# 概要

今回の課題において, 課題2のプロセッサのパイプライン化に取り組んだ.

ブロック図を含めたプロセッサの仕様, ソースコード, シミュレーションの結果を示す.

# 仕様

プロセッサのパイプライン化にともない, 授業で取り上げられたプロセッサとは仕様を変更した.

主な変更点とパイプラインの仕様を以下に示す.

## 1.1ステージ分割

命令フェッチ (IF), 命令デコード (ID), 演算実行 (EX), メモリアクセス (MEM), ライトバック (WB)の5ステージに分割した.

## 1.2モジュール分割

各ステージを1つのモジュールとし, その次のステージへのパイプラインレジスタをモジュール内に配置した. また, レジスタとメモリは別モジュールとして定義した.

パイプライン化のために次の3つのモジュールを新たに追加した.

* 制御信号を生成する制御ユニット
* フォワーディングを行うフォワーディングユニット
* ハザードを検出するハザード検出ユニット

新たなモジュールの詳細は別途後述する.

## 1.2ブロック図（課題2.1）

設計したプロセッサ全体のブロック図を次ページの図1に示す. なお図1は, パタソン, ヘネシー（2011, p. 347）の図4. 60を一部改変のうえ引用した. 簡単のため, 信号線の名前を省くなどしてある.

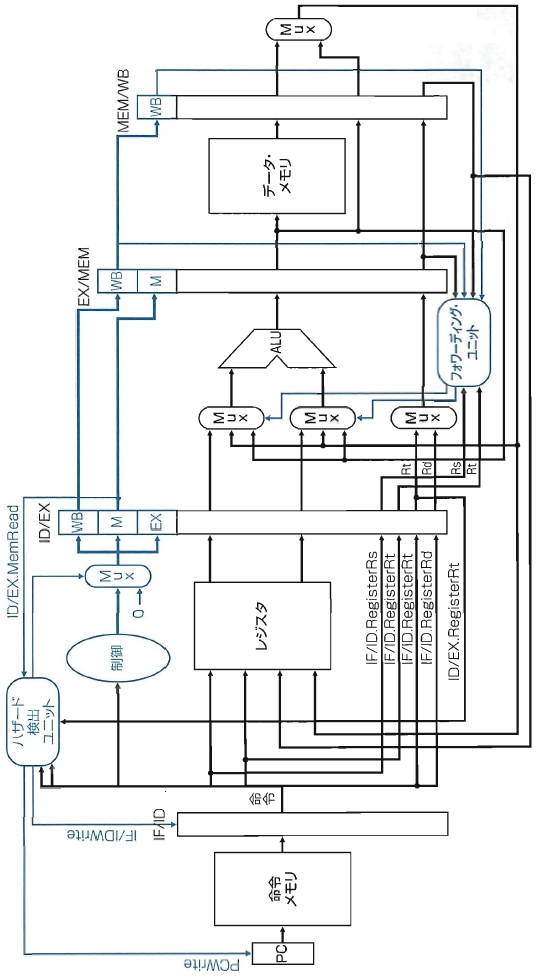


図 設計したプロセッサのブロック図

パタソン, ヘネシー（2011, p. 347）の図4. 60を一部改変

## 1.3新たなモジュールについて

### 制御ユニット

信号線のマルチプレクサの制御信号や, レジスタとメモリのアクセス制御信号など生成するためのモジュールである. IDステージにおいて命令を入力とし, 各制御信号を一元的に生成する. 生成された制御信号は, パイプラインレジスタを通して後続のステージへと送られる.

生成する制御信号の一覧とその意味を表1に示す.

表 制御信号とその意味

|  |  |  |
| --- | --- | --- |
| 信号名 | 0のときの意味 | 1のときの意味 |
| ctrl\_reg\_dst | レジスタ書き込み時に、第2オペランド（rt）のアドレスに書き込む | レジスタ書き込み時に、第1オペランド（rd）のアドレスに書き込む |
| ctrl\_alu\_src | ALUの第2入力を第2オペランド（rt）にする | ALUの第2入力を即値または変位にする |
| ctrl\_branch | 分岐命令でない | 分岐命令である |
| ctrl\_mem\_read | メモリ読み込みを行わない | メモリ読み込みを行う  （読み込みのビット幅も指定する） |
| ctrl\_mem\_write | メモリ書き込みを行わない | メモリ書き込みを行う  （書き込みのビット幅も指定する） |
| ctrl\_reg\_write | レジスタ書き込みを行わない | レジスタ書き込みを行う |
| ctrl\_mem\_to\_reg | ロード命令でない | ロード命令である |

また, 命令ごとの制御信号値を表2に示す. 空欄は制御信号がどちらの値であっても構わないので, 実装上は値が0になるように設計した.

表 命令ごとの制御信号値

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 命令 | ctrl\_reg\_  dst | ctrl\_reg\_  write | ctrl\_alu\_  src | ctrl\_mem\_  read | ctrl\_mem\_  write | ctrl\_mem\_  to\_reg | ctrl\_branch |
| add | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| addi | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| sub | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| lui | 0 | 1 |  | 0 | 0 | 0 | 0 |
| and | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| andi | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| or | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| ori | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| xor | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| xori | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| nor | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| sll | 1 | 1 |  | 0 | 0 | 0 | 0 |
| srl | 1 | 1 |  | 0 | 0 | 0 | 0 |
| sra | 1 | 1 |  | 0 | 0 | 0 | 0 |
| lw | 0 | 1 | 1 | 11(W) | 0 | 1 | 0 |
| lh | 0 | 1 | 1 | 10(HW) | 0 | 1 | 0 |
| lb | 0 | 1 | 1 | 01(B) | 0 | 1 | 0 |
| sw |  | 0 | 1 | 0 | 11(W) |  | 0 |
| sh |  | 0 | 1 | 0 | 10(HW) |  | 0 |
| sb |  | 0 | 1 | 0 | 01(B) |  | 0 |
| beq |  | 0 | 0 | 0 | 0 |  | 1 |
| bne |  | 0 | 0 | 0 | 0 |  | 1 |
| blt |  | 0 | 0 | 0 | 0 |  | 1 |
| ble |  | 0 | 0 | 0 | 0 |  | 1 |
| j |  | 0 |  | 0 | 0 |  | 1 |
| jal |  | 1 |  | 0 | 0 | 0 | 1 |
| jr |  | 0 |  | 0 | 0 |  | 1 |

### フォワーディングユニット

データハザードを削減するためのフォワーディングを行うためのモジュールである. 実際には, フォワーディングが必要か判断し, フォワーディングのためのマルチプレクサへの制御信号を生成している.

フォワーディングの必要の判断には, 次のようなロジックを用いた.

|  |
| --- |
| IF (MEMステージの命令がレジスタ書き込み命令である  AND MEMステージの命令の書き込みレジスタが,  EXステージの命令の読み込みレジスタに等しい)  THEN  MEMステージからフォワーディング  ELSE IF (WBステージの命令がレジスタ書き込み命令である  AND WBステージの命令の書き込みレジスタが,  EXステージの命令の読み込みレジスタに等しい)  THEN  WBステージからフォワーディング  ELSE  フォワーディングしない |

このフォワーディングユニットを用いても, ロード命令が書き込むのと同じレジスタを直後の命令が読みだそうとする場合には, ハザードを無くすことができない. なぜならば, ロード命令でメモリから値が読み込まれるのは, ロード命令がMEMステージ（の後半）にあるときで, そのとき直後の命令は既にEXステージにあるからである.

よって, この場合にはハザードを検出し, 1ステージ分だけパイプラインをストールさせなければならない. その役割を担うのが, 次のハザード検出ユニットである.

### ハザード検出ユニット

ロード命令の直後の命令で, ロード命令が書き込むレジスタを読み込む場合に, ハザードを検出してパイプラインをストールさせるためのユニットである.

ハザードの検出には次のようなロジックを用いた.

|  |
| --- |
| IF (EXステージの命令がロード命令である  AND EXステージの命令の書き込みレジスタが, EXステージの命令の読み込みレジスタに等しい)  THEN  ストール発生  ELSE  そのまま続行 |

このユニットはIDステージにおいて動作する. ストールするときには, 命令を書き換えるのではなく, 制御ユニットからの制御信号を０にネゲートするとともに, IFステージでのPCの更新とパイプラインレジスタの更新を差し止める. これにより, IDステージに今ある命令が後続のステージで何も読み書きできなくなるとともに, 次のクロックサイクルにおいても同じ命令がIDステージにあることを実現する.

制御ハザードの際のパイプラインフラッシュの仕様は, このモジュールで取り扱うわけではないが, ハザードつながりということで, ここでその詳細を述べておく.

PCを分岐先のものに更新するタイミングについて

分岐先のPCを知ることができるステージは, j, jal命令などはIDステージ, beq命令などはEXステージであるが, beq命令などで実際に分岐するかの判断は, ALUの演算後にALUのゼロフラグやサインフラグを利用する仕様にしてある.

分岐命令ができるだけ早いステージにある段階でPCを分岐先のものに更新できれば, パイプラインフラッシュすべき命令数が少なくなるが, 今回は簡単のため, 分岐命令がMEMステージにある段階でPCを分岐先のものに更新する設計にした. これにより, フラッシュすべき命令は, 分岐命令の直後の2命令となる.

フラッシュはストールと同様に, 命令書き換えではなく制御信号を0にネゲートすることによって実現した.

# ソースコード（課題2.2）

ソースコードのファイル構成は以下のようになっている.

* header.v

defineなど共通の設定

* IF.v

命令フェッチステージ

* ID.v

命令デコードステージ, レジスタ, 制御ユニット, ハザード検出ユニット

* EX.v

演算実行ステージ, フォワーディングユニット

* MEM.v

メモリアクセスステージ, メモリ

* WB.v

ライトバックステージ

* processor.v

トップモジュール

ソースコードはレポートの末尾に別途添付する.

# シミュレーション（課題2.3）

合計4つのシミュレーションについて, その概要とModelsimの出力を示す. 4つのシミュレーションの内訳は, パイプライン動作について1つ, フォワーディング機能について2つ, 分岐命令について1つである.

## パイプライン動作のシミュレーション

レジスタに値を順に格納するだけの命令列1を実行した. Modelsimの出力結果を図2に示す.

|  |
| --- |
| addi r1, r0, 1  addi r2, r0, 2  addi r3, r0, 3  addi r4, r0, 4  addi r5, r0, 5 |

命令列

図2下部の\_regsがレジスタの値を示しており, 第4クロックサイクル以降の各クロックサイクルで, 順に値が格納されていっており, パイプラインとして正常動作していることがわかる.

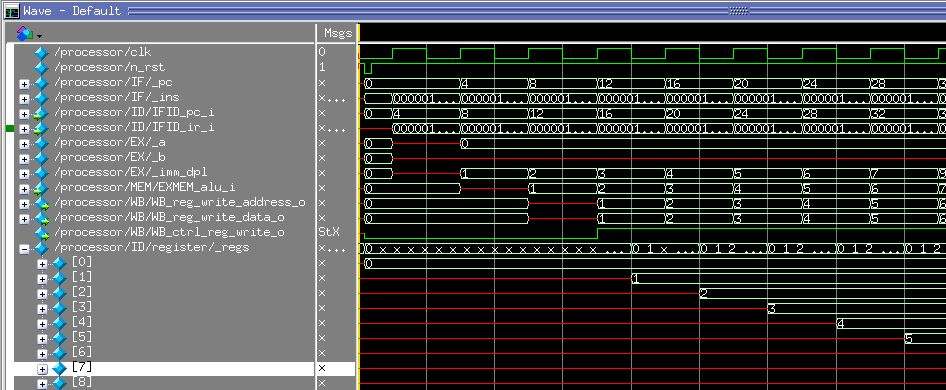
## 3.2 フォワーディング機能のシミュレーション

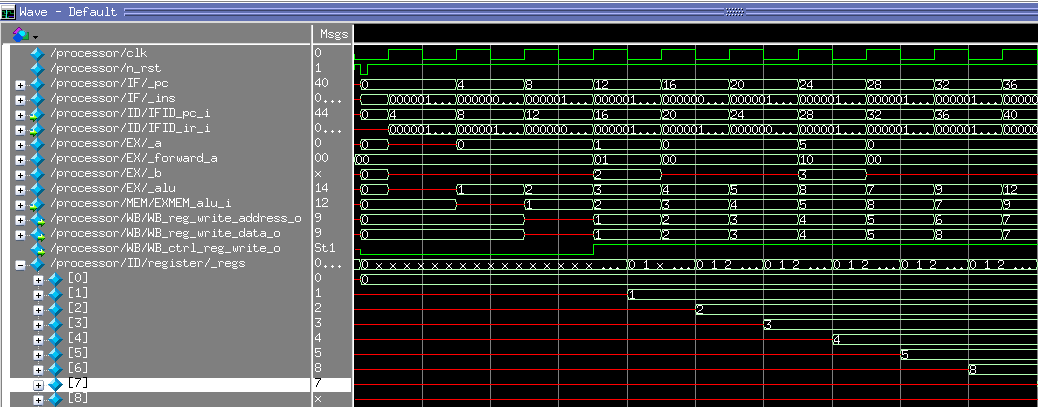
データハザードのため, フォワーディングが必要となる命令列2を実行した. 命令列2において, 3行目と6行目のadd命令でフォワーディングが必要となる. Modelsimの出力結果を図3に示す.

|  |
| --- |
| addi r1, r0, 1  addi r2, r0, 2  **add r3, r1, r2**  addi r4, r0, 4  addi r5, r0, 5  **add r6, r5, r3** |

命令列

\_forward\_aがALUの入力線\_aにフォワーディングを行うかどうかの制御線であり, 値が2'b01のとき, 直前のクロックサイクルでMEMステージにあるALUの出力結果（1）をフォワーディング, 値が2'b10のとき, 直前のクロックサイクルでEXステージにあるALUの出力結果（5）をフォワーディングしており, フォワーディング機構が動作していることがわかる.

 図 シミュレーション結果1

 図 シミュレーション結果2

## 3.3 メモリから読んだ値を直後に使う際にストールさせるシミュレーション

フォワーディングを行ってもデータハザードが解消できない, ロード命令の直後にその値を用いる場合のシミュレーションを行った. 命令列3を実行し, Modelsimの出力結果を図4に示す.

|  |
| --- |
| addi r1, r0, 100  addi r2, r0, 200  sw r1, 10(r0)  sw r2, 20(r0)  lw r3, 10(r0)  lw r4, 20(r0)  **add r5, r4, r3** |

命令列

ストールすべき命令（7行目のadd命令）がIDステージにあるとき, 1クロックサイクル分PCの更新が28で止まっており, またこのとき制御ユニットの制御信号（ctrl\_reg\_write）が0にネゲートされ, WBステージに命令が到達したときに, レジスタに何も書き込まれないことが確認できた.

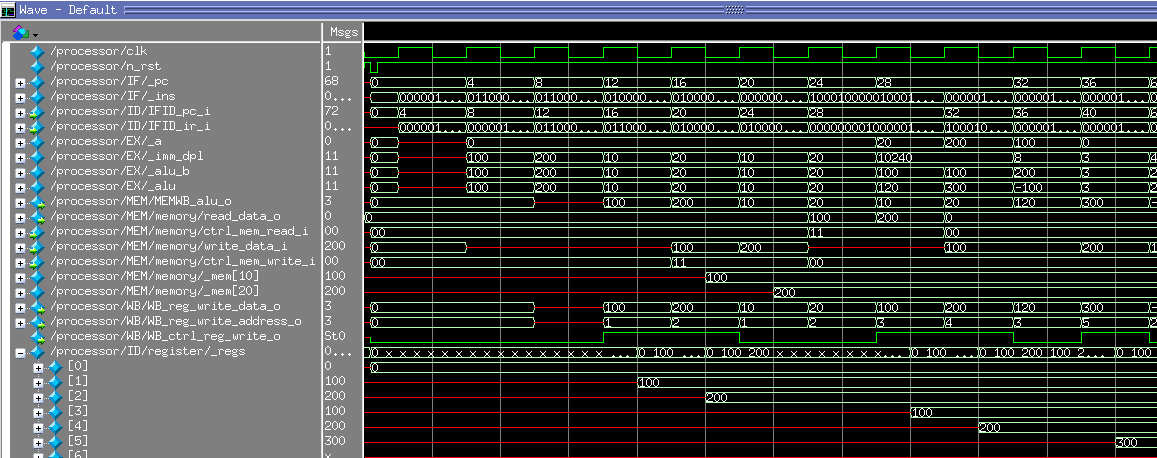
## 3.4 分岐命令のシミュレーション

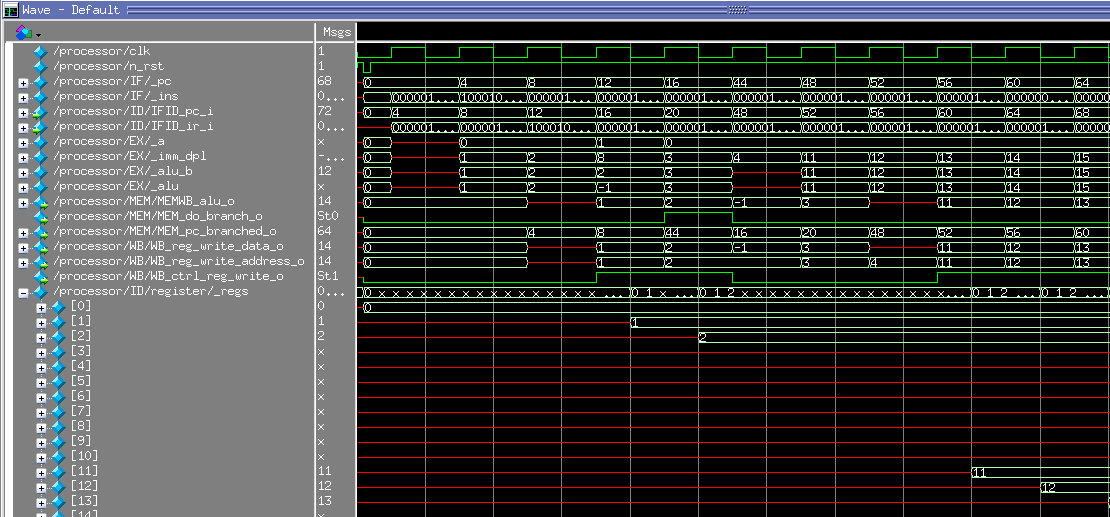
分岐命令を実行する命令列4のシミュレーションを行った. Modelsimの出力結果を図5に示す.

|  |
| --- |
| addi r1, r0, 1  addi r2, r0, 2  **blt r2, r1, label1**  addi r3, r0, 3  addi r4, r0, 4  addi r5, r0, 5  addi r6, r0, 6  addi r7, r0, 7  addi r8, r0, 8  addi r9, r0, 9  addi r10, r0, 10  **label1: addi r11, r0, 11**  addi r12, r0, 12 |

命令列

分岐命令（3行目のblt命令）実行時には, PCが分岐先のものに更新されるとともに, その時点でID, EXステージにある命令の制御信号が0にネゲートされ, WBステージに到達してもレジスタに何も書き込まれないことが確認できた.

 図 シミュレーション結果3

 図 シミュレーション結果4

# 参考文献

1. 坂井修一. 実践 コンピュータアーキテクチャ. 東京, コロナ社, 2009.
2. 坂井修一. コンピュータアーキテクチャ. 東京, コロナ社, 2004.
3. パターソン・デイビッド, ヘネシー・ジョン. コンピュータの構成と設計（上）. 第4版. 東京, 日経BP社, 2011.