先進計算機構成論 03

東京大学大学院 情報理工学系研究科 創造情報学専攻
塩谷 亮太
shioya@ci.i.u-tokyo.ac.jp

質問と回答とか

- 第一回講義も出席していましたが、回答可能時間を過ぎていたため出席処理が行なえませんでした。

質問と回答とか

- 今回の講義でシミュレータとエミュレータという言葉が登場したと思うのですが、この2つの違いは何でしょうか
- 普段抽象化して捉えている回路構造が非常に複雑な構成になっていることがわかり、このサイズにこれだけ膨大な回路を収めていることに驚きました。

質問と回答とか

- 理論的に納得するだけなら簡単なのですが、理解できているかどうかは実装できるかどうかみたいなところがあるので、RISCVエミュレータの作成とFPGAを使ってCPUを作ってみようと思います。"
  - むかしやっていた CPU 実験の手引きを移植したものが下記にあるので，よければこれを参考にしてみてください
  - https://github.com/shioya-lab/cpu-exercise

質問と回答とか

- 本題とはあまり関係ありませんが、以前東大のReedbushを用いてあるプログラムのFLOPS値を測定していた時に同じプログラムを用いてるにも関わらず深夜に測定した際と日中に測定した際での値が異なってしまったことがあありました。もしかするとあれはDVFSの影響だったのでしょうか

- アルゴリズム及びアーキテクチャと消費電力の関係についての問題です。具体的に言うと、並列化などの高スループット化手段は計算機の性能に非常に大きな影響を与えますが、その様な手段の利用は消費電力にどのような影響を与えますか？

質問と回答とか

- 遅延や消費電力といったものについては、学部時代の論理回路の授業では扱っていなかったので興味深かったです。（私は学部は他大学の情報系出身で東大出身ではないので東大での論理回路の授業がどうだったかはわかりませんが……）
- 学部が情報系ではない（物理学科です）ので独学で論理回路を勉強してあまりよくわからなかったですが、論理回路が実際にどのように構成されているかわかり非常に面白かったです。

質問と回答とか

- 自分現在はRISC-Vベースのハードウェア開発フレームワーク「Chipyard」を利用してカスタムアクセラレータの研究を行っているので、本日の講義RISC-Vの部分はかなり助かりました。もし可能であれば、今後もRISC-Vに関しての内容をより多く紹介してくれるとうれしいです。

質問と回答とか

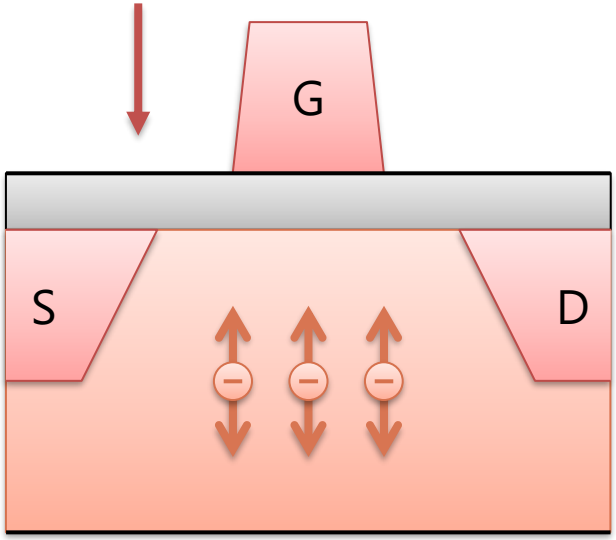
- 質問なのですが、寄生容量の充放電を考えなければならないのはどうしてですか？また、CPUののオーバークロックという言葉を耳にしたことがあるのですが、あれは無理やりゲートに高電圧をかけてチャネルを厚くしているということですか？

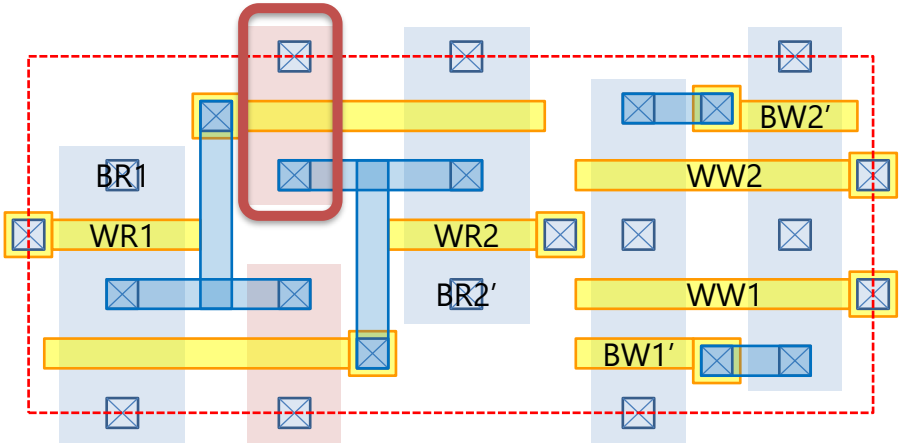
質問と回答とか

- スタンダートセル内のどの回路を使うかはどのように決めているんでしょうか？

- フルカスタムレイアウトの図のそれぞれの部分が何を表しているのかがよくわかりませんでした...

  - ◇ 左図を上から見たのが，右図の赤枠部分
  - ◇ 最近は Fin-FET と言われるものなど，もうちょっと違う構造になってる





質問と回答とか

- IntelやAMDは、実際どの程度フルカスタムなのかというか、論理合成に頼っているのかが気になります。
- フルカスタム・レイアウトのところで、実際にお絵描き(のように)して設計している、という話を聞いて、やはりこのような３次元構造の設計は人間の方がコンピュータよりも強いのだなあと思いました。全部お絵かきで頑張ったらどれぐらい早い回路になるのか気になりました

質問と回答とか

- 論理回路、ディジタル回路は履修したわけではないですが院試のために勉強していました。やったことの延長線上がしれると楽しいですね。
- めちゃくちゃレイヤが下がって少しびっくりしました。CMOS等は学部の授業でやって以来で結構内容を忘れていました。

前回のまとめ

- 目的：これらの具体的なイメージを持つ
  - CPU の論理的な構造や動作と，物理的な回路の関係
  - それら回路の遅延
- 論理回路の復習から始めて，遅延が何できまるのかまで
  - 論理回路と，その設計
  - CMOS による実現
  - 遅延

今日の内容

- 回路の消費電力
  1. クロックの消費電力
  2. アーキテクチャの違いによる消費電力の違い
  3. FPGA による回路
- 命令パイプラインの基礎
  - ◇ パイプライン化によるスループットの向上
  - ◇ ハザード

- この講義資料では，一部，五島先生の「ディジタル回路」の講義資料の図を使用しています

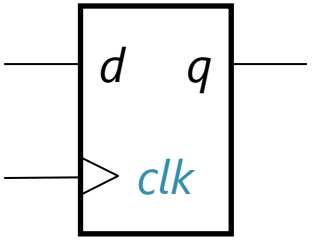
消費電力について

CPU やその他回路の消費電力について

- 回路の消費電力について
  1. クロックの消費電力
  2. アーキテクチャの違いによる消費電力の違い
  3. FPGA による回路

クロックによる消費電力

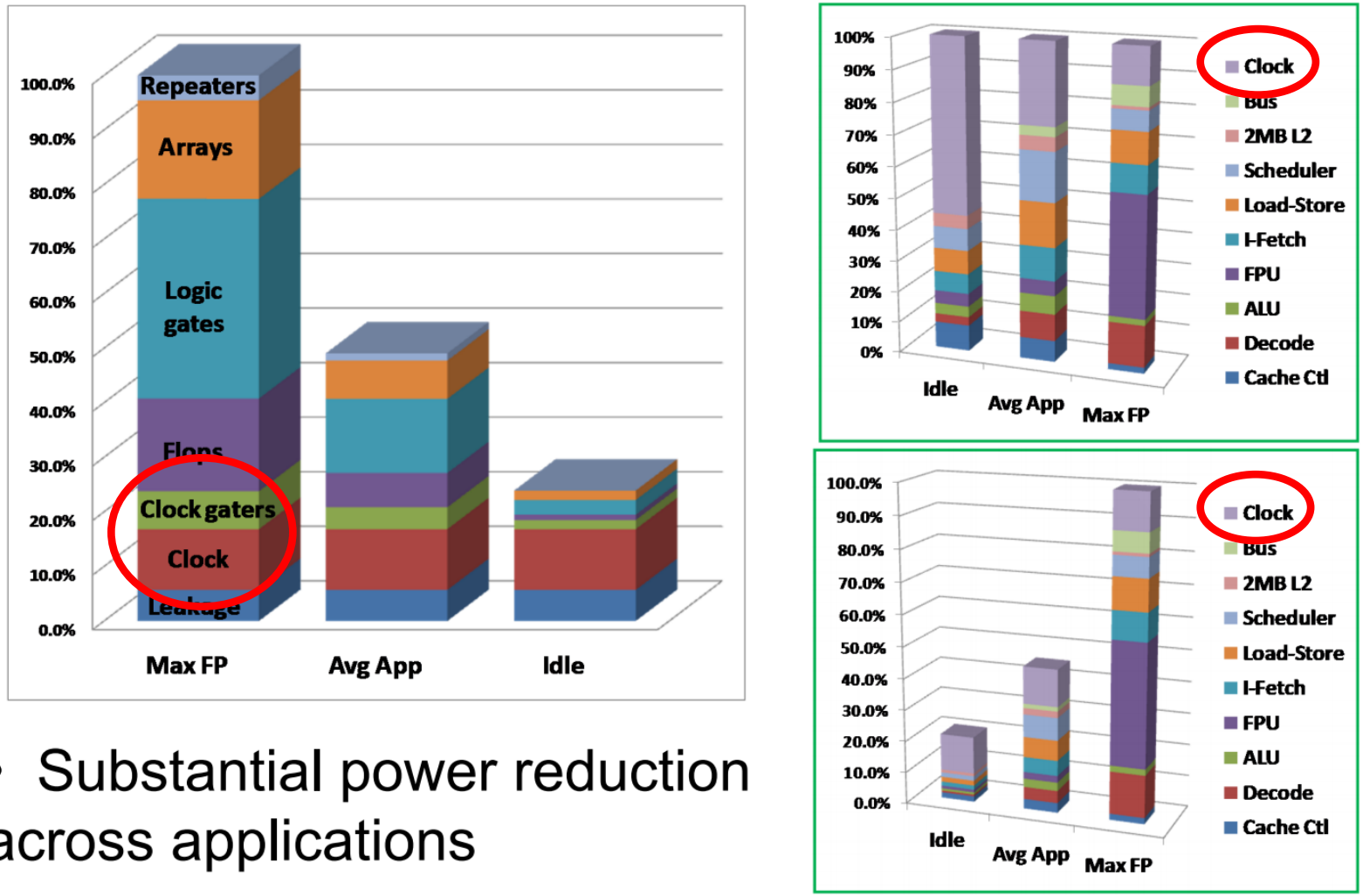
- クロック信号：
  - ◇ 記憶素子の更新タイミングを制御



- 具体的な D-FF の動作：
  - ◇ クロックの立ち上がりのたびに，d の値がサンプリング
  - ◇ その値が次のサイクルの間 q から出力される

- クロックによって消費される電力は非常に大きい
  - ◇ CPU 全体で消費される電力の数割におよぶこともある
  - ◇ なぜただ同期をとるためだけに，それほど電力が食われるのか？

AMD Steamroller の消費電力のうちわけ
実際にクロックが大きな割合を占めることがわかる

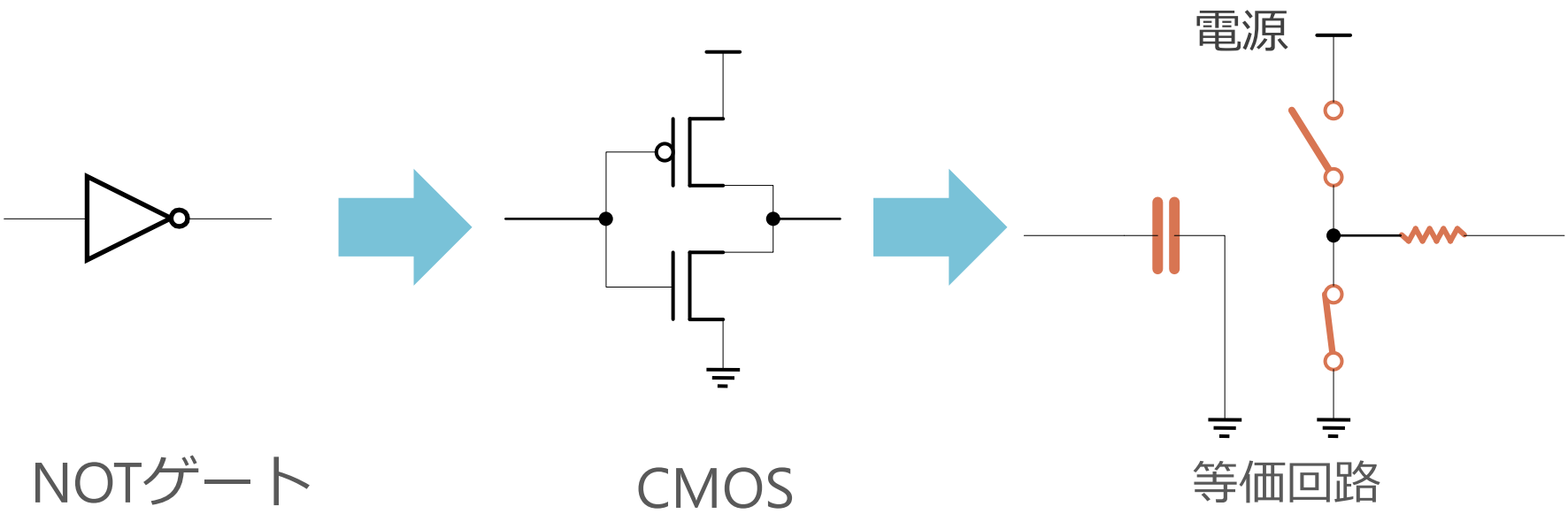


- Substantial power reduction across applications

クロックによる消費電力

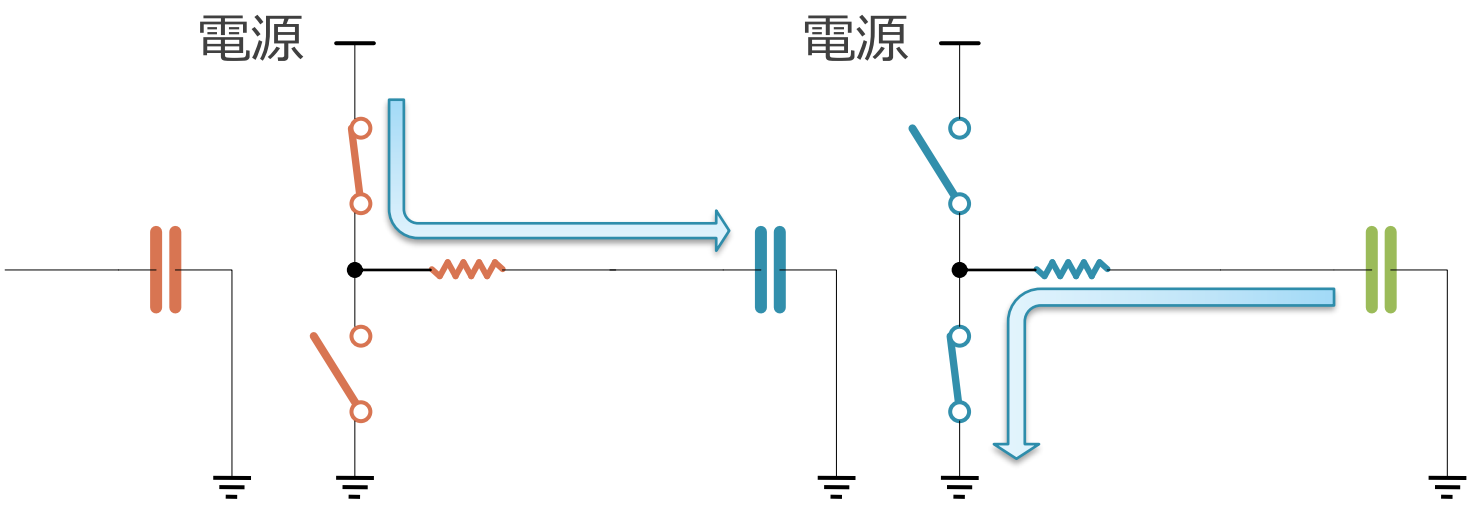
- なぜ更新タイミングの制御でそんなに電力が消費されるのか？
- CMOS 回路の消費電力により説明
  - ◇ 結局，コンデンサの充放電の話
  - ◇ 前回の復習からはじめる

CMOS ゲートの等価回路



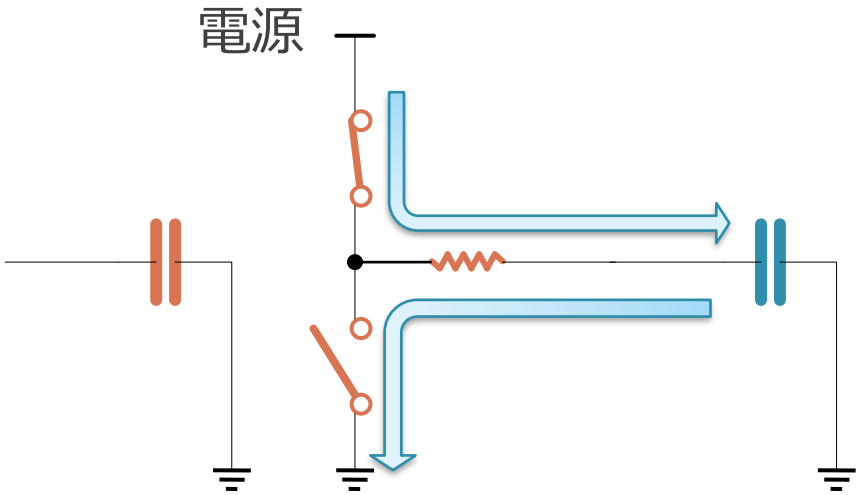
- 抵抗 & コンデンサと，連動したスイッチによって表せる
  - ◇ コンデンサに充電：下のスイッチがON
  - ◇ コンデンサを放電：上のスイッチがON

CMOS ゲートの遅延の実体



- 遅延：コンデンサの充放電にかかる時間
  1. あるゲートのスイッチが切り替わる
  2. 次の段のゲートへの充放電が開始
  3. 次の段のスイッチが切り替わる
  4. ・・・

消費エネルギー

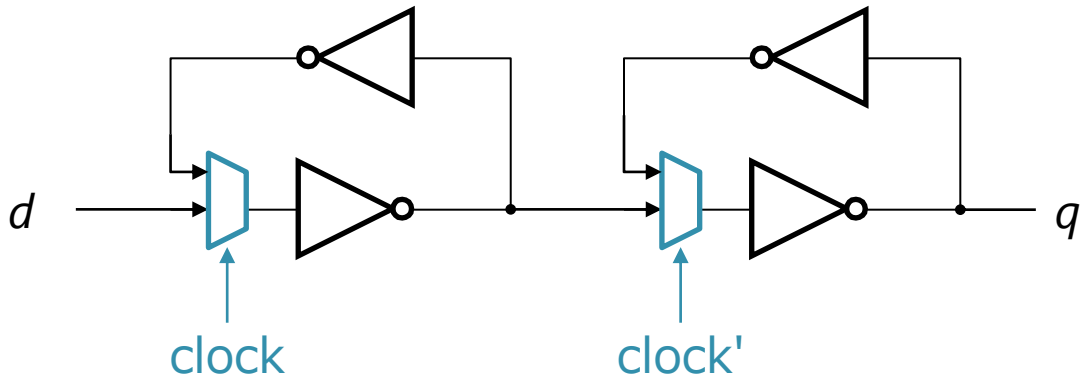


- 消費エネルギーは，主にコンデンサへの充放電で消費される
  - ◇ 消費エネルギーは電圧の二乗に比例：$E = CV^2$
  - ◇ 電荷 $Q = CV$ が，電圧 V の分だけ電源から GND へ移動するから
- 実際には，回路の性質に応じて充放電の回数は変化する
  - ◇ アクティビティ・ファクタ（α）= スイッチング発生確率 に比例

消費エネルギーの補足

- その他に，リーク電流と呼ばれるものもある
  - ◇ トランジスタを OFF にしていても，流れ続けてしまう電流
- 分類：
  - ◇ 充放電によるもの：動的（dynamic）消費電力
  - ◇ リークによるもの：静的（static）消費電力
- 通常は，静的消費電力は多くても数割で動的消費電力が主体
  - ◇ 先ほどの Steamroller では，リークは1割未満

D-FF の回路とクロックによる消費電力



- D-FF の構造：
  - ◇ リング状に繋がっている NOT ゲート
  - ◇ クロックによって切り替えられるマルチプレクサ
- クロックによる消費エネルギー：
  - ◇ マルチプレクサ（のトランジスタ）への充放電で消費
  - ◇ クロック信号が反転するごとに発生

クロックによる消費電力が大きくなる理由

- 理由１：CPU 全体の D-FF で毎サイクル必ず充放電が行われるため
  1. クロックなので毎サイクル必ず反転する
     - アクティビティ・ファクタは１
  2. 充放電されるトランジスタの総数もすごく多い

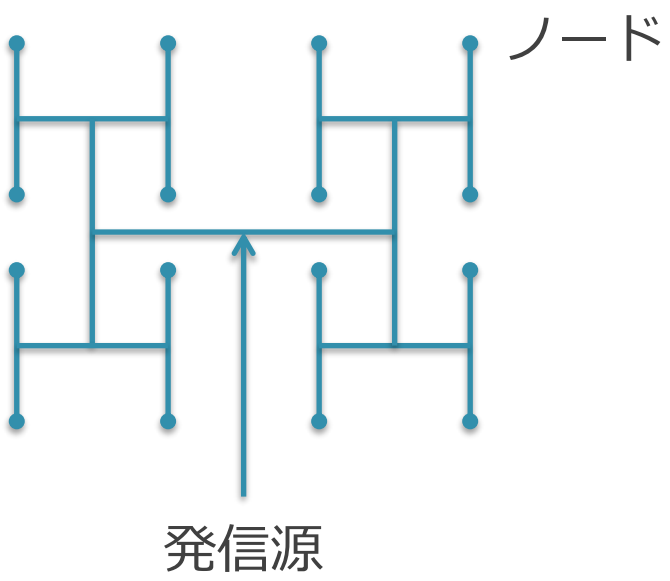
クロックによる消費電力が大きくなる理由

- 理由２：クロック供給のための配線が長大なため
  - ◇ 配線も寄生コンデンサを作るので，そこで充放電が起きる

1. クロックでは，配線の総延長がすごいことになる
   - ◇ すべての D-FF が単一のクロック発信源まで接続

2. スキュー（skew）を無くすために，配線はより長くなる
   - ◇ スキュー：D-FF 間のクロックの到達のずれ
   - ◇ クロック発信源から各 D-FF までの配線長が等しくなるように配置

クロックの配線方法の例：H-TREE



- H-TREE
  - ◇ クロック発信源から各ノードへの配線長が全て等しくなる
  - ◇ しかしその分，物理的に近いノードであっても遠回りになる

H-TREE による配線の例：IBM Power PC

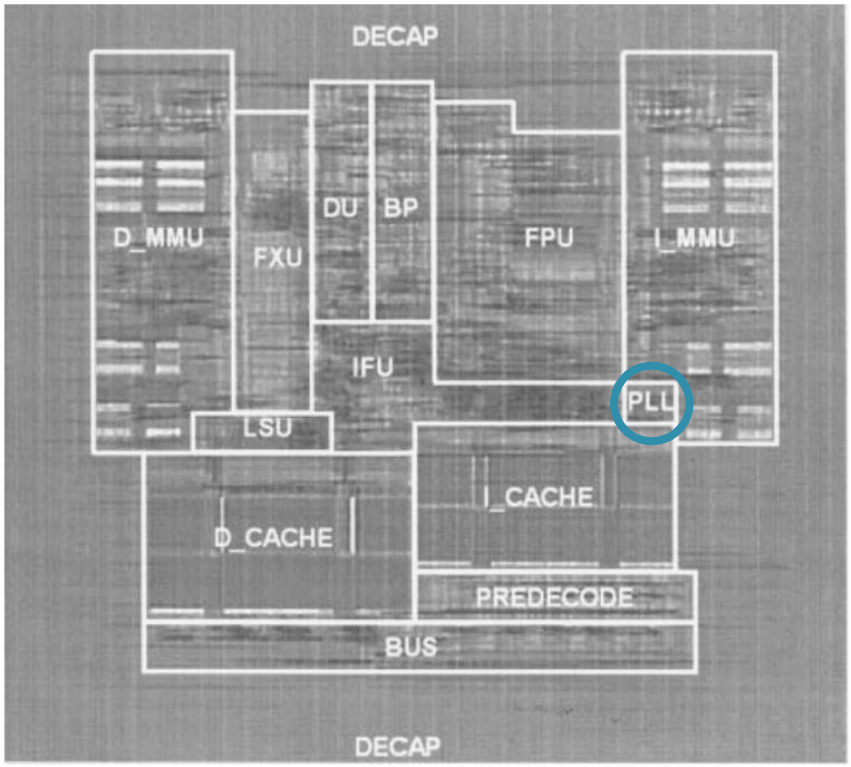


Figure 5.4.2: Die micrograph and floorplan.

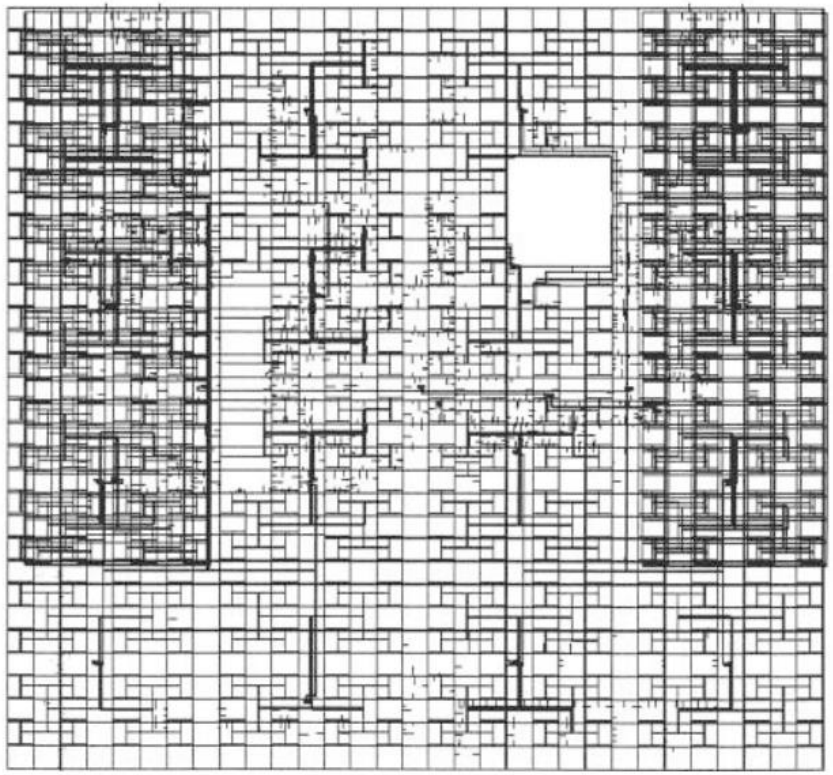


Figure 5.4.7: Clock distribution.

- P. Hofstee, N. Aoki, D. Boerstler, P. Coulman1 , S. Dhong, B. Flachs, N. Kojima, O. Kwon, K. Lee, D. Meltzer2 , K. Nowka, J. Park, J. Peter, S. Posluszny, M. Shapiro3 , J. Silberman2 , O. Takahashi, B. Weinberger, MP 5.4 A 1GHz Single-Issue 64b PowerPC Processor, ISSCC 2000 より

クロックの消費電力のまとめ

- クロックによる消費エネルギー：充放電で消費
  - D-FF 内にあるトランジスタ
  - クロックを配るための配線
- 大きくなる理由：
  - クロック信号が反転するごとに充放電が毎回発生
  - トランジスタ数や配線長が膨大

CPU やその他回路の消費電力について

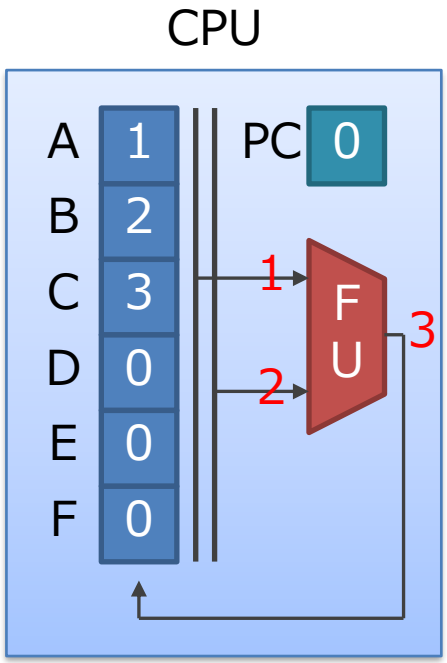
- いくつか補足
  1. クロックの消費電力
  2. **アーキテクチャの違いによる消費電力の違い**
  3. FPGA による回路

回路の遅延と消費エネルギー

- 回路の消費エネルギー：
  - ◇ 主にコンデンサへの充放電で消費
- おおざっぱには，トランジスタ数に比例すると考えて良い
  - ◇ トランジスタ数が増えると，コンデンサが増える
  - ◇ トランジスタ数 ≒ 回路面積

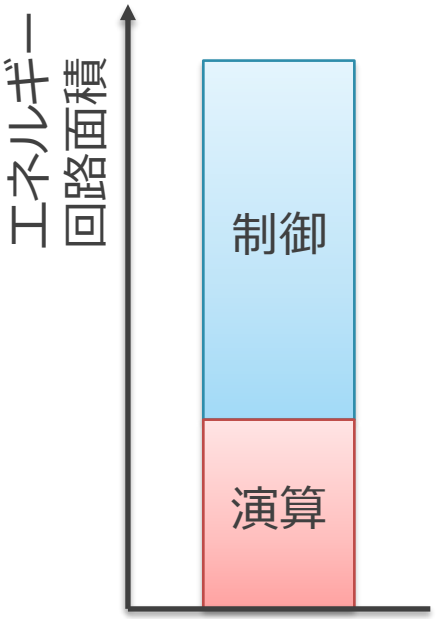
命令を処理するのに必要な回路

- 内訳：
  - ◇ 命令の読み出し
  - ◇ デコード
  - ◇ レジスタ読み書き
  - ◇ メモリの読み書き
  - ◇ （命令のスケジューリングや投機関係など
  - ◇ 演算



回路は命令制御と演算に大きく分けられる

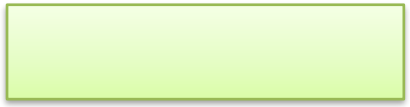
- 命令制御：アーキテクチャによって大きく異なる
  - ◇ 命令の読み出し
  - ◇ デコード
  - ◇ レジスタ読み書き
  - ◇ （命令のスケジューリングや投機関係など
- 演算：基本的に同じ
  - ◇ 論理演算，算術演算，浮動小数点演算
  - ◇ これら演算単体は，どのアーキでも同じことをする



いろいろな回路の規模

- 1bit NAND 演算器：　4 トランジスタ
  （小さすぎて見えない
- 64bit 整数加算器：　4k トランジスタ

- MIPS R3000 プロセッサ：　115k トランジスタ



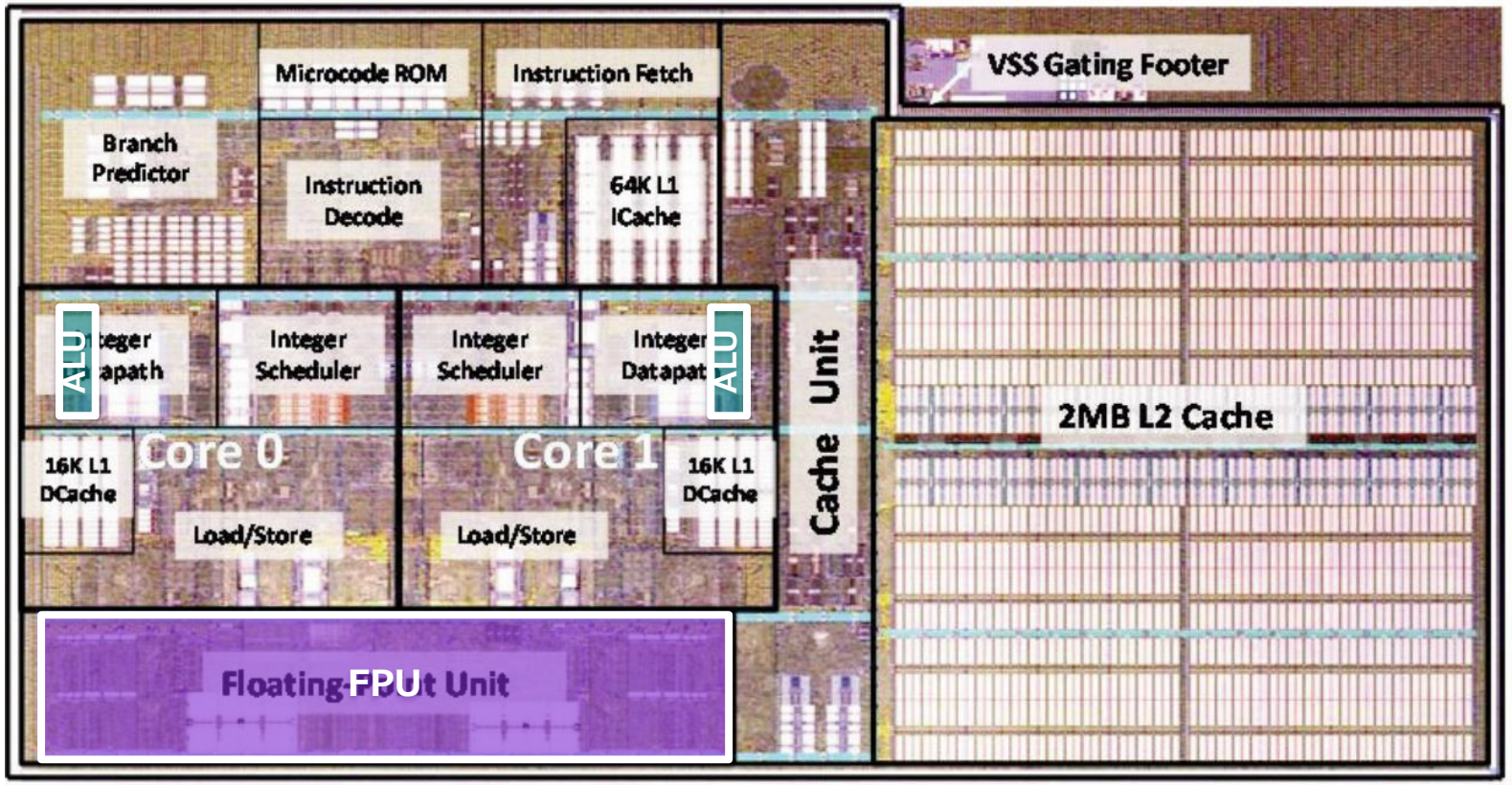
- 64bit 浮動小数点 乗算+加算器：　200kトランジスタ

Ryota Shioya, Masahiro Goshima, and Hideki Ando: A Front-end Execution Architecture for High Energy Efficiency, MICRO 2014 より

回路規模の例からわかること

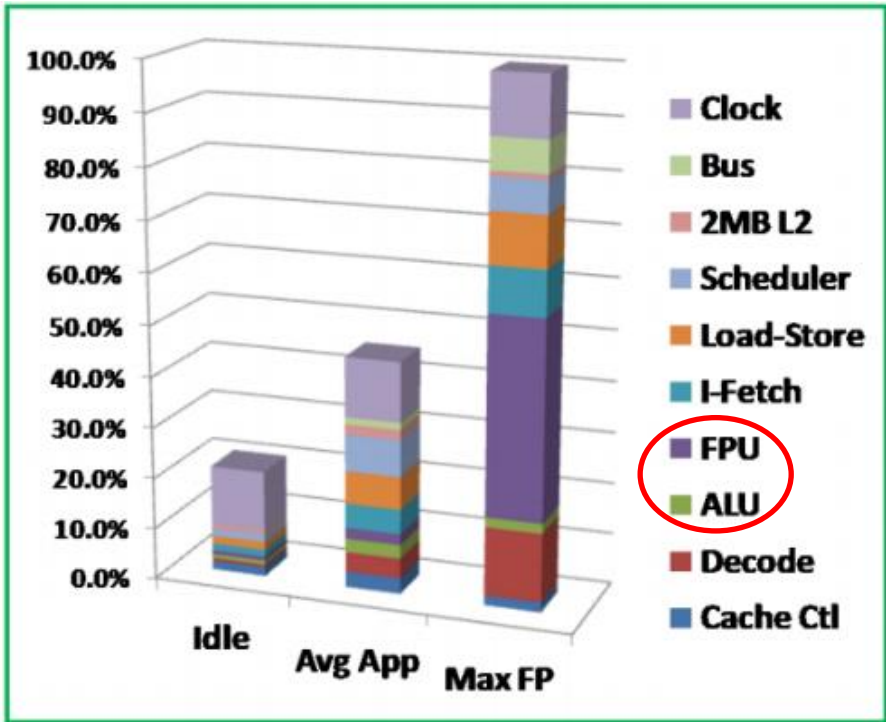
- MIPS プロセッサ全体に対する相対的な大きさで考える：
  - 1ビット論理演算：
    - 命令制御がほぼ全てを占める
  - 64 ビット加算
    - 命令制御が大半を占める
  - FP 演算
    - 演算器の方が大きい
- 消費エネルギーも，これに準じた大きさとなる

AMD Bulldozer のチップ写真



- Tim Fischer1 , Srikanth Arekapudi2 , Eric Busta1 , Carl Dietz3 , Michael Golden2 , Scott Hilker2 , Aaron Horiuchi1 , Kevin A. Hurd1 , Dave Johnson1 , Hugh McIntyre2 , Samuel Naffziger1 , James Vinh2 , Jonathan White4 , Kathryn Wilcox, Design Solutions for the Bulldozer 32nm SOI 2-Core Processor Module in an 8-Core CPU, ISSCC 2011 より

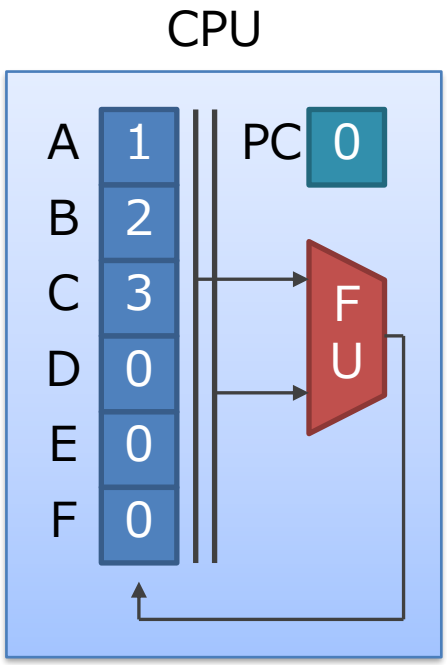
AMD Steamroller の消費電力

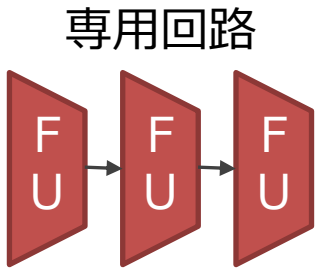


- ALU：整数演算器
  - ◇ 全体からみると，大きな割合を占めていない
- FPU：FP 演算器
  - ◇ 稼働時（Max FP）には，かなり大きな割合を占める

エネルギーは命令制御と演算の比率によって決まる

- CPU：演算以外の制御部分が大きい
  - ◇ 基本１つの命令が１つのデータを操作
  - ◇ 高速化のため命令の実行順を入れ替える等もする
- GPU：制御部分が相対的に小さい
  - ◇ １つの命令で多数のデータを操作
  - ◇ 命令フェッチ/デコードなどに必要な分が減る
- 専用回路：制御部分やレジスタがない
  - ◇ そもそも命令で処理しない
  - ◇ 演算器のみが繋がったような構造に流し込む





使いやすさは，おおむね制御部分の大きさに比例

- CPU：制御部分が大きい
  - ◇ ほっといても，ハードが（ある程度）勝手に並列実行してくれる
  - ◇ プログラマは一番楽
- GPU：制御部分が小さい
  - ◇ 単一の命令で複数のデータを操作
  - ◇ 規則正しくデータが並んでいるようにお膳立てしないと性能がでない
- 専用回路：制御部分がない
  - ◇ そもそもプログラムを実行できない
  - ◇ 目的ごとに回路の設計からしないといけない

CPU やその他回路の消費電力について

- 消費電力について
  1. クロックの消費電力
  2. アーキテクチャの違いによる消費電力の違い
  3. **FPGA による回路**

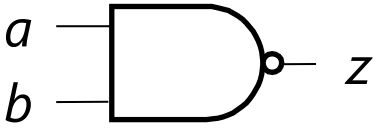
FPGA の場合

- FPGA：Field-Programmable Gate Array
  - ◇ 中身を書き換えることのできる回路
- FPGA で専用回路を作れば，いいことばかり？
  - ◇ 設計の敷居が下がりつつ，電力効率もよくなる？
  - ◇ CPU で実行されるプログラムの処理を専用回路に置き換える
- そんなに単純な話ではない
  1. FPGA の仕組み
  2. FPGA でうまく行く場合と行かない場合

FPGA の仕組み

- 書き換え可能なテーブルにより，回路を実現
  - LUT：Look up Table
    - 真理値表そのものを保持するテーブル
    - 事前にこれを所望の回路の真理値表に設定しておく
  - 入力をインデックスとしてテーブルにアクセスし，出力
    - 下の NAND の場合，a と b の 2ビットのインデクス

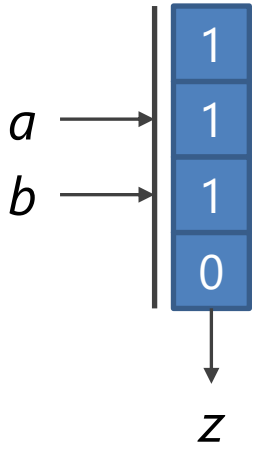
NAND



真理値表

a	b	z
0	0	1
0	1	1
1	0	1
1	1	0

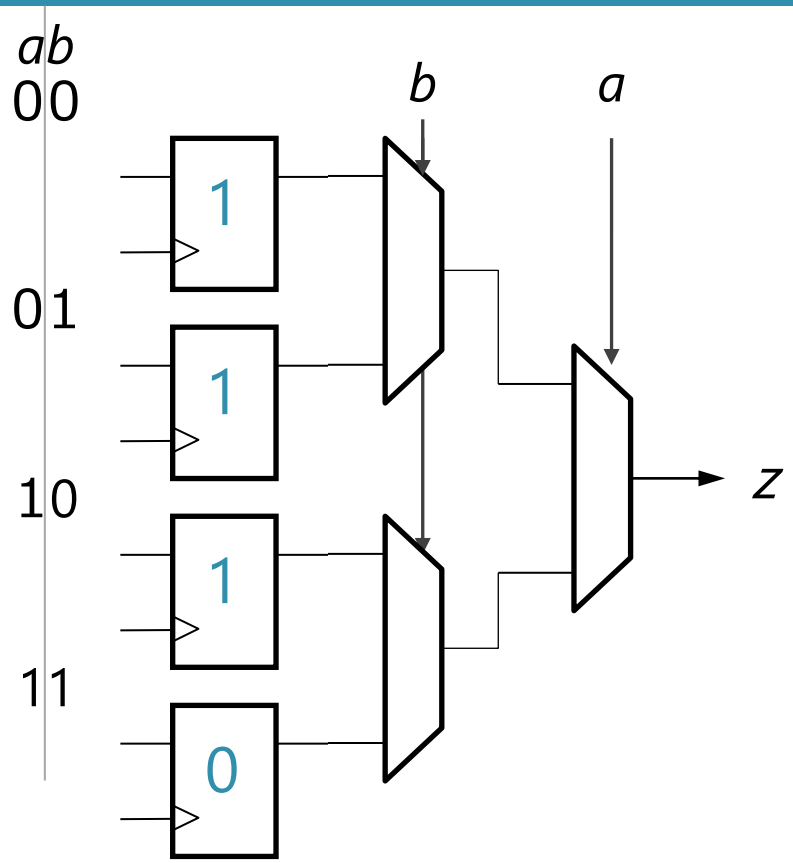
LUT



LUT の回路量の見積もり

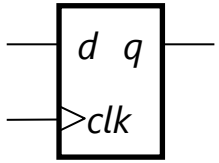
NAND

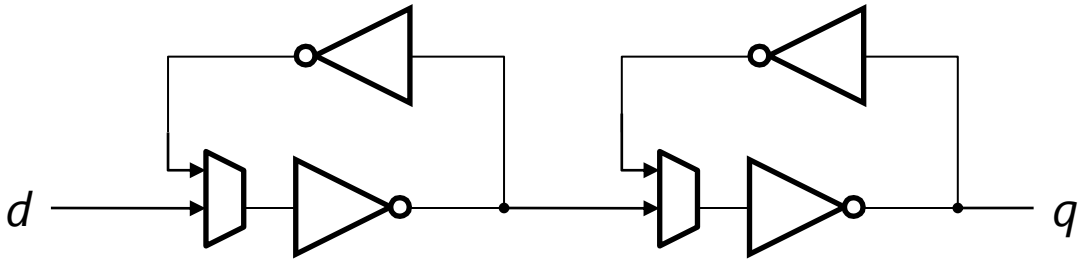
a	b	z
0	0	1
0	1	1
1	0	1
1	1	0

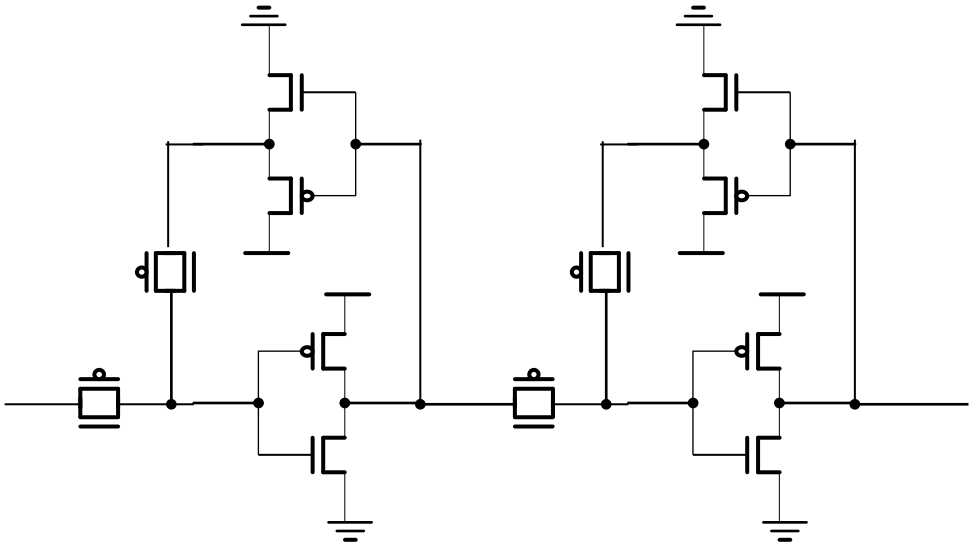


- ４エントリの LUT を D-FF で構成してみる
  - ◇ 中身を憶える 4つの D-FF
  - ◇ 場所を指定して選択する2段のマルチプレクサ

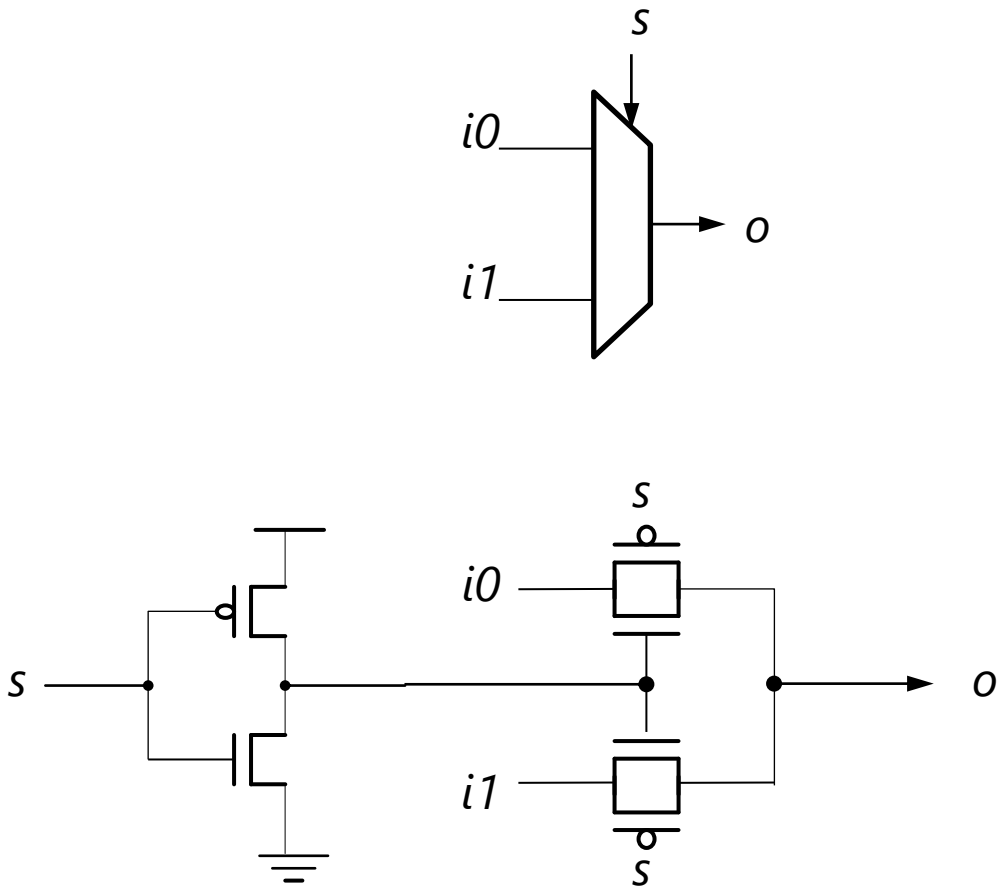
D-FF：トランジスタ 16個







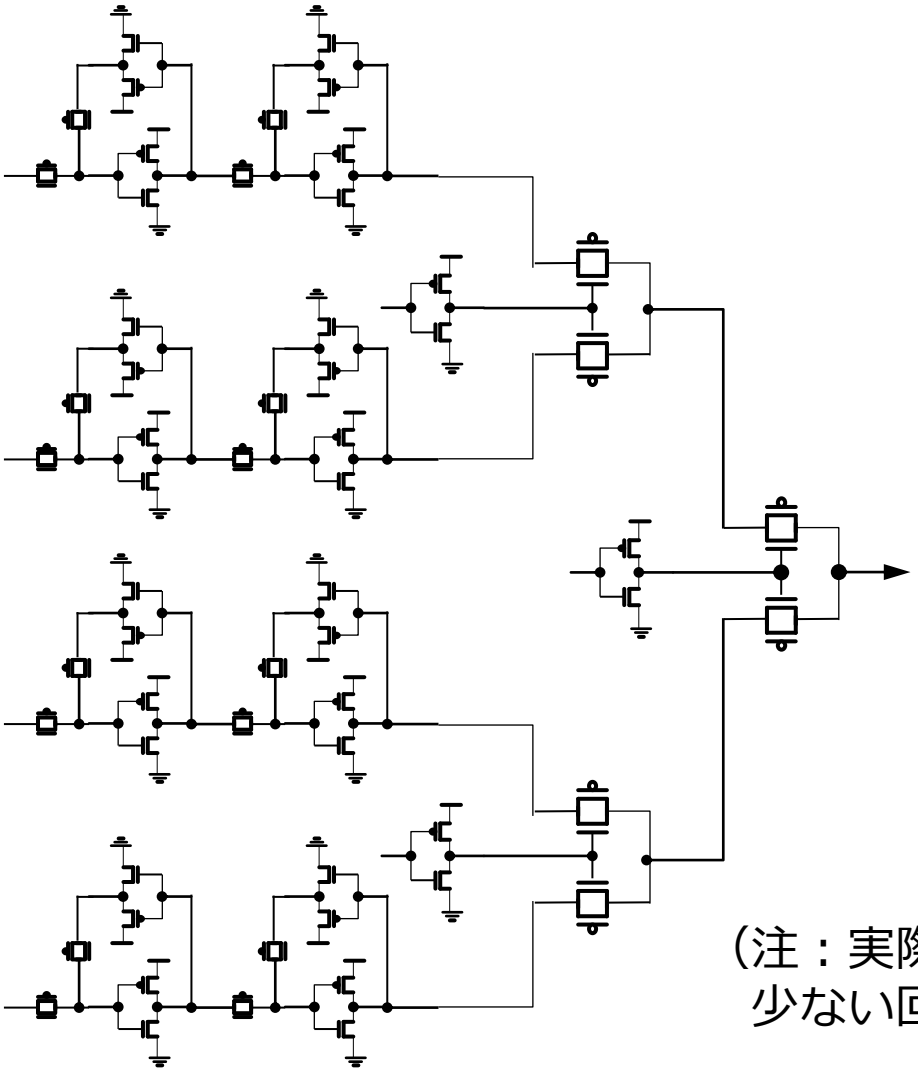
マルチプレクサ：トランジスタ 6個

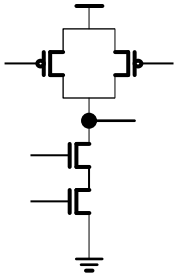


LUT vs. NAND
同じ回路を LUT で実現するのはものすごく効率が悪い

LUT：16*4+6*3=82

NAND： 4





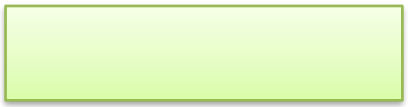
（注：実際にはもう少しトランジスタ数の
少ない回路でできています

LUT で回路を構成した場合

- このほかに，LUT 間を結合するためのネットワークも必要
- 結果として，直接回路を作るより１～２桁は下記の特性が悪化
  - ◇ 回路面積
  - ◇ 遅延
  - ◇ エネルギー

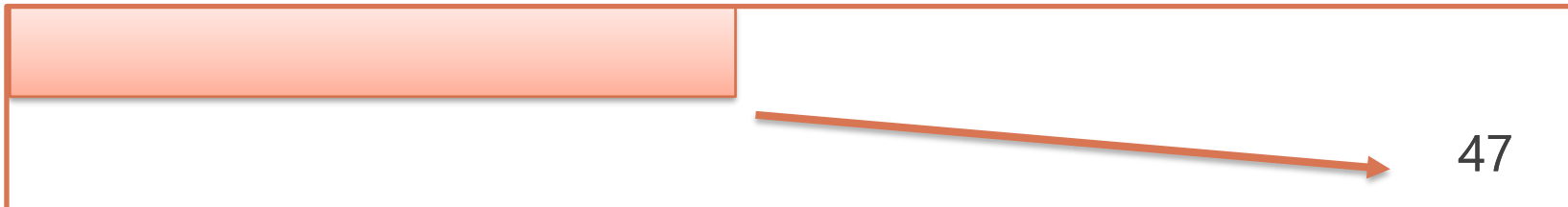
CPU から FPGA にしたときに良い場合・悪い場合

- MIPS R3000 プロセッサ：　115k トランジスタ
  - ◇ これの上で動くプログラムを FPGA に置き換えた場合を考える



- 1bit NAND 演算器：　4 トランジスタ
  - ◇ LUT によって 82 トランジスタになっても十分上記より小さい
    （小さすぎて見えない
- 64bit 整数加算器：　4k トランジスタ
  - ◇ 20倍大きくなり 80k になると，MIPS でやるのとほとんど変わらない

- 64bit 浮動小数点 乗算+加算器：　200kトランジスタ
  - ◇ FPGA にすると巨大になりすぎるし，何もおいしくない



FPGA の特性のまとめ

- トレードオフによって，最終的な優劣がきまる
  - FPGA 良い点：専用回路をくめば，命令制御に必要な資源が不要
    - データの受け渡のためのレジスタ・ファイルなども不要になる
  - FPGA 悪い点：回路としては一般に 1 桁から 2 桁程度性能が悪化
- 演算の種類と複雑さで FPGA 化したときにおいしいかどうかは決まる
  - 既に CPU や GPU に演算器が載っているような FP 演算などは逆効果になりかねない
- 実際には，FPGA にはよく使われる回路は LUT ではないものが入っていることも多い
  - 加算器や乗算器など

ここまでのまとめ

1. クロックの消費電力
   - ◇ クロックの消費電力が CPU 全体に占める割合は大きい
   - ◇ チップ全体の D-FF とそれへの配線を毎サイクル充放電するから

2. アーキテクチャの違いによる消費電力の違い
   - ◇ 回路は命令制御と演算に大きく分けられる
   - ◇ 命令制御に回路を割くと消費電力が大きくなるが,
     プログラマは楽に

3. FPGA による回路
   - ◇ FPGA は直接回路を作るのと比べるとかなり効率が悪い
   - ◇ CPU で動いているプログラムを FPGA の専用回路にした場合,
     おいしいかどうかは演算の複雑さで決まる

命令パイプライン

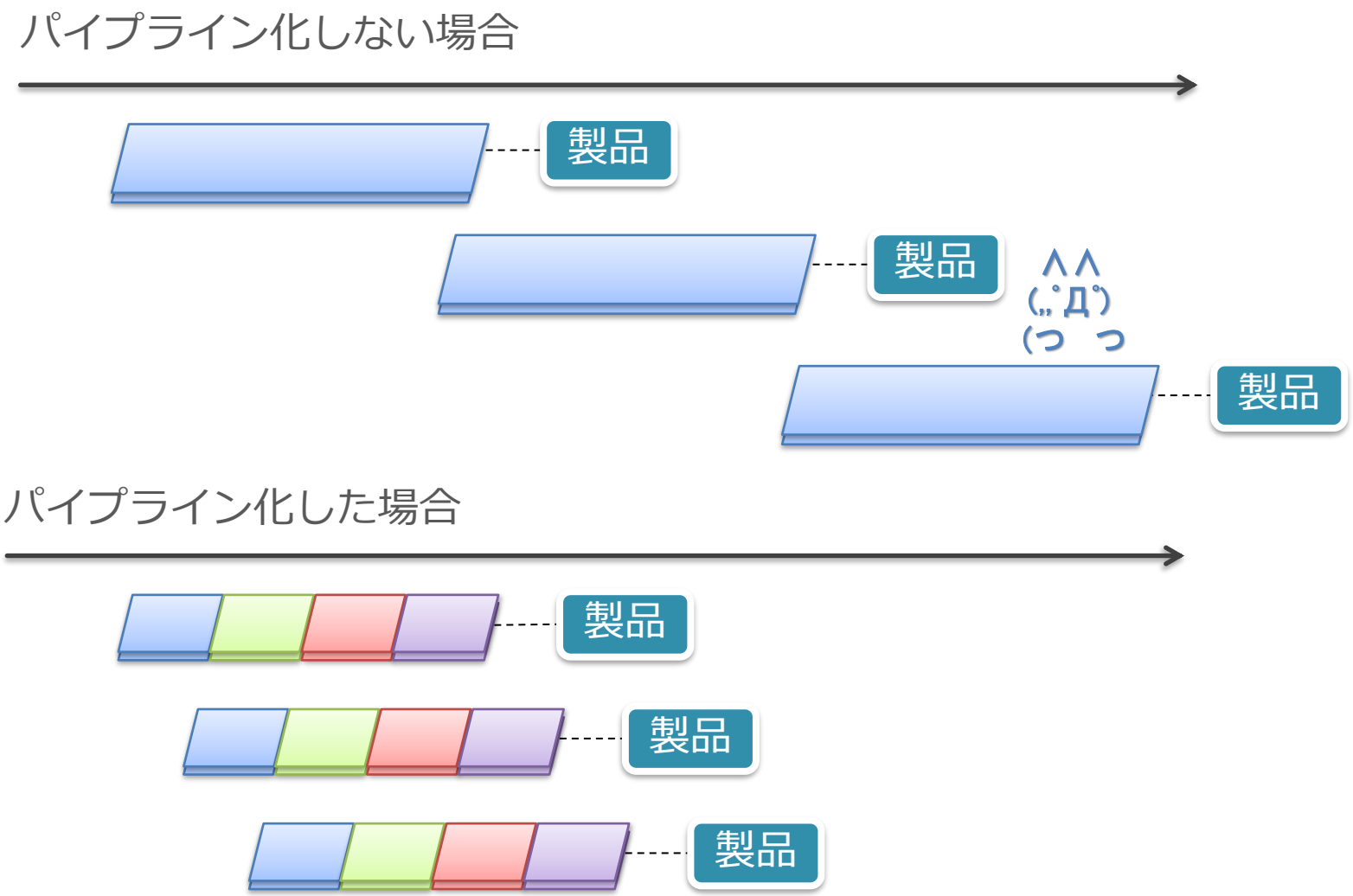
導入：工場のラインを考える

- ベルトコンベアのラインの上を製品が流れていく
  - ◇ ４人の人が，それぞれの工程の作業をおこなって完成
- 上のように1つしか製品をながさないと，
  - ◇ 各人は他の人が作業している間はヒマ

導入：工場のラインを考える

- 実際の工場：複数の製品を同時に流す
  - ◇ 各工程を並列して処理することによりスループットを向上
  - ◇ さっきの4倍の速度で製品ができあがっていく
- これが 命令パイプライン

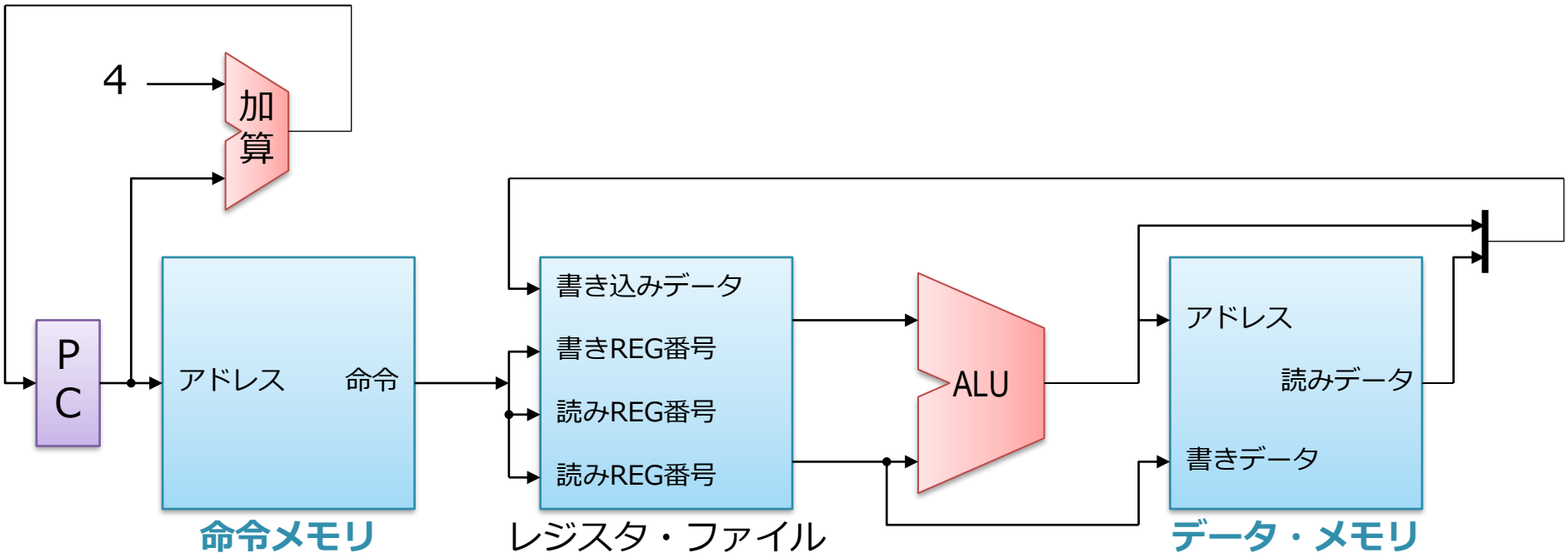
パイプライン化による性能向上



命令パイプライン

1. シングル・サイクル・プロセッサの動作
   - ◇ パイプライン化を前提とした構造のものを使って復習
   - ◇ 全ての命令の処理が１サイクルで完結
2. 上記のパイプライン化
   1. 具体的にどうパイプライン化するか
3. ハザード

ベースとなるシングル・サイクル・プロセッサ



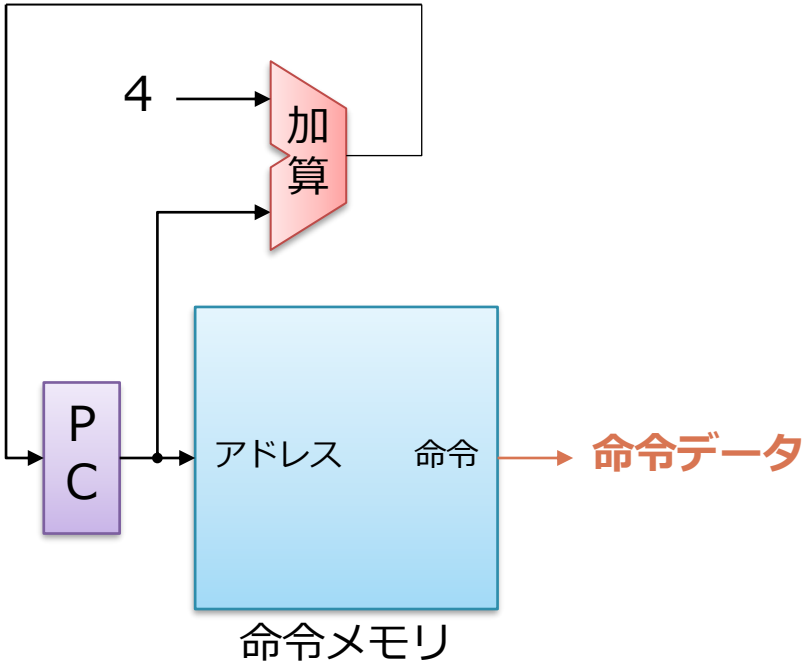
- 以前説明したものとの違い：
  - ◇ メモリが命令メモリとデータメモリに別れている
  - ◇ 算術 & 論理演算，ロード，ストアのみを実行可能
    - □ 分岐とジャンプは，簡単のために今は考えない

1命令の実行フェーズ

- 実行フェーズ
  1. フェッチ
  2. デコード
  3. レジスタ読み出し
  4. 実行
  5. レジスタ書き戻し

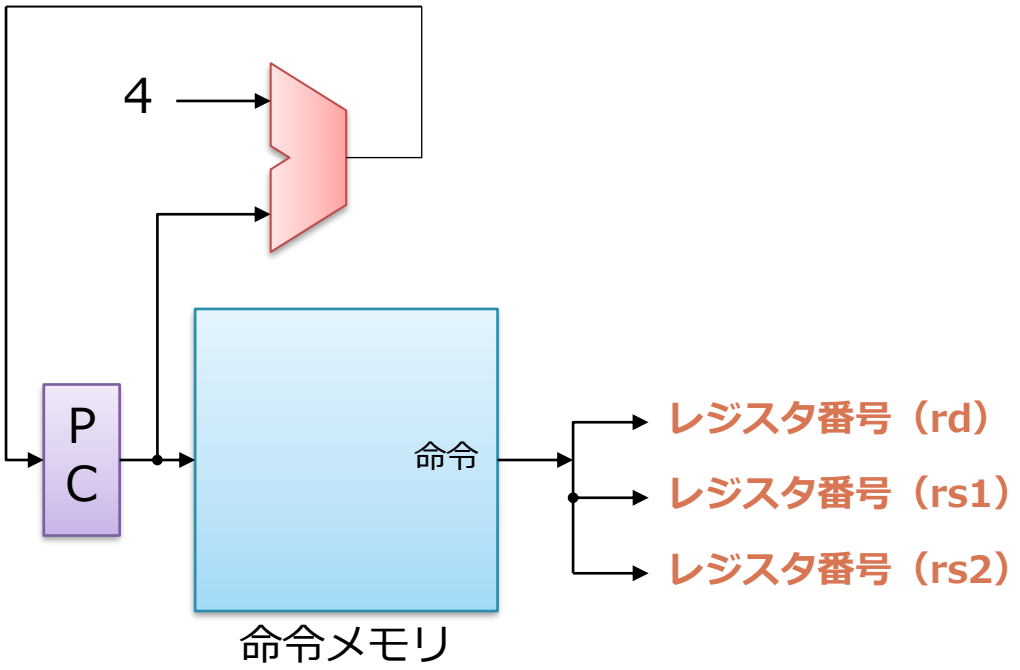
- RISC-V の加算命令を実行する流れをざっとみる

命令フェッチ

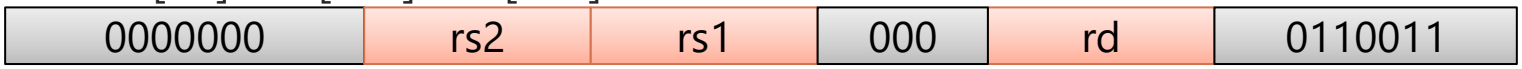


- 命令メモリから命令を読み出す
  - 命令メモリを順に読んでいくため，PC は毎サイクル加算される
  - 足している４は，RSIC-V では命令の幅が４バイトだから
  - 基本的に，この部分はどの命令でも変わらない

命令デコード

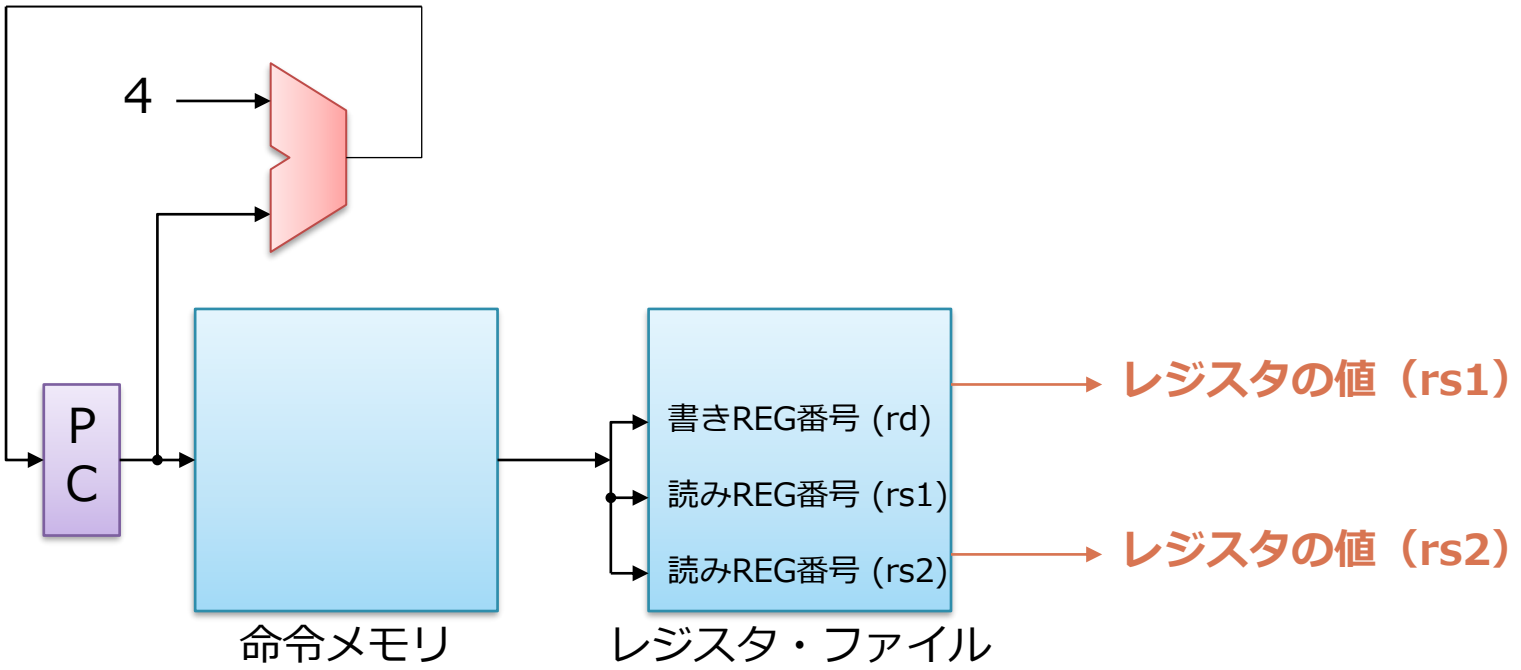


ADD : x[rd] ← x[rs1] + x[rs2]



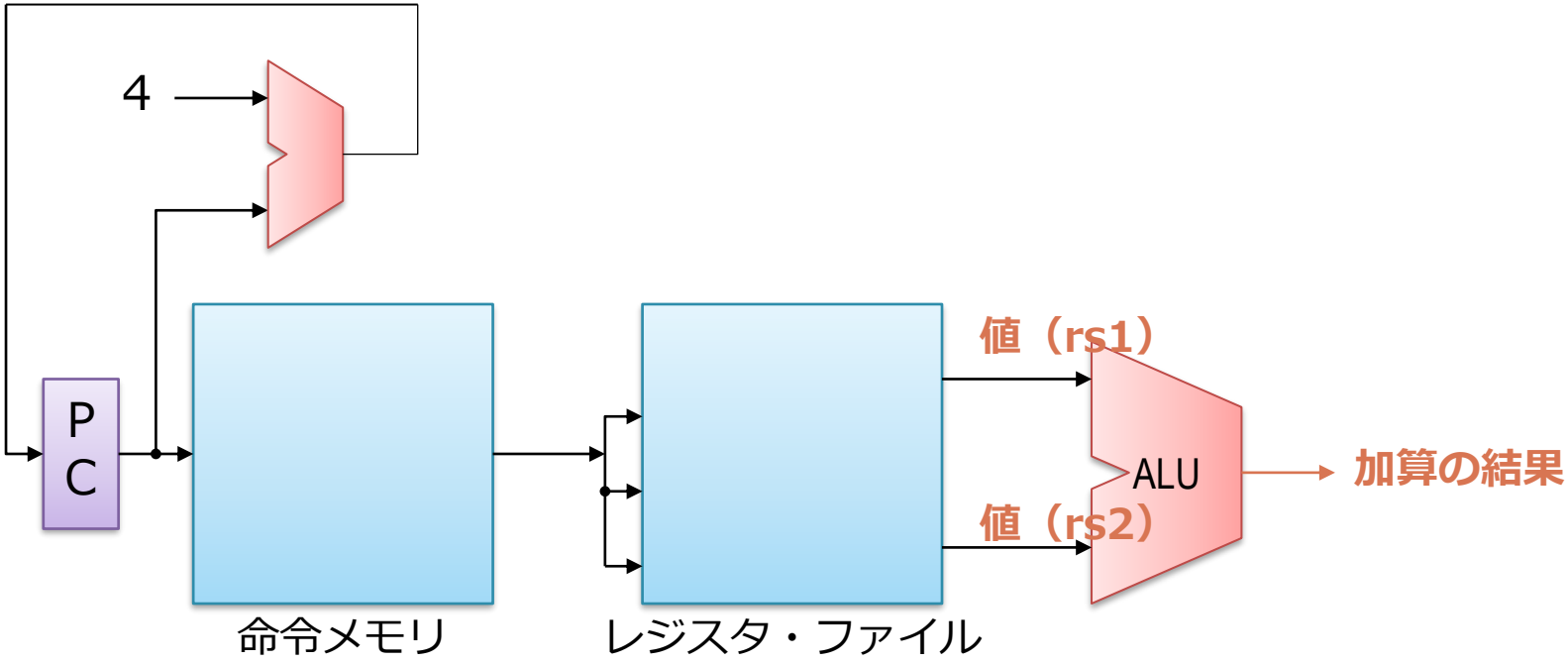
- 取り出した命令からレジスタ番号を表す部分のビットを取り出す
  - ソース（rs1, rs2）とディスティネーション（rd）

レジスタ読み出し



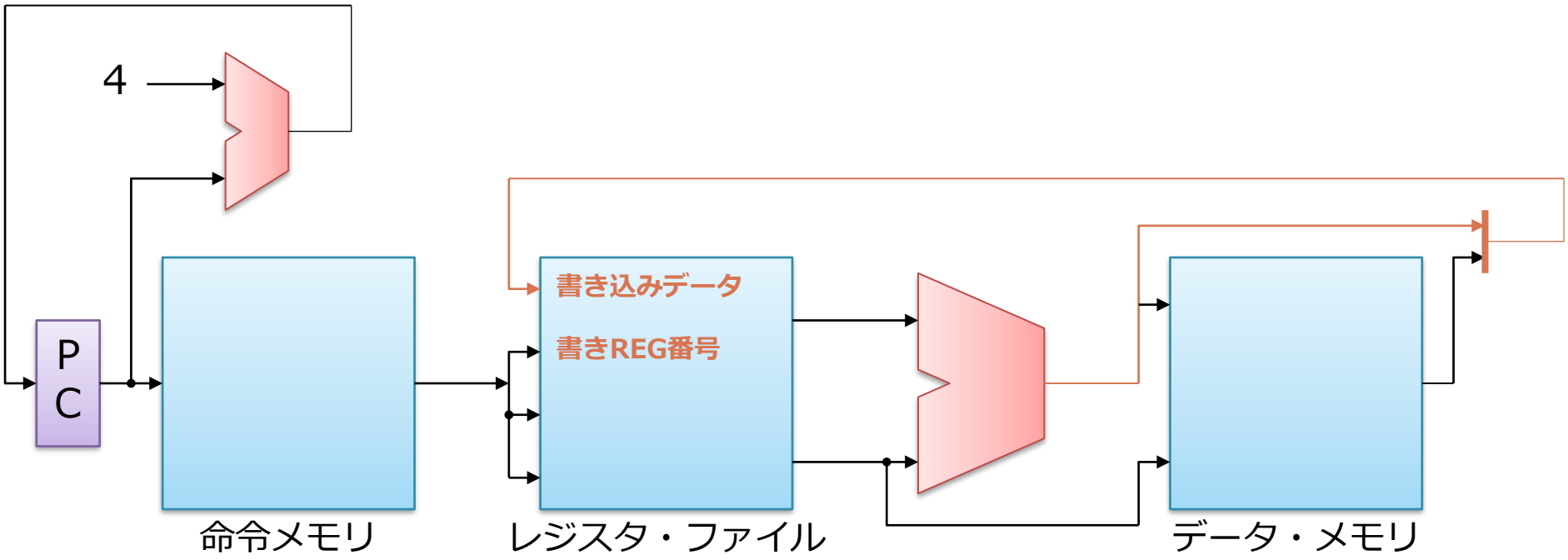
- デコードで得られたレジスタ番号を使って RF にアクセス
  - ◇ ソース・オペランドの値を読み出す

実行



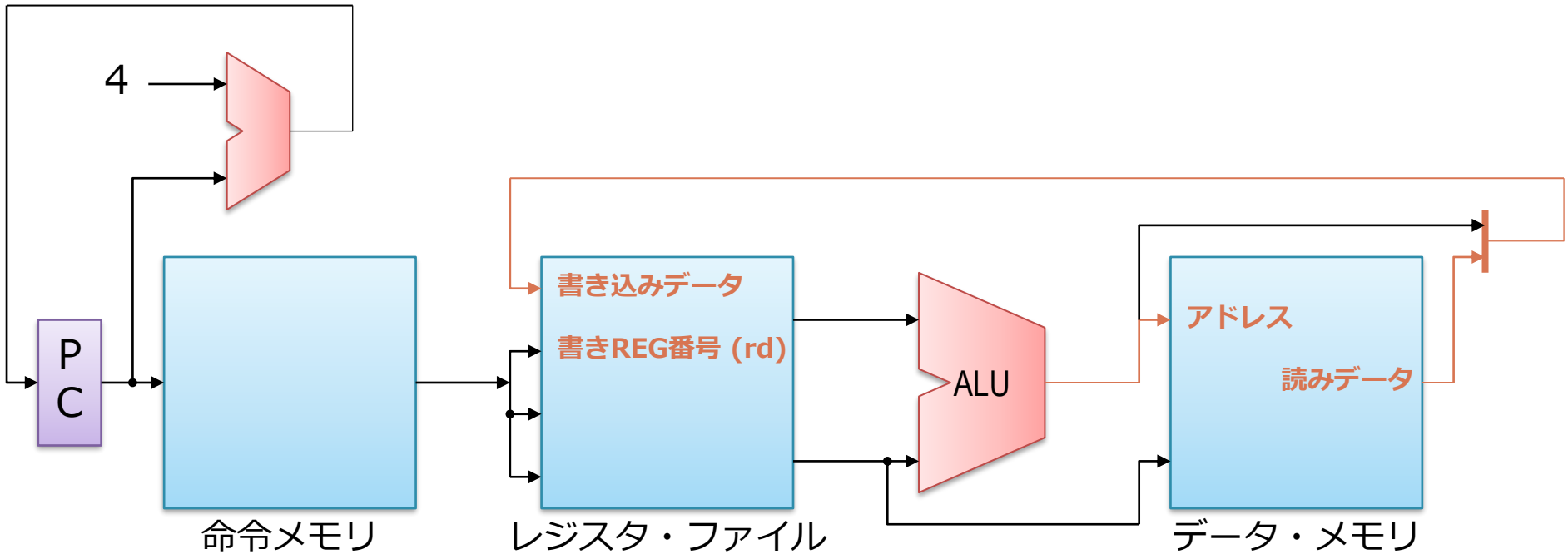
- RF から読みだした２つの値を加算

レジスタ書き戻し



- 加算の結果をレジスタ・ファイルに書き戻す
  - ◇ データ・メモリには用がないので何もしない

ロードの場合：メモリ・アクセスが加わる

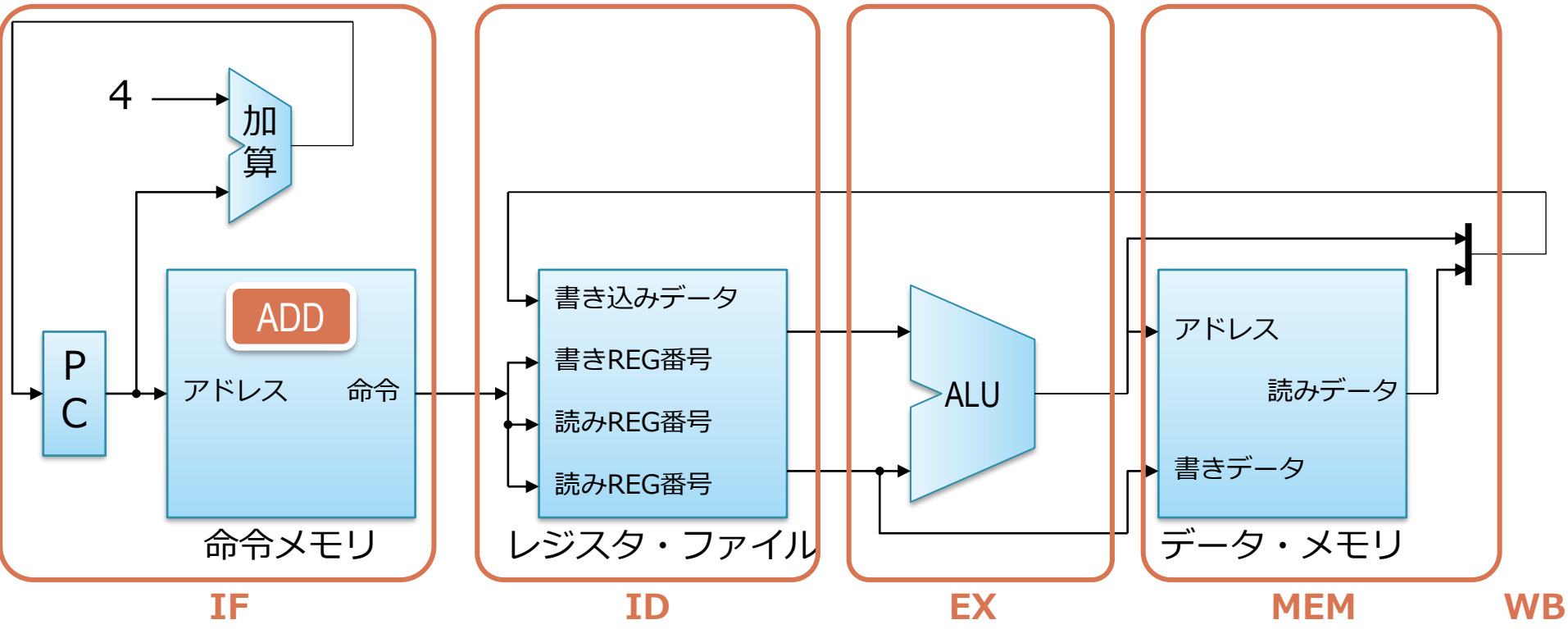


LW : x[rd] ← (x[rs1] + immediate)

immediate[11:0]	rs1	010	rd	0000011

- 加算命令との違い：
  - ◇ アドレスの計算（x[rs1] + immediate）を ALU でやる
  - ◇ 得られたアドレスでデータ・メモリにアクセス

各処理は基本的には左から右に流れる

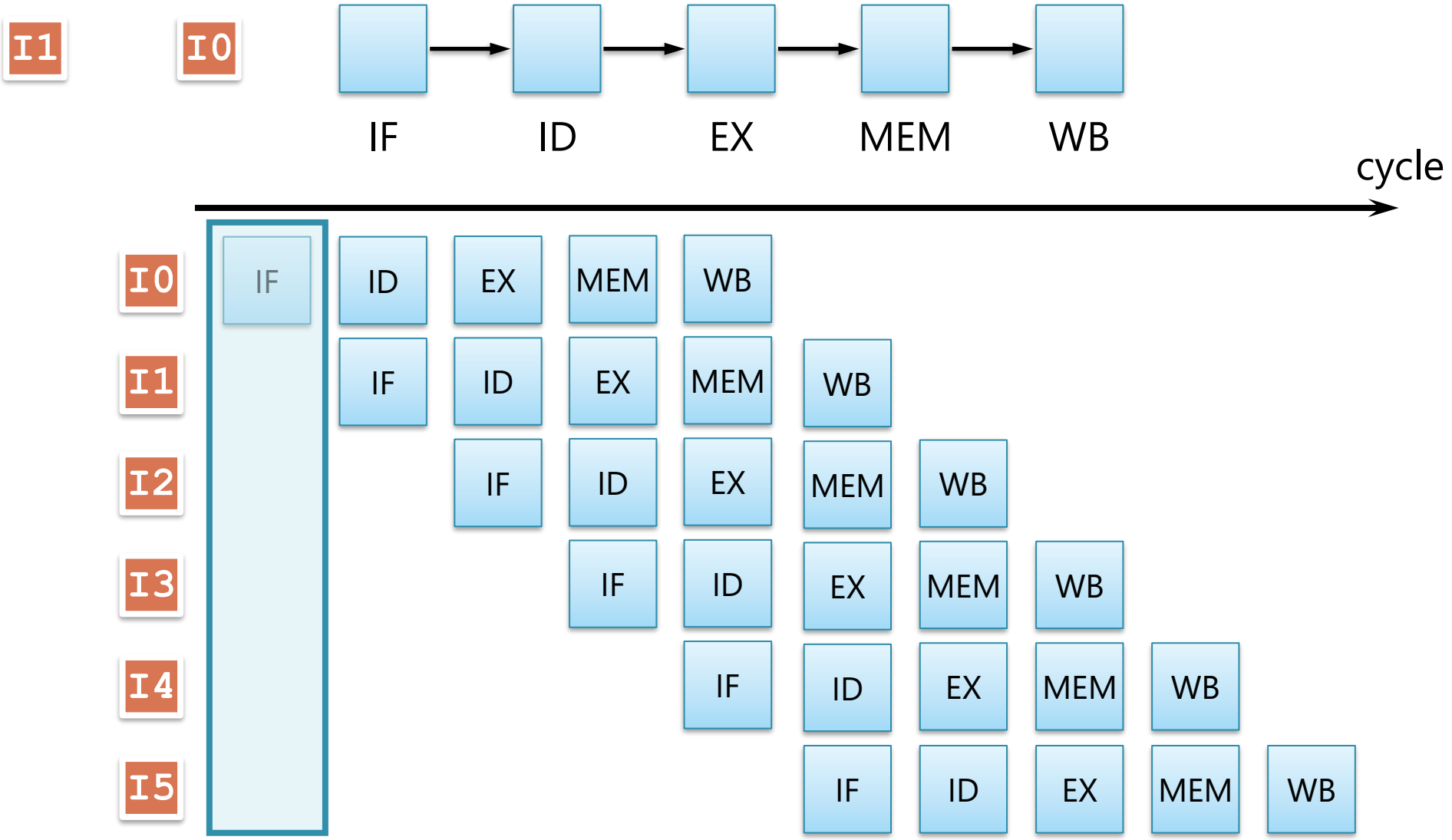


- 特定のユニットで仕事をしている間，他の部分は遊んでいる
- パイプライン化
  - ◇ これをもとに，導入で話したように処理をオーバーラップさせる

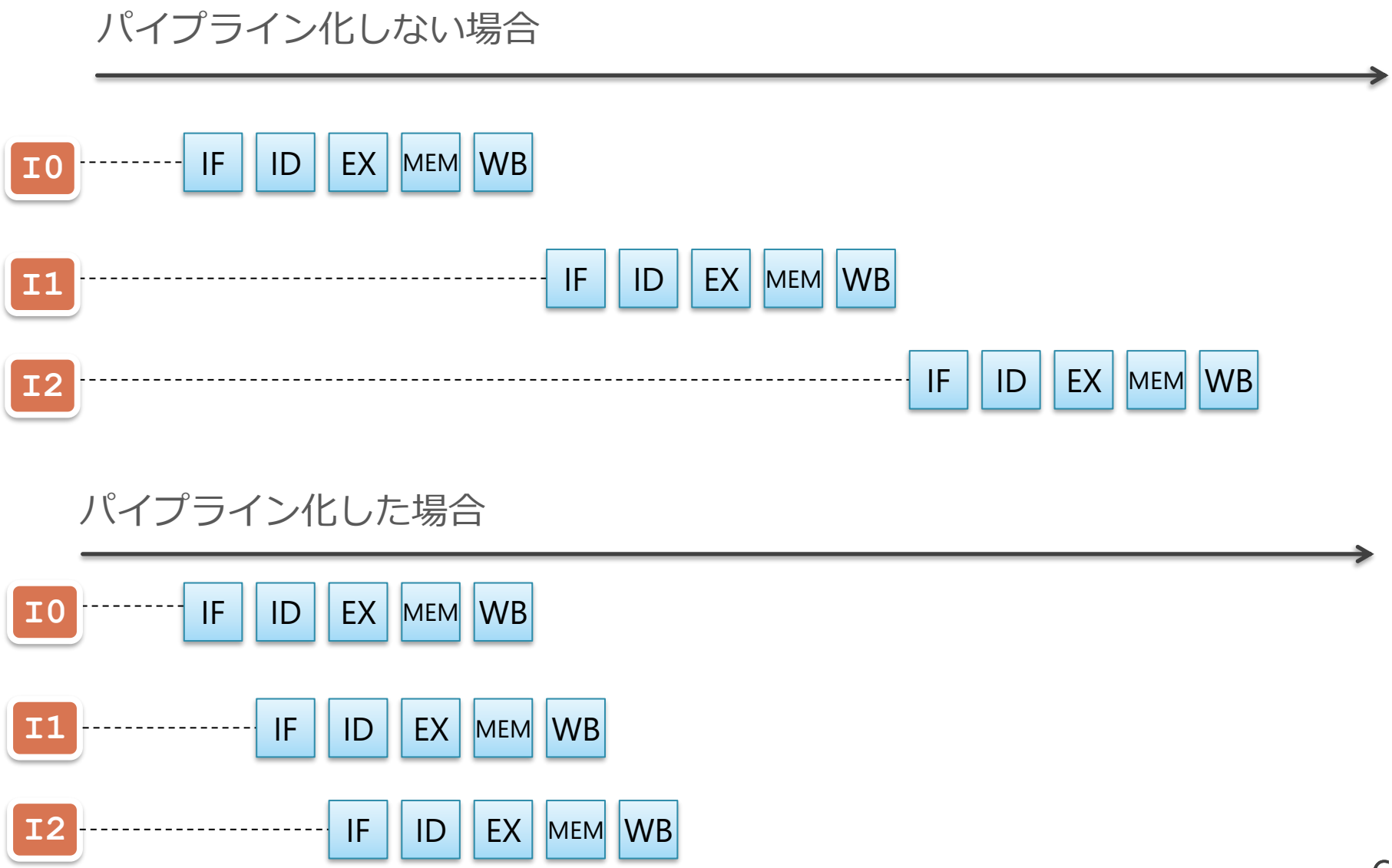
パイプライン化

- 回路のまとまりをオーバラップさせる単位にする
  - ◇ この単位をステージと呼ぶ
- ステージ
  1. IF： 命令フェッチ
  2. ID： デコードとレジスタ読み出し
  3. EX： 実行
  4. MEM：メモリ・アクセス
  5. WB： レジスタ書き込み

命令パイプラインの実行の様子



パイプライン化による性能（スループット）向上

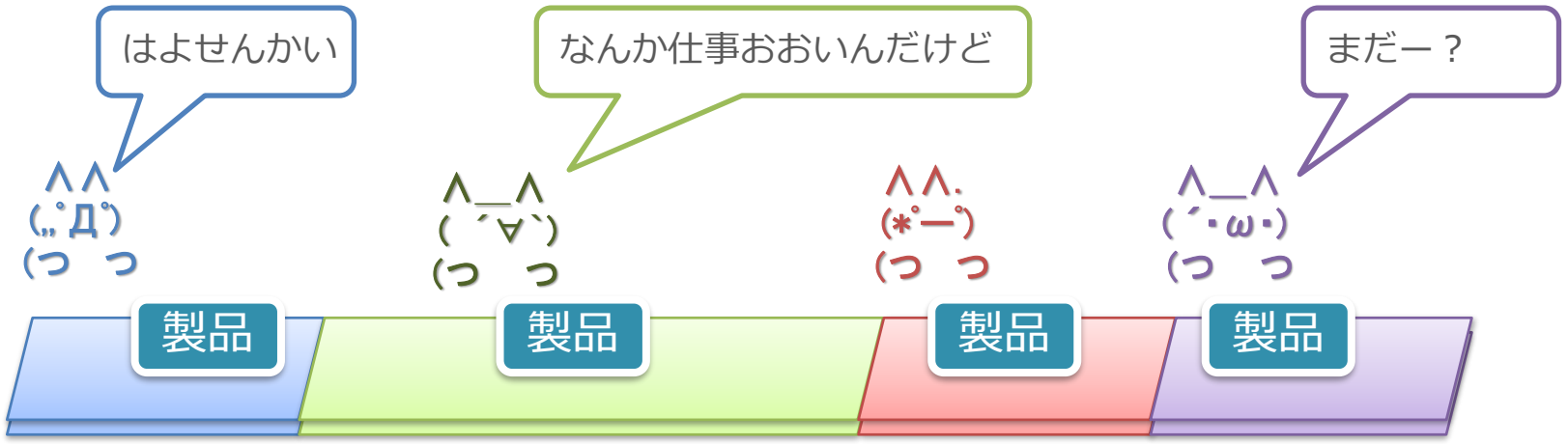


パイプライン化の効果

- レイテンシ (latency)：　短くならない（か，やや延びる）
  - ◇ 一続きの処理が始まってから終わるまでにかかる時間
  - ◇ この場合，1命令の始まりから終わりまでの処理時間
  - ◇ 原理的に短くならない（ステージ間にFF が入る分のびる）
- スループット (throughput)：　ステージ数倍だけ上がる
  - ◇ 単位時間当たりの処理量
  - ◇ この場合，単位時間あたりに実行される命令数

ステージはどこで切るか

- 大きな回路のまとまりをステージにする
  - ◇ 回路のまとまりが大きい → 遅延も大きい
- この遅延の大きさが揃っていないと，綺麗にうごかない
  - ◇ パイプライン全体は，一番遅いステージの遅延にあわせて動く
  - ◇ 他の人が仕事が終わったからと言って，先に送れない
- 良くない例：緑の人だけ仕事が多いので，全体が動かせない



ステージはどこで切るか

- ステージ
  1. IF：　命令フェッチ
  2. ID：　デコードとレジスタ読み出し
  3. EX：　実行
  4. MEM：メモリ・アクセス
  5. WB：　レジスタ書き込み

- 上記では，デコードとレジスタ読み出しが ID ステージにまとめられている
  - ◇ デコードにかかる遅延はほとんどない
  - ◇ 読み出した命令からオペランドを取り出すのは，単に信号線を繋ぐだけで良い

命令パイプライン

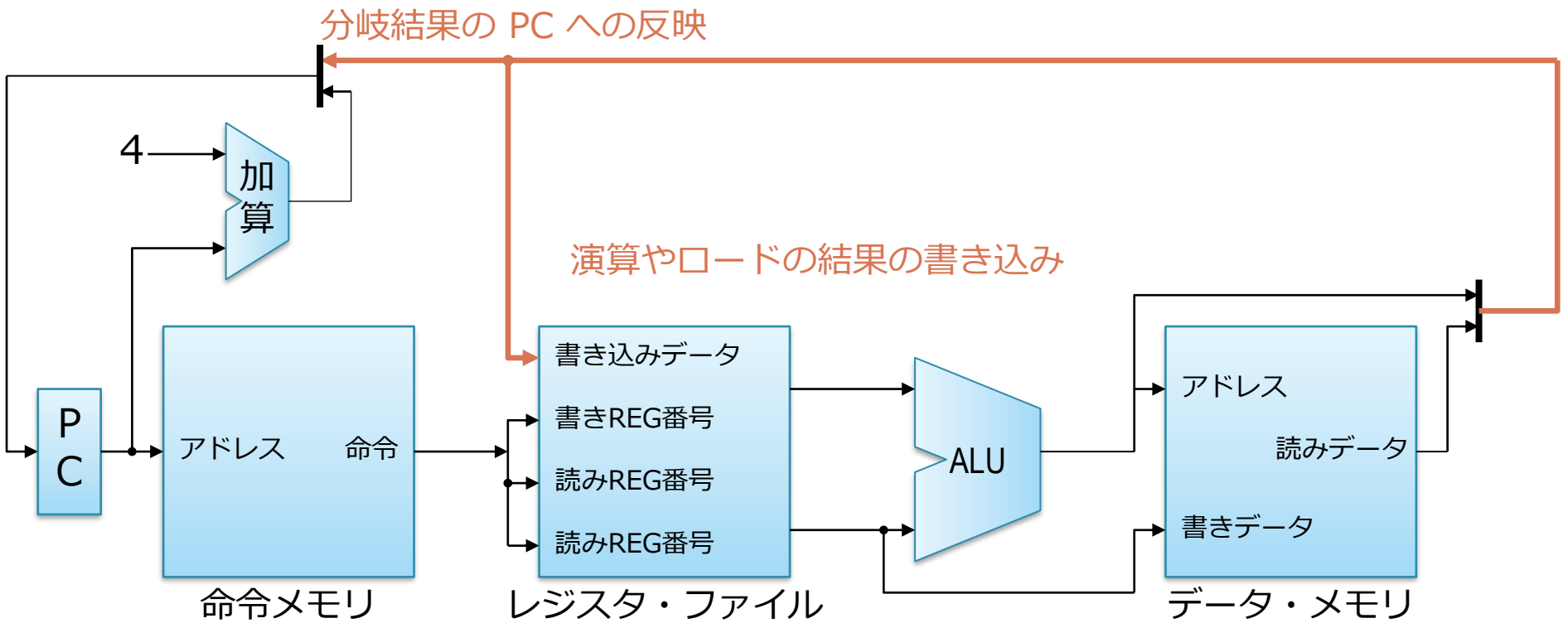
1. シングル・サイクル・プロセッサの動作
   - ◇ パイプライン化を前提とした構造のものを使って復習
   - ◇ 全ての命令の処理が１サイクルで完結
2. 上記のパイプライン化
   1. 具体的にどうパイプライン化するか
3. **ハザード**

ハザード

- パイプライン・ハザード
  - ◇ パイプライン動作を妨げる要因

- 分類：
  1. 非構造ハザード：　　バックエッジによる
     a. データ・ハザード：　データ依存
     b. 制御ハザード ：　　制御依存（分岐命令）
  2. 構造ハザード ：　　ハード資源の不足による

バックエッジ：逆方向（右から左）にいく信号

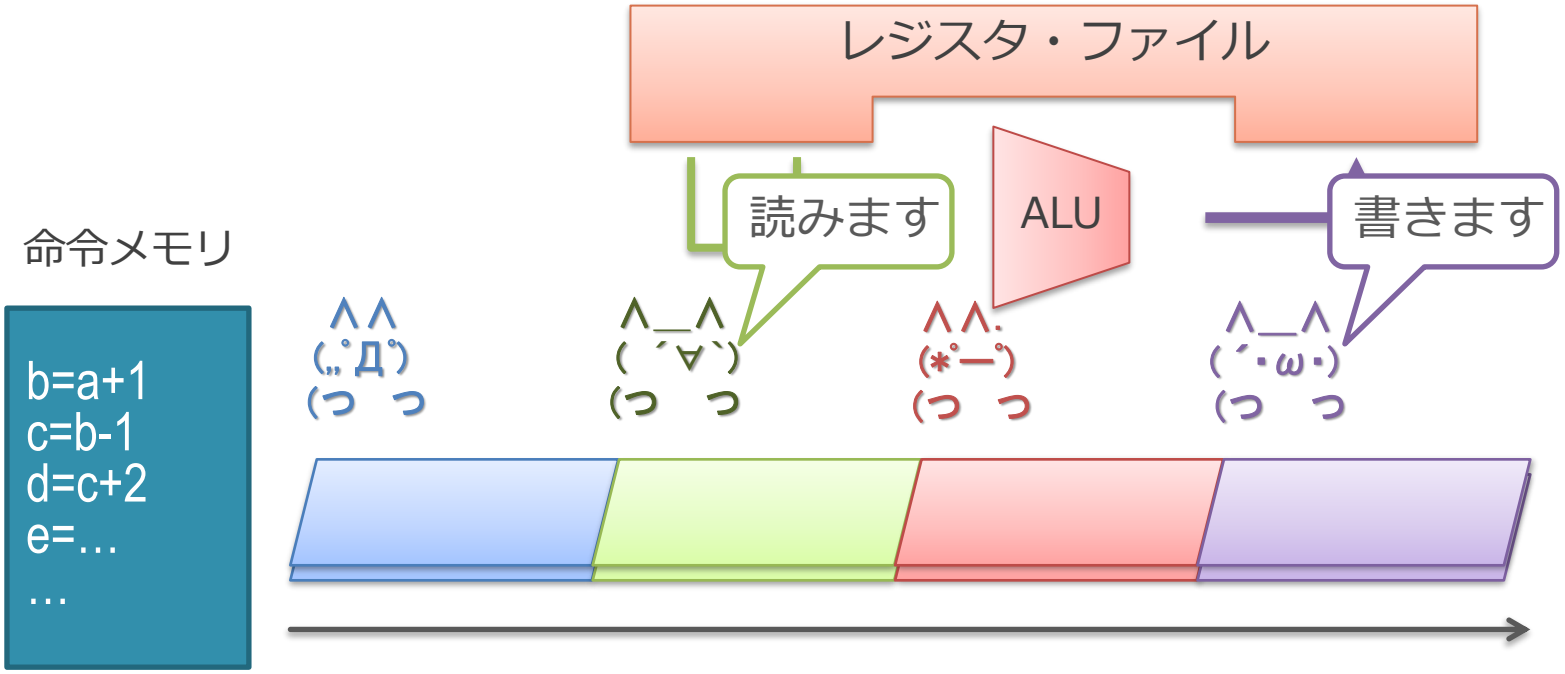


- バックエッジがあるため，命令を単純に流せない場合がある
  - ◇ 工場のラインのように，一方向に流せない

分岐命令の処理と制御ハザード

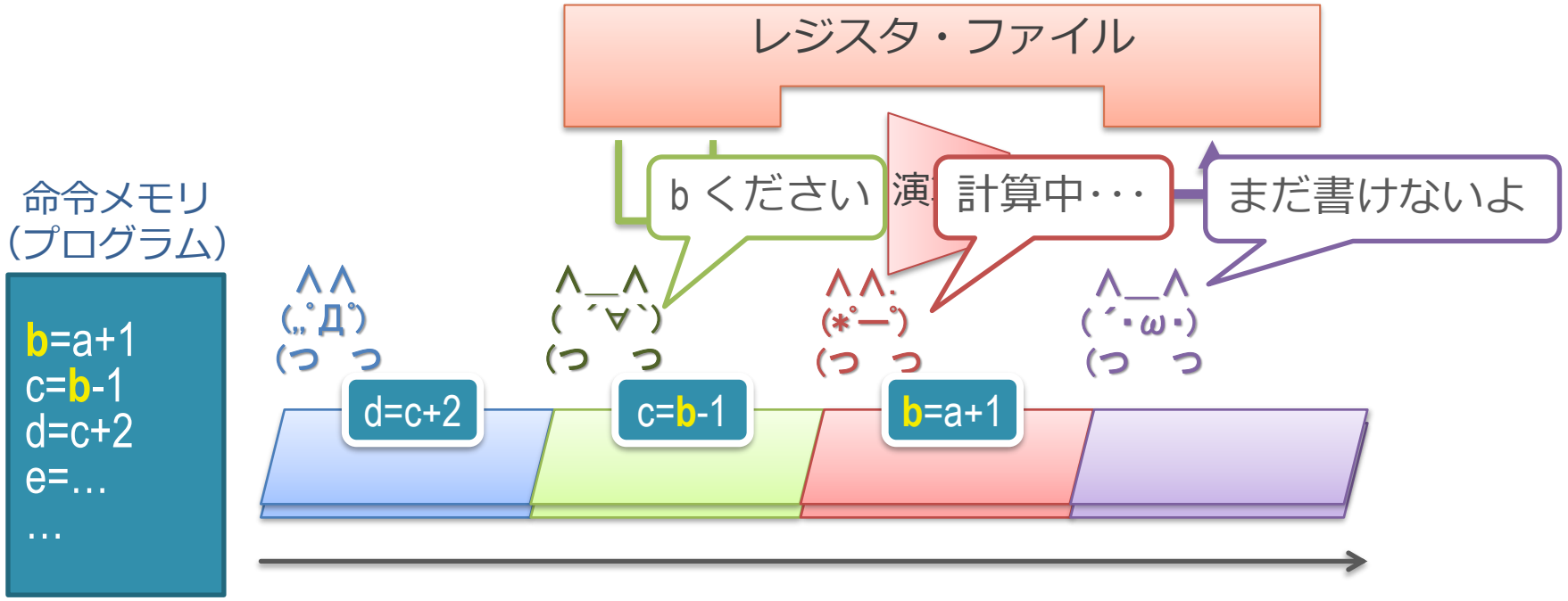
- 「if a > 0」の結果は最終段の (´･ω･) の人まで反映出来ない
  - ◇ 先頭は次に a=a+1 と a=a-1 のどちらを取り込めばいいのかわからない
- このままでは先に進めないので
  - ◇ 最も単純には，シングル・サイクルの時と同じだけ待つ

データ・ハザード



- レジスタ・ファイルへのアクセス
  - ◇ 演算の入力は(´∀`) の人がレジスタ・ファイルから読み出す
  - ◇ 演算の結果は(´･ω･) の人がレジスタ・ファイルに書き込む

データ・ハザード



- データ・ハザード
  - (´∀`) の人が **b**=a+1 の結果を読もうとしても,
  - (*ﾟ―ﾟ) の人がまだ計算中でレジスタ・ファイルに書いていない
  - (´･ω･) の人が計算結果をかけるのは次のサイクル
    - レジスタ・ファイルから読めるのはさらにその後
    - これも，単純には書き込みが終わるまでまつ

パイプラインのまとめ

- それぞれ以下について説明
  - ◇ 命令パイプライン
  - ◇ ハザードの基本
- 次回以降では，ハザードの詳細や対策についてまとめる
  - ◇ 単純に待っていたのでは，性能がすごい落ちる

まとめ

- 回路の消費エネルギー
  1. クロックの消費電力
  2. アーキテクチャの違いによる消費電力の違い
  3. FPGA による回路
- 命令パイプラインの基礎
  - ◇ パイプライン化によるスループットの向上
  - ◇ ハザード

- この講義資料では，一部，五島先生の「ディジタル回路」の講義資料の図を使用しています

出欠と感想

- 本日の講義でよくわかったところ，わからなかったところ，質問，感想などを書いてください（なんか一言書いてね）
  - LMS の出席を設定するので，そこにお願いします
  - パスワード：power
- 意見や内容へのリクエストもあったら書いてください