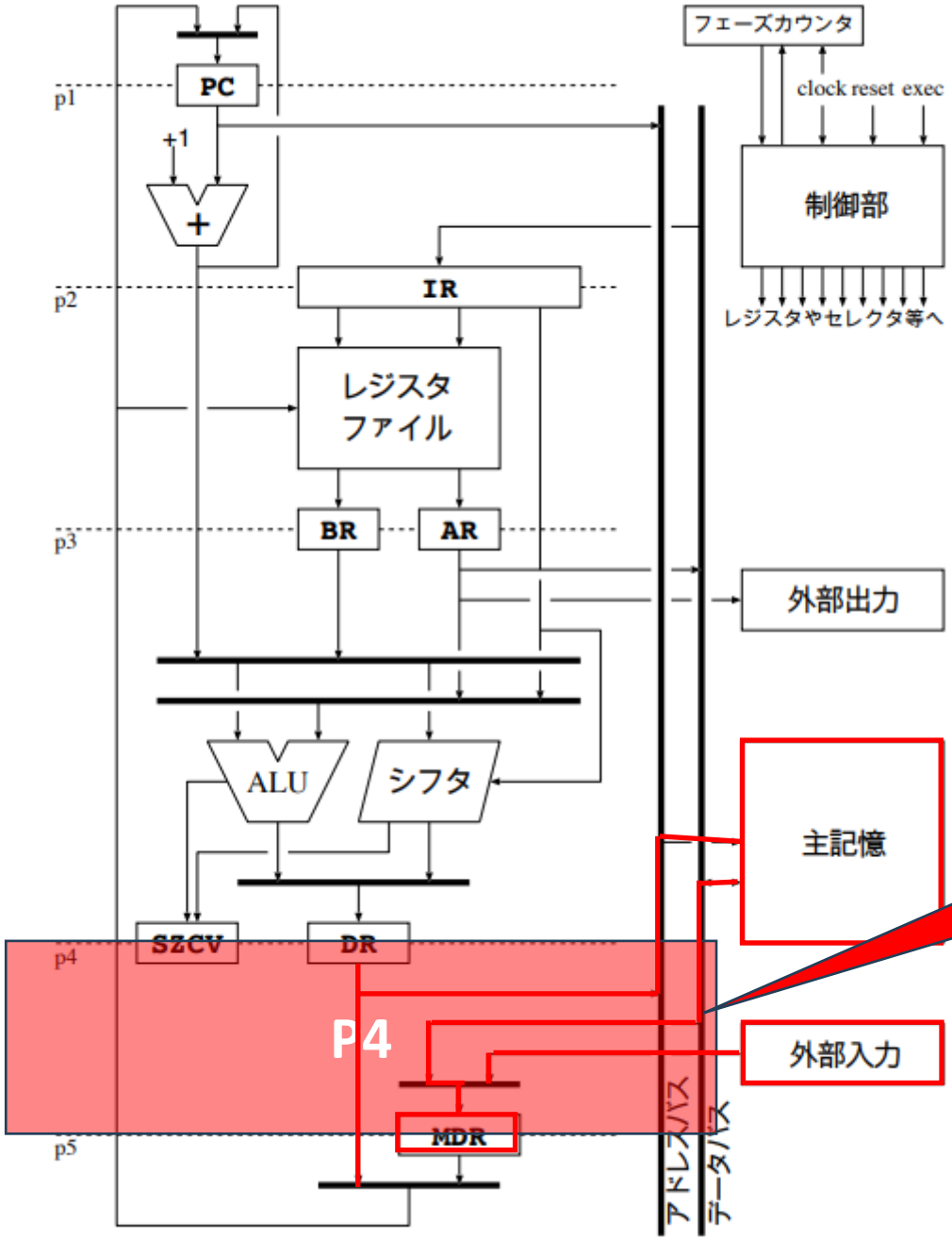

フェーズカウンタ
clock reset exec
制御
レジスタやセレクタ等へ
p1
PC
+1
+
P1
IR
p2
レジスタ
ファイル
BR
AR
p3
外部出力
ALU
シフタ
主記憶
SZCV
DR
p4
外部入力
MDR
p5
アドレスバス
データバス
メインメモリから命令フェッチしてIRに格納



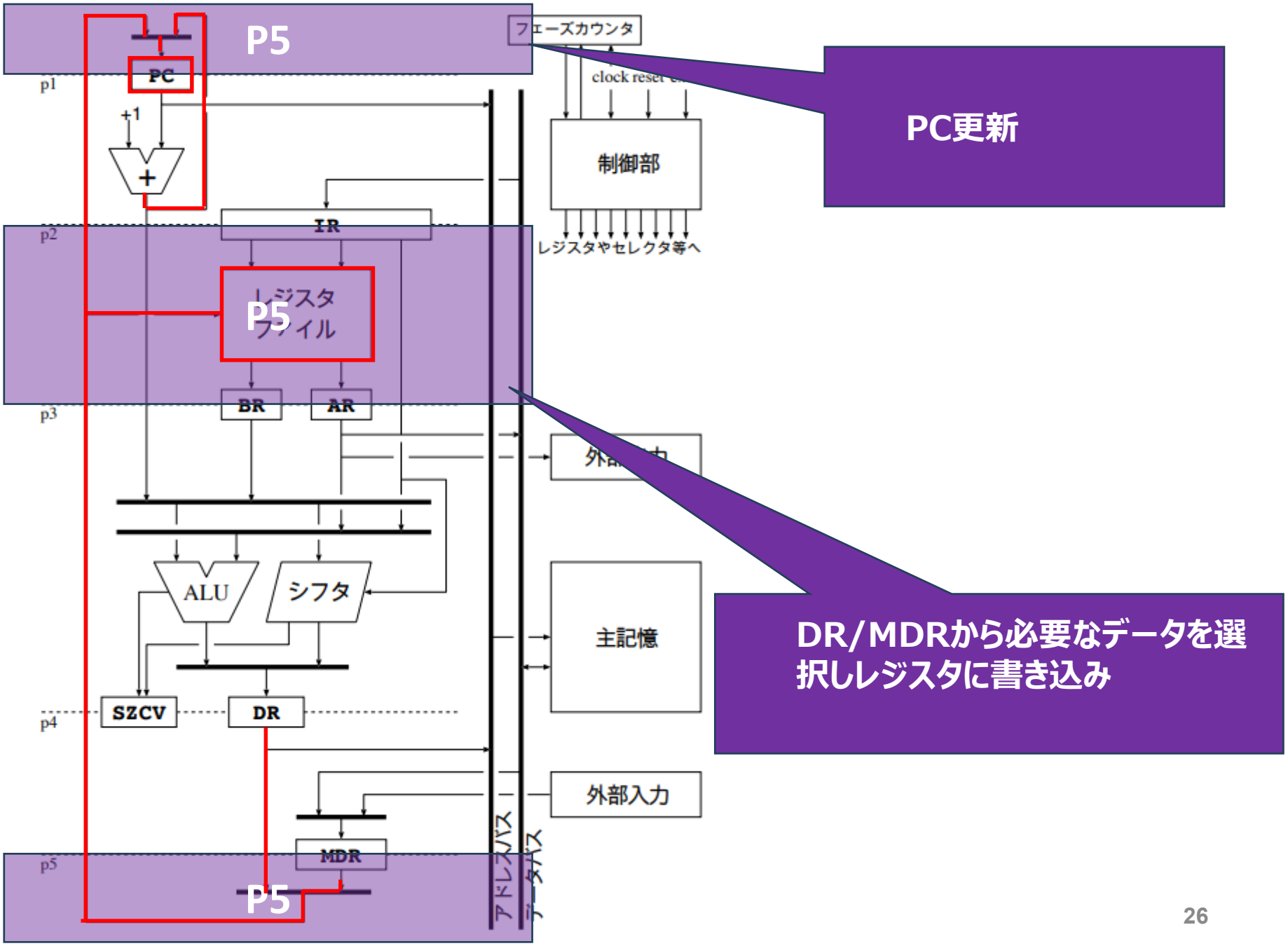
- 制御回路による命令デコード
- レジスタからデータを読み出しAR, BRに格納



- 演算に必要なデータを選択しALUにより演算。
- 結果をDRに格納



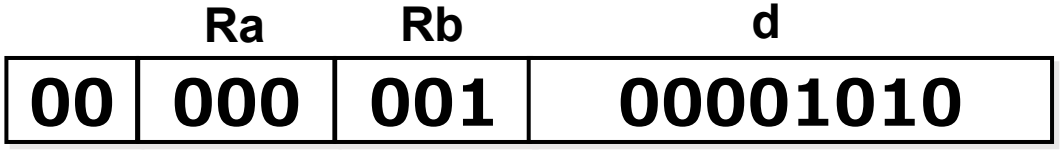
メモリへのアクセス、または外部入力からの取得。
値をMRDに格納


P5
PC
p1
+1
+
IR
p2
レジスタ
ファイル
P5
BR
AR
p3
ALU
シフタ
SZCV
DR
p4
MDR
p5
P5
フェーズカウンタ
clock reset
制御部
レジスタやセレクタ等へ
外部入力
主記憶
外部入力
アドレスバス
データバス
PC更新
DR/MDRから必要なデータを選択しレジスタに書き込み

命令実行の例

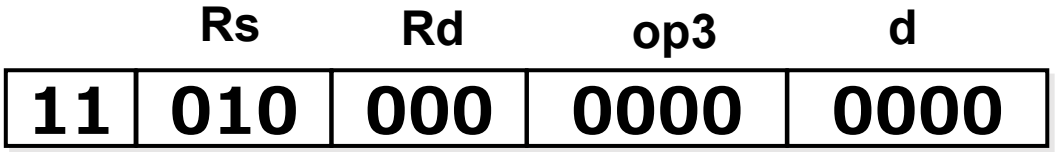
- ロード命令: プログラムカウンタ100
  - LD R0, 10(R1)

略記 | 0 | 0 | 1 | 10 |



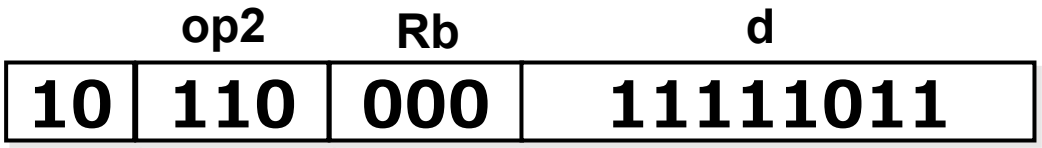
- 加算命令： プログラムカウンタ101
  - ADD R0, R2

略記 | 3 | 2 | 0 | 0 | 0 |

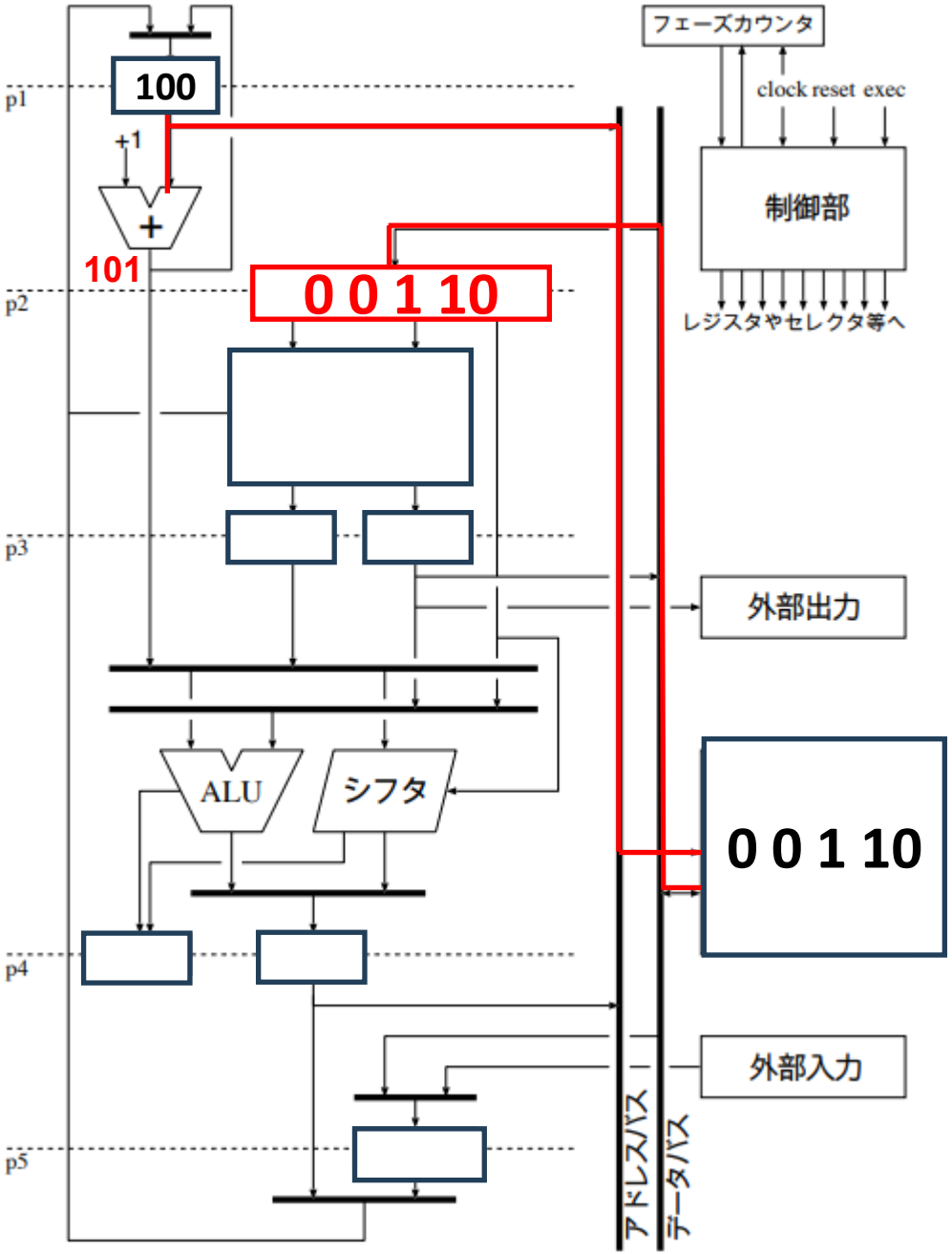


- 無条件分岐命令: プログラムカウンタ102
  - B -5

略記 | 2 | 6 | 0 | -5 |



LD R0, 10(R1)　P1（命令フェッチ）



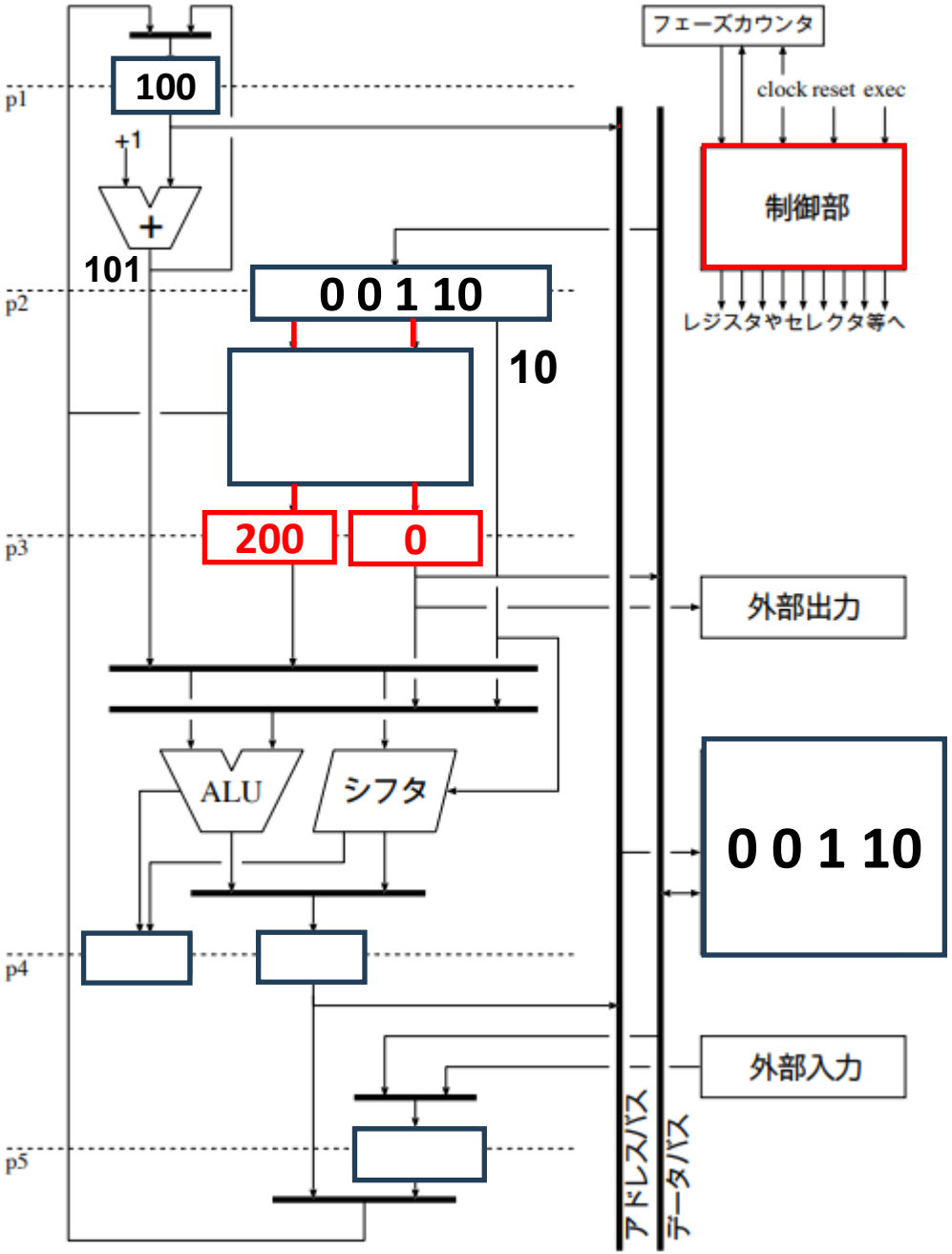
0	0
1	200
2	5
…	…
6	0
7	0

レジスタファイルの中身

…	…				
100	0	0	1	10	
101	3	2	0	0	0
102	2	6	0	-5	
…	…				
210	1000				

メインメモリの中身

LD R0, 10(R1)　P2（命令デコード、レジスタ読み出し）



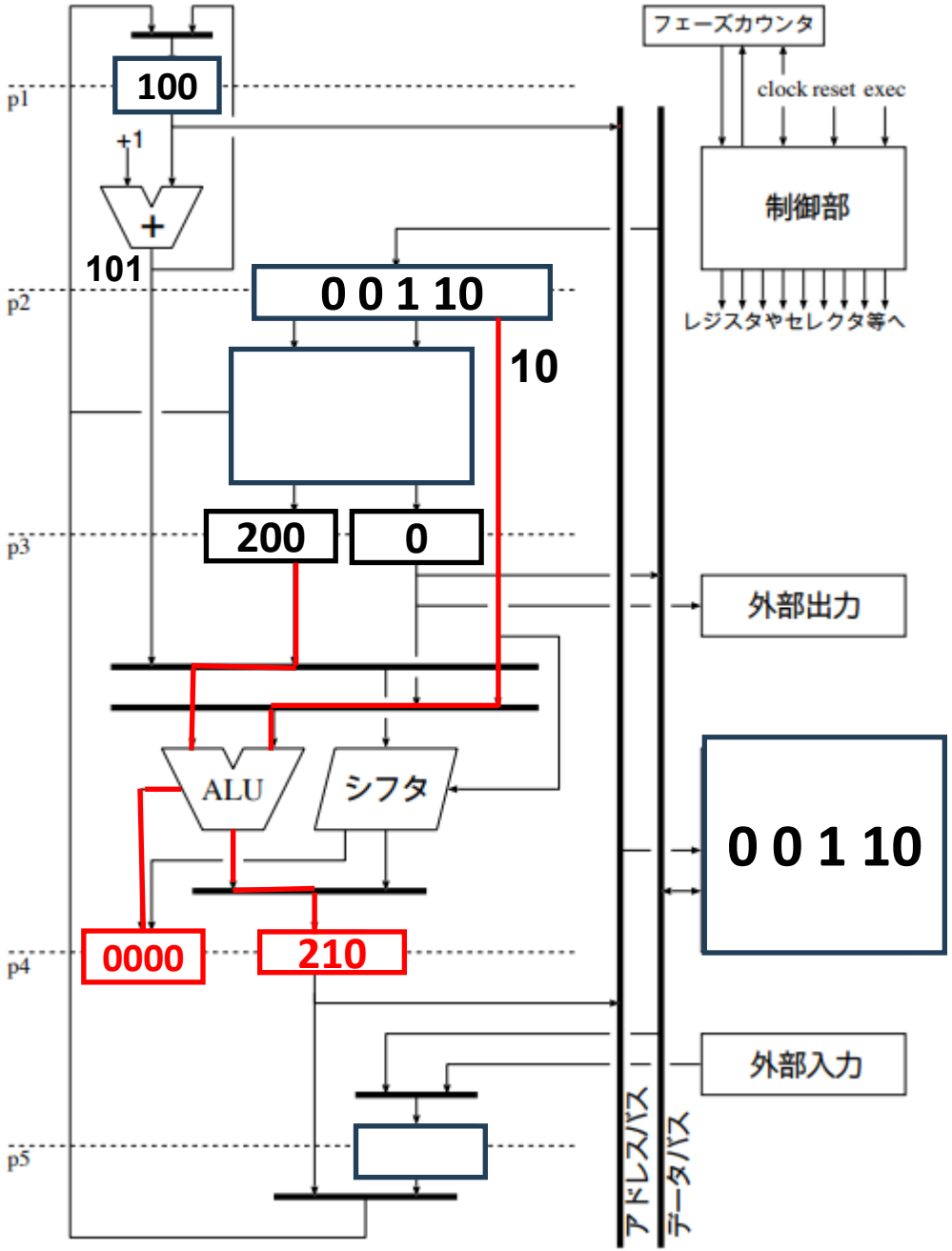
0	0
1	200
2	5
…	…
6	0
7	0

レジスタファイルの中身

…	…				
100	0	0	1	10	
101	3	2	0	0	0
102	2	6	0	-5	
…	…				
210	1000				

メインメモリの中身

LD R0, 10(R1)　P3（演算）



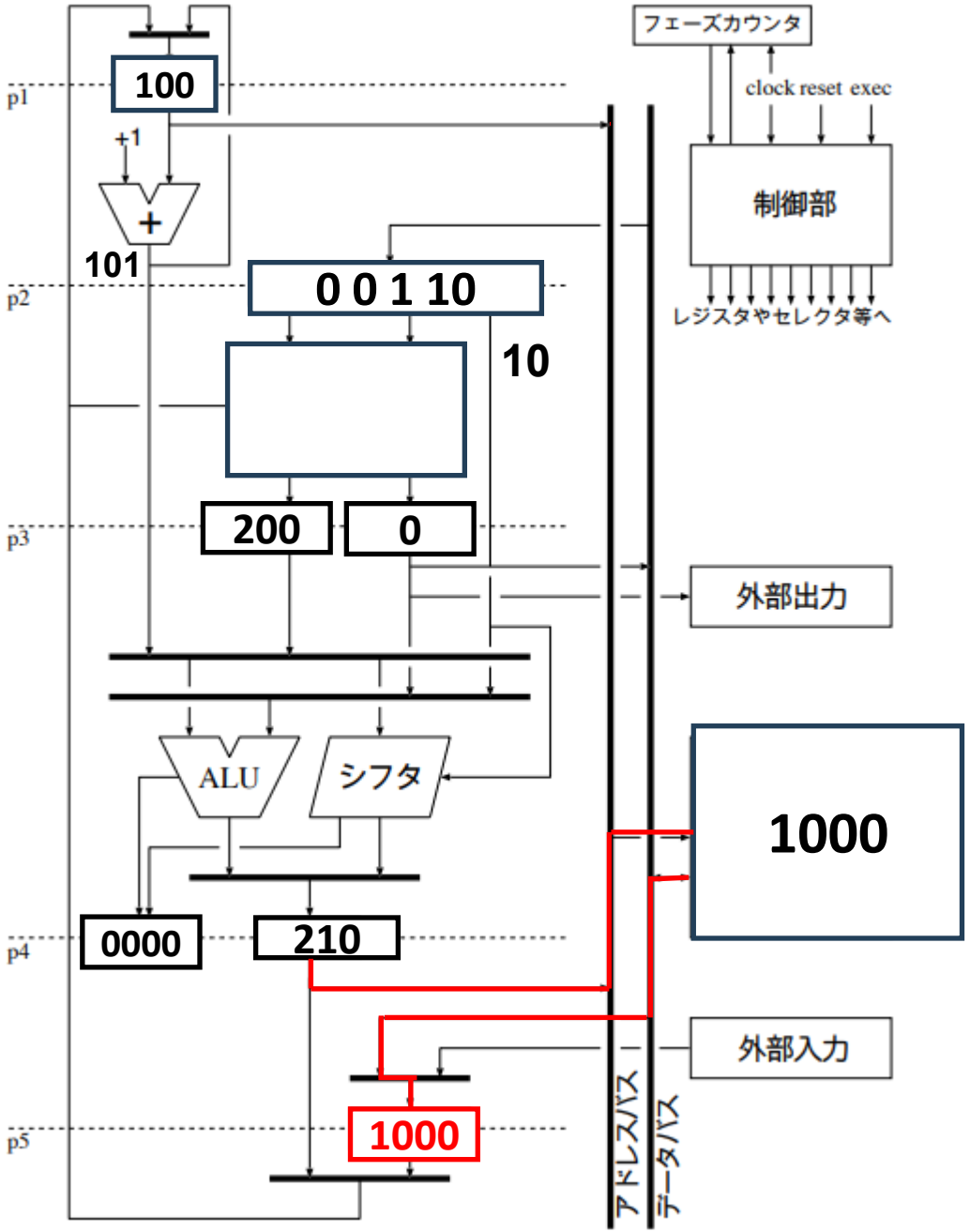
0	0
1	200
2	5
…	…
6	0
7	0

レジスタファイルの中身

…	…				
100	0	0	1	10	
101	3	2	0	0	0
102	2	6	0	-5	
…	…				
210	1000				

メインメモリの中身

LD R0, 10(R1)　P4（メモリアクセス）



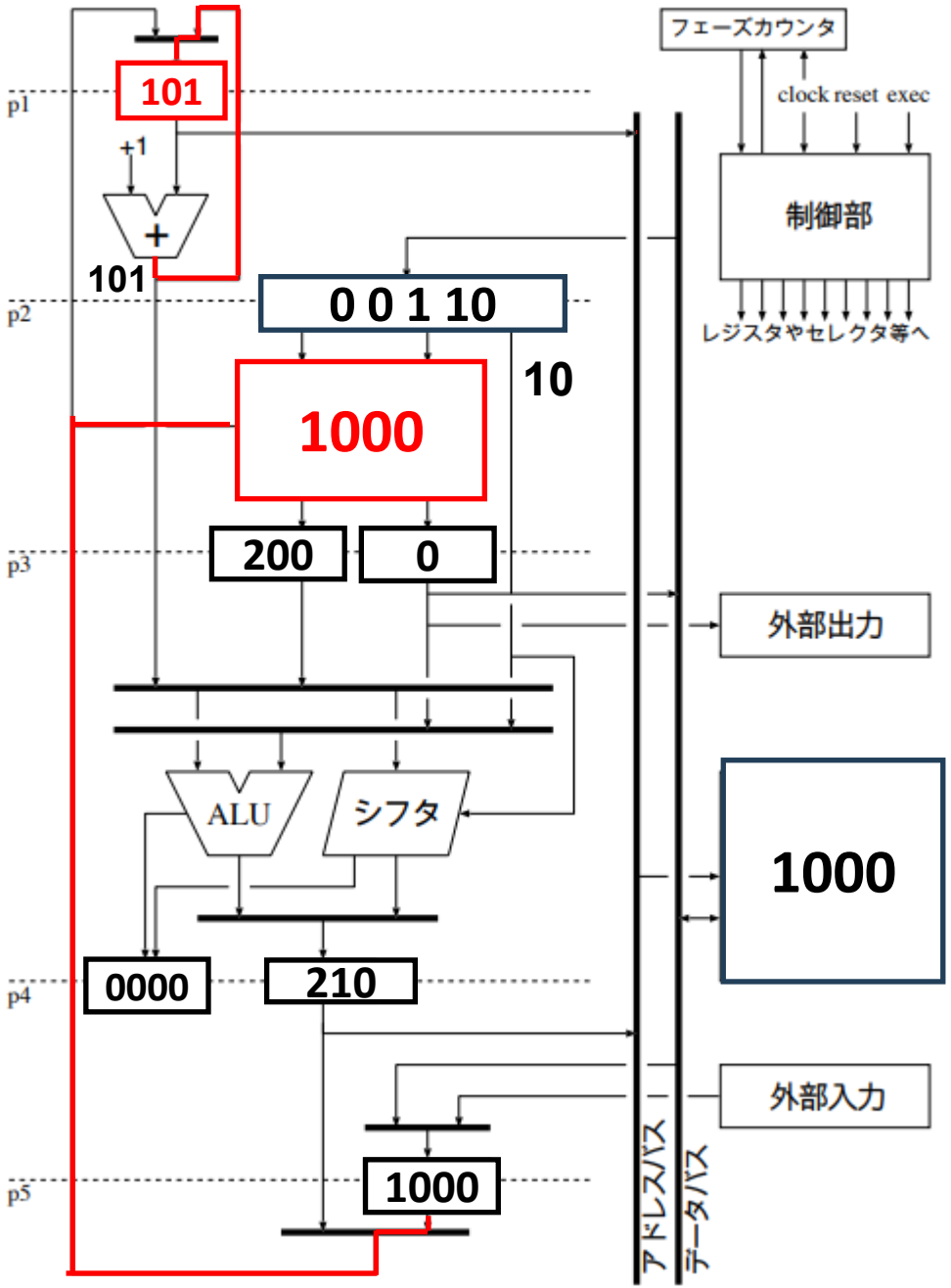
0	0
1	200
2	5
…	…
6	0
7	0

レジスタファイルの中身

…	…				
100	0	0	1	10	
101	3	2	0	0	0
102	2	6	0	-5	
…	…				
210	1000				

メインメモリの中身

LD R0, 10(R1)　P5（レジスタ書き込み、PC更新）



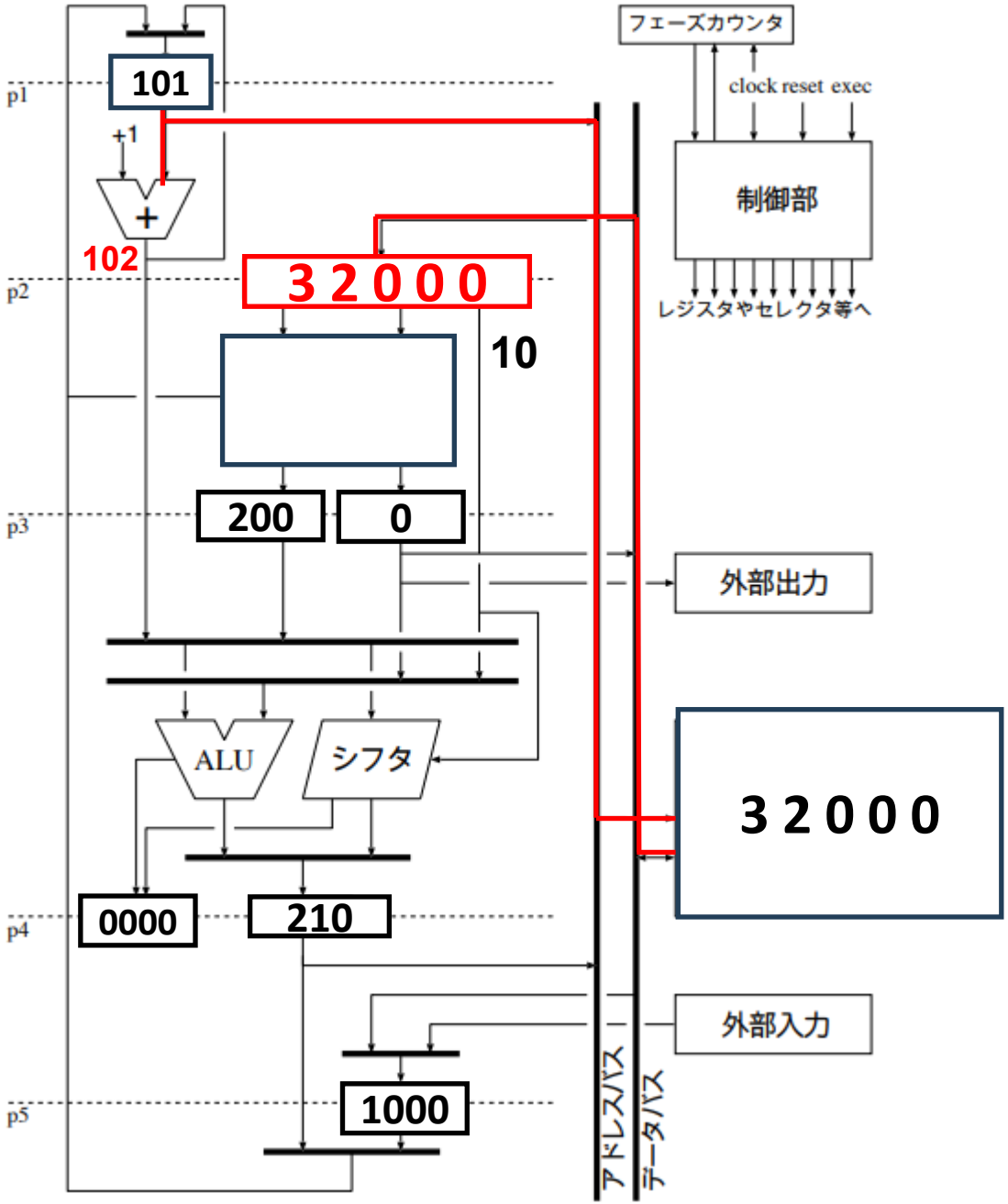
0	1000
1	200
2	5
…	…
6	0
7	0

レジスタファイルの中身

…	…				
100	0	0	1	10	
101	3	2	0	0	0
102	2	6	0	-5	
…	…				
210	1000				

メインメモリの中身

ADD R0, R2　P1（命令フェッチ）



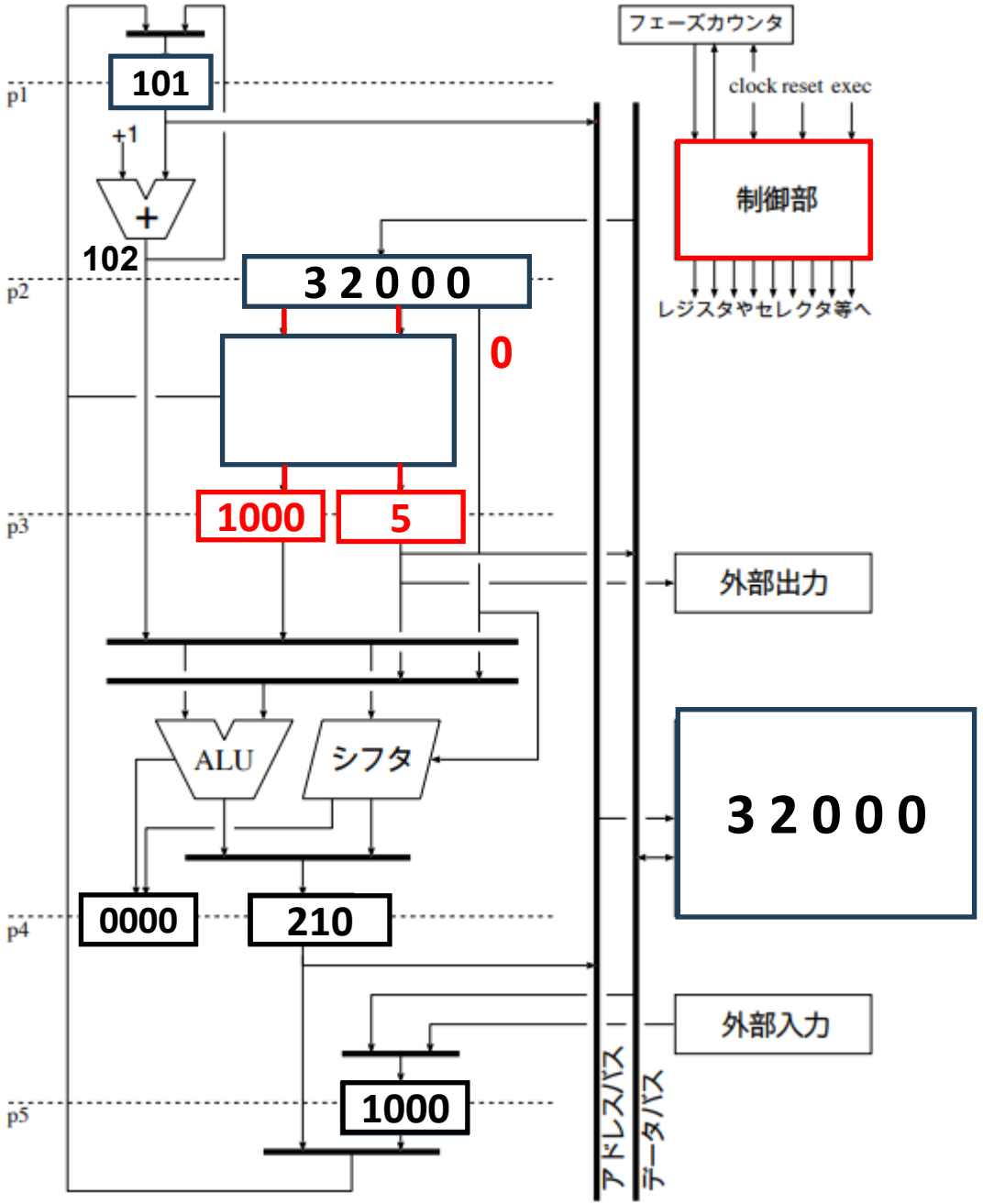
0	1000
1	200
2	5
…	…
6	0
7	0

レジスタファイルの中身

…	…				
100	0	0	1	10	
101	3	2	0	0	0
102	2	6	0	-5	
…	…				
210	1000				

メインメモリの中身

ADD R0, R2　P2（命令デコード、レジスタ読み出し）



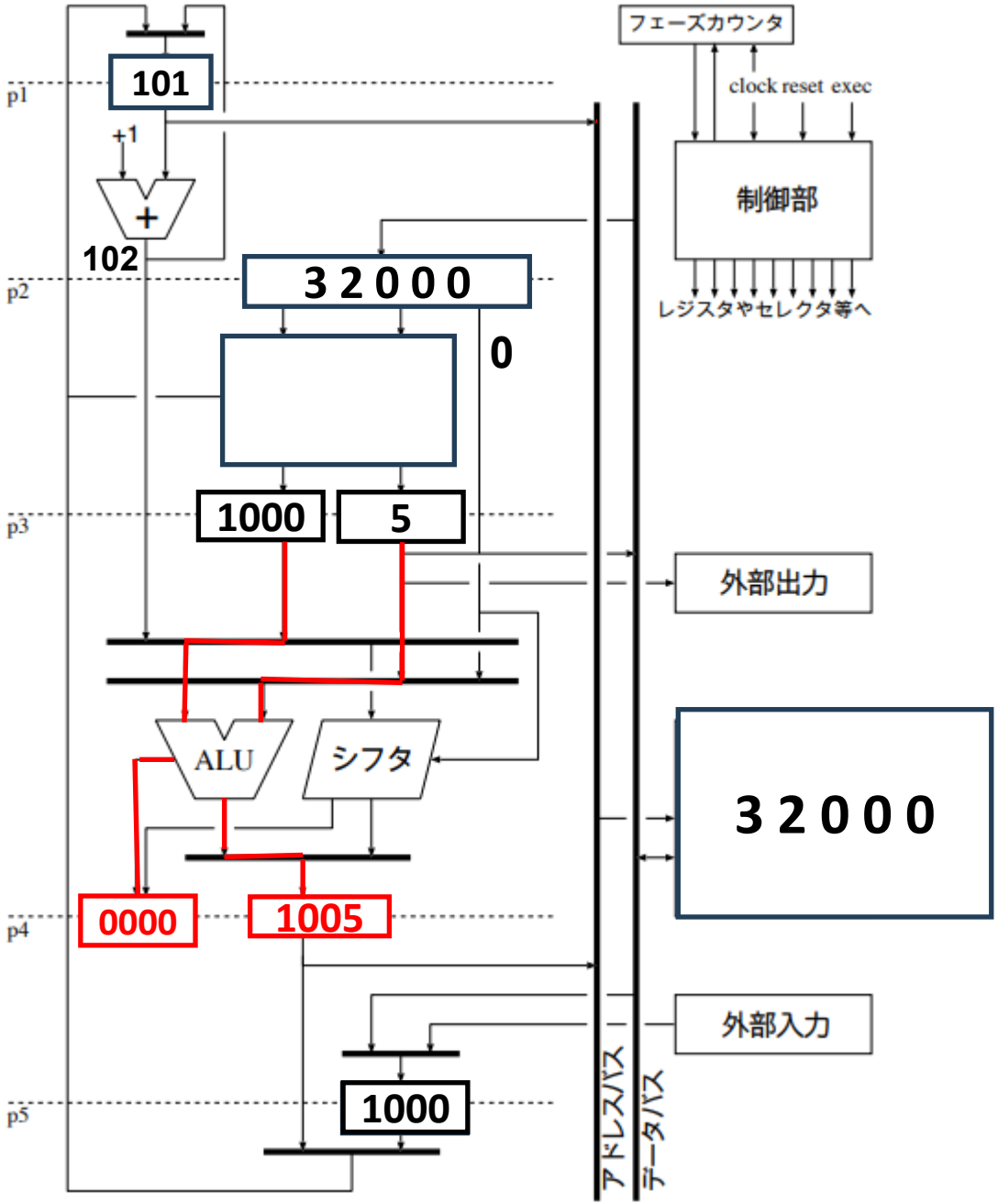
0	1000
1	200
2	5
…	…
6	0
7	0

レジスタファイルの中身

…	…				
100	0	0	1	10	
101	3	2	0	0	0
102	2	6	0	-5	
…	…				
210	1000				

メインメモリの中身

ADD R0, R2　P3（演算）



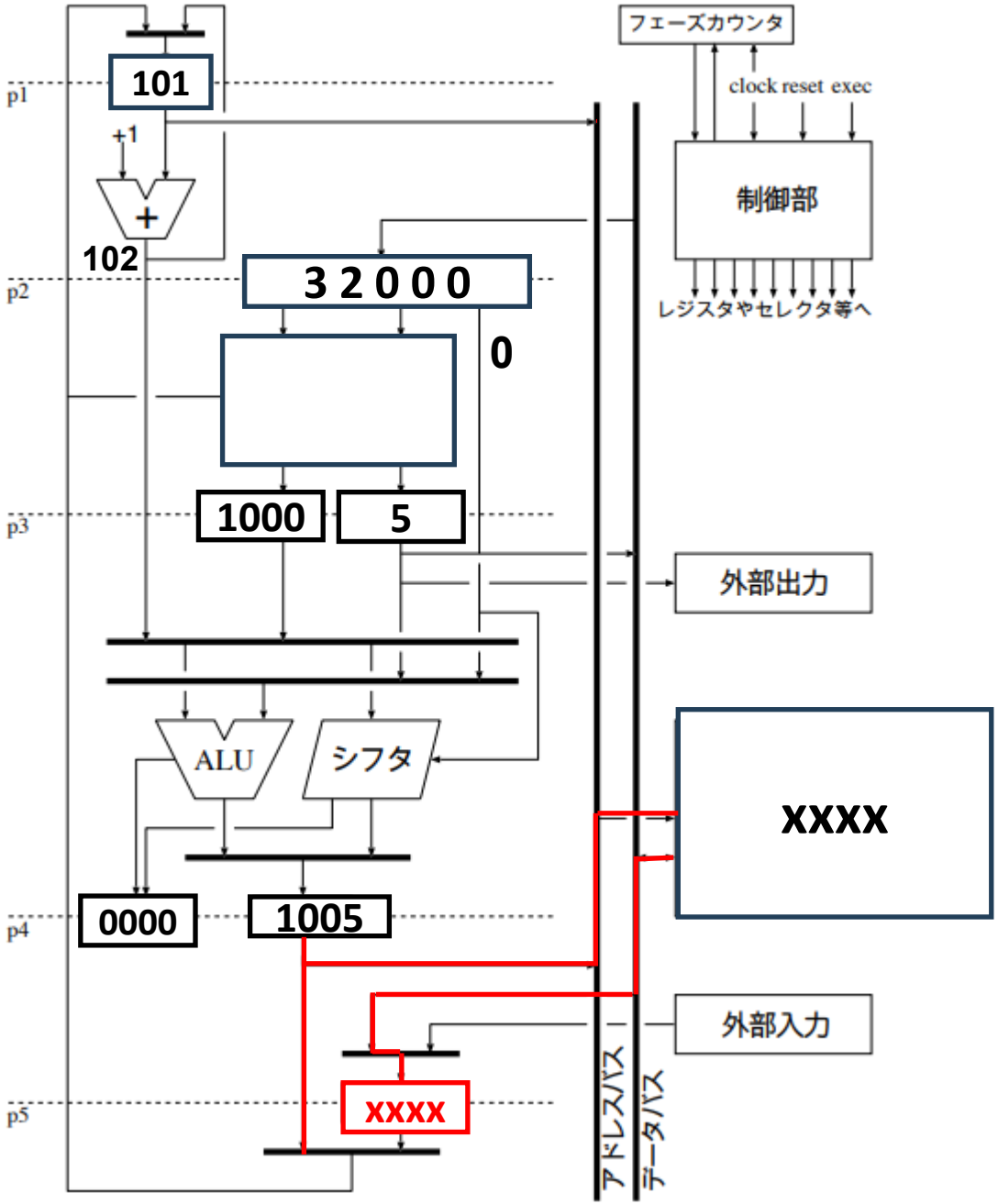
0	1000
1	200
2	5
…	…
6	0
7	0

レジスタファイルの中身

…	…				
100	0	0	1	10	
101	3	2	0	0	0
102	2	6	0	-5	
…	…				
210	1000				

メインメモリの中身

ADD R0, R2　P4（メモリアクセス）



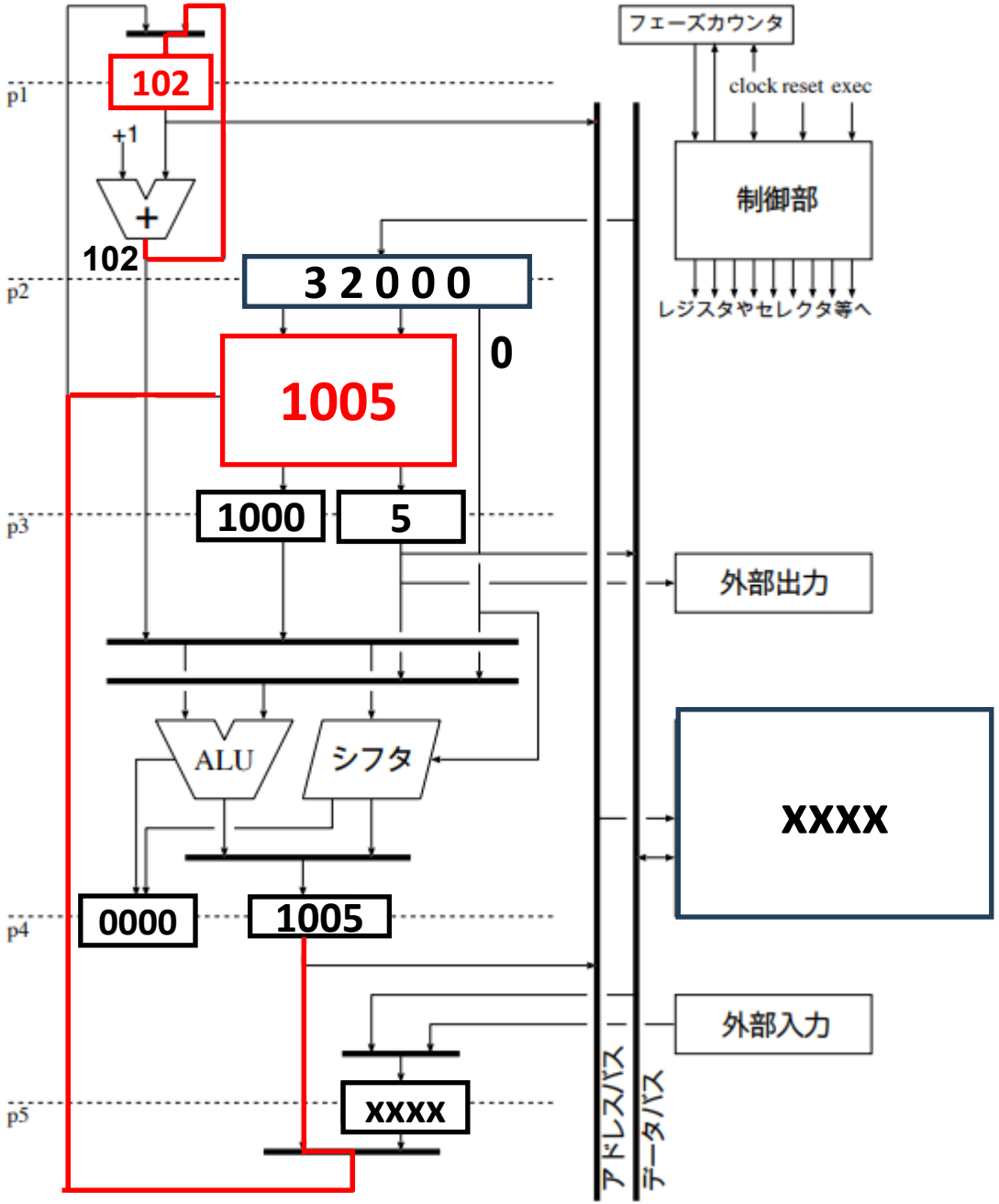
0	1000
1	200
2	5
…	…
6	0
7	0

レジスタファイルの中身

…	…				
100	0	0	1	10	
101	3	2	0	0	0
102	2	6	0	-5	
…	…				
210	1000				

メインメモリの中身

ADD R0, R2　P5（レジスタ書き込み、PC更新）



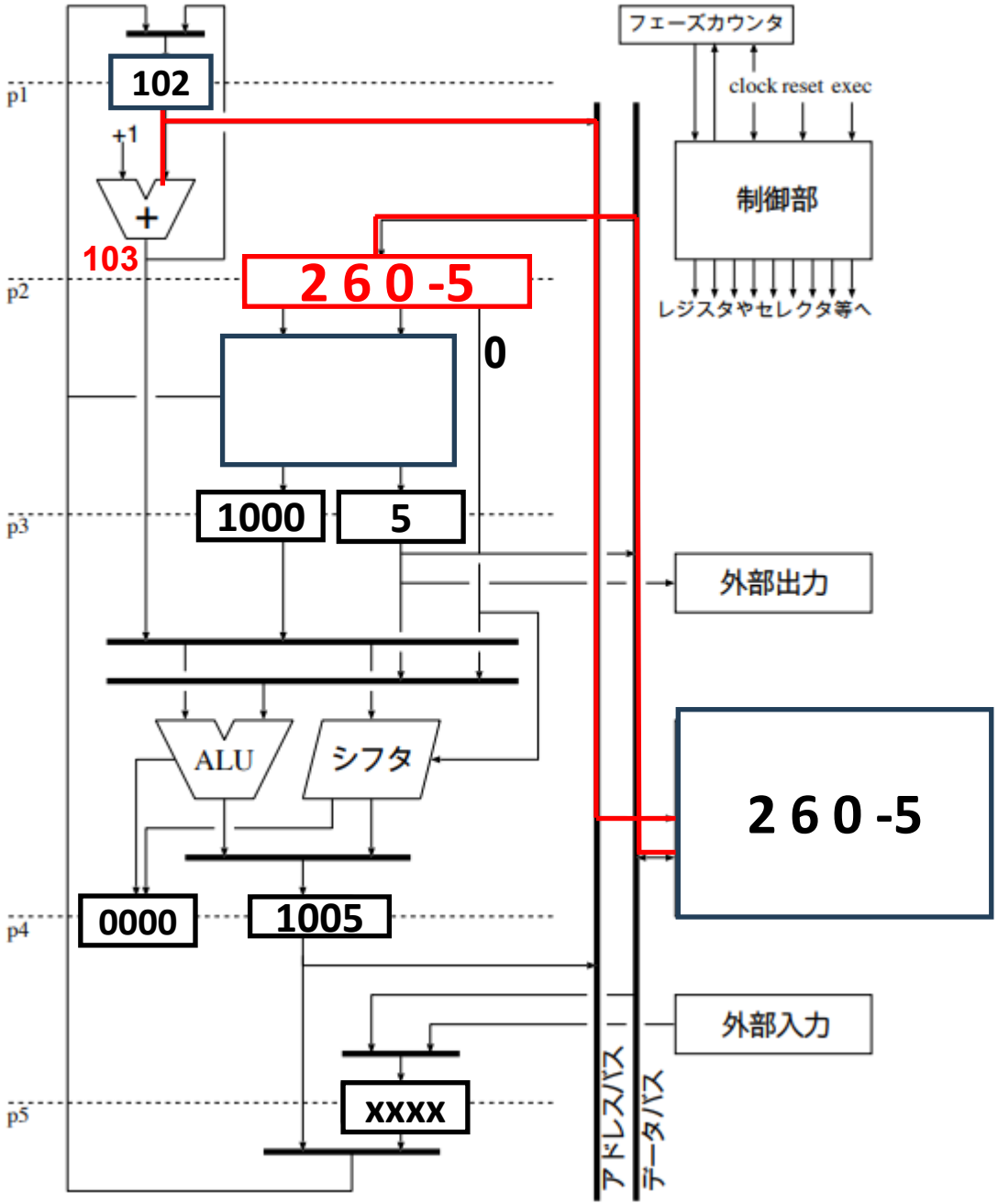
0	1005
1	200
2	5
…	…
6	0
7	0

レジスタファイルの中身

…	…				
100	0	0	1	10	
101	3	2	0	0	0
102	2	6	0	-5	
…	…				
210	1000				

メインメモリの中身

B　P1（命令フェッチ）



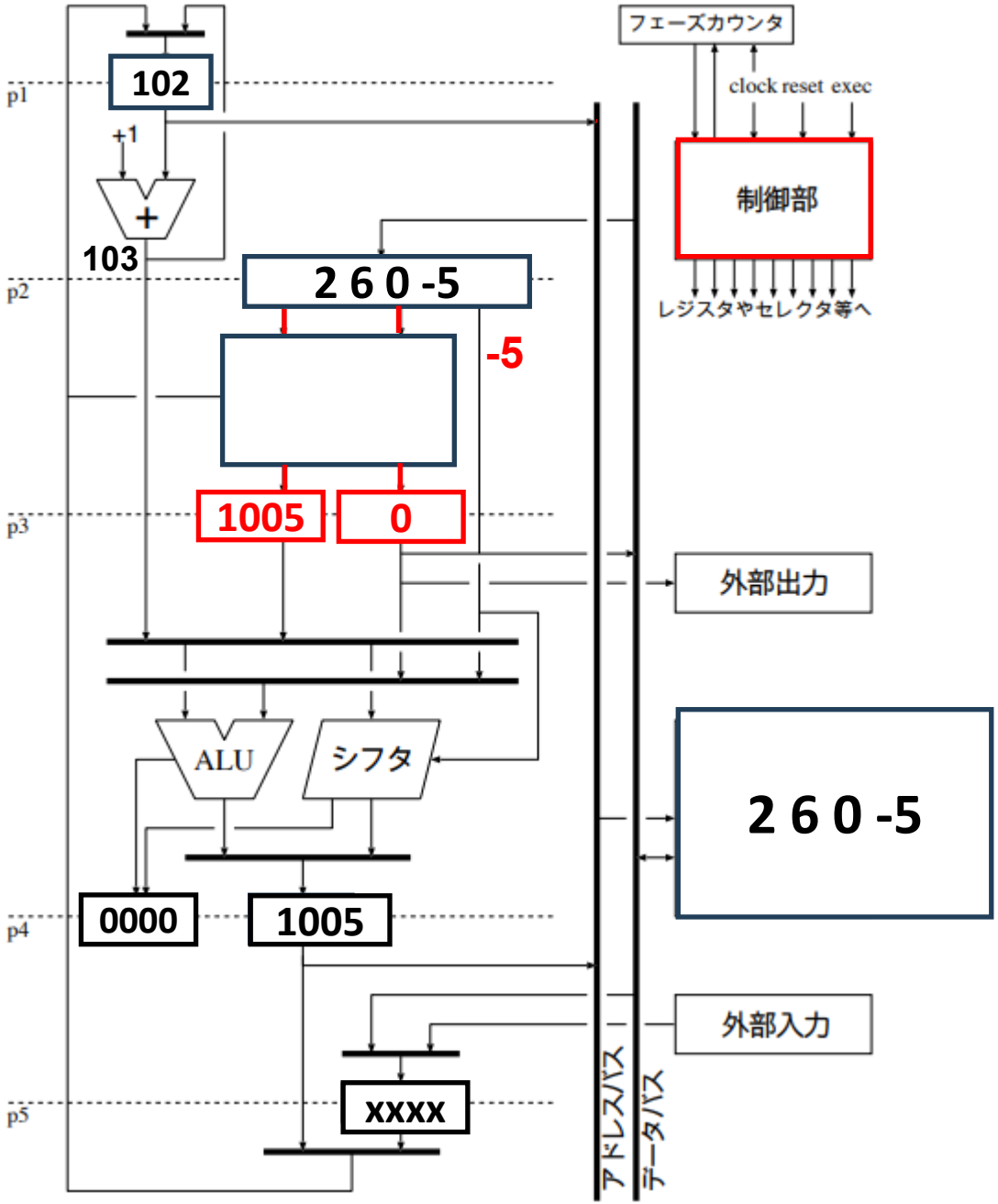
0	1005
1	200
2	5
…	…
6	0
7	0

レジスタファイルの中身

…	…				
100	0	0	1	10	
101	3	2	0	0	0
102	2	6	0	-5	
…	…				
210	1000				

メインメモリの中身

B-5　P2（命令デコード、レジスタ読み出し）



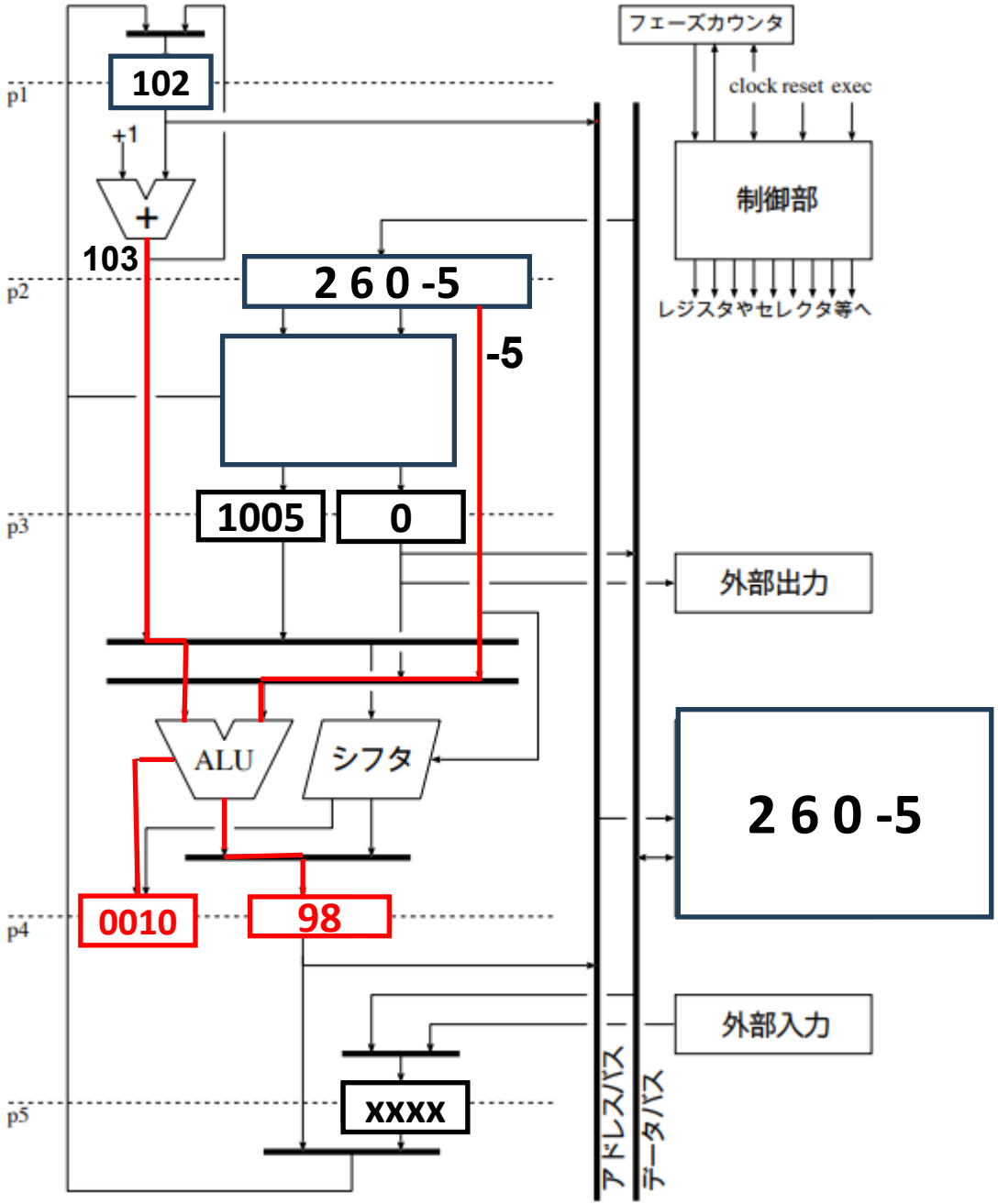
0	1005
1	200
2	5
…	…
6	0
7	0

レジスタファイルの中身

…	…				
100	0	0	1	10	
101	3	2	0	0	0
102	2	6	0	-5	
…	…				
210	1000				

メインメモリの中身

B-5　P3（演算）



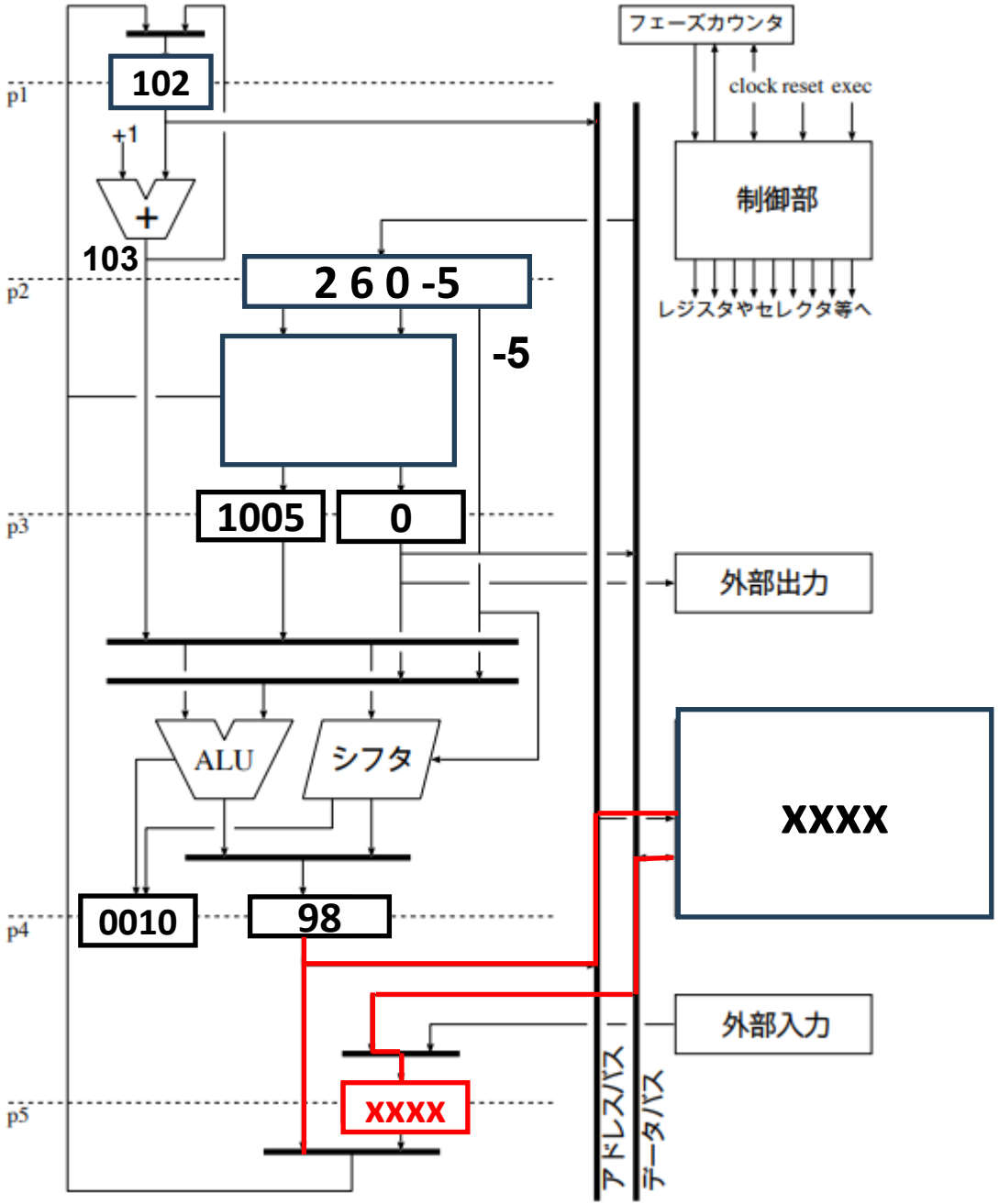
0	1005
1	200
2	5
…	…
6	0
7	0

レジスタファイルの中身

…	…				
100	0	0	1	10	
101	3	2	0	0	0
102	2	6	0	-5	
…	…				
210	1000				

メインメモリの中身

B-5　P4（メモリアクセス）



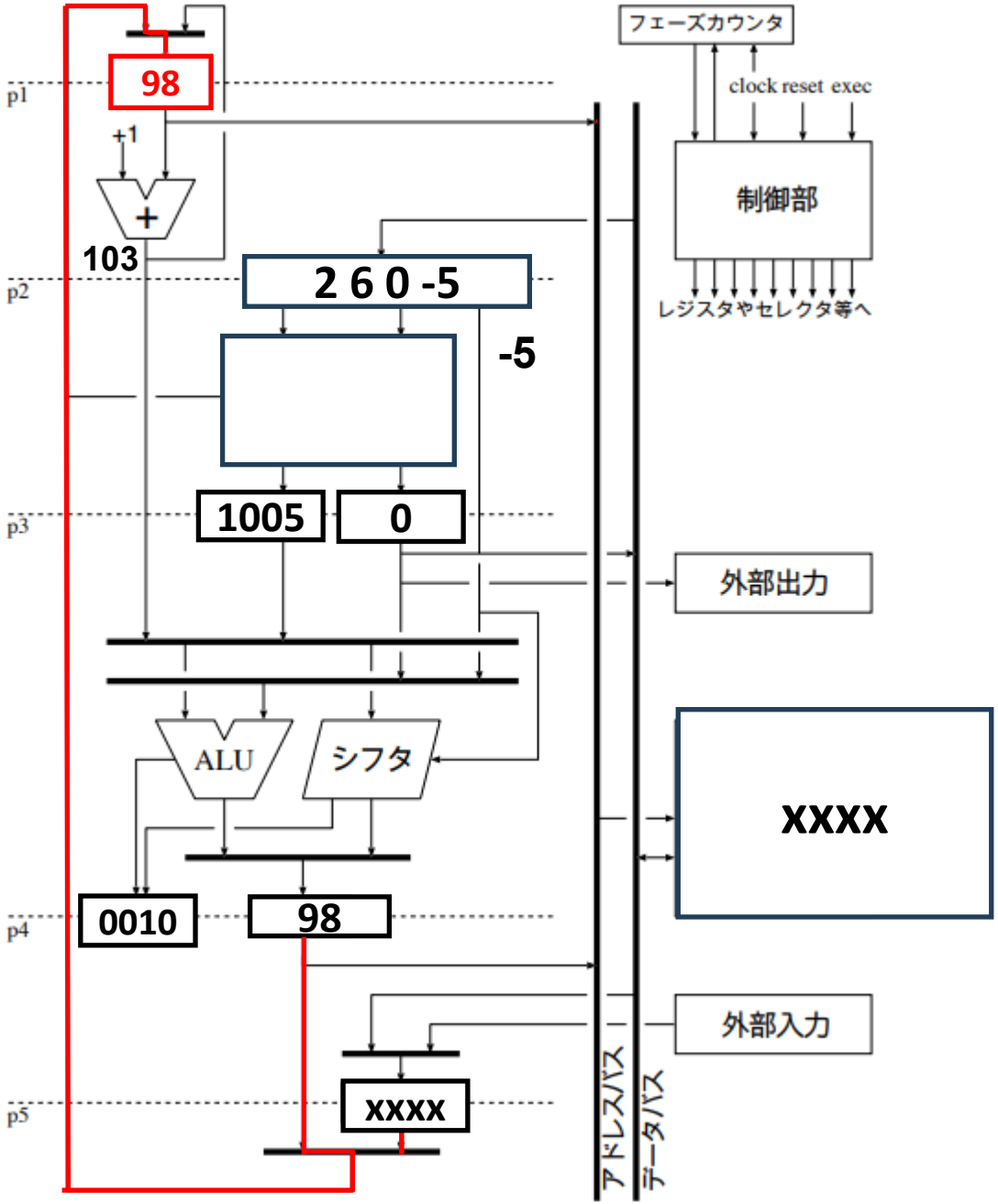
0	1005
1	0200
2	0005
…	…
6	0000
7	0000

レジスタファイルの中身

…	…				
100	0	0	1	10	
101	3	2	0	0	0
102	2	6	0	-5	
…	…				
210	1000				

メインメモリの中身

B -5　P5（レジスタ書き込み、PC更新）



0	1005
1	0200
2	0005
…	…
6	0000
7	0000

レジスタファイルの中身

…	…				
100	0	0	1	10	
101	3	2	0	0	0
102	2	6	0	-5	
…	…				
210	1000				

メインメモリの中身

課題とレポート

課題

- SIMPLE/Bに改良、拡張を加えた**独自プロセッサ**の設計、実装、評価
  - SIMPLE/Bの仕様を満たすプロセッサの作成
  - SIMPLE/Bに改良・拡張を加えた独自プロセッサの作成
    - 命令セットアーキテクチャの改良：命令の追加、条件分岐の１命令化
    - マイクロアーキテクチャの改良：ハーバードアーキテクチャ、パイプライン、スーパースカラ、マルチコア
    - など
  - SIMPLE/Bと独自プロセッサの性能を比較し、実施した改良や拡張がもたらす有効性を定量的に評価。
    - 最高クロック周波数、実行命令数、実行サイクル数
    - ゲート数（LUT数）
    - など

中間レポート、デモ

- 中間デモ 5/8(金)
  - FPGAボード上で何らかの命令が動作しているところを見せる
  - 進捗状況を報告する
  - 時間は１グループ５分
- 中間レポート 5/8(金)
  - 概要：独自プロセッサを作成するにあたり必要となる仕様書と実施報告書
  - 提出方法：githubの**2026-simple-teamXX** のリモートリポジトリに，middleという**タグ名のリリース**を作成して提出
  - 提出物
    A. アーキテクチャ検討報告書
    B. 方式設計仕様書
    C. 機能設計仕様書
    D. 実施状況報告

最終レポート、デモ

- 最終デモ 5/29(金)
  - 完成したプロセッサの性能をアピールできるプログラム（応用プログラム）を作成し、プロセッサ上で動作させる。
  - 「SIMPLE/Bに比べてこんな点がすぐれている」というセールストーク
  - 時間は１グループ５分ほど
- 口頭試問

最終レポート、デモ

- 最終レポート 6/5(金)
  - 概要：独自プロセッサを使用するにあたり必要となるマニュアル、仕様書と実施報告書
  - 提出方法：githubの**2026-simple-teamXX** のリモートリポジトリに，finalという**タグ名のリリース**を作成して提出
  - 提出物
    A. 最終成果物のユーザーズマニュアル
    B. アーキテクチャ拡張仕様書
    C. 性能評価仕様書
    D. 機能設計仕様書
    E. 実施報告書

レポートはHPで指定されている体裁に従うこと。
従っていない場合は再提出となります。

課題におけるLLMの利用について

- LLMの利用は禁止ではありませんが、課題に取り組むうえで利用した場合は、レポートに以下の点を明記してください。
  - 使用した目的
  - 使用したプロンプト
  - 使用したモデル
  - LLMからの出力
- HDLコードの作成において初めからLLMにソースコードを生成させることは学習の効果を大きく下げてしまいます。また正しいコードの生成には効果的なプロンプトを入力する必要があり、生成されたコードが正しいかどうかを判断するにも、ある程度の実装のスキルが必要があります。

大学の実験はそのスキルを身に着ける良い機会です。ぜひLLMの利用は最低限に、自らの力で課題に取り組むことをお勧めします。

評価

- 出席、最終成果物の完成度、グループ内での貢献度、レポート、デモ、口頭試問により総合的に評価
- グループ内での貢献度はハードウェア面での貢献度を見る
  - 応用プログラムのみを担当し、プロセッサの設計にかかわっていない場合は評価されないので注意
- 課題はグループとして取り組むが、評価は個人ごとに行う

ソート速度コンテスト

- データをソートする時間を競うコンテスト
  - データ
    - 16bitの符号付整数1024個
    - ランダム、昇順ソート済み、降順ソート済みの3種類
  - 時間の定義：完了までのサイクル数×クロック周波数
  - 3種類のデータ各々の処理時間の平均値
- 過去の先輩の記録も記載。ぜひ参加して、歴代最高記録を狙ってください
  http://isle3hw.kuis.kyoto-u.ac.jp/contest/index.html

実験の進め方・Tips

アーキテクチャの仕様を決める

- 命令セットアーキテクチャ
  - どんな命令を追加するか
  - Reservedのどの部分を利用するか
  - 追加した命令が応用プログラムに生かせるとよい
- マイクロアーキテクチャ
  - パイプライン化
  - メモリ構成
  - 分岐予測
  - など
- プロセッサのコンセプトを決めて、そこからどのような機能拡張をするかを考えるのもよい
  - ソート速度コンテスト1位を取れるプロセッサ
  - 周波数が150MHz以上で動作するプロセッサ など

モジュール構成と分担を決める

- どのようなモジュール構成にするか
  - プロセッサ全体をどう分割するか
    - フェーズ単位？機能単位？
  - 各レジスタはどの単位に属するか
- どう分担を分けるか
  - 制御系とデータパス系
  - サブデザインとトップデザイン＆インタフェース
  - 基本機能と拡張機能
  - 同じ機能ブロックの違うバージョンをそれぞれ設計
  - グループの全員がプロセッサに分担をもつように

    一人がプロセッサを全て担当し、もう一人がソフトウェア（応用プログラムやアセンブラの改造など）を担当する、はダメ

進め方、スケジュールを決める

- 進め方
  - 部品から作るか、トップデザイン（部品はダミー）をまず用意するか。両方から攻めるのもあり。
- スケジューリング
  - 中間レポート「何らかの命令が動作」の実現時期と機能をどう設定するか（中間まで約3週間。意外と短い）
  - 最終成果物の仕様は中間レポートまでに決める。修正は後でもできる。
  - 工程が後戻りすることもある。スケジュールは余裕を持って立てておくこと。

検証（デバッグ）環境を整える

- テストベンチを作り込む
  - 必要な入力信号の変化をすべてテストベンチに記載しておくことで、シミュレーションを立ち上げるだけで、動作確認ができるようになる。
- 実機での検証環境の構築
  - 回路規模が大きくなってくると、シミュレーションでのデバッグは大変。
  - ボード上のLEDや7セグLEDを利用して、プロセッサの内部信号（レジスタや制御信号の値）を表示できるようにする。

メモのすすめ

- 実験を進める中で悩んだ箇所、詰まった個所、考えたこと、試したこと、どのように解決したかをメモしておく。それがレポートの考察になる。
- Gitのコミット時のコメントを活用するのも良い。