プロセッサ設計演習

2021年07月05日

B4 ウン　クアン　イー

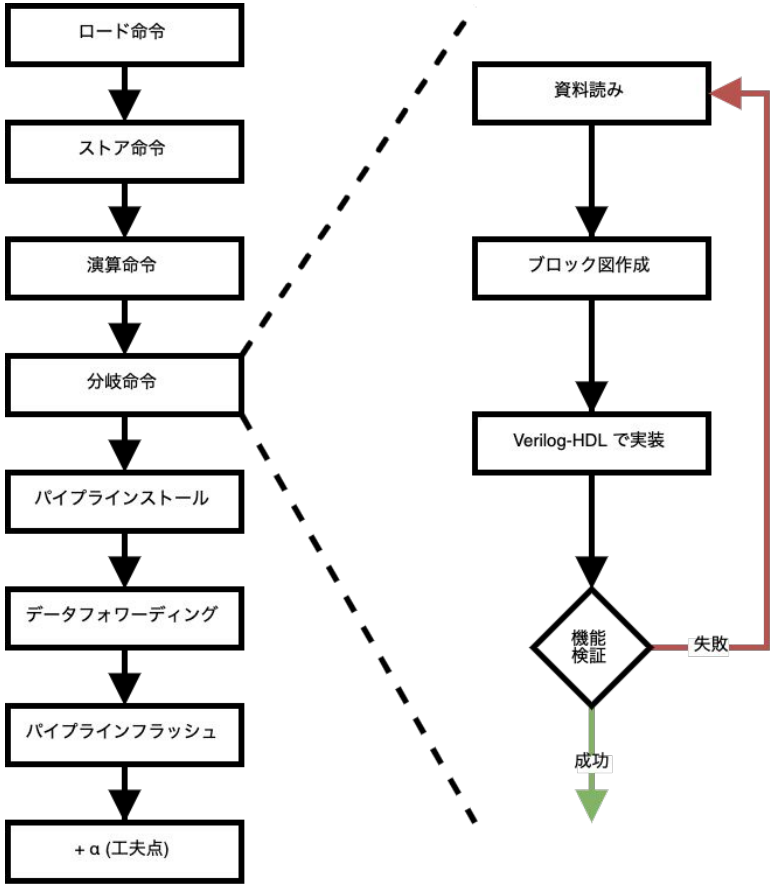
プロセッサの設計過程

1. 設計過程

2. 機能検証

※プロセッサの仕様は予稿を参照いただきたい

1. 設計過程


ロード命令
ストア命令
演算命令
分岐命令
パイプラインストール
データフォワーディング
パイプラインフラッシュ
+ α (工夫点)
資料読み
ブロック図作成
Verilog-HDL で実装
機能
検証
失敗
成功

検証方法: シミュレーション

テストプログラム	検証する機能
load	ロード命令、レジスタへの書き込み
store	ストア命令、レジスタからの読み出し
p2 前半	演算命令 (R 形式, I 形式)、 データフォワーディング、パイプラインストール
p2 後半	分岐命令、パイプラインフラッシュ
hello	総合 (プロセッサ全体の機能)
それ以外の C プログラム	総合 (プロセッサ全体の機能)

性能評価方法

1. プログラム実行クロックサイクル数

2. 論理合成

性能評価用プログラムと設定

・ベンチマークプログラム MiBench [1]

・指標: クロックサイクル数

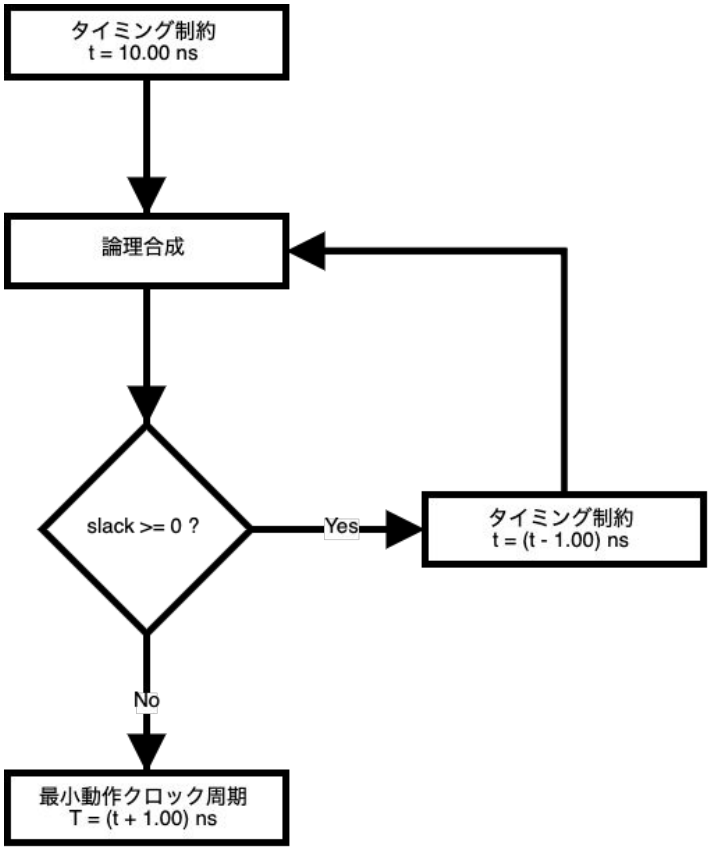
・データのサイズ: test (選択肢: large, small, test)

・コンパイラの最適化レベル: 3

[1] M.R. Guthaus, J.S. Ringenberg, D. Ernst, T.M. Austin, T. Mudge, and R.B. Brown. Mibench: A free, commercially representative embedded benchmark suite. In Proceedings of the Fourth Annual IEEE International Workshop on Workload Characterization. WWC-4 (Cat. No.01EX538), pp. 3-14, 2001.

論理合成の方法

・タイミング制約を 10.00ns から
　1.00ns ずつ減らしていく
・slack の値が 0 より小さくなったら止める
・動作クロック周期、面積と消費電力



工夫点

1. 動的分岐予測
   クロックサイクル数の削減

2. クリティカルパスの短縮
   クロック周期の削減

3. 改善後の論理合成

説明内容

1. 対象命令
2. 分岐方向と分岐アドレスの予測法
3. 分岐予測の動作
4. 動的分岐予測の評価

無条件分岐命令

・必ず分岐する

・JAL, JALR

条件付き分岐命令

・条件文によって分岐方向が決まる

・BEQ, BNE, BLT, BGE, BLTU, BGEU

分岐方向の予測

・2bit 予測器を用いる

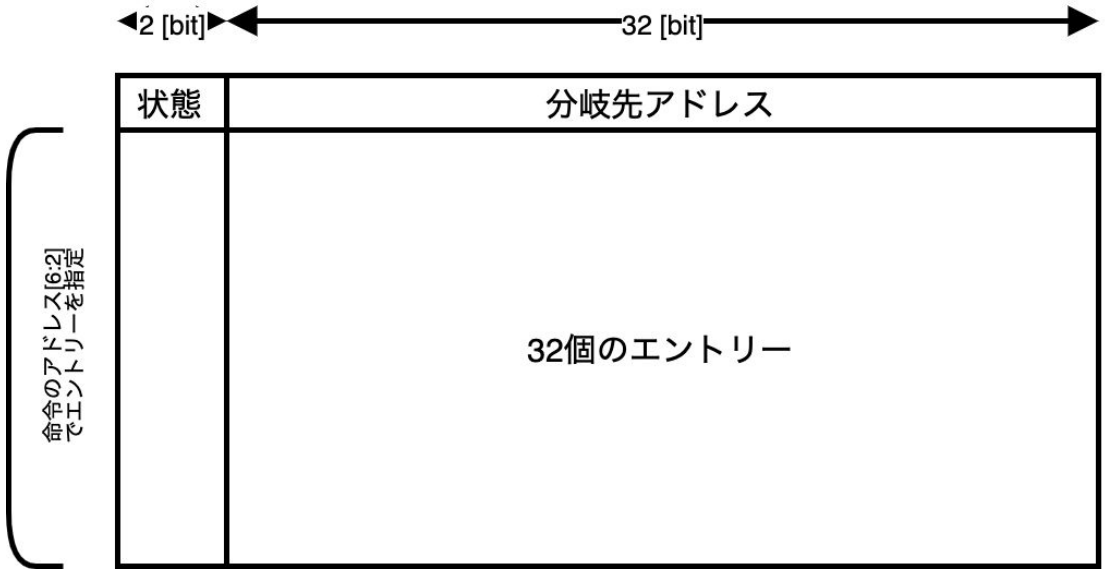
・2bit の状態信号を基に分岐方向を予測する

状態信号	分岐方向の予測
00 (STRONG_NOT_TAKE)	分岐しない
01 (WEAK_NOT_TAKE)	分岐しない
10 (WEAK_TAKE)	分岐する
11 (STRONG_TAKE)	分岐する

2. 分岐方向と分岐アドレスの予測法 (2/2)

ローカル予測機の参照テーブル

- 命令ごとに記録する
  - 状態信号
  - 分岐先アドレス
- エントリー数: 32 (5bit)
- 対象命令のアドレス [6:2]



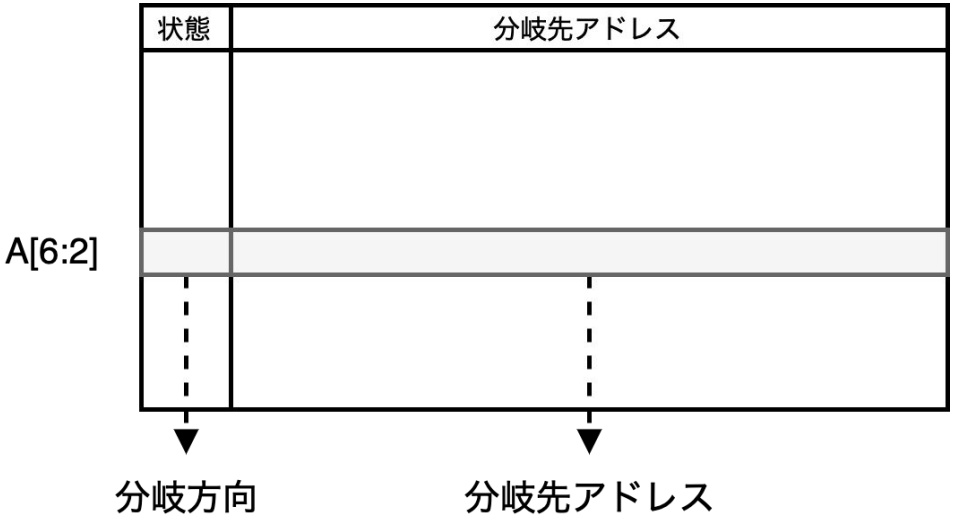
3. 分岐予測の動作 (1/5)

IF ステージ

・命令 A をフェッチする

・IF ステージに対象命令かどうかを確認する

・分岐方向と分岐先アドレスを予測する



3. 分岐予測の動作 (2/5)

EX ステージ

・命令 A の分岐方向を判定する

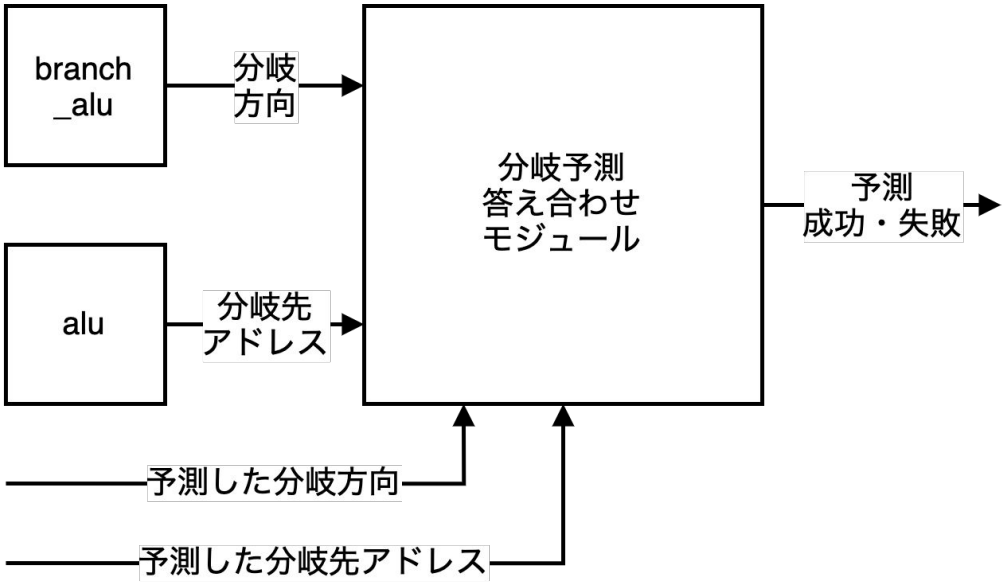
・命令 A の分岐先アドレスを計算する

・答え合わせをする

　・予測した分岐方向・分岐先アドレス
　　のどれかが違っていれば、予測失敗

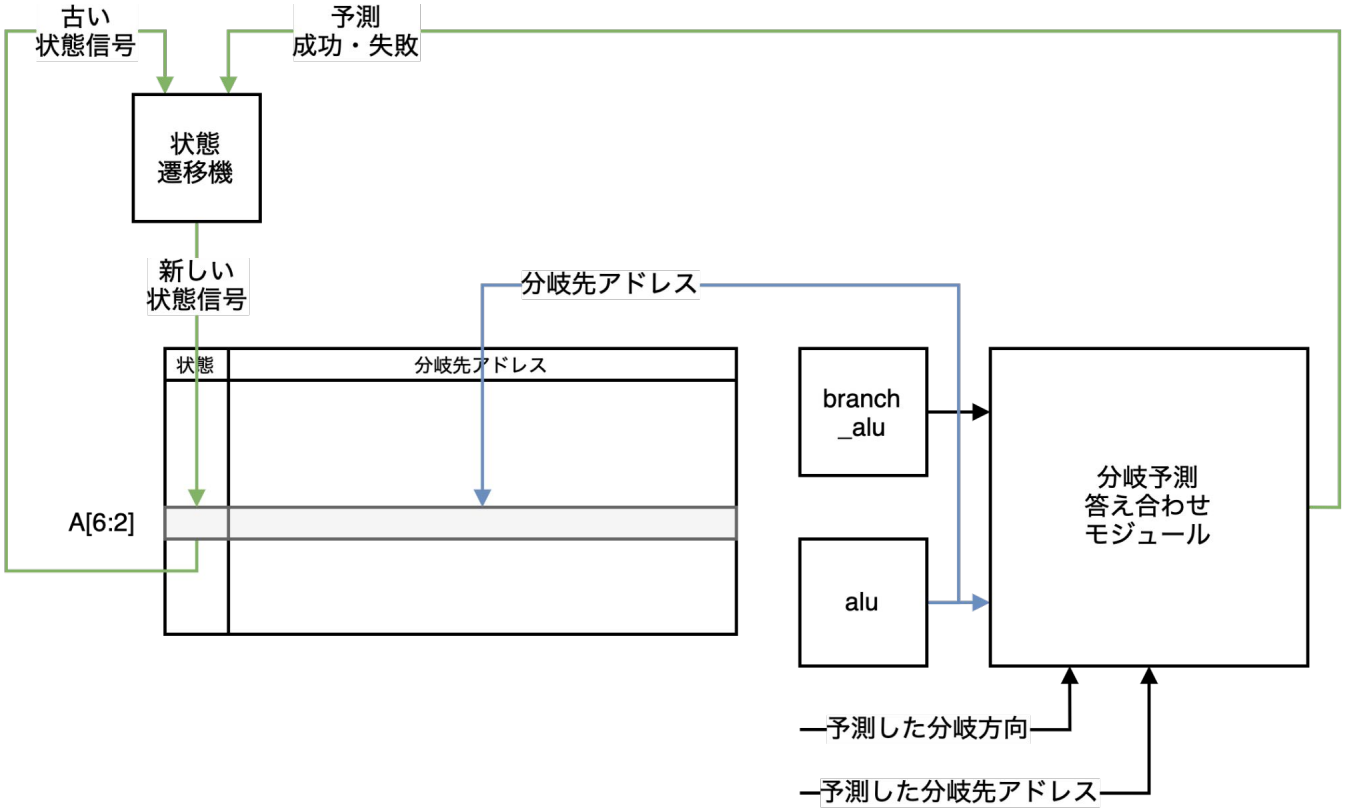
　・予測失敗→分岐する (PC の更新)

　・予測成功→分岐しない



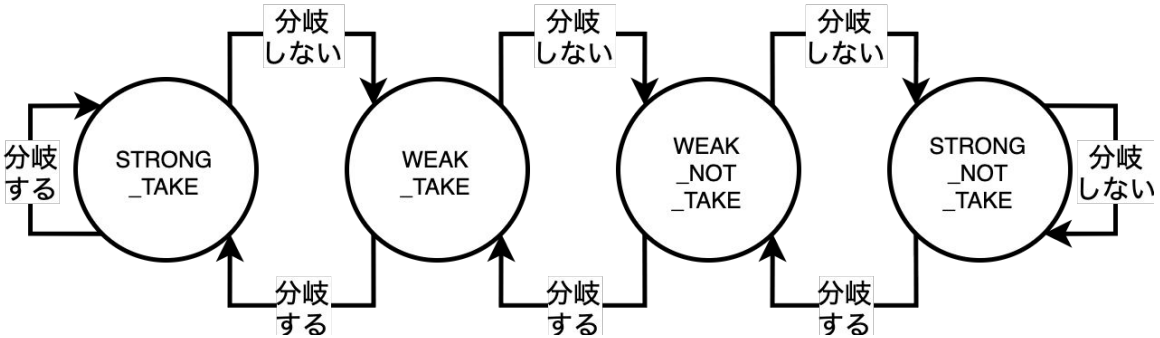
3. 分岐予測の動作 (3/5)

参照テーブルの更新



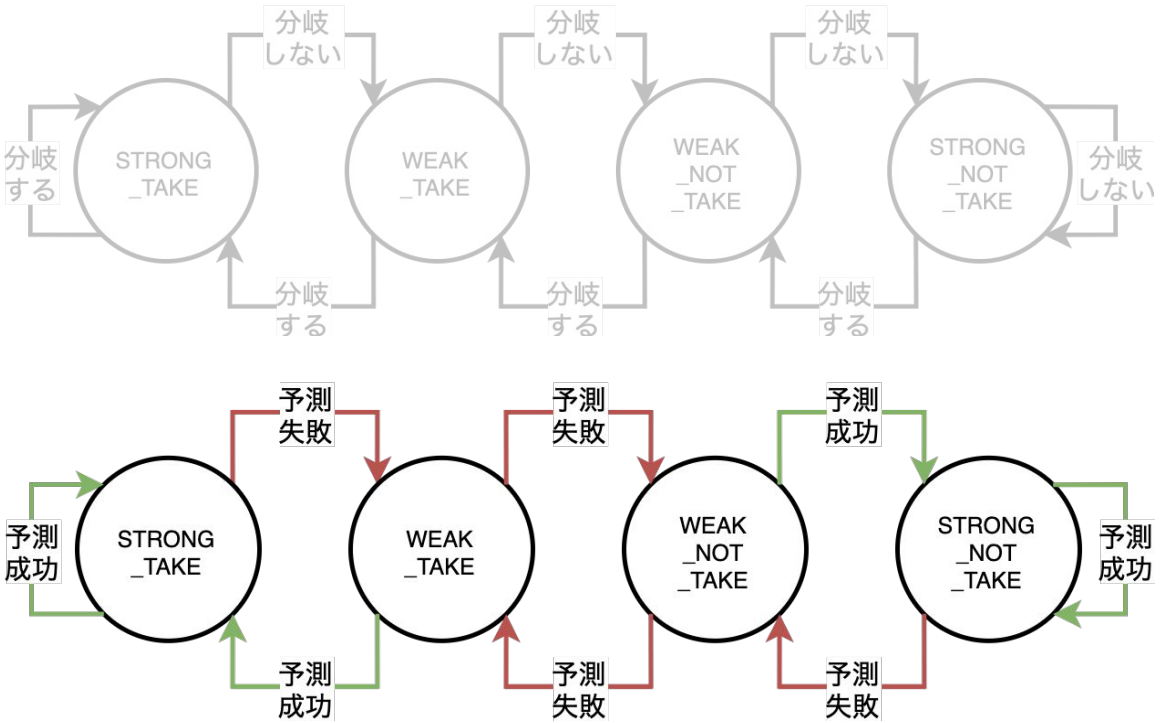
3. 分岐予測の動作 (4/5)

状態信号の状態遷移図



3. 分岐予測の動作 (5/5)

状態信号の状態遷移図



4. 動的分岐予測の評価 (1/2)

評価指標: クロックサイクル数

- $\text{クロックサイクル数減少率} = \frac{\text{分岐予測実装前} - \text{分岐予測実装後}}{\text{分岐予測実装前}} \times 100\%$
- 平均で 26% 減少した

プログラム	分岐予測実装前のクロックサイクル数	分岐予測実装後のクロックサイクル数	クロックサイクル数減少率 [%]
stringsearch	10594	6966	34.25
bitcnts	56040	44680	20.27
dijkstra	4079473	3048011	25.28

※ 分岐予測器の予測失敗率については予稿を参照いただきたい

4. 動的分岐予測の評価 (2/2)

評価指標: 動作クロック周期 (論理合成)

プロセッサ	最小動作クロック周期 [ns]	動作周波数 [MHz]	面積 [μm²]	消費電力 [mW]
分岐予測実装前	6.00	167	357534.7228	7.5732
分岐予測実装後	9.00	111	936675.8334	10.6318

実行時間 ＝ 実行クロックサイクル数 × 動作クロック周期

⬆️ ⬇️ (−26%) ⬆️ (+50%)

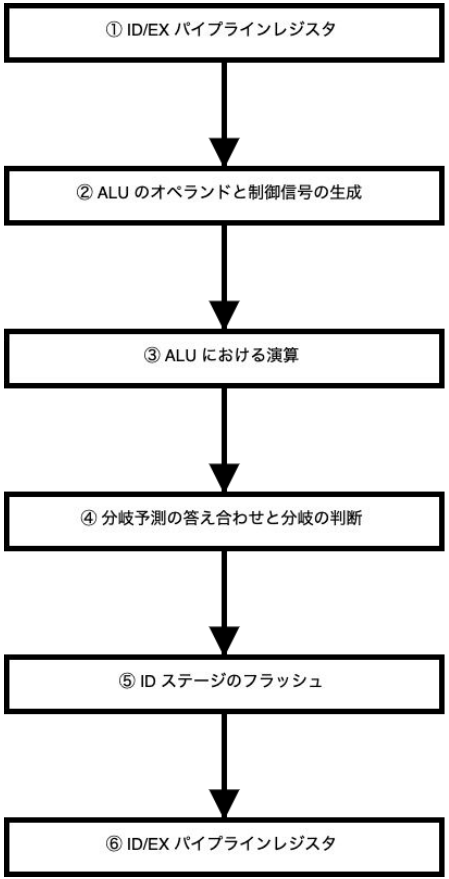
説明内容

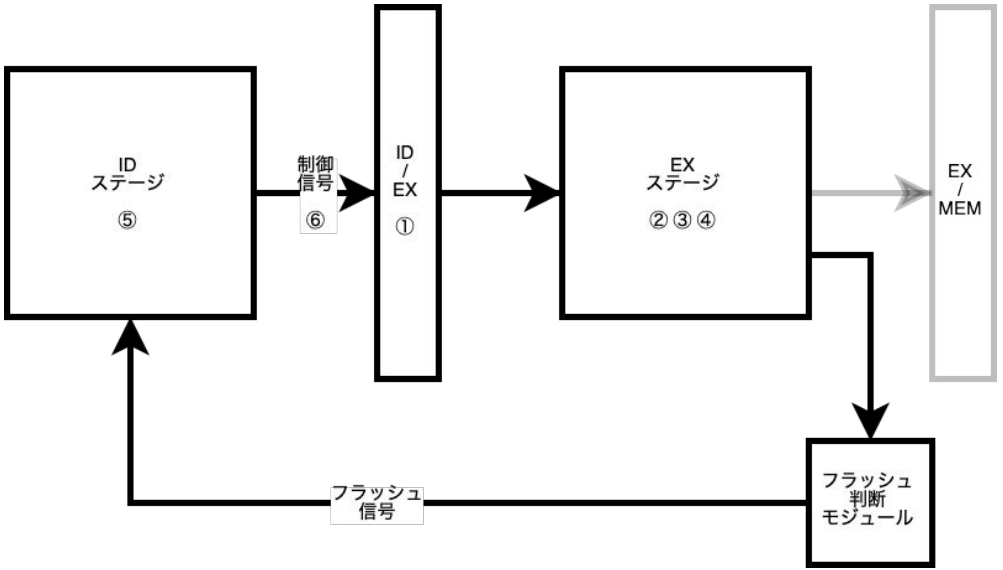
1. クリティカルパス
2. 改善案 ①
3. 改善案 ②

クリティカルパスとは

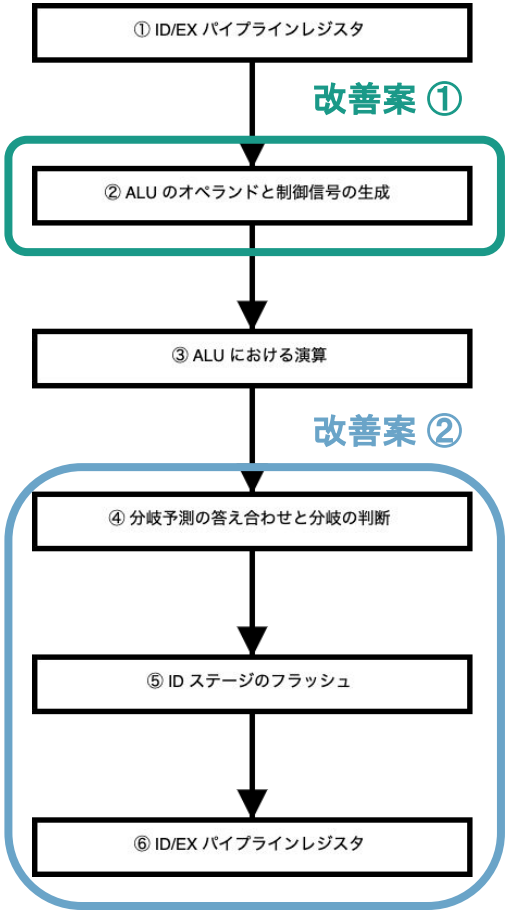
・1クロック周期の中で、一番処理時間のかかる “道”
・この処理時間を遅延時間とも呼ばれる
・クリティカルパスの遅延時間によって、最小動作クロック周期が決まる
  ・クリティカルパスが短くなれば、最小動作クロック周期も短くなる

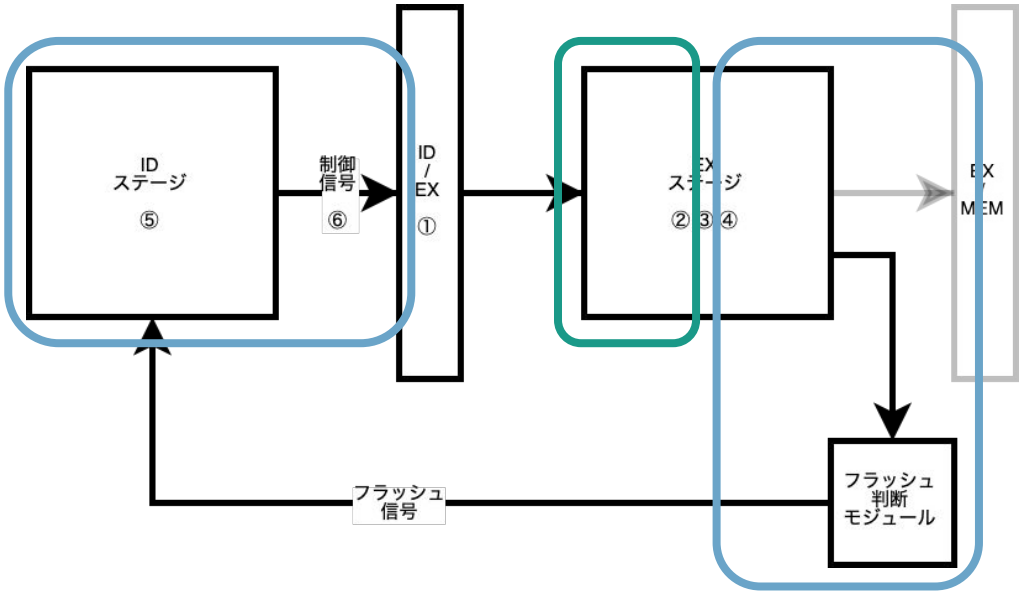
1. クリティカルパス (2/3)





1. クリティカルパス (3/3)





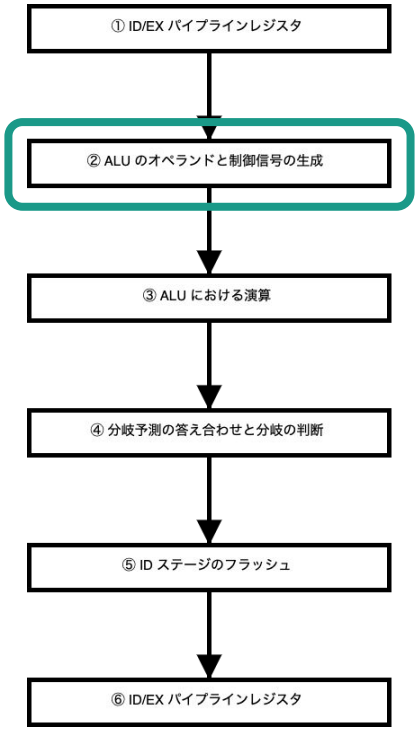
2. 改善案 ① (1/2)

良くなかった点

・ALU のオペランドの生成時間が
　高い割合を占めた

改善案

・ALU のオペランド生成回路を
　ID ステージに移動した
・EX ステージでは、ALUの制御信号のみ
　を生成する



2. 改善案 ① (2/2)

良くなかった点

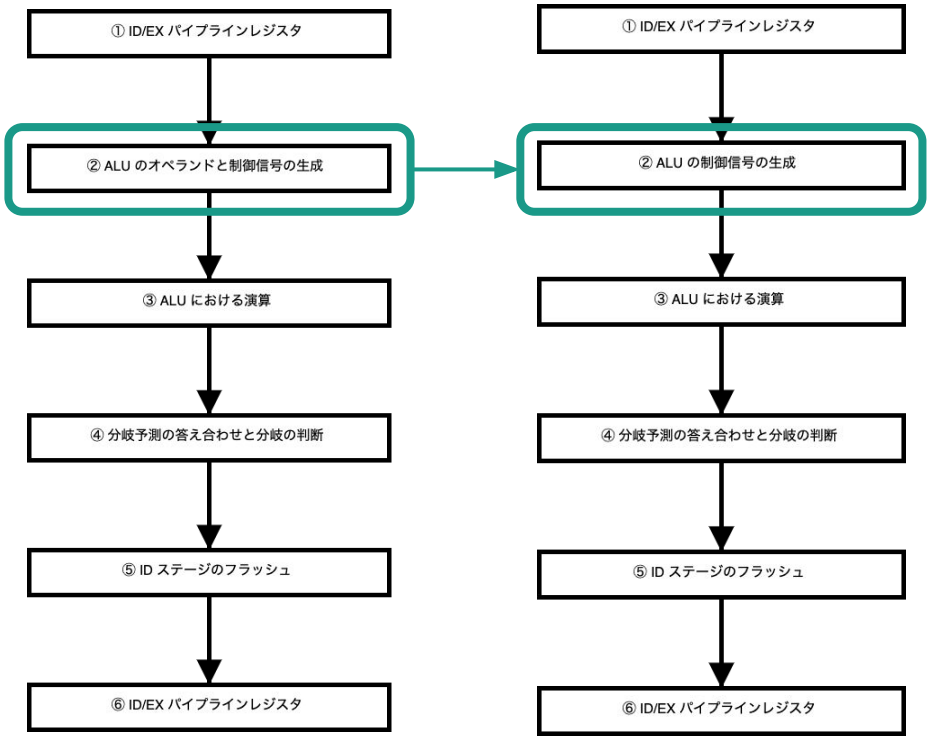
・ALU のオペランドの生成時間が
　高い割合を占めた

改善案

・ALU のオペランド生成回路を
　ID ステージに移動した
・EX ステージでは、ALUの制御信号のみ
　を生成する

結果

・クロック周期: 9ns → 8ns (> 6ns)



良くなかった点

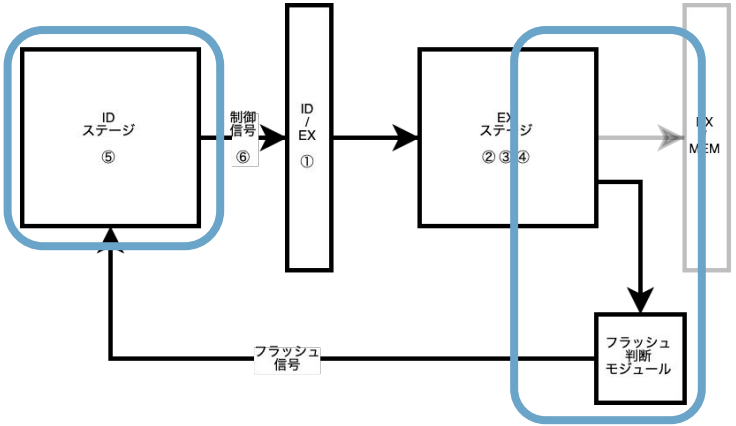
・パイプラインのフラッシュをステージ内で行った

・ID ステージがクリティカルパスに含まれた

改善案

・ID ステージをクリティカルパスから除く

・パイプラインレジスタで
　パイプラインのフラッシュを行う



3. 改善案 ② (2/2)

良くなかった点

・パイプラインのフラッシュをステージ内で行った

・ID ステージがクリティカルパスに含まれた

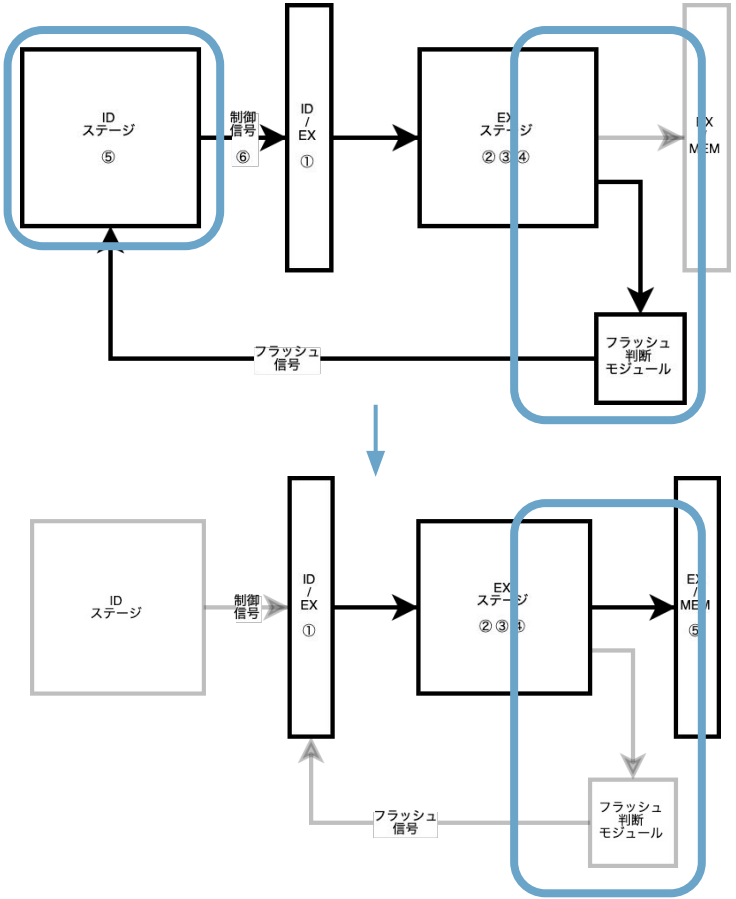
改善案

・ID ステージをクリティカルパスから除く

・パイプラインレジスタで
　パイプラインのフラッシュを行う

結果

・クロック周期: 8ns → 5ns (< 6ns)



論理合成の結果

プロセッサ	最小動作クロック周期 [ns]	動作周波数 [MHz]	面積 [μm²]	消費電力 [mW]
分岐予測実装前	6.00	167	357534.7228	7.5732
分岐予測実装後	9.00	111	936675.8334	10.6318
クリティカルパス短縮後	5.00	200	761303.0413	15.7896

実行時間 ＝ 実行クロックサイクル数 × 動作クロック周期

⬇️ ⬇️ (−26%) ⬆️ (−17%)

まとめ

1. 原理を理解したといっても、設計・実装することが困難

2. 同期、先輩、教員たちのサポートがなければできなかった
ありがとうございました!

予備

プロセッサの仕様

1. 命令セット

2. 例外・割り込み処理

3. パイプライン処理

4. ハザード対処

1. 命令セット

命令	内容	形式
lui	load upper immediate	U
auipc	add upper immediate to pc	U
jal	jump and link	J
jalr	jump and link register	J
beq	branch equal	B
bne	branch not equal	B
blt	branch less than	B
bge	branch greater than or equal	B
bltu	branch less than unsigned	B
bgeu	branch greater than or equal unsigned	B
lb	load byte	I
lh	load halfword	I
lw	load word	I
lbu	load byte unsigned	I
lhu	load halfword unsigned	I
sb	store byte	S
sh	store halfword	S
sw	store word	S
addi	add immediate	I
slti	set less than immediate	I
sltiu	set less than immediate unsigned	I
xori	exclusive or immediate	I
ori	or immediate	I
andi	and immediate	I
slli	shift left logical immediate	I
srli	shift right logical immediate	I
srai	shift right arithmetic immediate	I

命令	内容	形式
add	add	R
sub	sub	R
sll	shift left logical	R
slt	set less than	R
sltu	set less than unsigned	R
xor	exclusive or	R
srl	shift right logical	R
sra	shift right arithmetic	R
or	or	R
and	and	R
ecall	environment call	I
csrrw	csr read and write	I
csrrs	csr read and set	I
csrrc	csr read and clear	I
csrrwi	csr read and write immediate	I
csrrsi	csr read and set immediate	I
csrrci	csr read and clear immediate	I
mret	machine-mode exception return	R

これらの命令がなくても
ベンチマークプログラム
が実行できる

サポートしている例外・割り込み処理（優先順位が高い順）

1. リセット
2. 不正命令
3. 命令アクセス・ミスアライメント
4. ECALL 命令

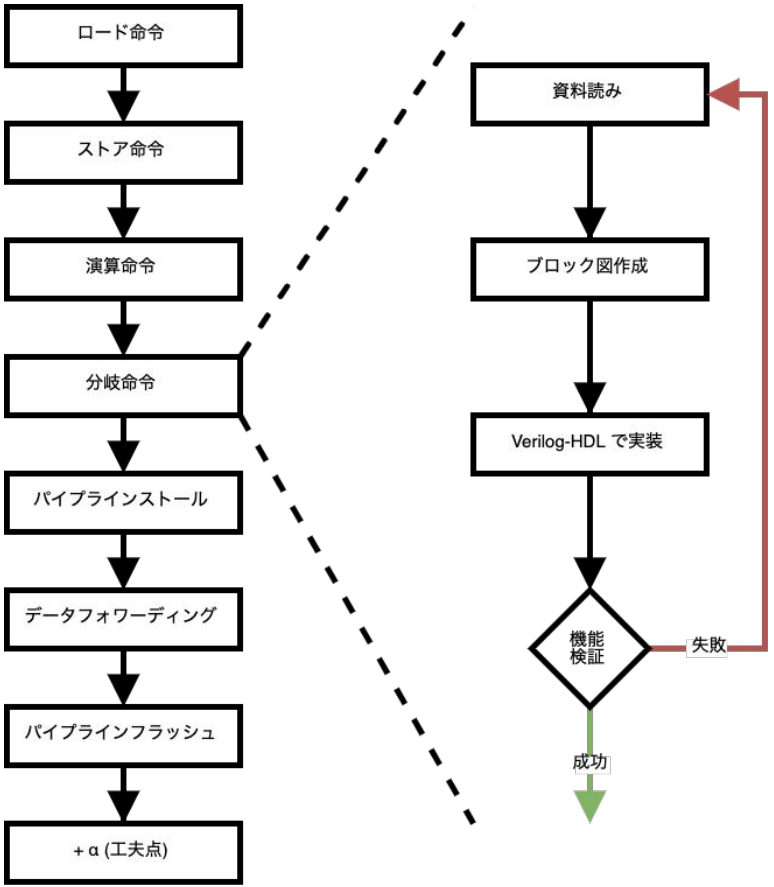
5段階パイプライン

- IF ステージ
- ID ステージ
- EX ステージ
- MEM ステージ
- WB ステージ

3. プロセッサの名前

opt

- 一歩ずつ進めてきた
  - one at a time
- 一時間あたり一歩
  - one-step per time



データハザード

・データフォワーディング
  ・RAW ハザードを解決する（解決できない状況がある）

・パイプラインストール
  ・データフォワーディングが解決できるような状況になるまでストールする

制御ハザード

・パイプラインフラッシュ
  ・プロセッサの内部状態（メモリと汎用レジスタ）に対する変更を無効化する

アセンブリプログラム

命令ごとのテスト

- load (ロード命令)
- store (ストア命令)
- p2 (演算と分岐命令)
- trap (ECALL 命令)

C プログラム

総合テスト

- hello
- napier
- pi
- prime
- bubblesort
- insertsort
- quicksort

性能評価結果

・クロック周期 (シミュレーション環境): 10 ns

・ $\text{クロックサイクル数} = \left\lceil \frac{\text{プログラム実行時間 (シミュレーション環境)}}{10} \right\rceil$

プログラム	クロックサイクル数
stringsearch	10594
bitcnts	56040
dijkstra	4079473

最小動作クロック周期の求め方

1. タイミング制約を 10.00ns に設定して論理合成を行う
2. slack (= タイミング制約 − 最大遅延時間) が 0 以上であることを確認する
3. タイミング制約を 1.00ns 減らして、1. と 2. を繰り返す
4. slack が 0 より小さい値になったら止める
5. 最小動作クロック周期 = slack が負になったタイミング制約 + 1.00ns

論理合成の結果

最小動作クロック周期 [ns]	動作周波数 [MHz]	面積 [μm²]	消費電力 [mW]
6.00	167	357534.7228	7.5732

2. 分岐方向と分岐アドレスの予測法 (2/4)

グローバル予測機		ローカル予測機
対象命令は1つの状態信号を共有 [2]	状態信号	対象命令は各自の状態信号を所有 [2]
「分岐する・分岐しない」と予測	条件付き分岐命令	「分岐する・分岐しない」と予測
「分岐する・分岐しない」と予測 （他の分岐命令に影響される）	無条件分岐命令	「分岐する」と予測する傾向あり

[2] John L. Hennessy and David A. Patterson. Computer Architecture A Quantitative Approach. Katey Birtcher, 6 edition, 2019.

2. 分岐方向と分岐アドレスの予測法 (3/4)

グローバル予測機		ローカル予測機
対象命令は1つの状態信号を共有 [2]	状態信号	対象命令は各自の状態信号を所有 [2]
「分岐する・分岐しない」と予測	条件付き分岐命令	「分岐する・分岐しない」と予測
「分岐する・分岐しない」と予測 （他の分岐命令に影響される）	無条件分岐命令	「分岐する」と予測する傾向あり

・無条件分岐命令は必ず分岐するので、「分岐する」と予測したい

・ローカル予測機を採用

[2] John L. Hennessy and David A. Patterson. Computer Architecture A Quantitative Approach. Katey Birtcher, 6 edition, 2019.

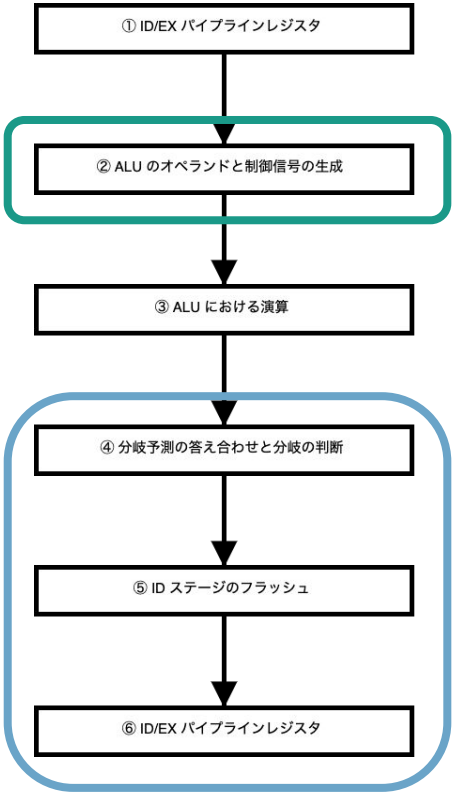
4. 動的分岐予測の評価 (2/3)

評価指標: 予測失敗率

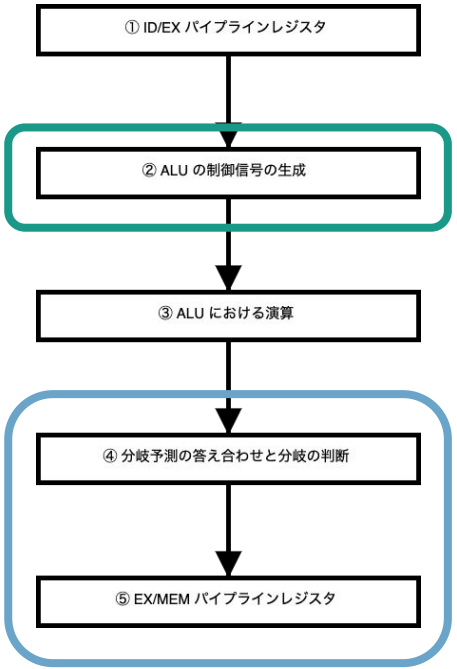
- $$\text{予測失敗率} = \frac{\text{予測失敗命令数}}{\text{分岐予測対象命令数}} \times 100\%$$

プログラム	分岐予測対象命令数	予測失敗命令数	予測失敗率 [%]
stringsearch	2113	131	6.20
bitcnts	9930	690	6.95
dijkstra	869932	12886	1.48

4. 改善後のクリティカルパス (1/2)

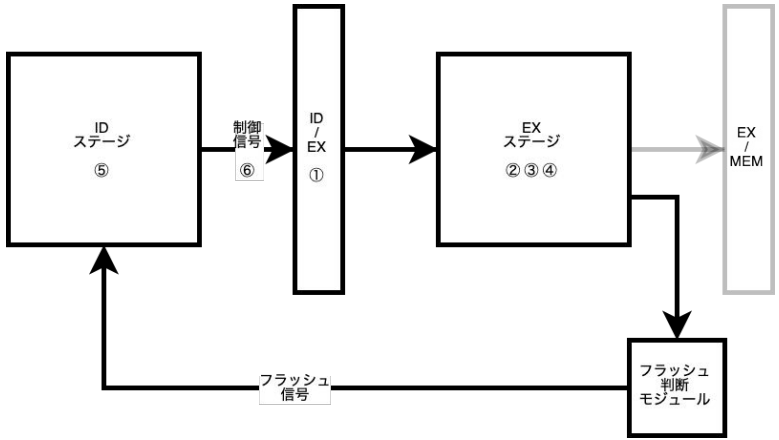


改善前

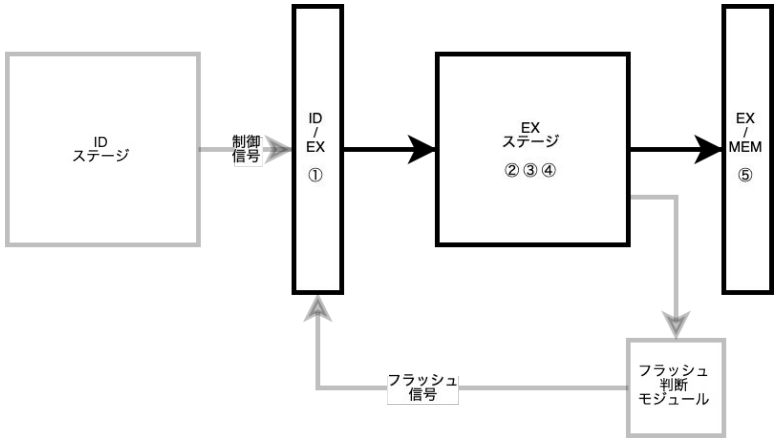


改善後

4. 改善後のクリティカルパス (2/2)



改善前



改善後