

---

東北大学 工学部 機械知能・航空工学科  
2015年度 5セメスター・クラスD

# 計算機工学

## 14. さらに勉強するために

大学院情報科学研究科  
鏡 慎吾

<http://www.ic.is.tohoku.ac.jp/~swk/lecture/>

# これまで何を学んだか

---

- 0と1の世界
  - 2進数, 算術演算, 論理演算
- 計算機はどのように動くのか
  - プロセッサとメモリ
  - 演算命令, ロード・ストア命令, 分岐命令
- 計算機はどのように構成されているのか
  - 組合せ回路  $\doteq$  論理関数
    - 論理式の標準形, 論理式の簡単化
  - 順序回路  $\doteq$  有限状態機械
  - メインメモリ, キャッシュメモリ

# 話し切れなかったこと

---

- 0と1の世界 → 実数は?
  - 2進数, 算術演算, 論理演算
- 計算機はどのように動くのか → プロセッサとメモリ以外は?
  - プロセッサとメモリ
  - 演算命令, ロード・ストア命令, 分岐命令
- 計算機はどのように構成されているのか → 高速化技術?
  - 組合せ回路 ≡ 論理関数
    - 論理式の標準形, 論理式の簡単化
  - 順序回路 ≡ 有限状態機械
  - メインメモリ, キャッシュメモリ

# Q. 実数はどのように表現するか？

---

## 浮動小数点数を用いる（教科書1.3節）

考え方の基礎は、いわゆる科学的記数法（指数表記）

- $6.02 \times 10^{23}$

- $1.602 \times 10^{-19}$

仮数部    指数部

仮数部は1 以上10 未満にする（正規化）

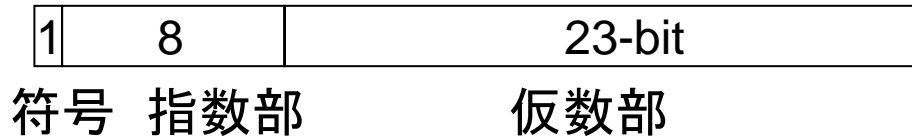
仮数部の桁数がいわゆる有効数字

これの2進数版が浮動小数点数

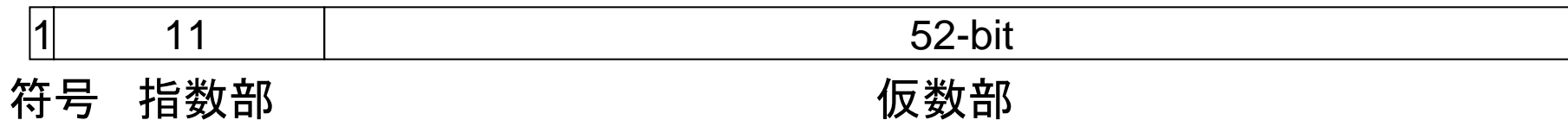
- 仮数部も指数も有限ビットの2進数で表す
- 指数部の底は 10 ではなく 2
- 仮数部は 1 以上 2 未満にする（正規化）

# IEEE 754 浮動小数点数

## 単精度 (C言語の float)



## 倍精度 (C言語の double)



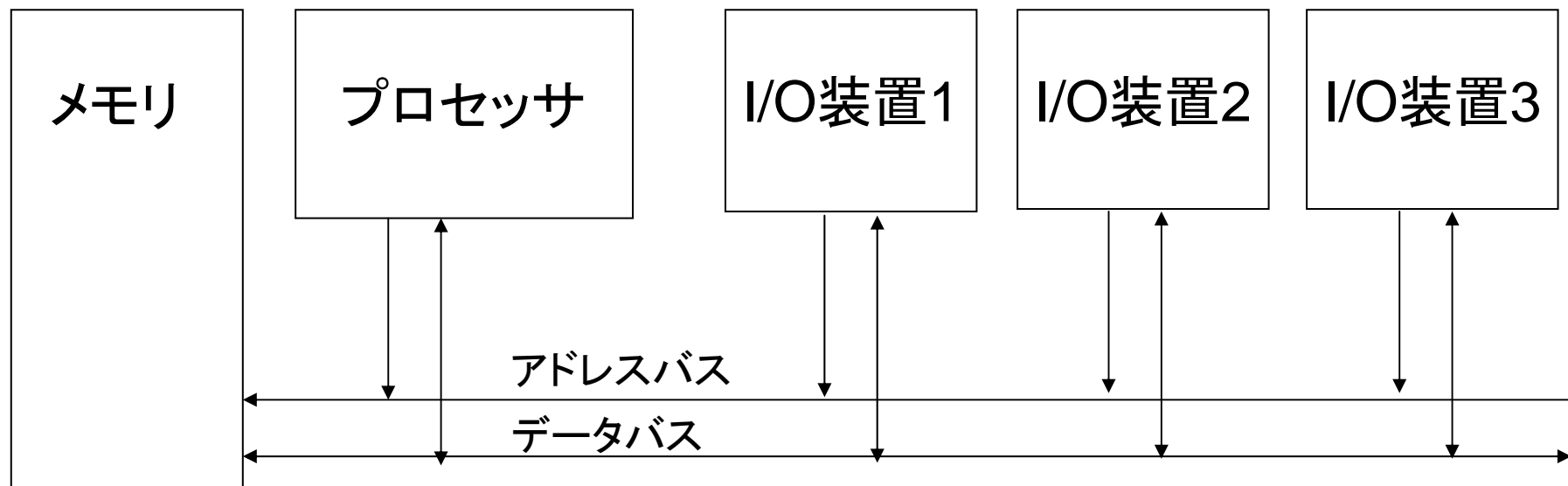
$$\textcircled{-} 1.\boxed{01101101} \times 2^{\boxed{1100110}}$$

符号部 仮数部 指数部

符号:  
0: 正  
1: 負

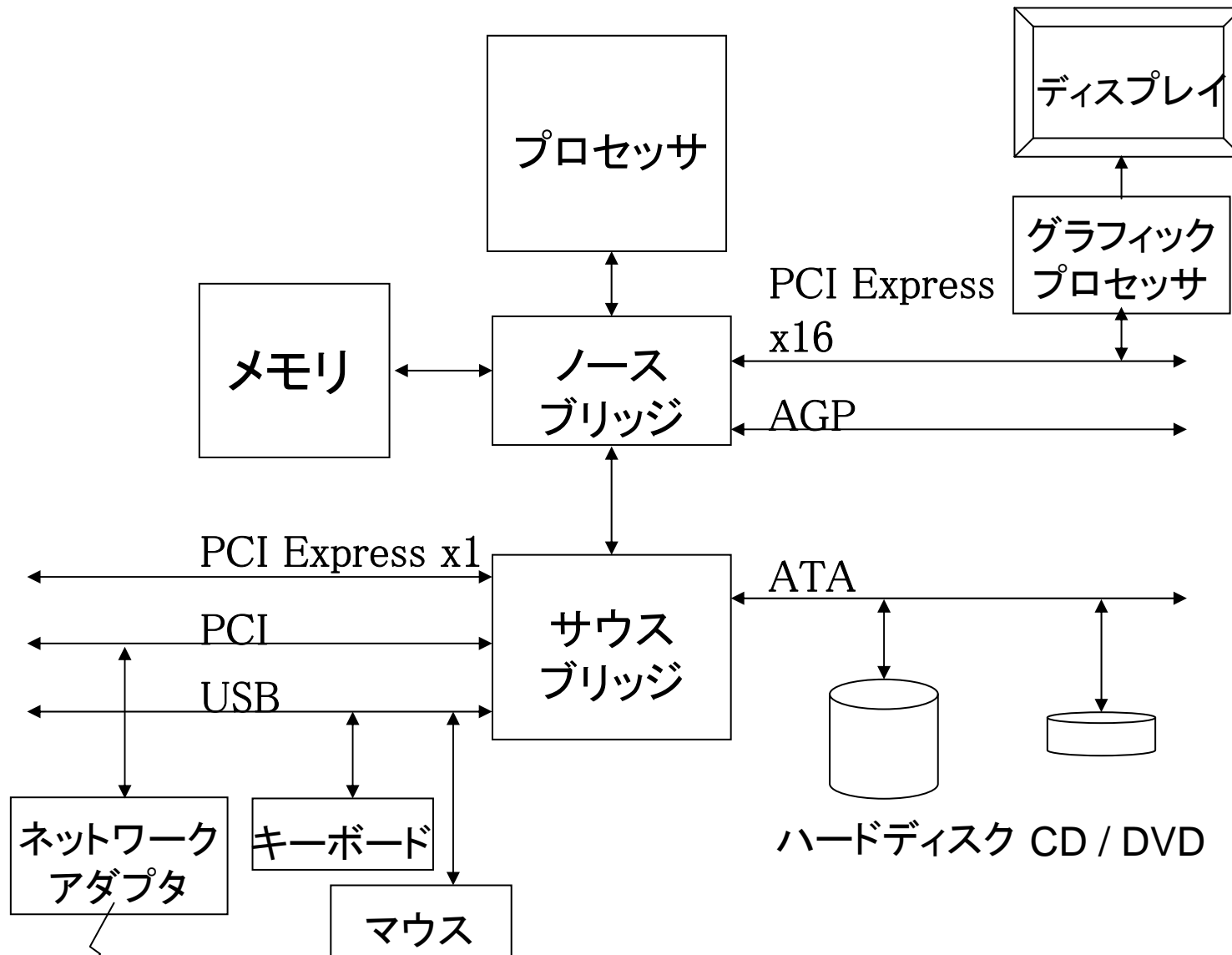
## Q. プロセッサとメモリ以外の機器は？

ディスプレイもキーボードもハードディスクも、**入出力装置**  
(Input/Output device, I/O)として扱われる(教科書付録C章)



- 装置ごとに割り当てられたアドレス範囲にアクセスすることで特定の I/O 装置と通信できる (memory mapped I/O 方式)
- I/O 専用の命令とアドレス空間が用意されているアーキテクチャもある (I/O 専用命令方式)

# PC用の入出力バスとの例



# Q. なぜ複数のプログラムが同時に走るのか

---

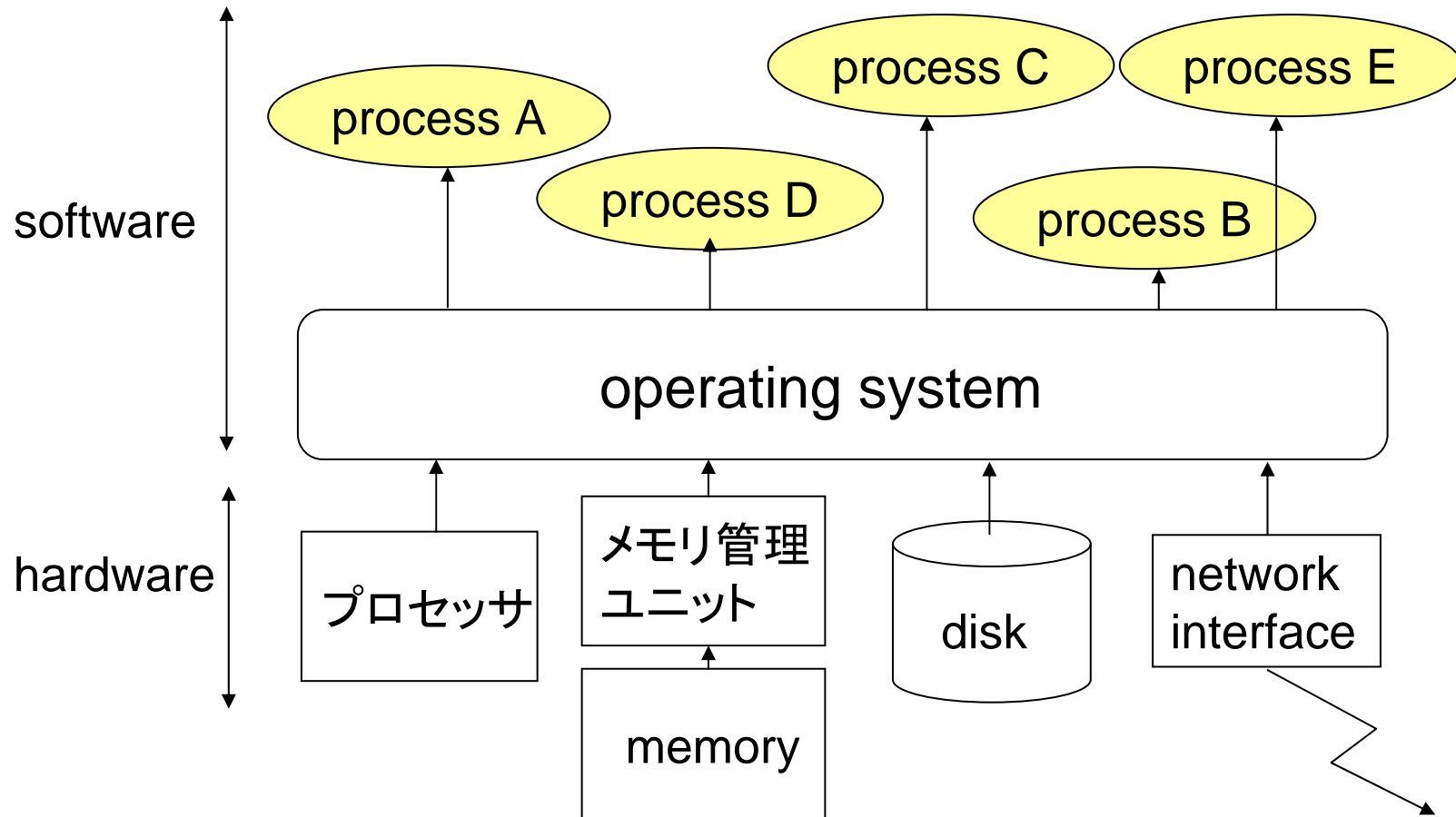
オペレーティングシステム (OS) と呼ばれるソフトウェアが、複数のプログラムの時分割切り替えを行っている (教科書付録D章)  
具体例: Windows, MacOS, Linux など

OSの役割:

- ハードウェアの詳細を隠蔽して、**抽象化されたマシンをプログラムに提供する**  
例: A社のハードディスク, B社のハードディスク, C社のUSBメモリ  
→ 「ファイル」という概念で統一的に操作できる
- 複数のプログラム, 複数のユーザの間で、**必要な資源(ハードウェア)を適切に管理する**  
例: 同時にディスクを読み書きしても大丈夫  
Word が不正なアドレスを読み書きしても, Excel には影響がない

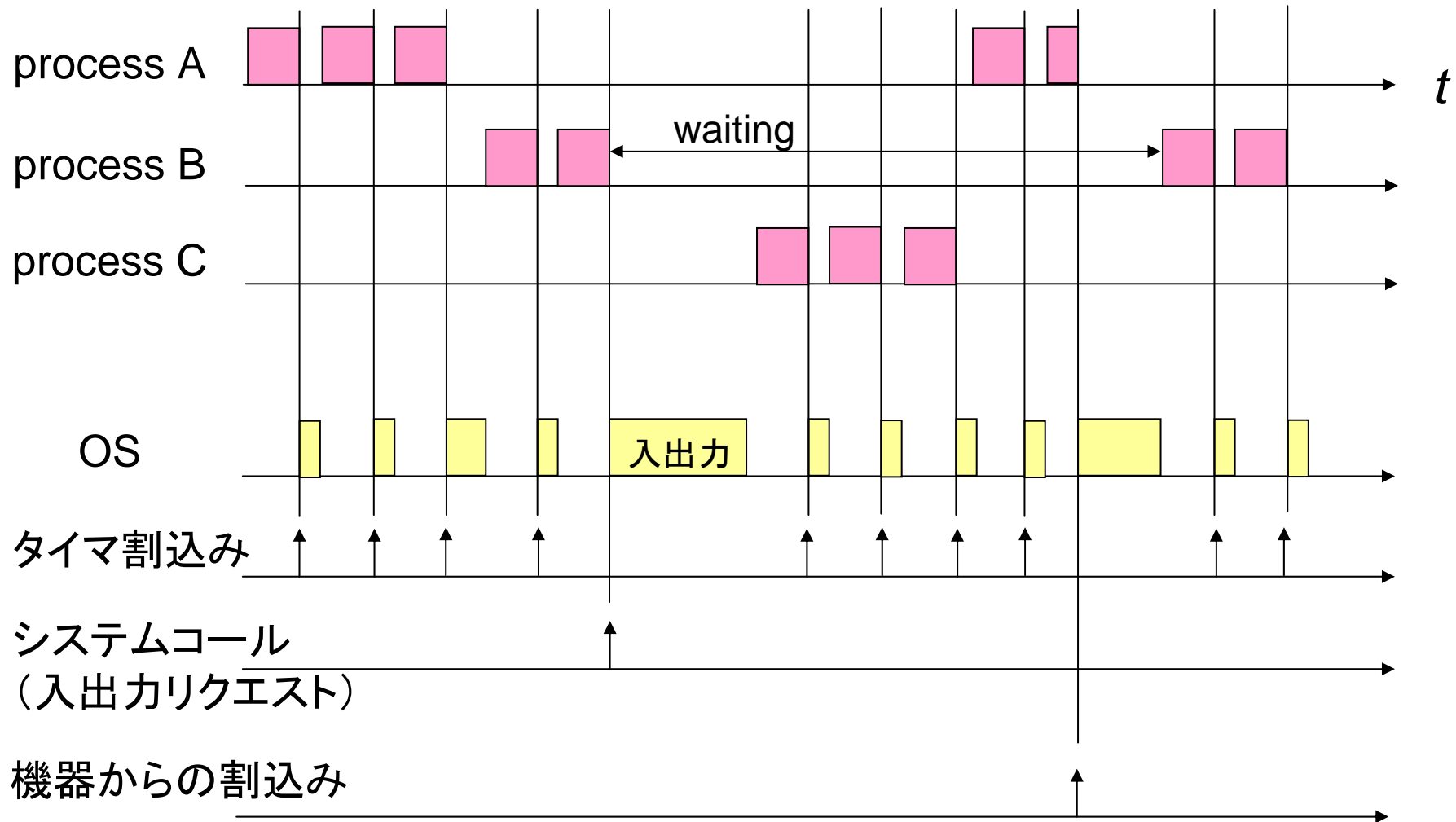


# OS の概念



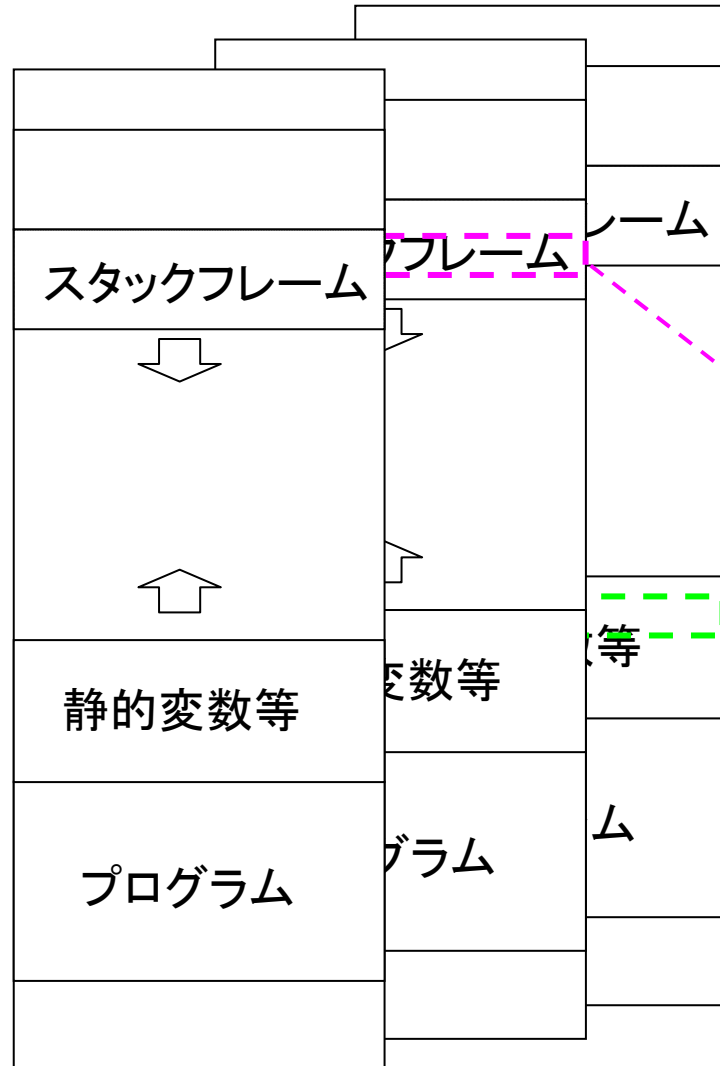
複数のプロセスにハードウェア資源を(多くの場合**時分割**で)割り当てるソフトウェア

# プロセッサ時間の割り当て

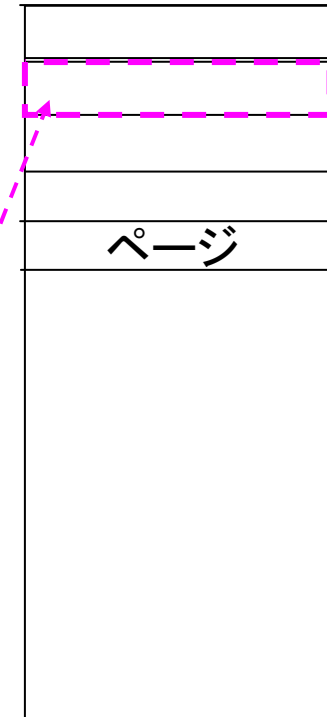


# メモリの割り当て(仮想記憶)

プロセスA プロセスB プロセスC

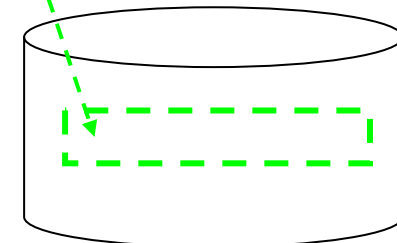


物理メモリ



メモリ管理ユニット  
(ハードウェア)  
+  
OS

ハードディスク



# Q. コンピュータはどのように高速化する?

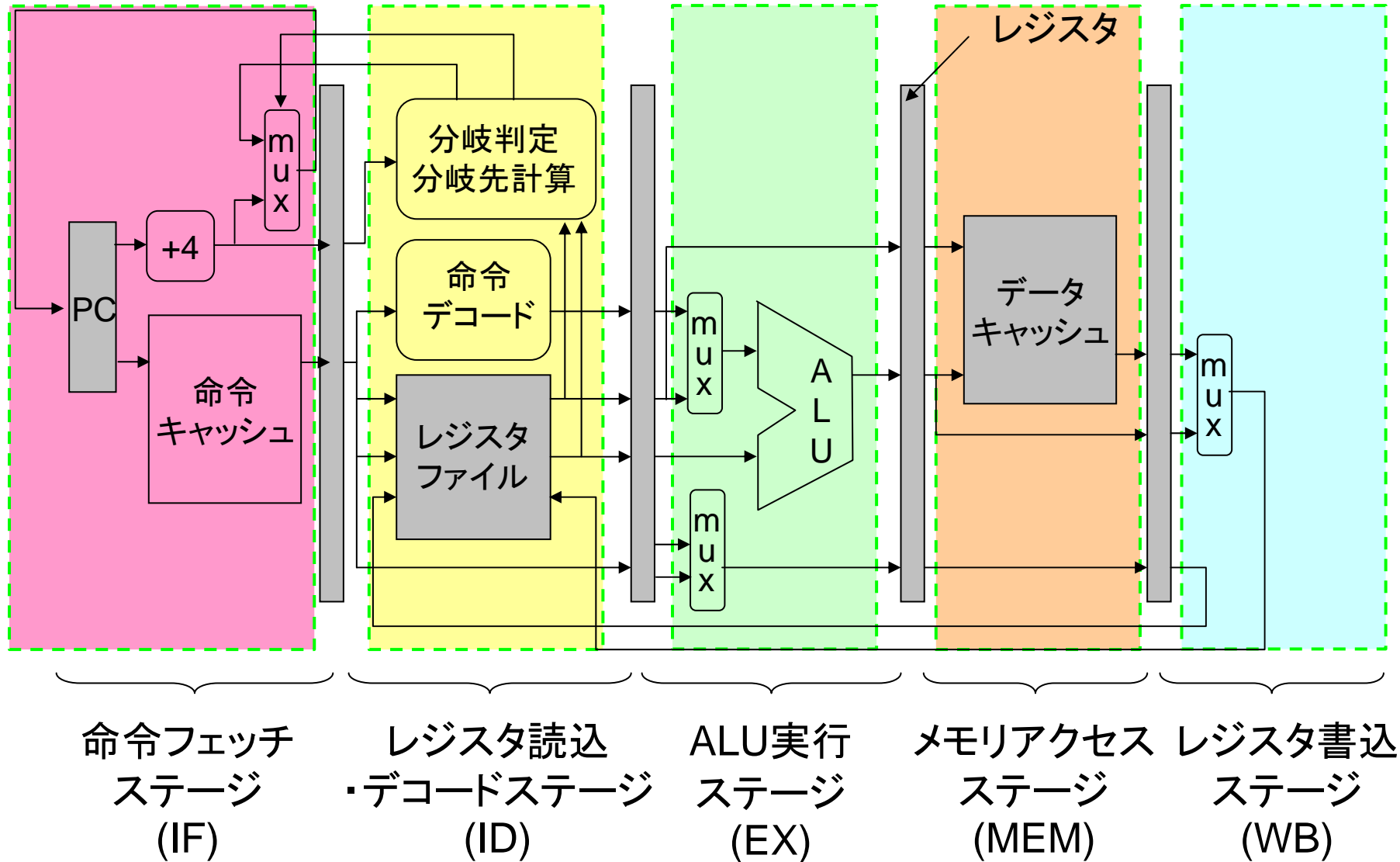
---

半導体素子の微細化により, トランジスタ動作速度の向上と, 大量の回路の利用が可能となってきた(教科書付録F章)

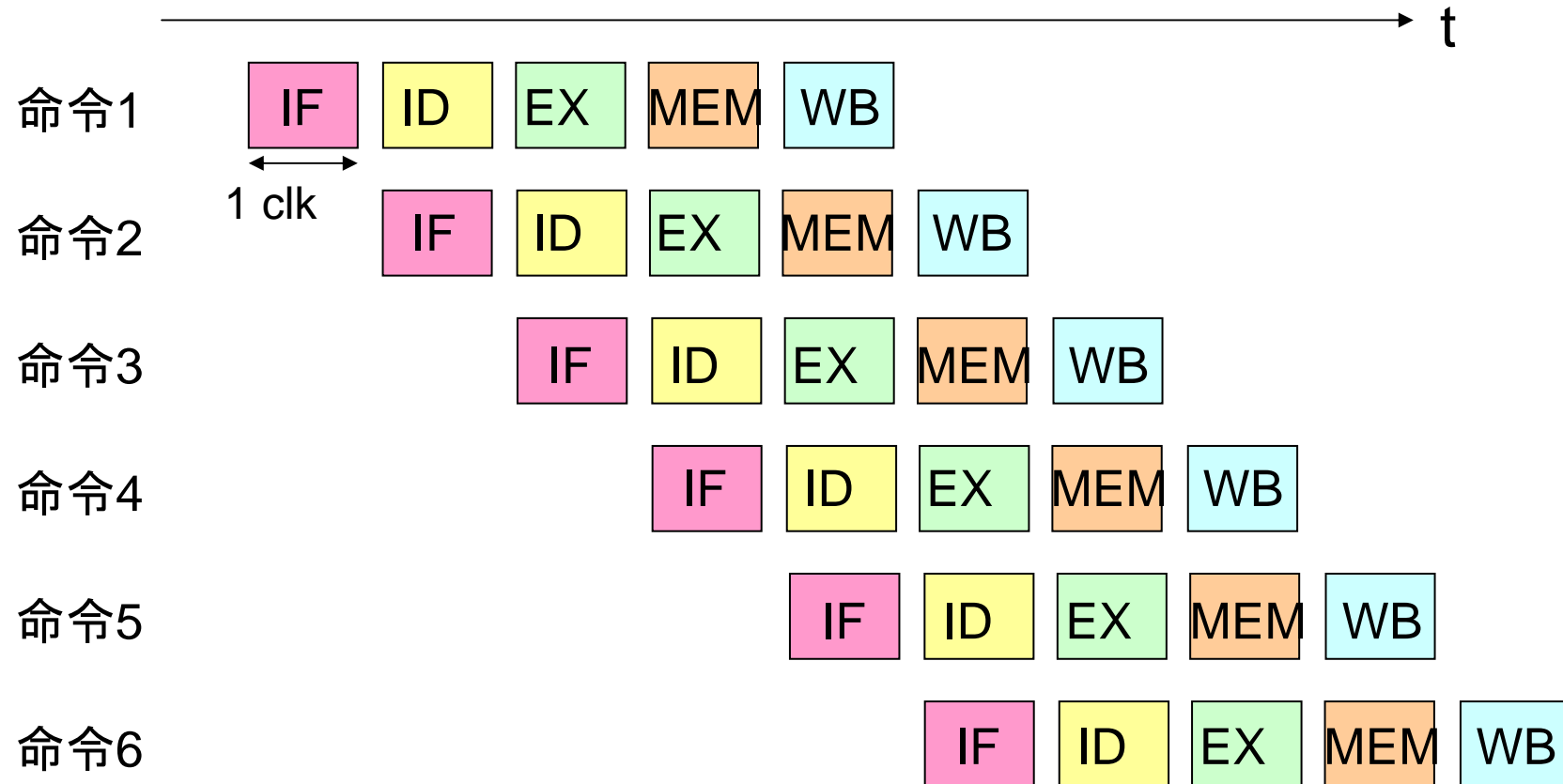
動作速度向上 → **クロックサイクル時間の短縮**  
大量の回路 → **並列処理**

クロック周波数の向上は既に頭打ちになっており, 並列処理の重要性が高まっている

# MIPSのパイプライン化

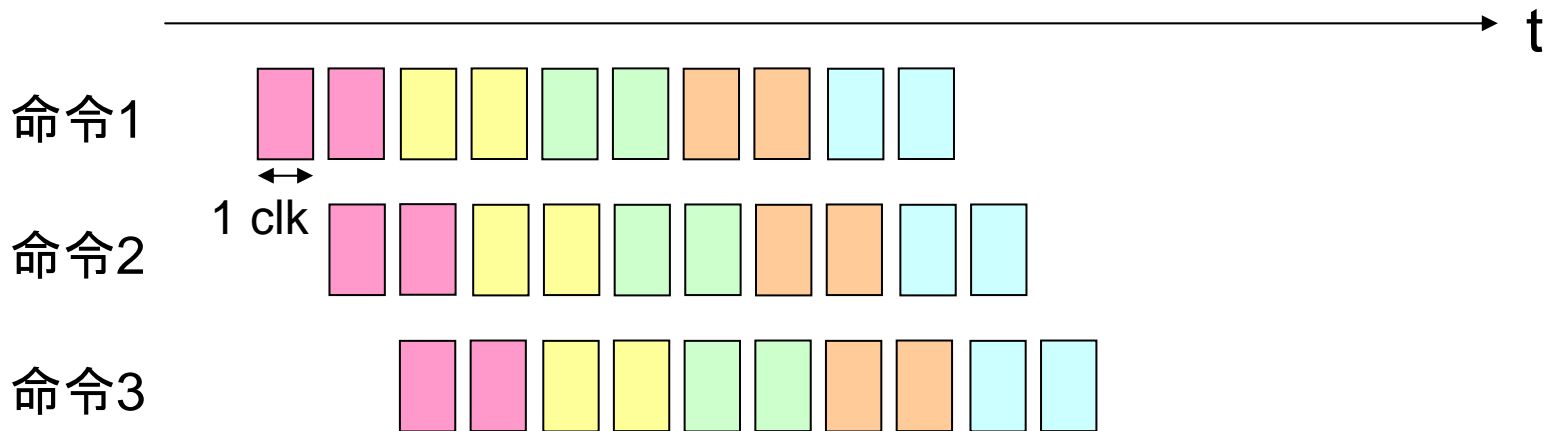
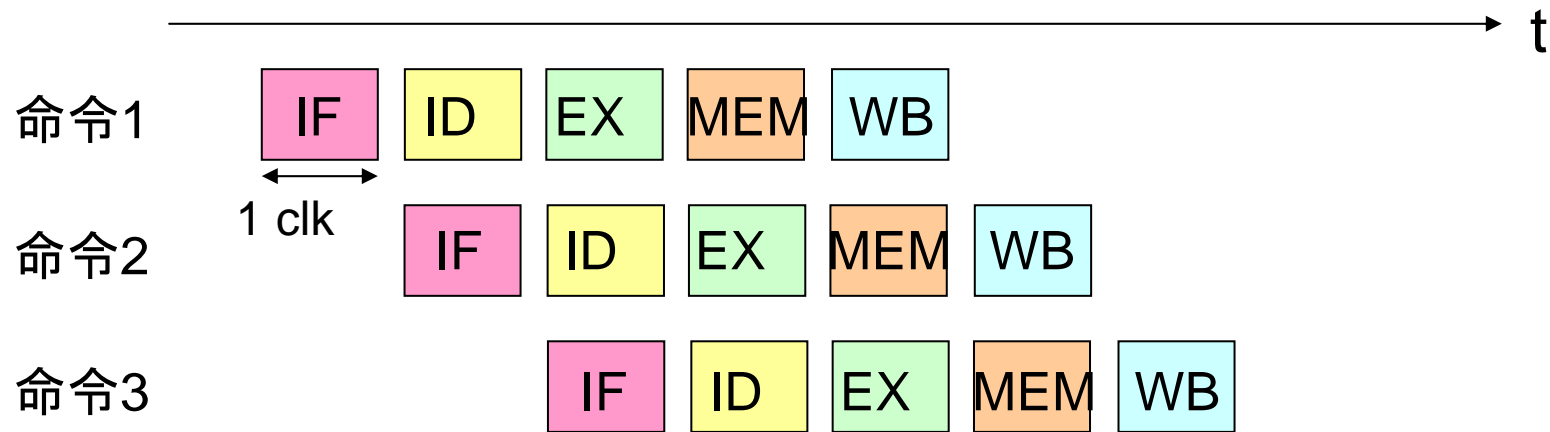


# 命令パイプライン



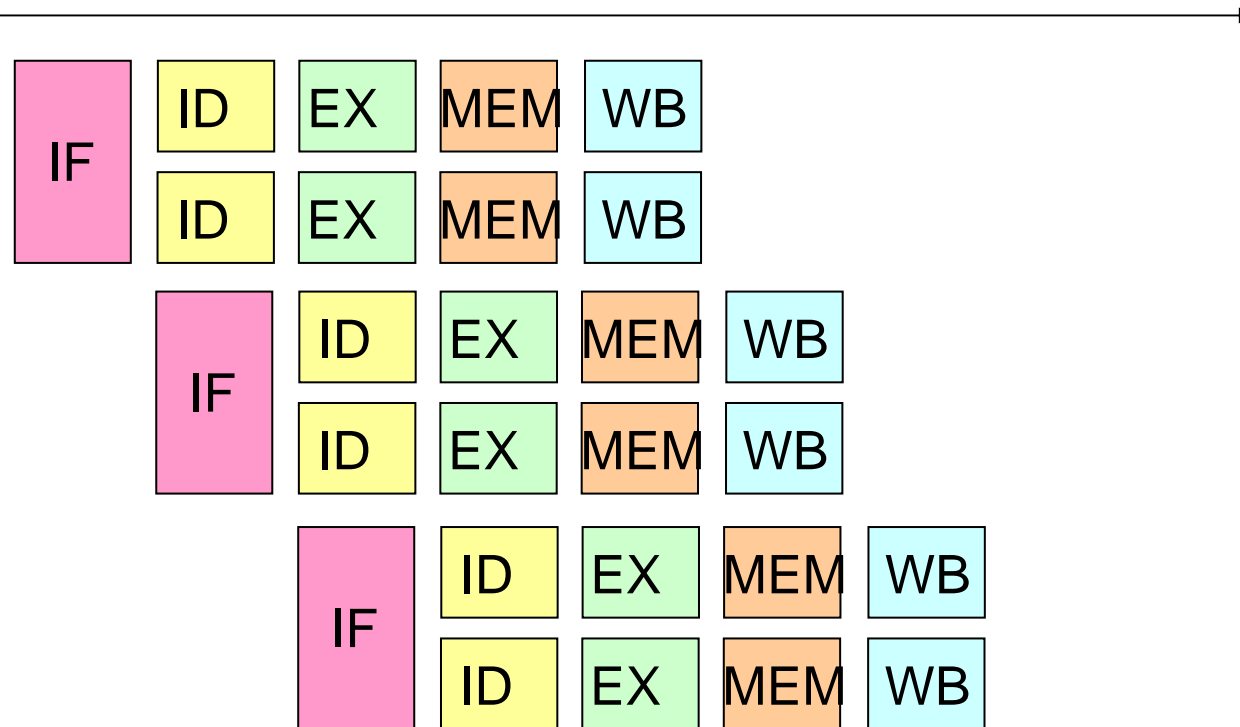
これも一種の並列処理(複数のステージが同時に動いている)

# パイプラインのさらなる細分化



e.g. Pentium 4 は 20段

# 命令レベル並列性

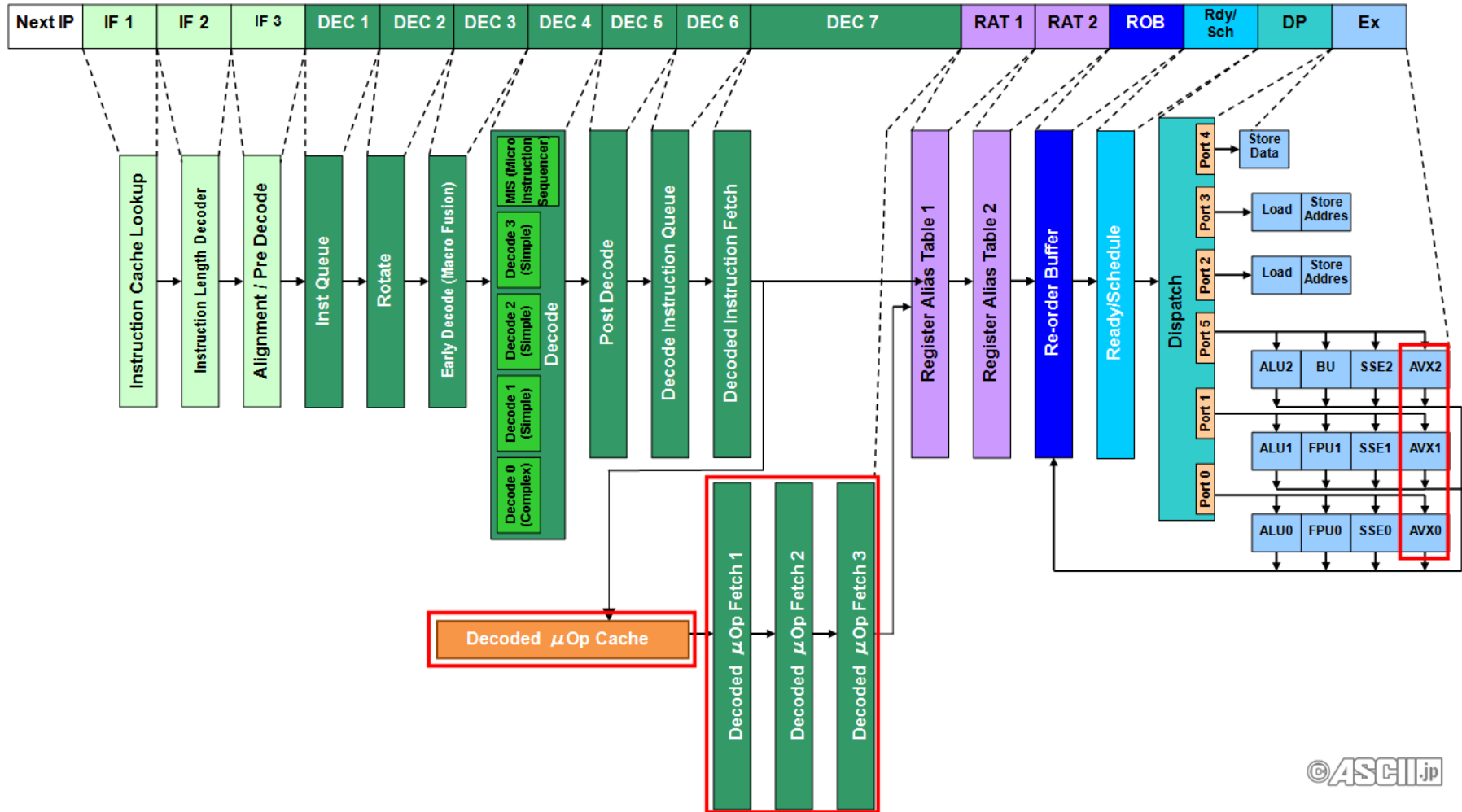


## スーパースカラ:

- 演算器を多重化し, 複数命令を同時実行
- 同時実行可能かどうかはハードウェアが動的に判定



# 例: Core i7 Sandy Bridge



<http://ascii.jp/elem/000/000/724/724498/>

# データ並列性

---

画像処理, 音声処理, ある種の科学技術計算など  
→ 同じ演算を多数のデータに適用することが多い  
(**SIMD**; Single Instruction stream Multiple Data stream)

SIMD型並列処理の実現形態:

**空間並列**: 同じ演算器を多数並べる

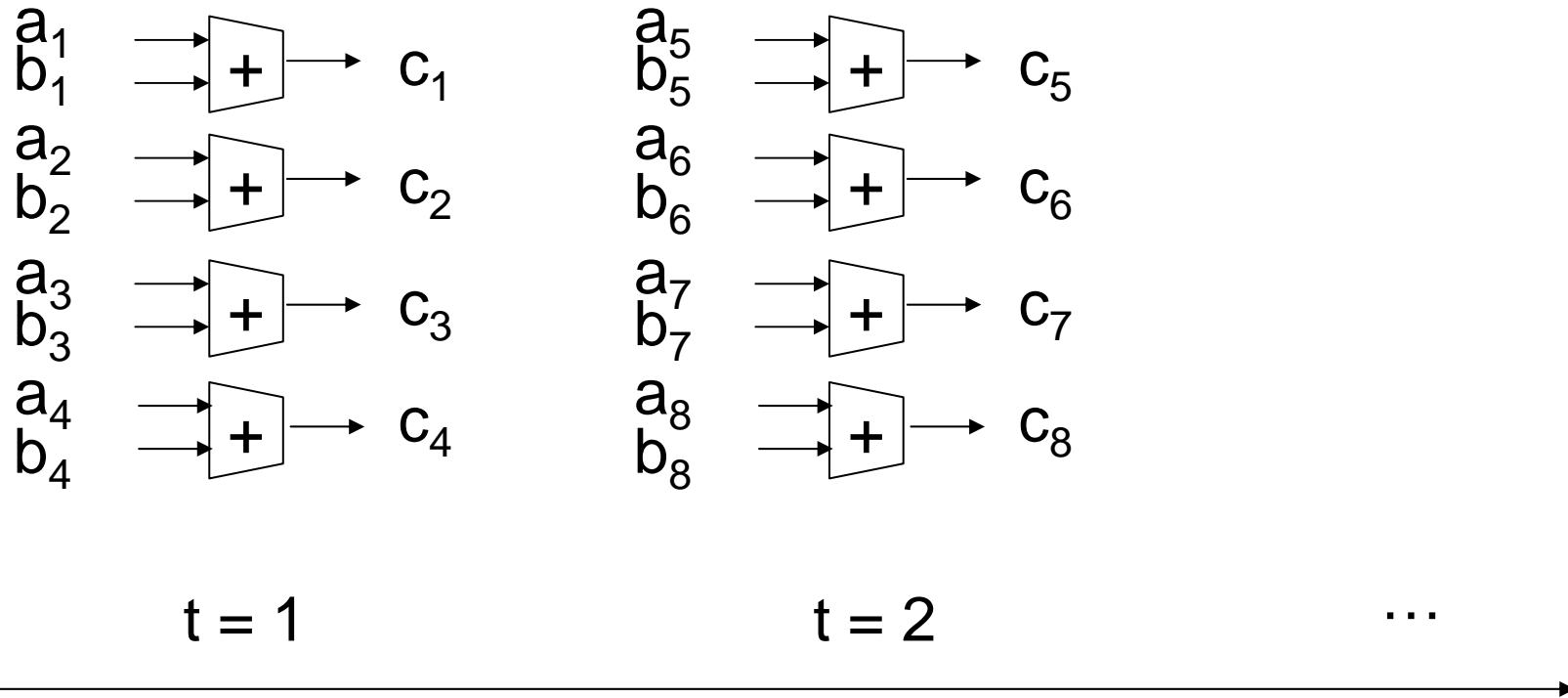
例) マルチメディア命令 (MMX, SSE 命令など)

例) GPU (Graphic Processing Unit)

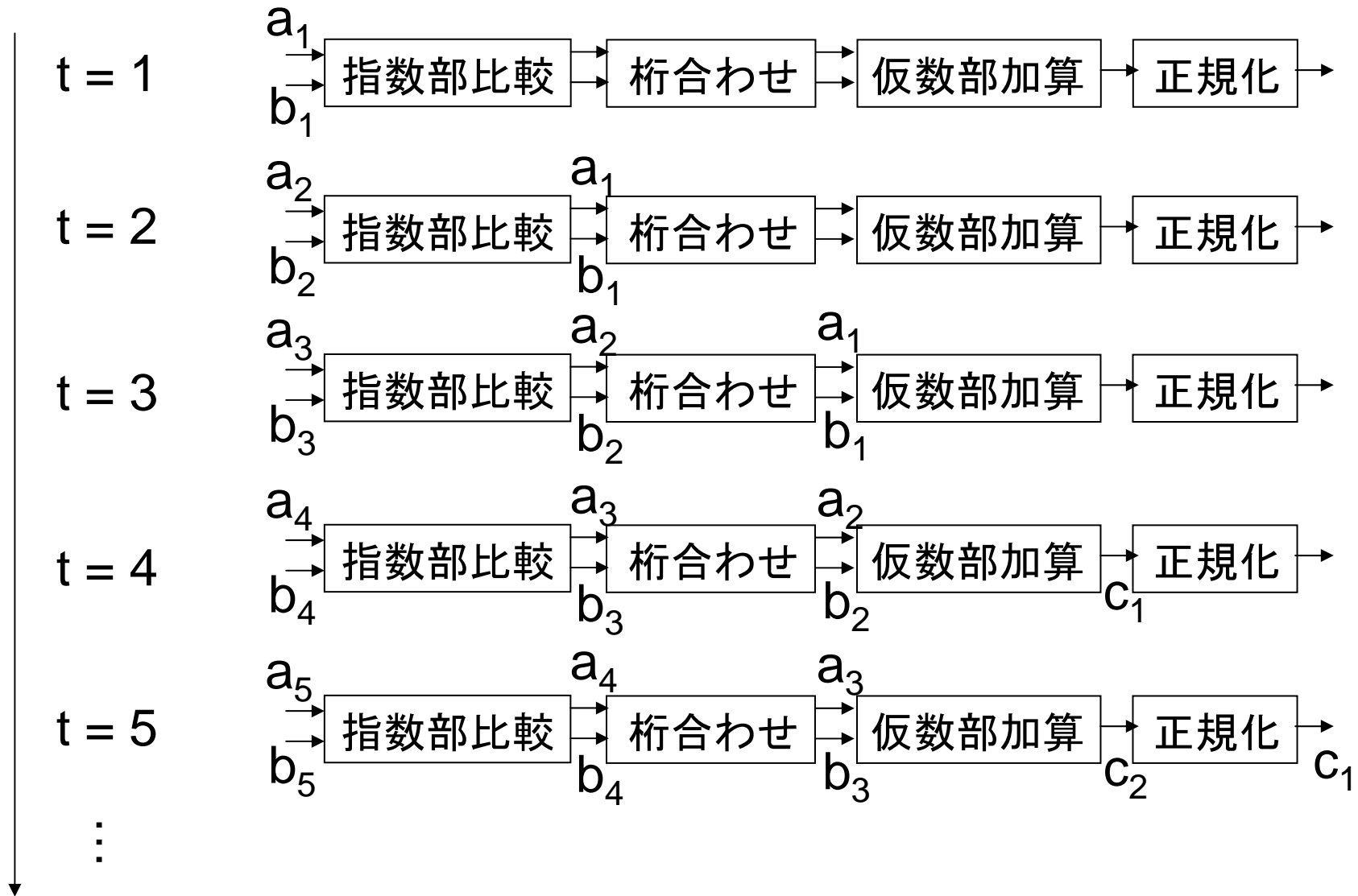
**時間並列**: 処理を複数のステージに分割してパイプライン化  
(ベクトル演算と呼ばれる場合がある)

# 空間並列の例

浮動小数点ベクトル  $[a_1, a_2, \dots, a_n]$  と  $[b_1, b_2, \dots, b_n]$  の加算

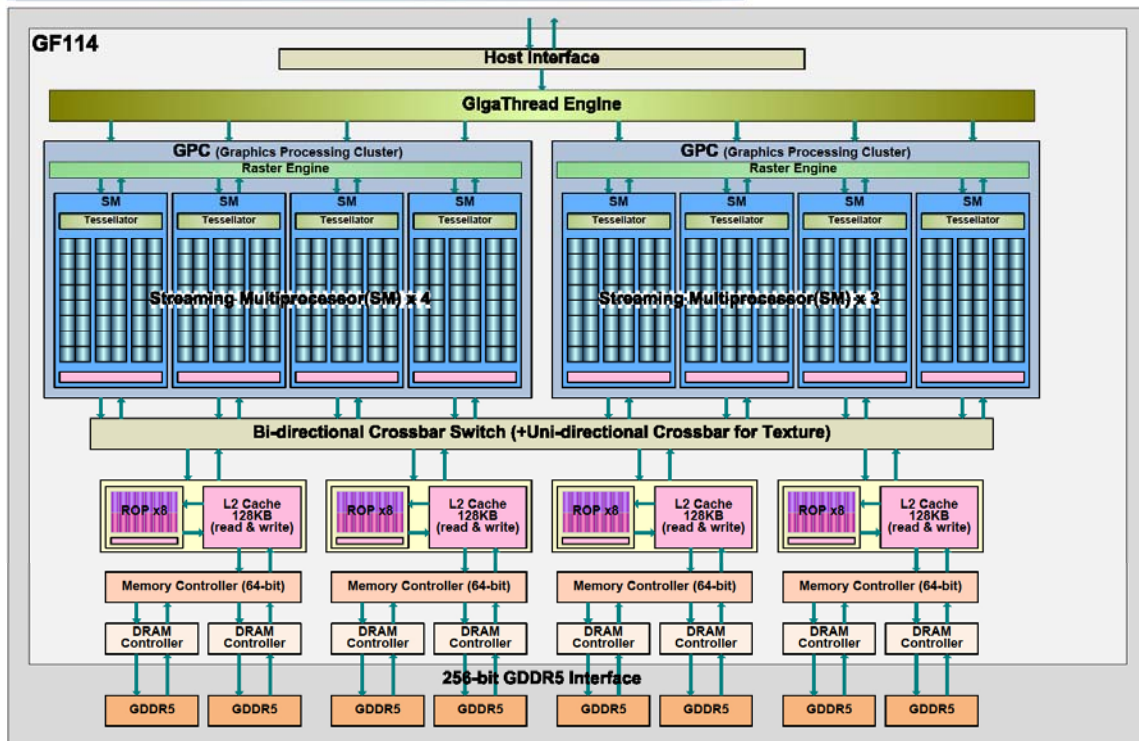


# 時間並列の例



# 例: GPU (Graphics Processing Unit)

GeForce GTX 560 Ti(GF114) Overview



Copyright (c) 2011 Hiroshige Goto All rights reserved.

NVIDIA GeForce GTX 560 Ti



[http://pc.watch.impress.co.jp/docs/column/kaigai/20110126\\_422573.html](http://pc.watch.impress.co.jp/docs/column/kaigai/20110126_422573.html)

<http://www.nvidia.co.jp/object/product-geforce-gtx-560ti-jp.html>

# スレッドレベル並列性

---

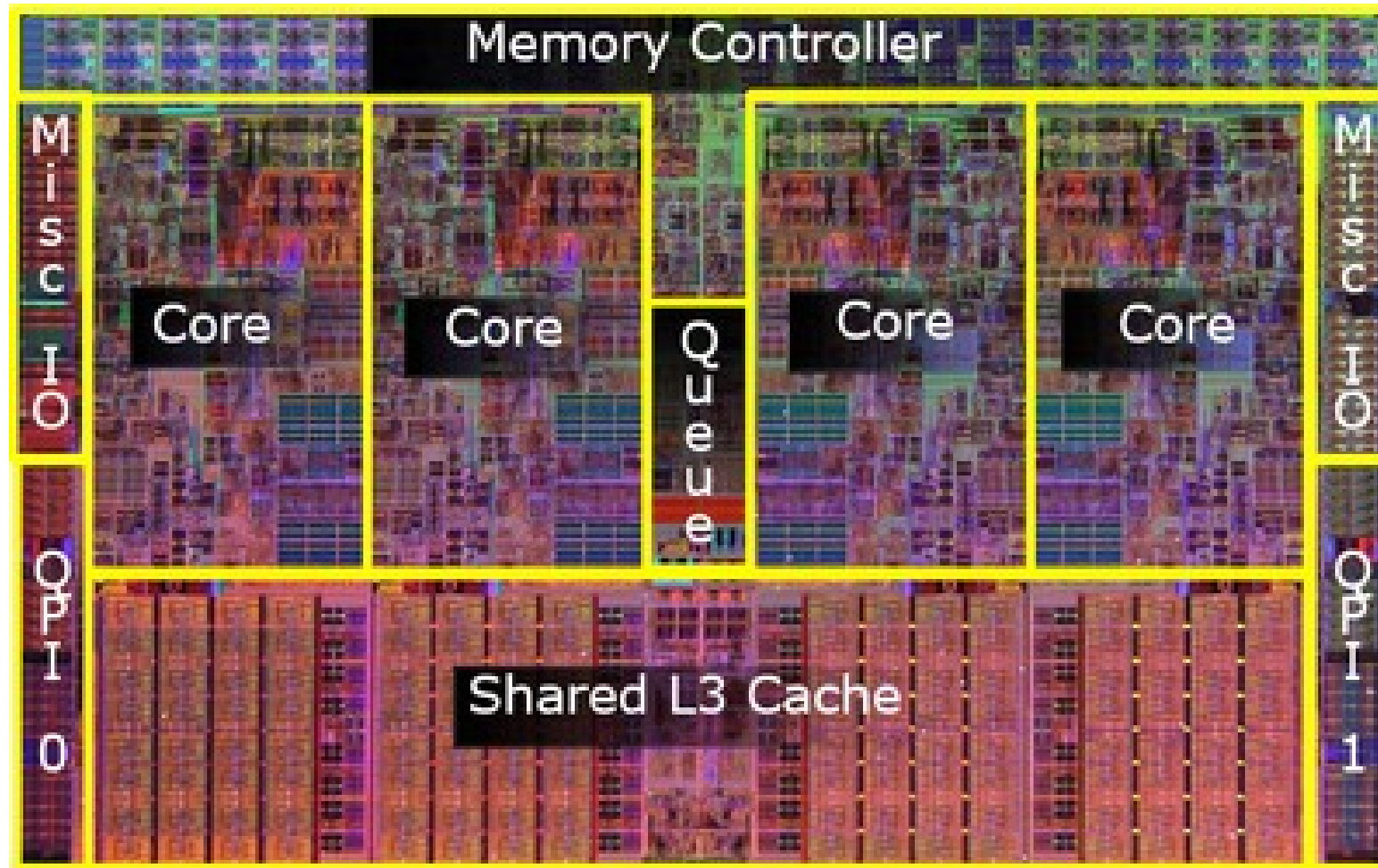
- クロックサイクル時間短縮: 消費電力の限界
- 命令レベル並列性: 3程度が限界
- データ並列性: アプリケーション依存

→ スレッドレベル並列性の活用へ

複数のプログラム, あるいはプログラム内の複数の処理の流れ (thread of control) からであれば, 同時に実行できる命令を容易に取り出すことができる

- 同時マルチスレッディング: スーパースカラプロセッサにおいて, 複数のスレッドからの命令を取り出して実行  
例) Intel の Hyper-Threading Technology
- マルチコア: 複数のプロセッサをチップ上に集積

## Intel Core i7 (2008)



# やっぱり話し切れなかったこと

---

コンパイラの動作(教科書9章):

- 高水準プログラミング言語からアセンブリ言語への変換
- 重要概念: 字句解析, 構文解析, 意味解析, コード生成

コンピュータネットワーク(教科書10章):

- 入出力装置を介して, コンピュータとコンピュータを接続できる. 多数のコンピュータ同士が接続され, ネットワークを構成する.
- 重要概念: パケット交換, TCP/IPモデル, IPアドレスとポート番号, DNS