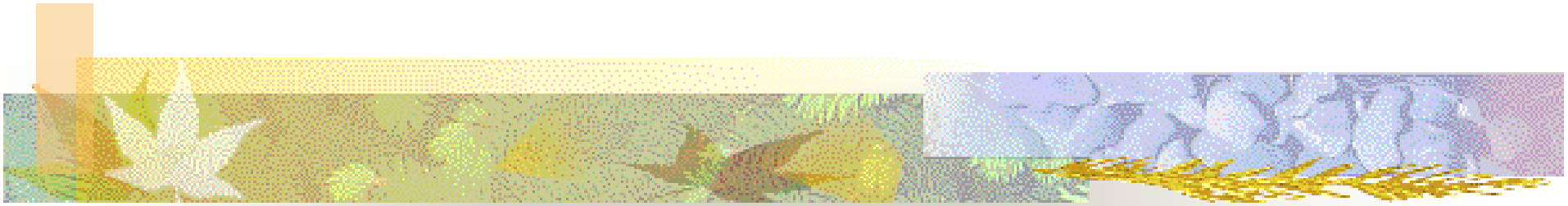
北海道大学
HOKKAIDO UNIVERSITY

コンピュータシステム
（アーキテクチャ　第６回）



工学部 情報エレクトロニクス学科
大学院 情報科学研究院 情報理工学部門
堀山 貴史

前回（アーキテクチャ第5回）の内容

アーキテクチャの基本知識（3）
（マイクロプロセッサ以降）

1. マイクロプロセッサの登場
2. マイクロプロセッサの進化
3. Apple 対 MS DOS / Windows (?)
4. UNIXとワークステーションの登場
5. Google登場
6. ムーアの法則

感じ取ってほしいこと

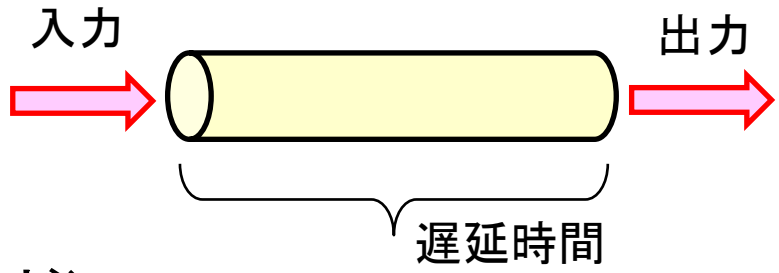
- アーキテクチャは、産業とリンクして発展してきた
- 次の時代を作るのは、皆さん
- ソフトウェアも、ハードウェアも分かって、
  すべてをフル活用できるように、色々と知ってほしい
  （ソフトもハードも、中身が分かっている人の方が、
  ブラックボックスで使うだけの人より強い）

今回の内容

並列処理アーキテクチャ

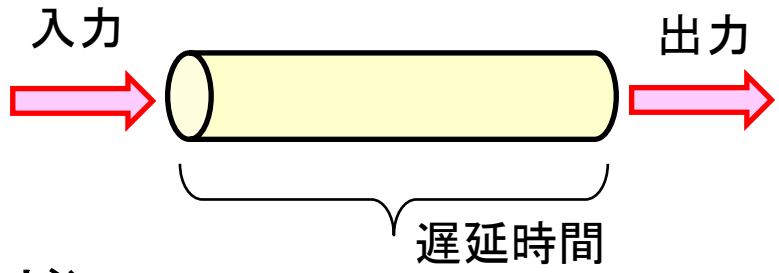
1. 計算機の処理能力の2つの要素
   - スループット、レイテンシ
2. 並列処理
   - 並行処理と並列処理、パイプライン並列とマルチプロセッシング並列
3. 並列処理の分類
   - SIMD と MIMD、粒度による分類
4. 並列アーキテクチャ
   - マルチプロセッサ・マルチコア(MIMD型)、SIMDプロセッサ
   - 命令パイプライン、演算パイプライン
5. ノイマン型計算機の限界
   - フォンノイマン・ボトルネック

1. 計算機の処理能力の2つの要素



- **スループット (Throughput)**
  - 単位時間あたりの処理能力
  - いわゆる処理速度(MIPS, MFLOPSなど)
  - データ転送で使う場合は断面を通過するデータ量(bit/s)
  - 入力データ量と出力データ量が同じならばデータ量＝処理量
- **レイテンシ (Latency)**

1. 計算機の処理能力の2つの要素

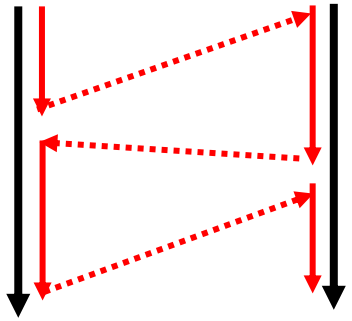


- **スループット (Throughput)**
  - 単位時間あたりの処理能力
  - いわゆる処理速度(MIPS, MFLOPSなど)
  - データ転送で使う場合は断面を通過するデータ量(bit/s)
  - 入力データ量と出力データ量が同じならばデータ量＝処理量
- **レイテンシ (Latency)**
  - 計算を始めてから答えが出てくるまでの遅延時間(sec)
  - メモリにランダムにアクセスする場合は、スループットだけでなくレイテンシが重要になる
  - 2次記憶などに連続的にデータ転送する場合は、レイテンシよりもスループットが重要

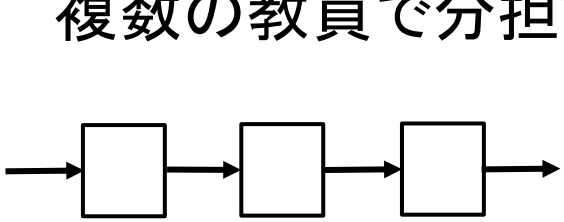
2. 並列処理

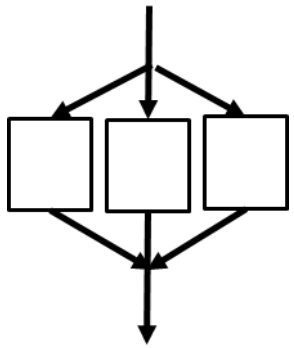
並行処理 と 並列処理

- 並行処理（concurrent processing）
  - 1個のプロセッサで、複数のプログラムを切り替えながら
    見かけ上同時に処理すること
    （例）１人の教員が複数の講義科目を同じ学期に担当する。



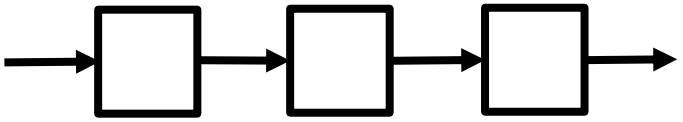
- 並列処理（multi-processing）
  - 複数のプロセッサで、1個のプログラムを
    手分けして同時に処理すること。
  - （例）1つの講義科目を3クラスに分け、
    複数の教員で分担する。



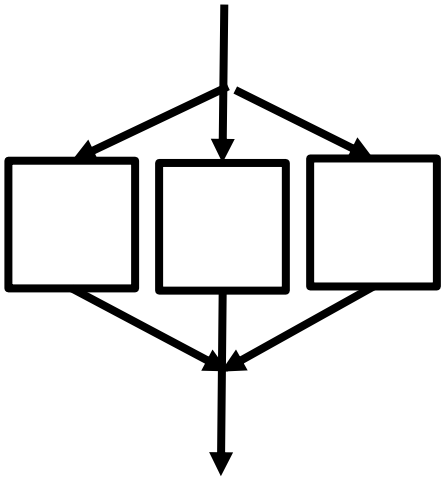


2種類の並列化

- **パイプライン型の並列化**
  - 問題を直列に分解して流れ作業で高速化
    （例）ベルトコンベア、エスカレータ
  - スループットが向上
  - 処理が円滑に流れるような工夫が必要

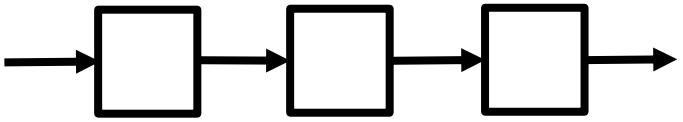


- **マルチプロセッシング型の並列化：**

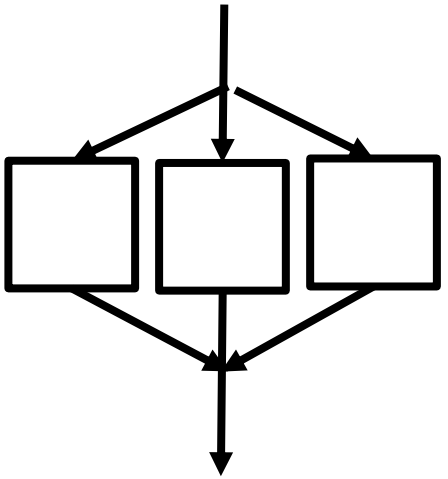


2種類の並列化

- **パイプライン型の並列化**
  - 問題を直列に分解して流れ作業で高速化
    （例）ベルトコンベア、エスカレータ
  - スループットが向上
  - 処理が円滑に流れるような工夫が必要



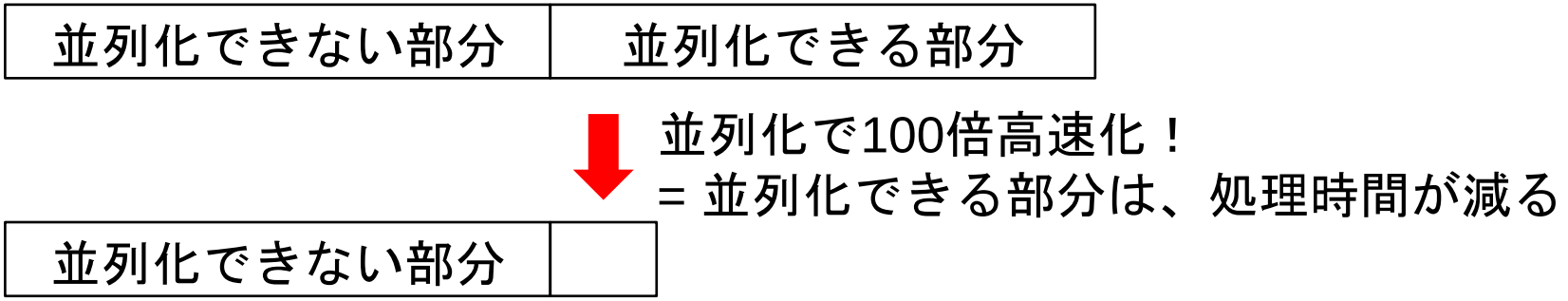
- **マルチプロセッシング型の並列化：**
  - 問題を並列に分割して計算して最後にまとめる
    （例）複数台並んだエレベータ
  - プロセッサ1台に比べて、スループットが向上
    レイテンシも向上する場合がある
  - 問題の分割のバランスを取る必要がある
  - 最後に計算結果をまとめるところも重要



**いずれの場合も待たされて遊んでいる
プロセッサが少なくなるように工夫すべき**

並列処理による高速化のチェックポイント

- **同時実行すべき処理が十分な個数あるか？**
  - N個に分割できたとして、高々N倍の高速化
- **どうしても並列化できない処理が全体の何%あるか？**
  - 全体の半分の処理量しか並列化できないとしたら、
    並列化により100倍高速化しても全体として高々2倍にしかならない



全体の計算時間は、もとの半分以上かかる
= 全体での高速化は、2倍以下

並列処理による高速化のチェックポイント

- **同時実行すべき処理が十分な個数あるか？**
  - N個に分割できたとして、高々N倍の高速化
- **どうしても並列化できない処理が全体の何%あるか？**
  - 全体の半分の処理量しか並列化できないとしたら、
    並列化により100倍高速化しても全体として高々2倍にしかならない
- **並列化の粒度(granularity)は？**
  - 粒度とは、分割単位の大きさ
  - 細かすぎるとプロセッサ間の通信コストが増えやすくなる
  - 粗すぎると並列度が上がりにくくなる
- **入力と出力のデータアクセスは？**
  - 入力や出力のデータ量に比べて、演算量が圧倒的に大きい場合は、
    あまり気にする必要はないが、そうでなければ、
    同時に多数のプロセッサがデータにアクセスするための工夫が必要

3. 並列処理の分類

- SIMD と MIMD
- 粒度による分類

命令とデータの流れによる分類

M. Flynnが1966年に提唱したノイマン型計算機の分類

- **SISD (Single Instruction, Single Data stream)**
  - 単一の命令列を単一のデータに対して実行する計算機
    一般的な逐次処理計算機
- **SIMD (Single Instruction, Multiple Data stream)**
  - 単一の命令列を複数のデータに対して実行する計算機
    多数のデータに対して同じ処理を行う並列計算機
- **MISD (Multiple Instruction, Single Data stream)**
- **MIMD (Multiple Instruction, Multiple Data stream)**

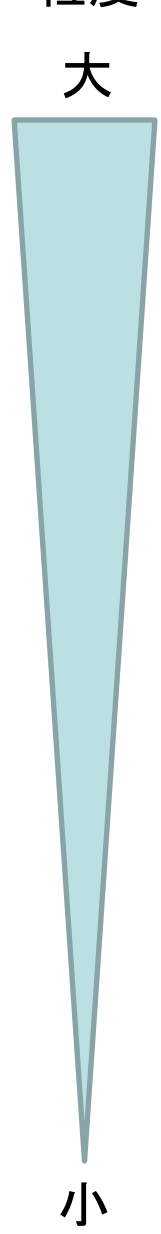
命令とデータの流れによる分類

M. Flynnが1966年に提唱したノイマン型計算機の分類

- **SISD (Single Instruction, Single Data stream)**
  - 単一の命令列を単一のデータに対して実行する計算機
    一般的な逐次処理計算機
- **SIMD (Single Instruction, Multiple Data stream)**
  - 単一の命令列を複数のデータに対して実行する計算機
    多数のデータに対して同じ処理を行う並列計算機
- **MISD (Multiple Instruction, Single Data stream)**
  - 複数の命令列を単一のデータに対して実行する計算機
    (あまり見かけないが、信頼性を高めるため、同じ計算を別の方法で計算して多数決を行う計算機システムなど)
- **MIMD (Multiple Instruction, Multiple Data stream)**
  - 複数の命令列を複数のデータに実行する計算機
    多数の異なるプログラムを同時に実行できる並列計算機

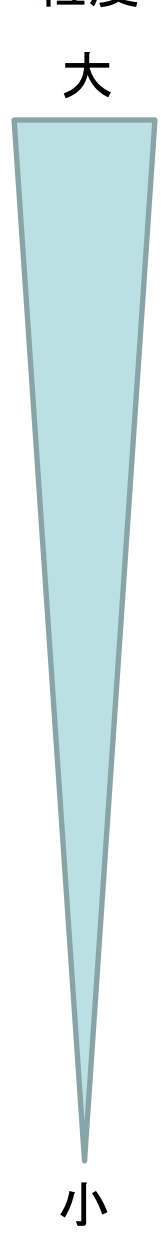
粒度による分類

- **プログラム並列**
  - プログラムごとの並列化
  - プロセスごとの並列化
    （プロセスとは、メモリ空間を割り当てられた実行単位のこと）
    （1つのプログラムが複数プロセスにより実行されることもある）
- **命令レベル並列**
  - プログラムを構成する命令を単位とする並列化
  - 命令の内部ステージ（命令フェッチ、主記憶とのデータ転送、各種演算など）を単位とする並列化
- **演算レベル並列**
- **ビットレベル並列**

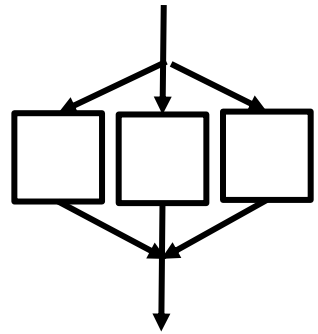


粒度による分類

- **プログラム並列**
  - プログラムごとの並列化
  - プロセスごとの並列化
    （プロセスとは、メモリ空間を割り当てられた実行単位のこと）
    （1つのプログラムが複数プロセスにより実行されることもある）
- **命令レベル並列**
  - プログラムを構成する命令を単位とする並列化
  - 命令の内部ステージ（命令フェッチ、主記憶とのデータ転送、各種演算など）を単位とする並列化
- **演算レベル並列**
  - 算術式を構成する演算を単位とする並列化
  - 1つの算術演算の内部ステージを単位とする並列化
- **ビットレベル並列**
  - 2進数のビット単位での並列化

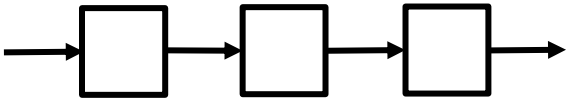


並列アーキテクチャの分類



- **マルチプロセッサ・マルチコア（MIMD型）**
  - プログラム単位のマルチプロセッシング
    （命令パイプライン、演算パイプラインも併用）
- **SIMD型プロセッサ**
  - 命令単位および演算単位のマルチプロセッシング

- **命令パイプライン**
  - 命令単位のパイプライン並列化



- **演算パイプライン**
  - 演算単位のパイプライン並列化およびマルチプロセッシング

- **FPGA, ASIC**
  - ハードウェアを用いたビットレベルの並列化
    （回路内のどこもかしこもが、並列に動く）

休 憩

- ここで、少し休憩しましょう。

- 深呼吸したり、肩の力を抜いてから、
  次のビデオに進んでください。

4. 並列アーキテクチャ (1)

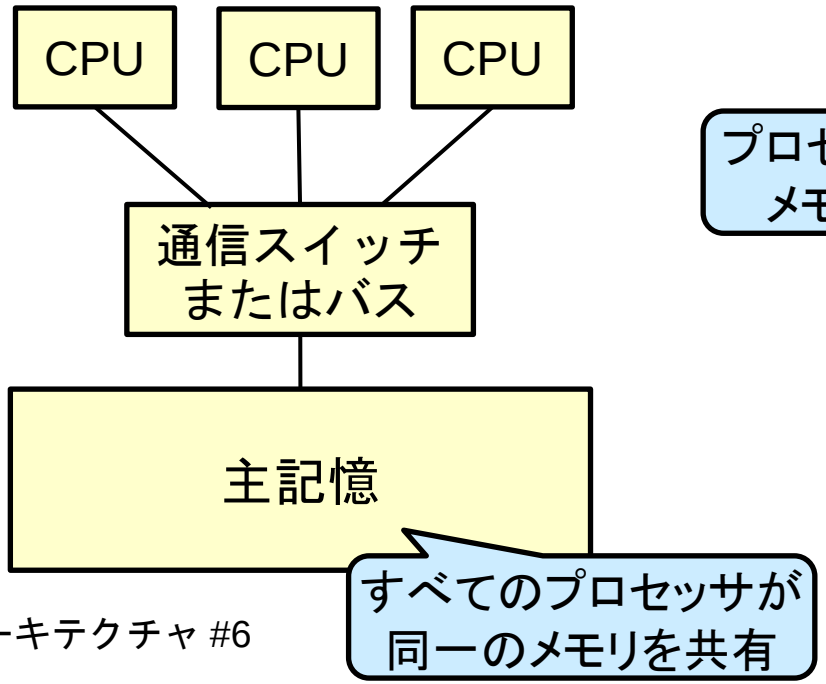
- マルチプロセッサ・マルチコア（MIMD型）
- SIMD型プロセッサ

マルチプロセッサ・マルチコア

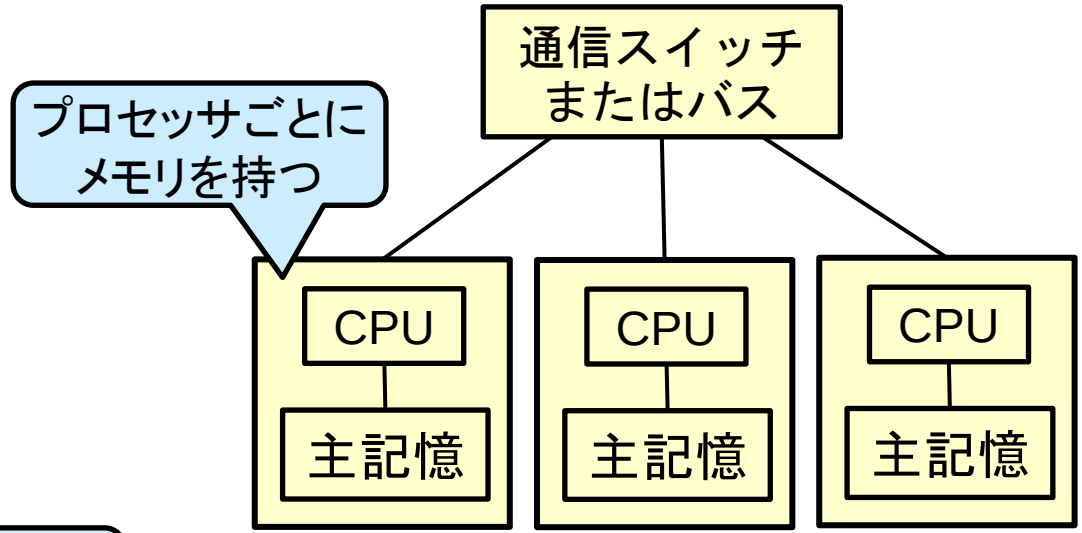
- **マルチプロセッサ(multi-processor)**：複数の独立した
  プロセッサが協調して動作する形式の並列マシン（MIMD型）

- プロセッサ同士の通信方法によって大きく2通りに分類できる

共有メモリ型

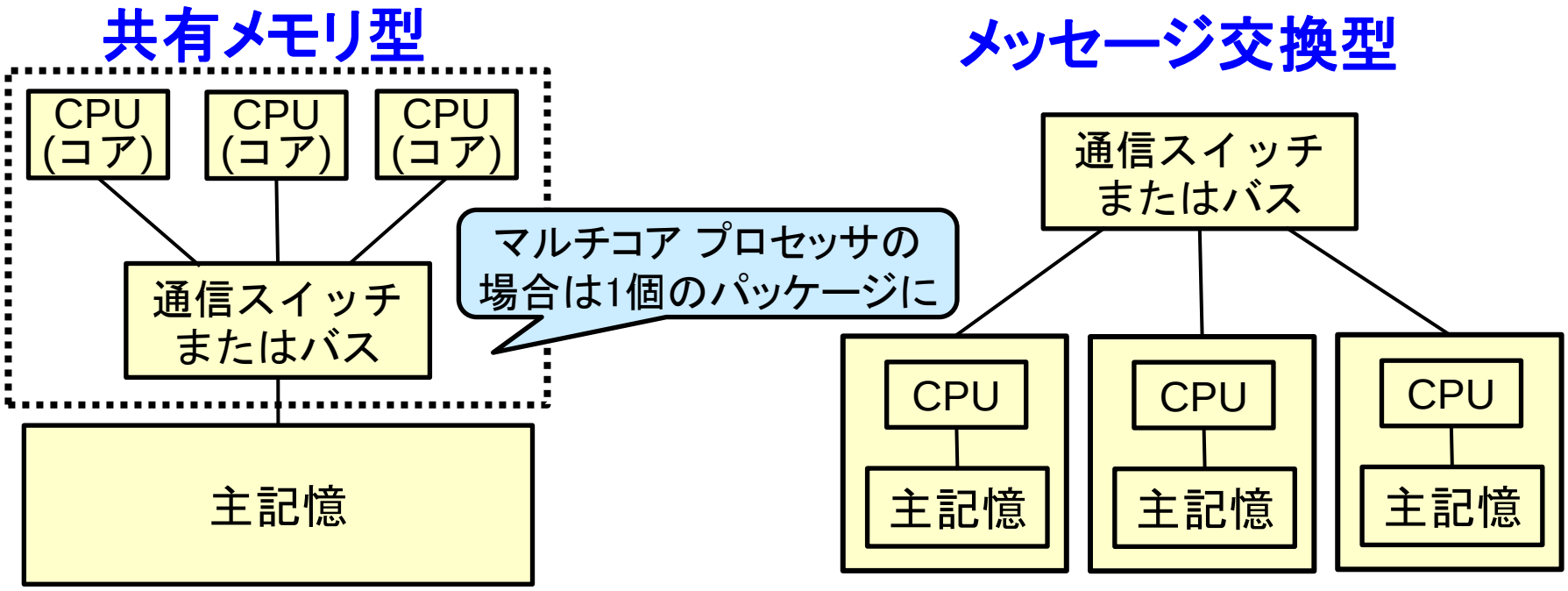


メッセージ交換型



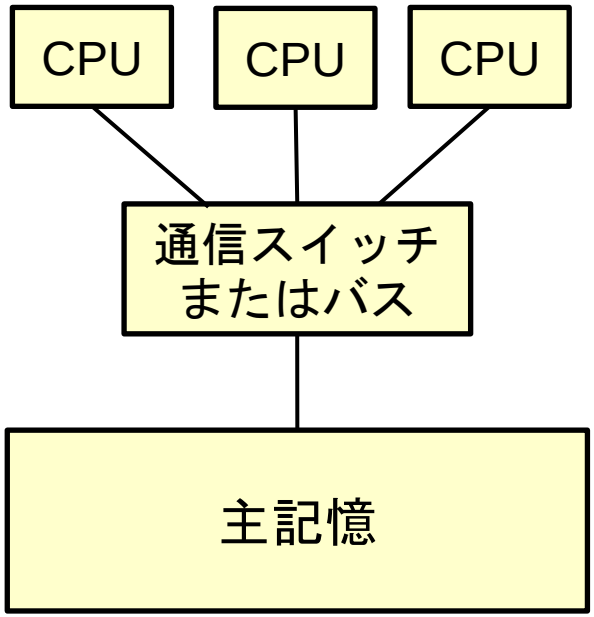
マルチプロセッサ・マルチコア

- **マルチプロセッサ(multi-processor)**: 複数の独立した
  プロセッサが協調して動作する形式の並列マシン(MIMD型)
- 1つのVLSIパッケージに詰め込まれたマルチプロセッサは
  マルチコア(multi-core)プロセッサと呼ばれる
- プロセッサ同士の通信方法によって大きく2通りに分類できる



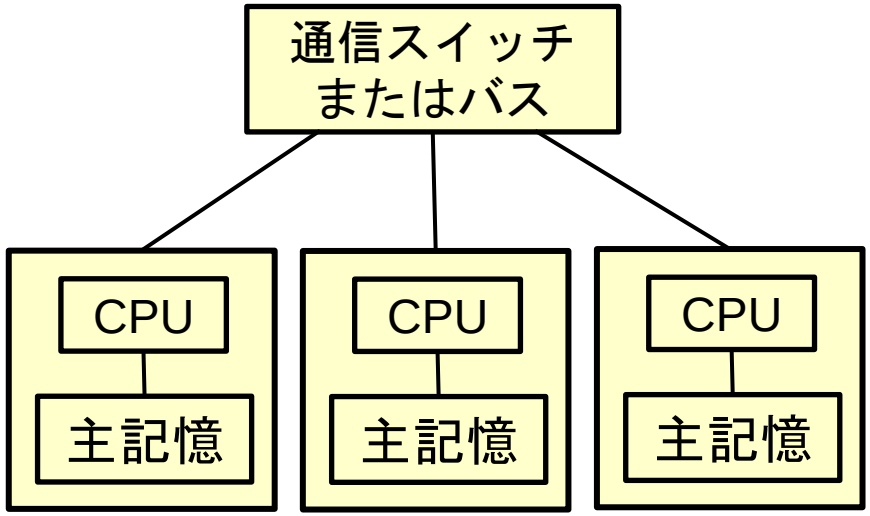
共有メモリ (Shared Memory)型

- プロセッサ(CPU)同士は、主記憶の読み書きを通して
  データを受け渡しする
  - 各プロセッサは共通のメモリアドレス空間を持つ
  - 命令デコーダやプログラムカウンタは別々に持つ(MIMD)
- 利点
  - 各プロセッサは主記憶と通信するだけ
    → 単一プロセッサ用プログラムを拡張して並列化しやすい
  - マルチコア構成も容易
- 欠点
  - 異なるプロセッサが同じ記憶場所に
    同時に書き込むことができない
  - プロセッサと主記憶の間の通信が集中すると、
    並列度を上げるのが難しくなる



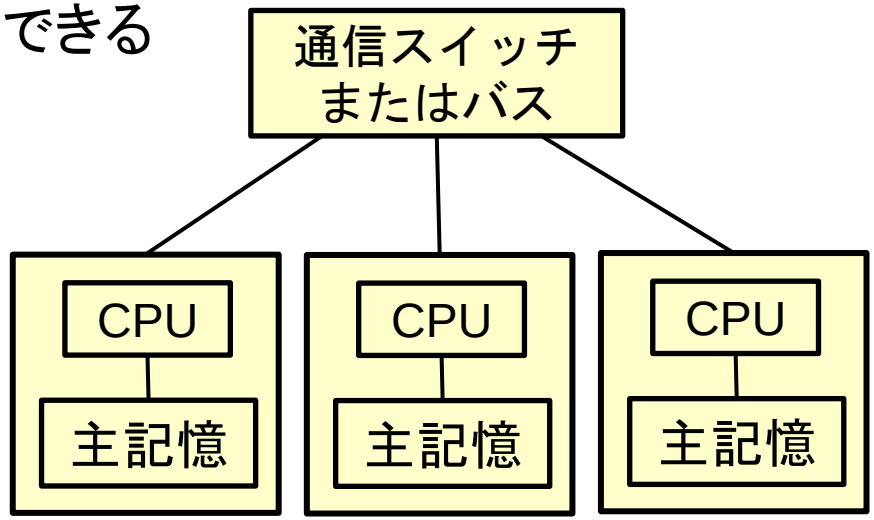
メッセージ交換(Message Passing)型

- 各プロセッサは、ローカルな主記憶を備え、プログラムやデータを個別に保持して独立に動作する
- プロセッサ同士で直接メッセージを通信する
  - プロセッサから見ると、LANやインターネットと同様な形式でパケットを送受信する（ただし専用設計された通信網なので非常に高速）
- プロセッサ間のネットワーク構造も種々工夫されている
  - 単一/多重バス・リング・フルメッシュ・ツリー・格子・トーラス・キューブ



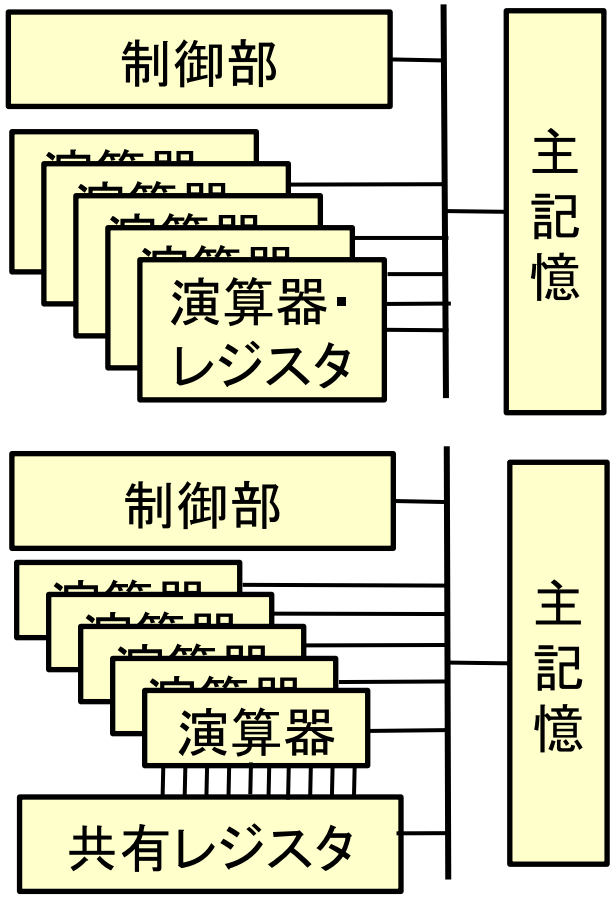
メッセージ交換(Message Passing)型

- 各プロセッサは、ローカルな主記憶を備え、
  プログラムやデータを個別に保持して独立に動作する
- プロセッサ同士で直接メッセージを通信する
  - プロセッサから見ると、LANやインターネットと同様な形式でパケットを送受信する（ただし専用設計された通信網なので非常に高速）
- プロセッサ間のネットワーク構造も種々工夫されている
  - 単一/多重バス・リング・フルメッシュ・ツリー・格子・トーラス・キューブ
- （利点）：専用設計されたネットワークを工夫することで
  プロセッサ数が多くても互いに高速通信できる
- （欠点）：明示的に通信を行うので、
  単一プロセッサとは異なる特有の
  並列プログラミングが必要に
- 最近のスパコンの多くは
  このタイプに属する（京、富嶽など）
  https://www.r-ccs.riken.jp/jp/fugaku/



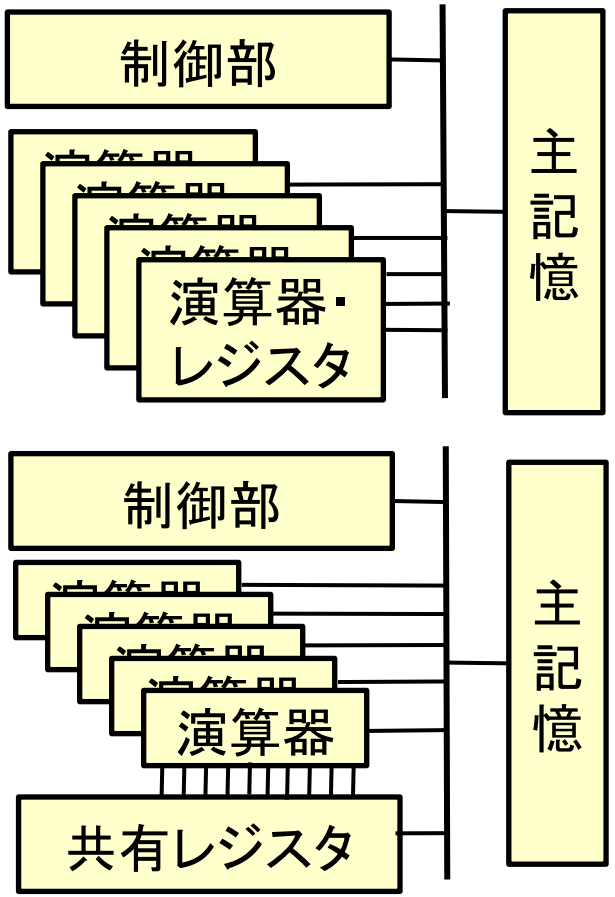
SIMDプロセッサ

- 命令デコーダなどの制御部を1か所に共通化し、同じ処理を実行する多数のプロセッサユニットを並べたもの
- どこまで共通化するかで、様々なバリエーションがある



SIMDプロセッサ

- 命令デコーダなどの制御部を1か所に共通化し、同じ処理を実行する多数のプロセッサユニットを並べたもの
- どこまで共通化するかで、様々なバリエーションがある
  - 各プロセッサが制御部以外のレジスタ・ALUなどを個別に持つもの（共有メモリ型と個別メモリ型の両方ありうる）
  - 制御部だけでなく多数のレジスタを並べて共有化し、各ALUが任意のレジスタに同時にアクセスできるようにしたもの
  - GPU（Graphic Processing Unit)もSIMDプロセッサの一種

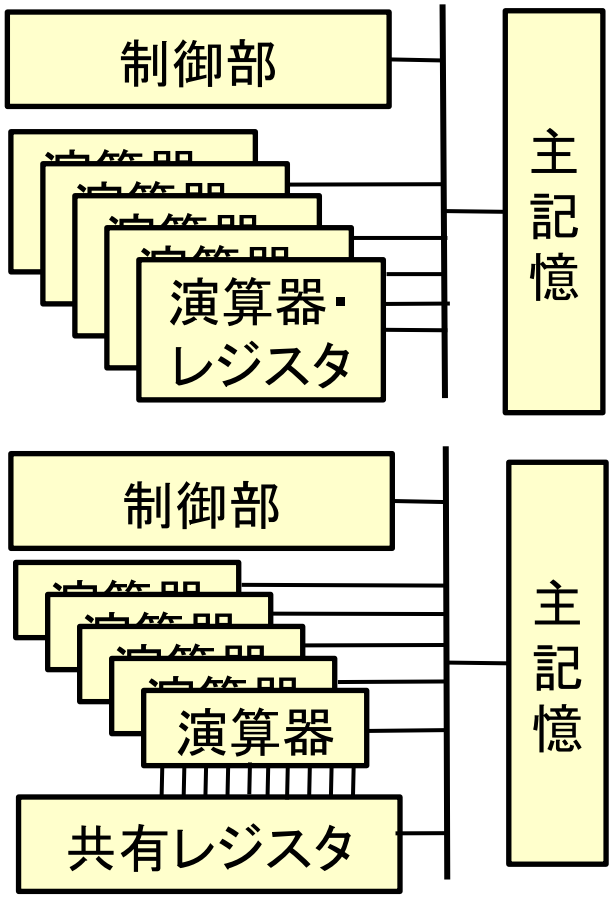


SIMDプロセッサ

- 命令デコーダなどの制御部を1か所に共通化し、
  同じ処理を実行する多数のプロセッサユニットを並べたもの
  – バリエーションがある

脱線： GPU

- もともとは、3次元グラフィックスの描画など画像処理のためのプロセッサ
- シンプルな演算器を多数搭載している
- GPGPU: 並列数の大きいプロセッサとして、一般的な並列計算にも利用されるように
  (最近だと、深層学習ベースのAI など)
- 知ってほしい事： 使い方は、皆さん次第
  アーキテクチャは産業とリンクして発展



– GPU(Graphic Processing Unit)も
  SIMDプロセッサの一種

休 憩

- ここで、少し休憩しましょう。

- 深呼吸したり、肩の力を抜いてから、
  次のビデオに進んでください。

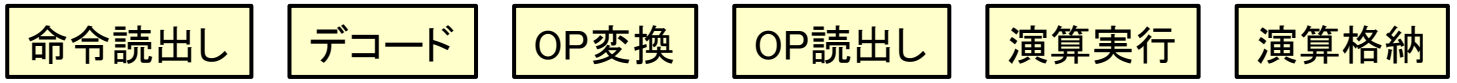
4. 並列アーキテクチャ (2)

- 命令パイプライン
- 演算パイプライン

命令パイプライン

第２回資料

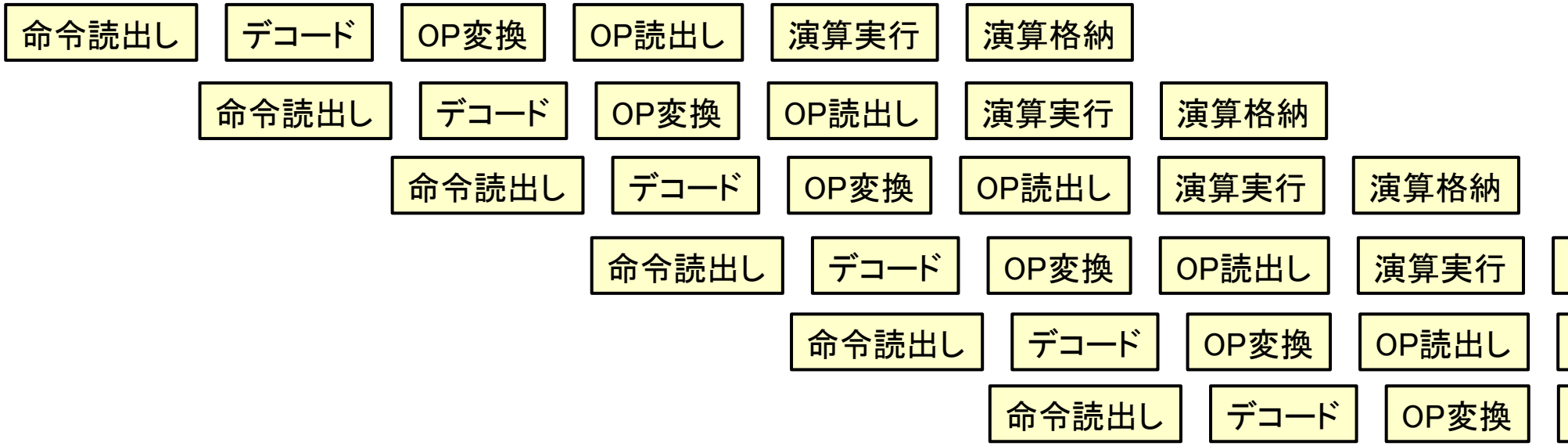
- 機械語１命令は、いくつかのステップに分けられる
  - （例）命令読み出し、デコード、オペランドアドレス変換、オペランド読み出し、演算実行、演算格納



命令パイプライン

第２回資料

- 機械語１命令は、いくつかのステップに分けられる
  - （例）命令読み出し、デコード、オペランドアドレス変換、オペランド読み出し、演算実行、演算格納
- 連続する機械語を、1ステップずつ流れ作業で処理することでスループットを向上できる
  - 各ステップはそれぞれ別々のハードウェアで処理する
  - 各ステップはほぼ同一の所要時間で動作するように設計
  - n ステップに分けてパイプライン化すれば n 倍の高速化



命令パイプラインの乱れと対策

- 命令パイプラインが乱れると処理速度が低下する
  - 先行命令の演算結果を後続の命令が使う場合

A = B ＋ C

D = A × 10　… Aの値が決まらないと、
次の命令が実行できない

命令パイプラインの乱れと対策

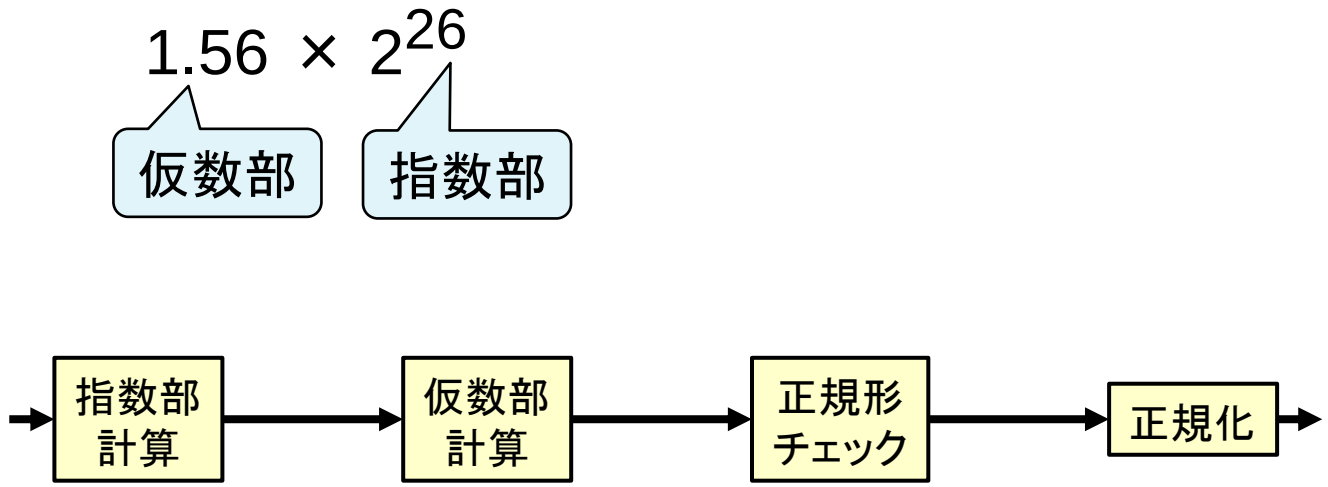
- 命令パイプラインが乱れると処理速度が低下する
  - 先行命令の演算結果を後続の命令が使う場合
  - 先行命令により後続命令のオペランドが設定される場合
  - 条件分岐や割り込みにより後続命令が変わる場合
  - 主記憶からの読み出しに時間がかかり待たされる場合
  - 連続する命令のメモリ読み出しのタイミングが重なる場合
  - 通常より時間がかかる演算命令が実行される場合

命令パイプラインの乱れと対策

- 命令パイプラインが乱れると処理速度が低下する
  - 先行命令の演算結果を後続の命令が使う場合
  - 先行命令により後続命令のオペランドが設定される場合
  - 条件分岐や割り込みにより後続命令が変わる場合
  - 主記憶からの読み出しに時間がかかり待たされる場合
  - 連続する命令のメモリ読み出しのタイミングが重なる場合
  - 通常より時間がかかる演算命令が実行される場合
- 命令パイプラインの乱れを抑える工夫
  - **ループバッファ**: 繰り返し1回分の命令をまとめてCPU内に取り込んで、主記憶アクセスを減らす
  - **多重命令ストリーム**: 条件分岐のYes/Noの2系統のパイプラインを用意し、両方とも先読みしておく
  - **分岐履歴テーブル**: 条件分岐のYes/Noの頻度に偏りがあるとき、頻度が高い方を先読みすることで平均速度を上げる

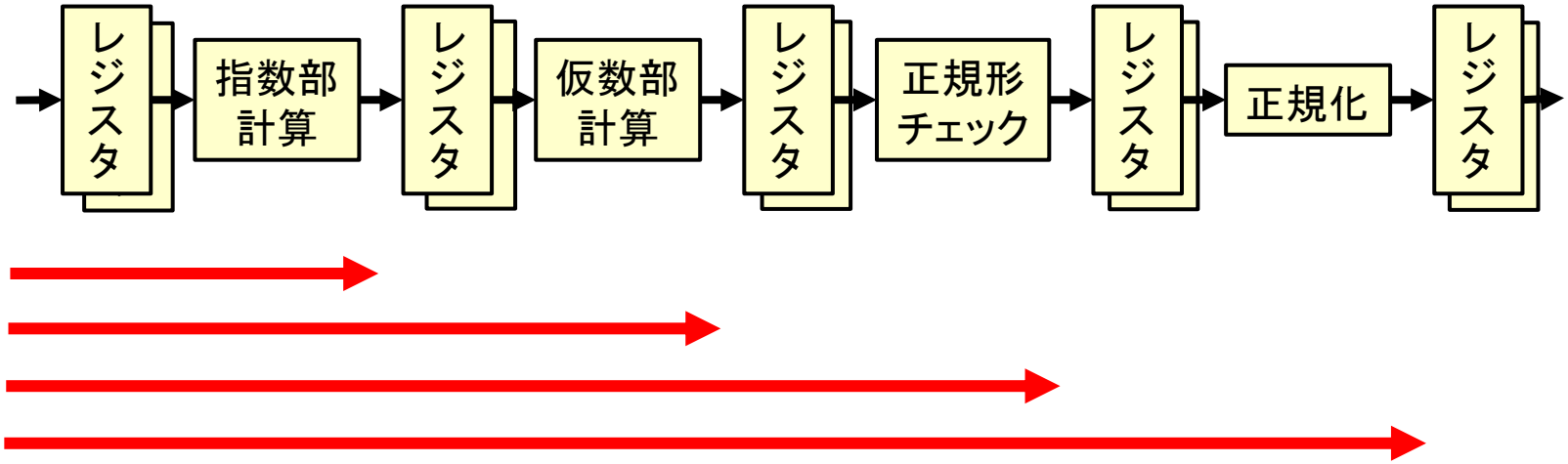
演算パイプライン

- 浮動小数点演算も、いくつかのステップに分けられる
  - （例）指数部計算、仮数部計算、正規形チェック、正規化処理



演算パイプライン

- 浮動小数点演算も、いくつかのステップに分けられる
  - （例）指数部計算、仮数部計算、正規形チェック、正規化処理
- 連続する演算を流れ作業で進めることで
  スループットを向上できる
  - 各ステップは別々のハードウェアで動作させる
  - 各ステップはほぼ同一の所要時間で動作するように設計



演算パイプラインとベクトルプロセッサ

- 演算パイプライン技法は、配列の各要素に同じ演算を順次実行するときに特に威力を発揮
  - SIMD並列化の一種
  - ベクトルや行列などを多用する科学技術計算に有効なので「ベクトルプロセッサ」とも呼ばれ、1980年代のスーパーコンピュータ開発競争で発展した

演算パイプラインとベクトルプロセッサ

- 演算パイプライン技法は、配列の各要素に同じ演算を順次実行するときに特に威力を発揮
  - SIMD並列化の一種
  - ベクトルや行列などを多用する科学技術計算に有効なので「ベクトルプロセッサ」とも呼ばれ、1980年代のスーパーコンピュータ開発競争で発展した
- スーパーコンピュータの典型的な構成
  - 複数のベクトルプロセッサ（高速演算器＋ベクトルレジスタ）
    ベクトルレジスタ：数百ワードのレジスタ群（主記憶に比べ相当高速）
  - スカラ部（逐次実行処理）
  - 主記憶制御部（階層的メモリ構成・キャッシュ）

5. ノイマン型計算機の限界

- ノイマン型計算機の特徴
  - プログラム内蔵方式
  - １次元に並んだメモリ空間
  - メモリ(主記憶)上のプログラムをCPUで逐次実行
  - メモリとCPUは、転送路を通して接続

ノイマン型計算機のボトルネック（フォンノイマン・ボトルネック）
命令もデータもすべてここを通るので、ここの性能がボトルネックになる

5. ノイマン型計算機の限界

- ノイマン型計算機の特徴
  - プログラム内蔵方式
  - １次元に並んだメモリ空間
  - メモリ(主記憶)上のプログラムをCPUで逐次実行
  - メモリとCPUは、転送路を通して接続

ノイマン型計算機のボトルネック（フォンノイマン・ボトルネック）
命令もデータもすべてここを通るので、ここの性能がボトルネックになる

- フォンノイマン・ボトルネックを改善する工夫の例:
  - 転送路（バス）を太く速くする（16→32→64ビット）
  - 命令の逐次読出し実行をやめてプログラムの記述方法を変える
  - メモリ(主記憶)のデータを読み出したら、
    プロセッサ内でできるだけ加工する（CPU内のレジスタは高速）

今回のまとめ

並列処理アーキテクチャ

1. 計算機の処理能力の2つの要素
   - スループット、レイテンシ
2. 並列処理
   - 並行処理と並列処理、パイプライン並列とマルチプロセッシング並列
3. 並列処理の分類
   - SIMD と MIMD、粒度による分類
4. 並列アーキテクチャ
   - マルチプロセッサ・マルチコア(MIMD型)、SIMDプロセッサ
   - 命令パイプライン、演算パイプライン
5. ノイマン型計算機の限界
   - フォンノイマン・ボトルネック