コンピュータアーキテクチャ Ⅰ

九州大学大学院システム情報科学研究院
安浦寛人、中西恒夫 、井上弘士

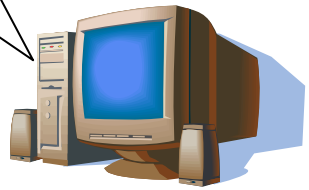
イントロダクション

コンピュータ処理に関する3つの疑問

Q1
「情報」をどのように表現しているのだろう？

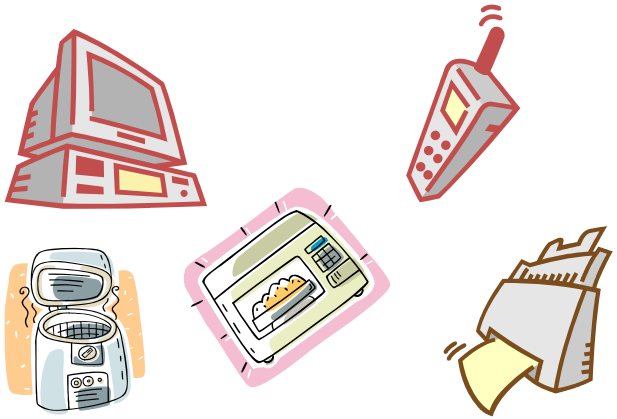
データ入力
（情報の収集）

Q3
どのような手順で「情報」を処理する（計算する）のだろう？



計算（情報の処理）

記憶（情報の蓄積）

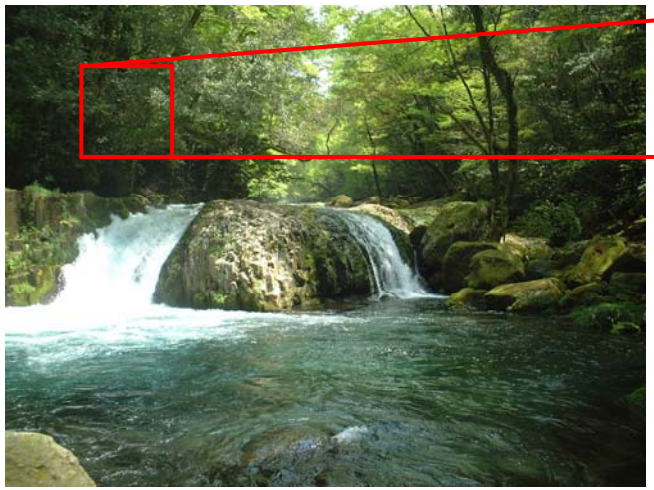


データ出力
（情報の表示）

Q2
どのようにして取得したデータの計算を行うのだろう？

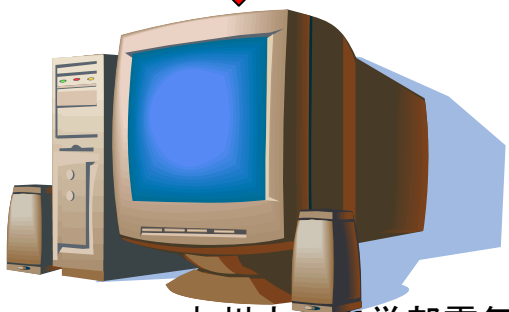
コンピュータの3つの常識(1)

常識1: 今日のコンピュータは、**2進法**で表現された情報を処理対象とする.



赤の強さ: 30
青の強さ: 57
緑の強さ: 79

赤の強さ: 00011110
青の強さ: 00111001
緑の強さ: 01001111



コンピュータの3つの常識(2)

常識2: 今日のコンピュータは、**スイッチング素子**を用いて2進法で表現されたデータの計算を行う.

0と1を物理現象に対応させることでデータを表現

例えば、高い電圧状態(1)と低い電圧状態(0)

コンピュータは，電子的にスイッチON/OFFを制御できる素子(スイッチング素子)を用いて，電圧の高低を制御する．このスイッチング素子を組合わせることで演算/制御回路を構成して計算する.

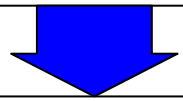
世代	西暦	スイッチング素子
第一世代	1950～1959	真空管
第二世代	1960～1968	トランジスタ
第三世代	1969～1977	集積回路
第四世代	1978～	LSI，VLSI

コンピュータの3つの常識(3)

常識3: 今日のコンピュータは**プログラム**の指示通りに動作する.

プログラム=2進数で表現される命令(**機械語**)の列

```
# 1 から $s1 までの和を $s0 に納めるプログラム
        add     $s1, $zero, $zero       # 0 + 0 → $s1(∴ 0 → $s1)
        addi    $s4, $zero, 1           # 0 + 1 → $s4(∴ 1 → $s4)
L1:     add     $s1, $s1, $s0           # $s1 + $s0→$s1
        sub     $s0, $s0, $s4           # $s0 – 1→$s0
        bne     $s0, $s4, L1            # Jump to L1 if $s0≠0
```



```
00000000000000001000100000100000
00100000000101000000000000000001
00000010001100001000100000100000
00000010000010100100000000100010
00010110000101000000000000000010
```

プログラミング言語

言語	わかりやすさ	説明
機械語	わかりにくい	算術演算，分岐などのプロセッサへの命令の並び．二進数で記述される．コンピュータが直接理解できるのは機械語のみ．
アセンブリ言語	↓	機械語と違ってADD，SUBなど命令を表す英単語や，命令のパラメータとなる数字等を用いて，プロセッサへの指示を記述する．読みやすいが，プロセッサのひとつひとつの命令は単純でありプログラムを理解するのは簡単ではない．アセンブリ言語はアセンブラによって機械語に変換される．
高級言語	わかりやすい	英単語，数式，記号など人間が理解しやすい抽象的な命令でプロセッサへの指示を記述する．高級言語はコンパイラによって同じ機能を有する機械語プログラムに変換するか，インタプリタという高級言語を解釈・実行する特別なプログラムを使って実行する． ex.) C, C++, Java, Fortran, Perl, COBOL, BASIC, Prolog, LISP, ...

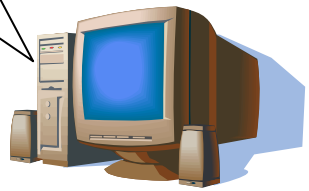
コンピュータ処理に関する3つの疑問

Q1
「情報」をどのように表現
しているのだろう？

データ入力
（情報の収集）

A1:情報を2進法で表現
（010110010101）

Q3
どのような手順で「情報」を処
理する（計算する）のだろう？



計算（情報の処理）

記憶（情報の蓄積）

A3:プログラムにより明示
的に指示

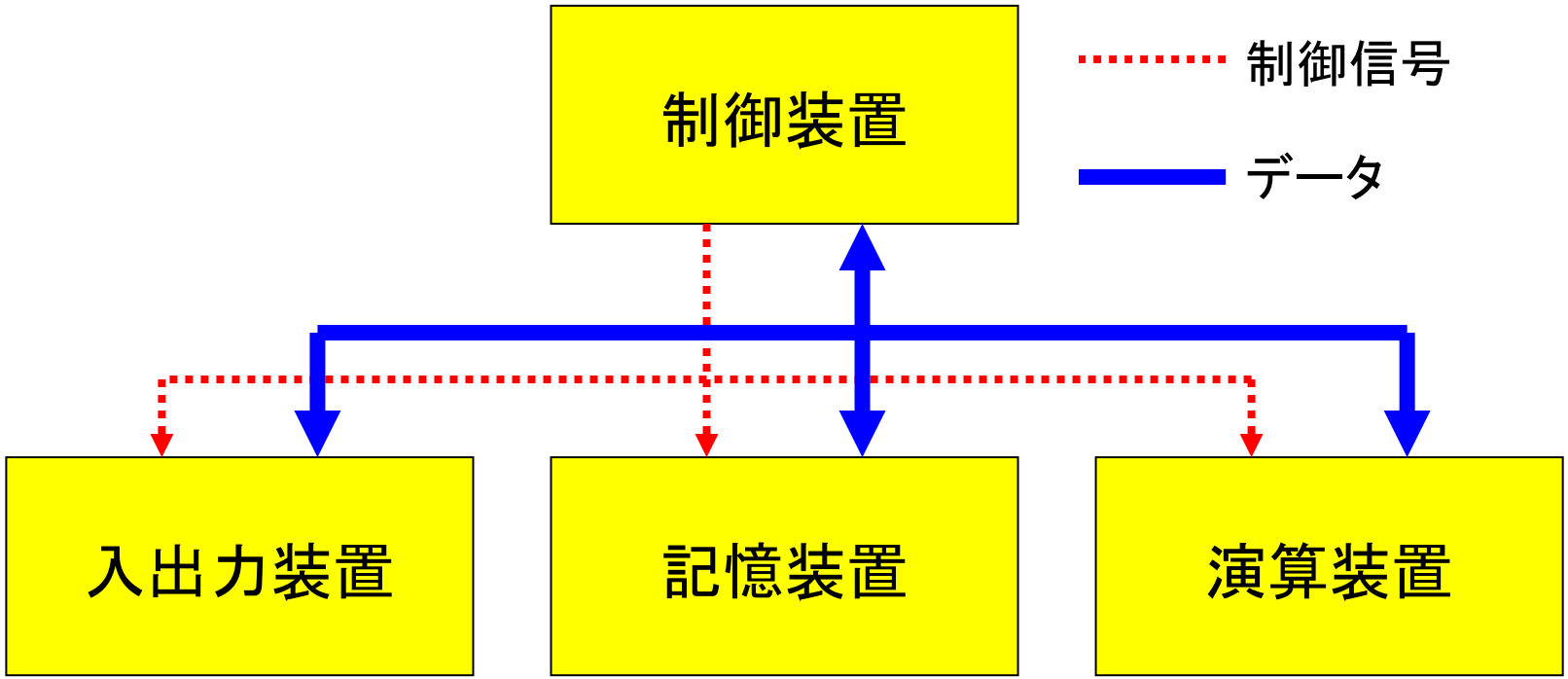
Q2
どのようにして取得データに
対する計算を行うのだろう？

データ出力
（情報の表示）

A2:スイッチング素子で（2進
数用）演算/制御回路を実現

ノイマン型計算機(1)

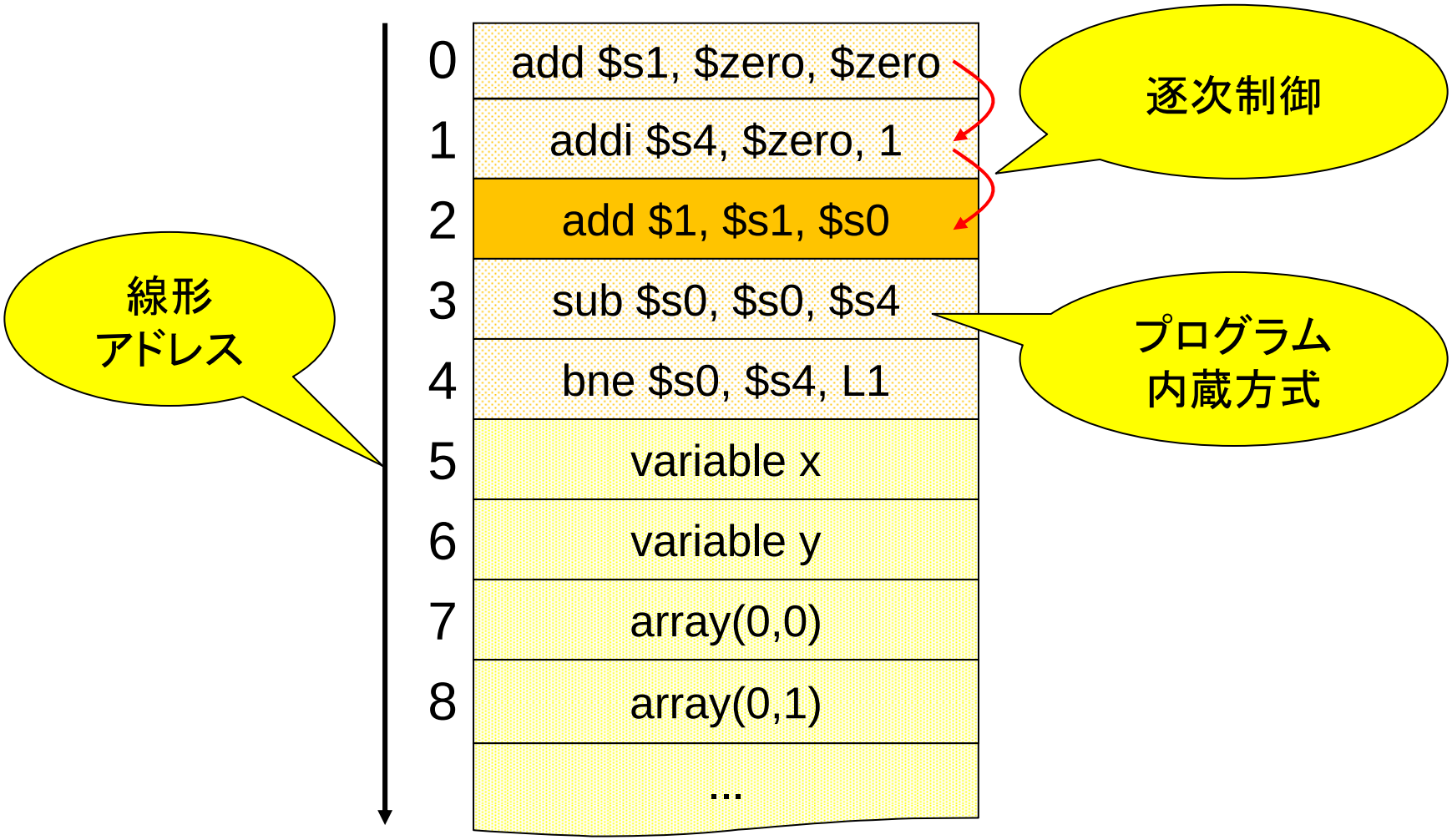
今日の多くの計算機はノイマン型計算機として設計されている.



ノイマン型計算機(2)

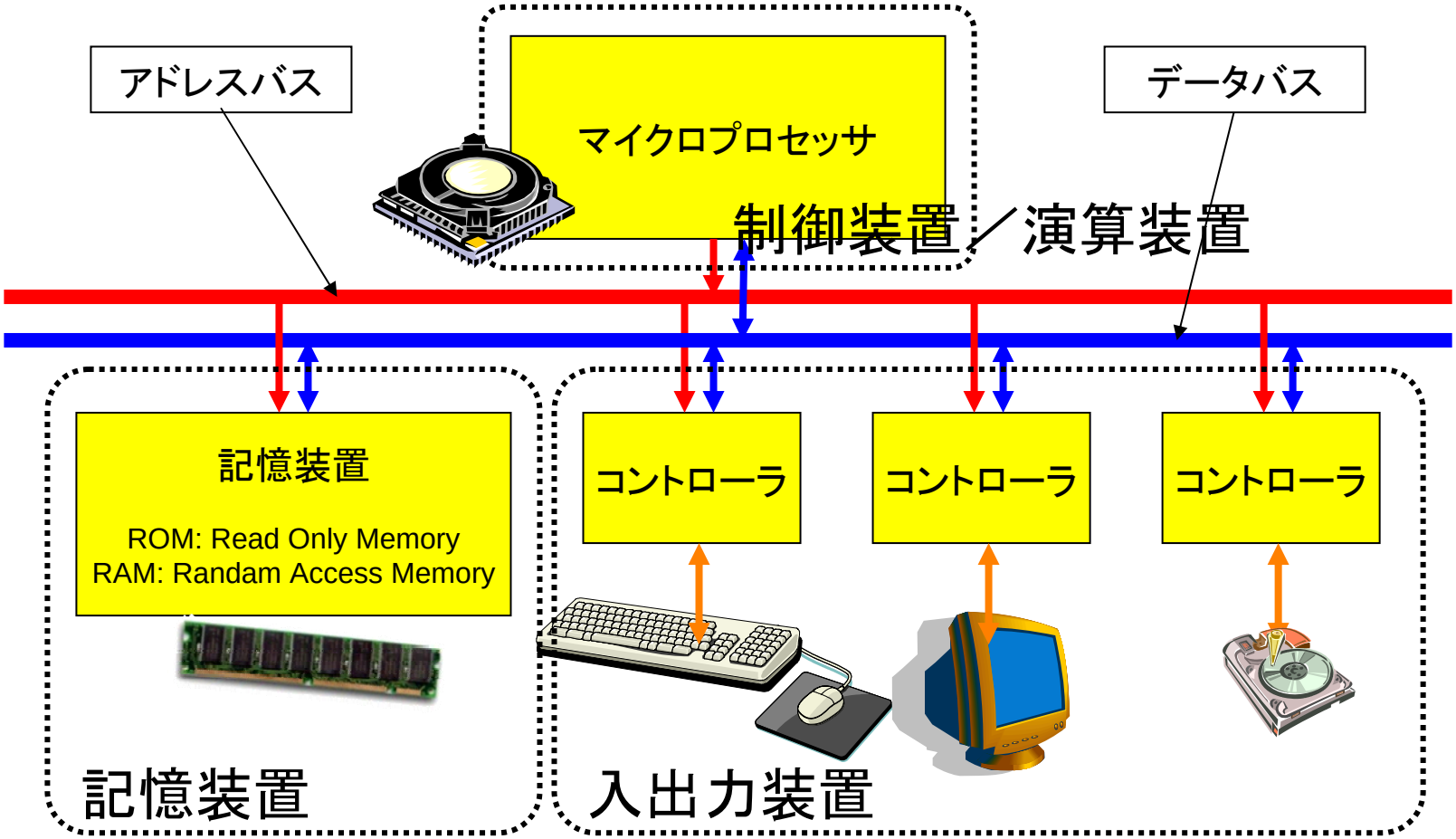
- **プログラム内蔵方式:** 命令とデータは主記憶中に区別なく置かれる. 命令とデータの区別は実行するプログラムによって行われる.
- **逐次制御:** 命令は主記憶からひとつひとつ取り出されて, 決められた順番で実行される.
- **線形アドレス:** 主記憶の各セルは0から順番に番号が振られる. この番号は番地(アドレス)と呼ばれ, 命令やデータのある場所を指示するのに用いられる.

ノイマン型計算機(3)


0
add $s1, $zero, $zero
1
addi $s4, $zero, 1
2
add $1, $s1, $s0
3
sub $s0, $s0, $s4
4
bne $s0, $s4, L1
5
variable x
6
variable y
7
array(0,0)
8
array(0,1)
...
逐次制御
線形
アドレス
プログラム
内蔵方式

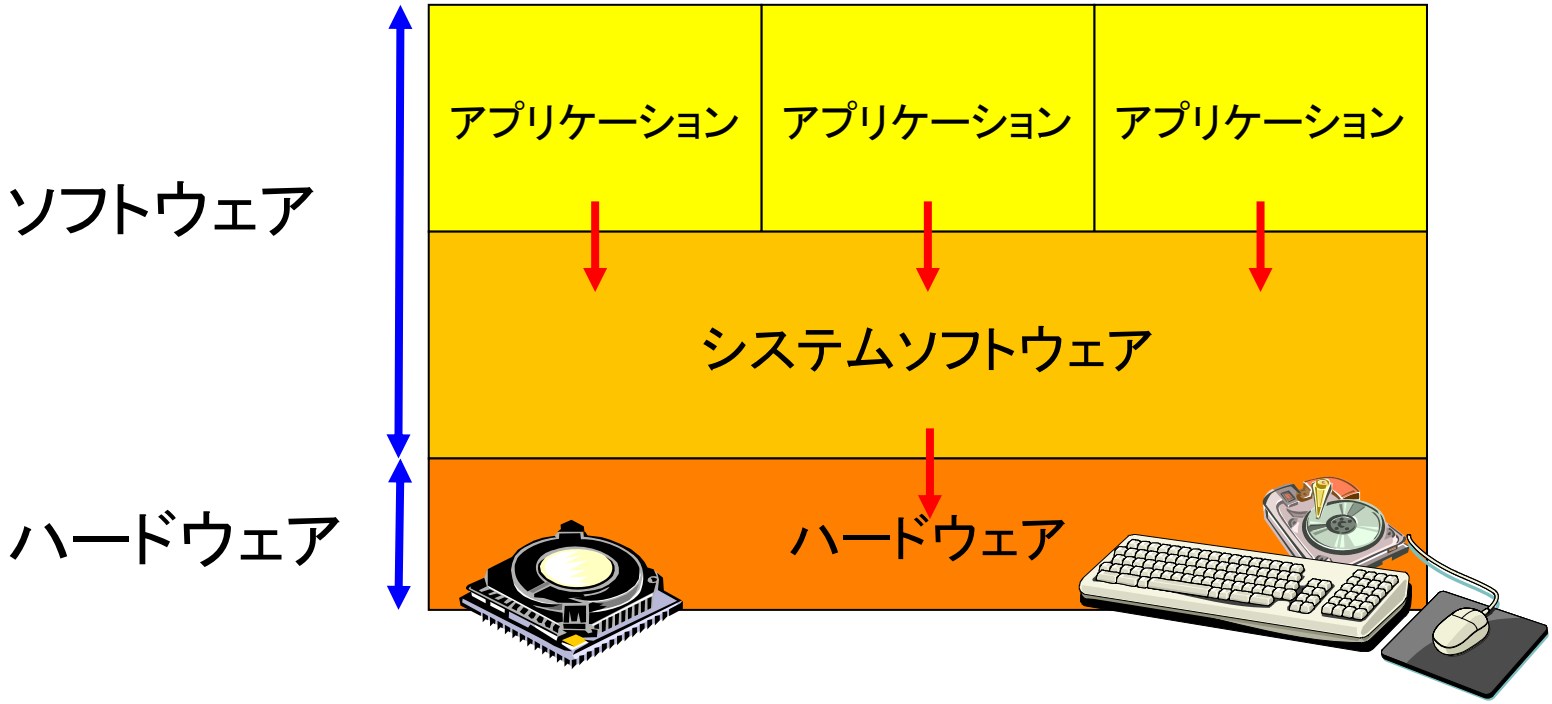
ノイマン型計算機(4)

ノイマン型計算機の実際の例



階層設計と抽象化(1)

階層設計: 上位の要素は下位の要素の提供する機能(インターフェース)のみを使って，自分の機能を実現する．=下位は自身の構造の詳細を隠して，インターフェースのみからサービスを提供している．(**抽象化**)

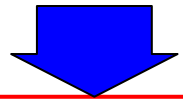


階層設計と抽象化(2)

階層設計すると…

- 下位がどのように実装されているかを考えずに，自身の設計・実装に集中できる.
- 下位の提供するインターフェースを共通化することで，上位の移植性が向上する.

プロセッサがソフトウェアに対して提供するインターフェース



命令セットアーキテクチャ

- 命令の種類
- 命令の二進数への符号化
- データの表現法 *etc.*