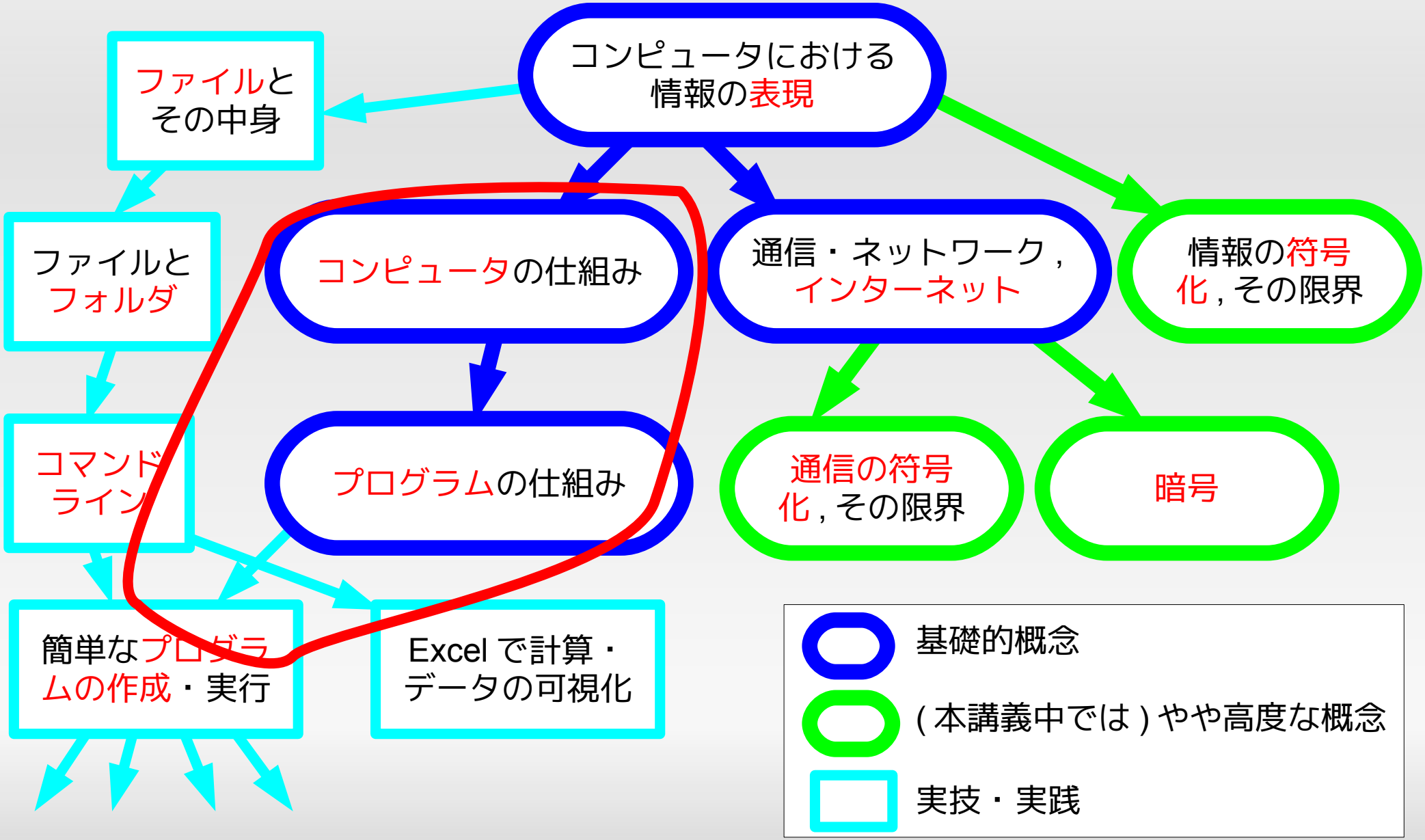
コンピュータが「計算」をする仕組み

- 田浦健次朗

本日の範囲


コンピュータにおける
情報の表現
ファイルと
その中身
ファイルと
フォルダ
コマンド
ライン
コンピュータの仕組み
プログラムの仕組み
通信・ネットワーク，
インターネット
情報の符号
化，その限界
通信の符号
化，その限界
暗号
簡単なプログラ
ムの作成・実行
Excel で計算・
データの可視化
基礎的概念
(本講義中では) やや高度な概念
実技・実践

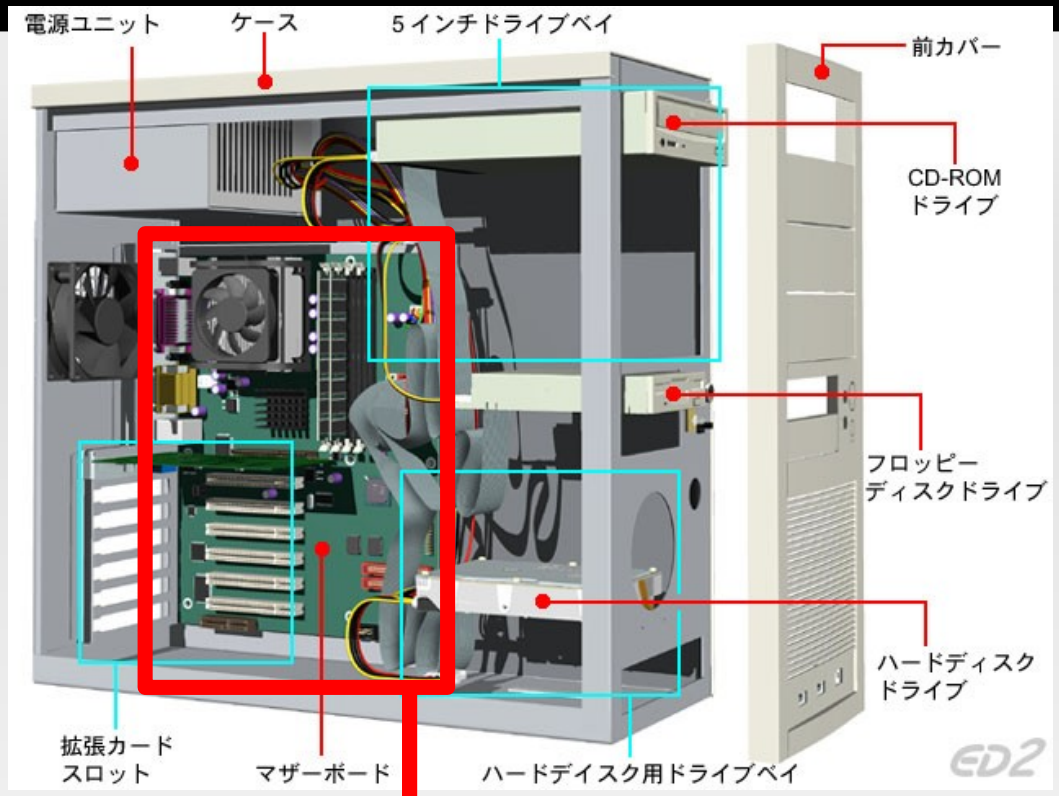
目標

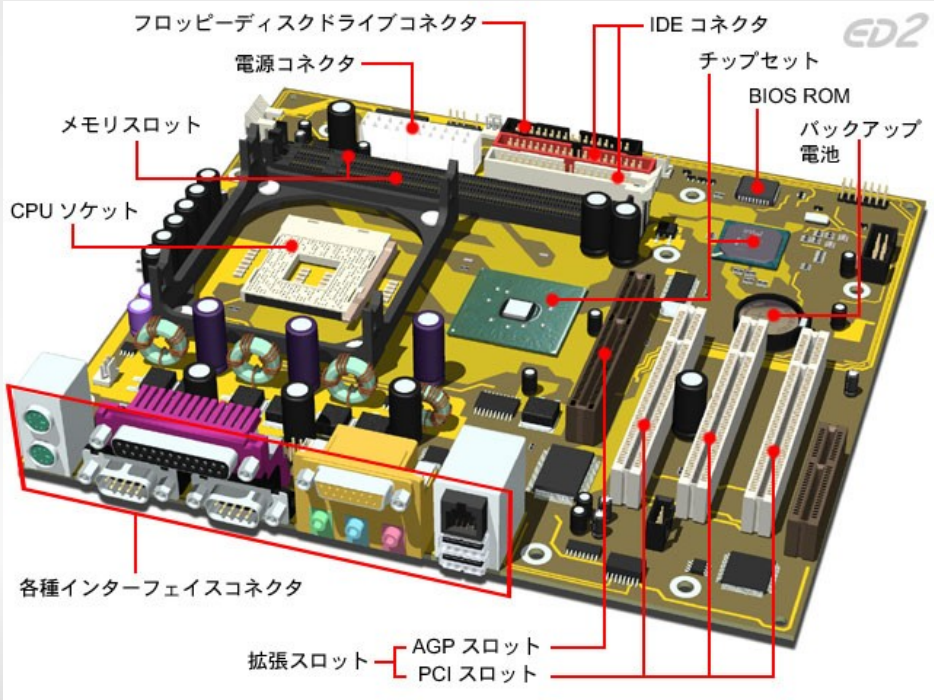
- コンピュータの物理的な構成を大雑把に理解
  - CPU, メモリ , 2 次記憶 , ディスプレイ , マウス , キーボード
- コンピュータは ,CPU によって「機械語のプログラム」を実行して計算をしていることを理解
- 機械語のエッセンスを理解
- 高級言語 (プログラミング言語) の基本アイデアを理解
- (演習) 具体的なプログラミング言語で簡単なプログラムを書いてみる

演習目標

- プログラミング言語 Python
- プログラムを書く時の考え方
- 具体的目標
  - 数列 , 微分方程式 (Excel でやった例と比較しながら)
  - 関数の最大・最小や根の計算
  - 各種の確率や期待値の計算

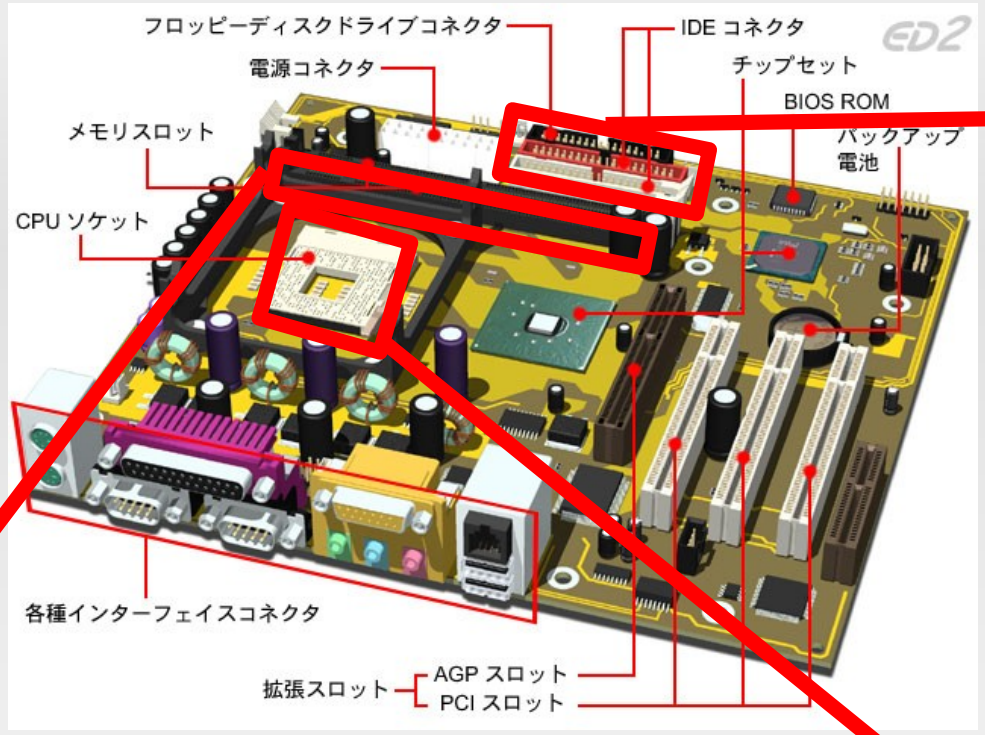
コンピュータの物理的構成



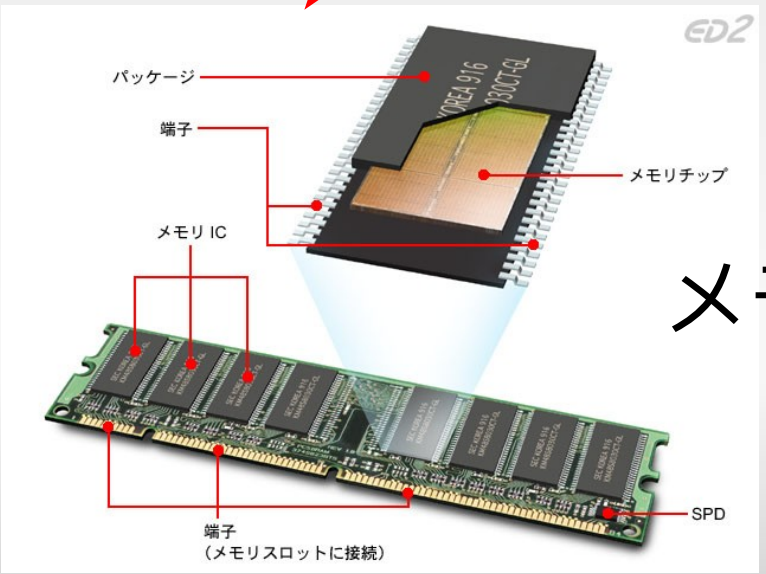


出典：「情報機器と情報社会の仕組み素材集」
http://www.sugilab.net/jk/joho-kiki/index.html

CPU, メモリ , 2 次記憶

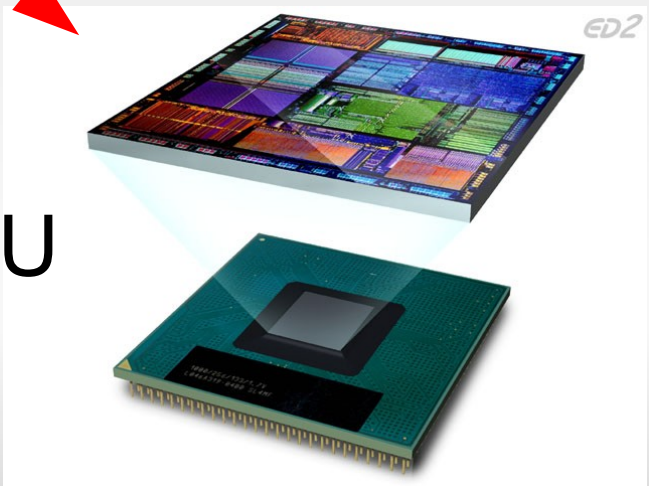


ハードディスク
(2 次記憶)



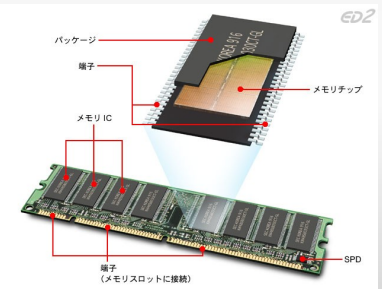
メモリ

CPU



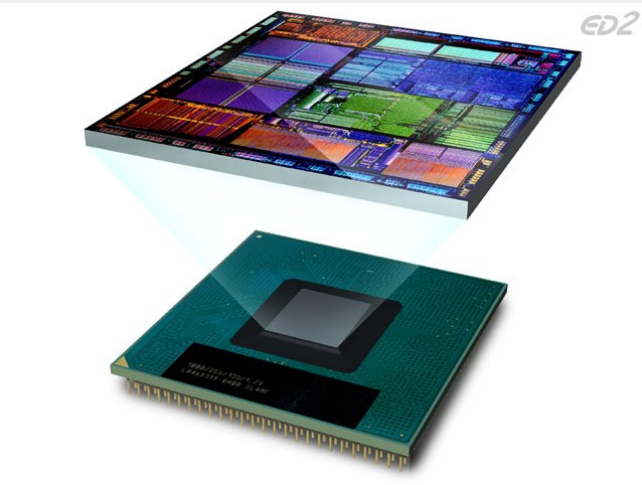
メモリ (主記憶)

- Bit (0 or 1) を保持できる (= 2 つの状態を区別できる) セルの集まり
- 実際には 8 bit (1 byte) より小さい単位でアクセスされることはない
  - 今時のメモリ 1 モジュール 数 GB (1GB=10^9 bytes)
- 各 byte に通し番号 (番地；アドレス) があり，指定した番地のデータを読み書きできる



CPU の役割

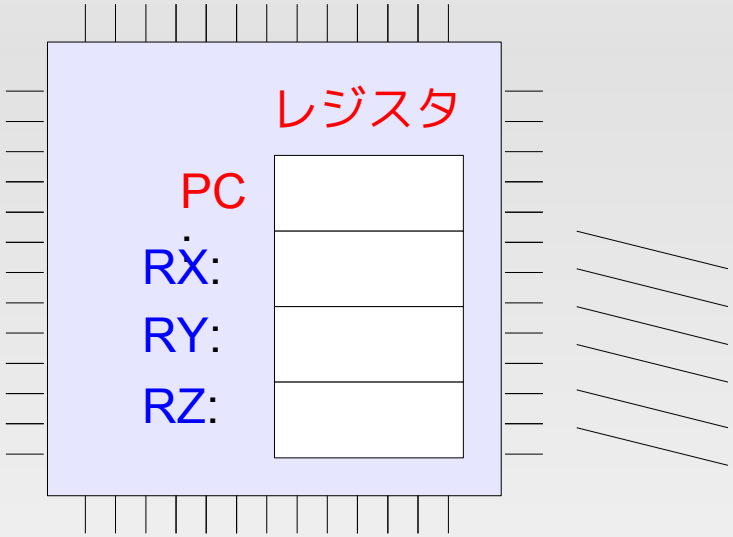
- コンピュータが動く =CPU が命令を実行する
  - メモリ上に書かれている「命令」を実行
  - 「命令」を実行した結果 , メモリ上のデータを書き換えることができる
- 電源を入れた瞬間から切るまで , プログラムを実行せずにやっていることはないといって過言ではない



ハードディスク (2 次記憶) の役割

- メモリ (主記憶) と同様 , 「記憶装置の一種」
- 主記憶との違い
  - 永続的 (= 電源を切っても消えない)
  - CPU が直接アクセスできるわけではない (= 一旦メモリに移動してから処理する)
  - 容量 / 費用 が大 (= 安い)
- ⇒ 2 次記憶は「ファイル」を格納する場所として使われている

計算の「原理」を知るための簡単化された CPU の構造



メモリ

0:		
1:	load 10,R1	命令列
2:	load 11,R2	
3:	add R1,R2,R3	
4:	store R3,12	
5:	halt	
6:		
7:		
8:		
9:		データ
10:	123	
11:	345	
12:		
13:		

番地 (アドレス)

レジスタ

- CPU 内部の少量のメモリ
- 主記憶とほとんど役割は同じ (足し算・掛け算などの「演算」はレジスタに対してのみしか行えない , という CPU もある)
- 特別重要なレジスタ : プログラムカウンタ (PC)
  - 次に実行すべき命令の番地を保持するレジスタ

CPU の動作

- 以下をひたすら繰り返す
  - PC に格納されている番地に格納されている命令 (PC が指す命令) を実行
  - 実行した命令が「停止命令 (halt) 」だったら終了
- 「命令を実行」すると
  - 命令で指定されたメモリ , レジスタ (普通一ヶ所) が書き換わる
  - 加えて ,PC が「次の命令」を指すように書き換わる
    - ほとんどの命令は PC を 1 命令分増やす⇒次の命令はメモリ上での「次の命令」)

CPU の動作のアナロジー

- 電源を入れるとマニュアル本の 1 ページ目を開き , 黙々と指示 (命令) にしたがう
- マニュアル本の脇に方眼紙 (メモリ) も置いてある
- マニュアルにかかれている指示の例
  - 方眼紙の XX マス目に YY と書け
  - 方眼紙の XX マス目と YY マス目の値を足し ,ZZ マス目に書け
- あるページを読み終わったら
  - たいがいの場合次のページへ進む
  - たまに ,「この質問の答えがハイなら XX ページ目へ進め (戻れ) 」

命令の種類 (大雑把な分類)

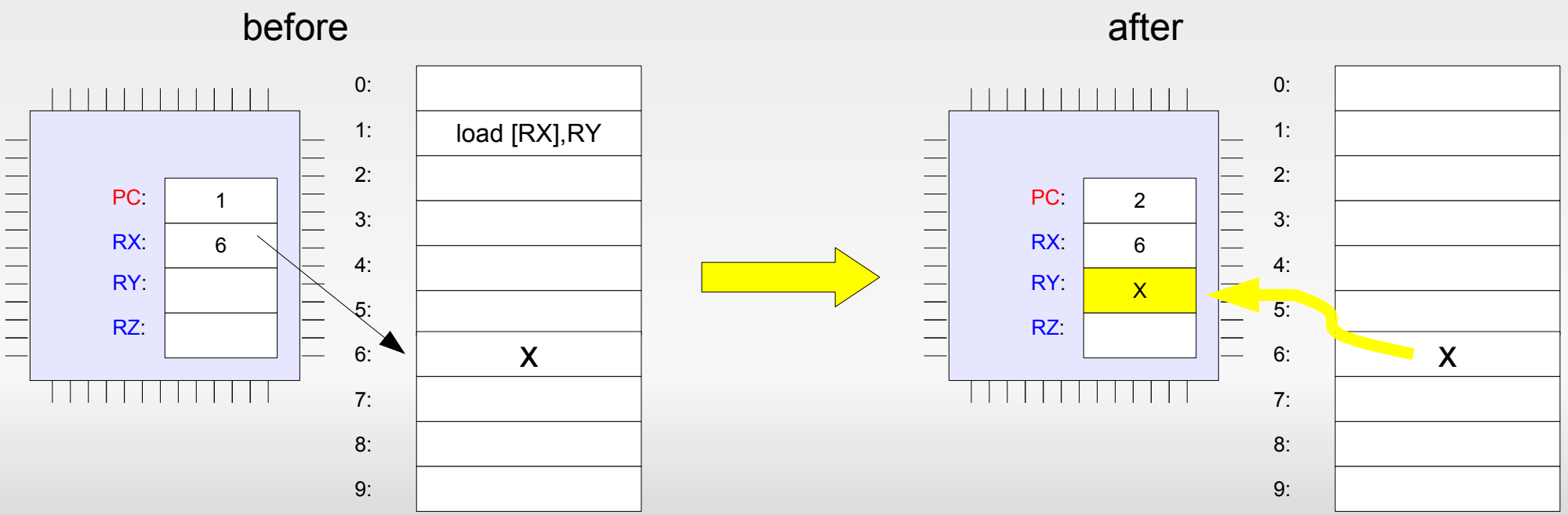
- メモリ・レジスタ間移動 (ロード・ストア)
- 演算 (加減乗除 ,etc.)
- 比較 (大小比較 , 等号比較 , etc.) と分岐
-
- CPU ごとに似て非なる記法 (機械語 , アセンブリ言語) で表す

記法上の注意

- 以下で「例をあげて」それらしく説明するが ...
- 現実の CPU の記法と似ているが , 架空のもの (教科書や過去問とも違う)
  - 現実の CPU は命令の種類が多い (Intel : > 700 個) し , 機種により記法が異なる
  - それらが多くなる理由 :
  - とにかく演算の種類が多い
  - 同じ動作の演算でも対象となる語の種類 (8bit, 16bit, 整数か小数点つきの数か) により命令が違う
- 要するに「原理は同じ」だが記法の「世界標準」はない (⇒ 細かい違いを気にしなくてよい)

ロード(メモリ→レジスタ)

- **`load [R1],R2`**
  - レジスタ R1 に格納されているメモリ番地に格納されている値をレジスタ R2 にコピー
- (**`load [100],R2`** は「100 番地」からロード)

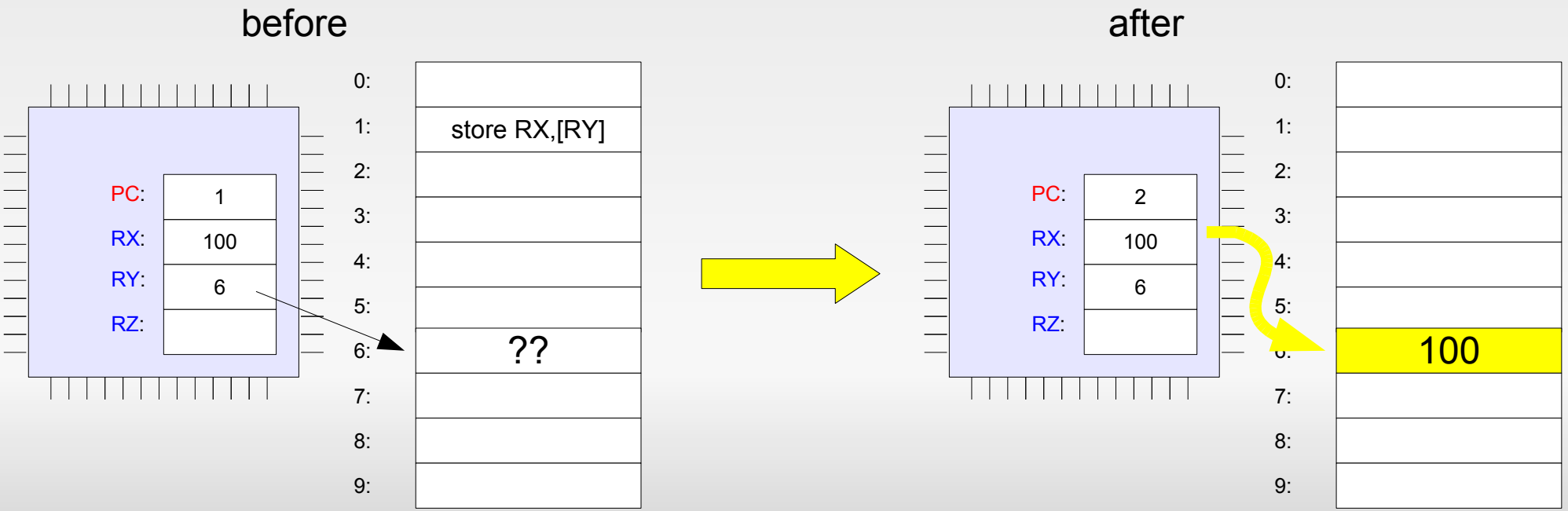


以降言葉の省略

- 混乱の恐れがないときは
  - レジスタ R に格納されている値 → R(の値)
  - レジスタ R に格納されている番地 → R が指す番地
  - R1 が指す番地に格納されている値 → R が指す番地の値
- と省略する
- 例 : **`load [R1],R2`**
  - R1 が指す場所の値を R2 にコピー

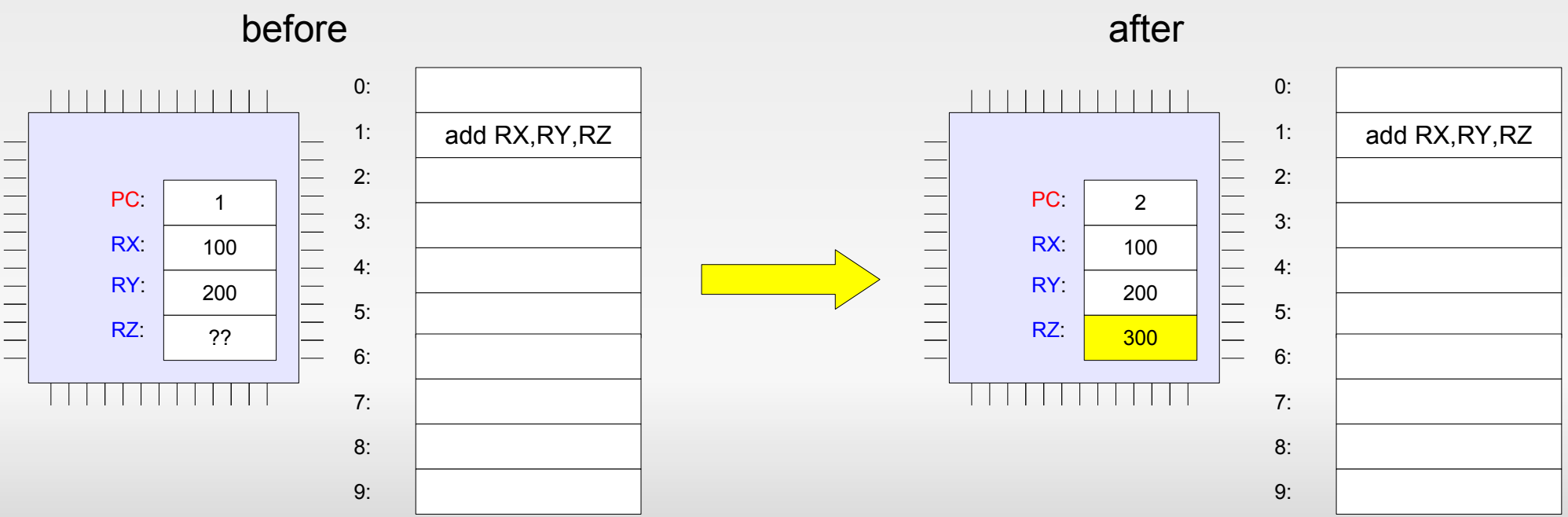
ストア (レジスタ→メモリ)

- **`store R1,[R2]`**
  - R1 を ,R2 が指す番地にコピー
- **(`store R1,[100]` は 100 番地にコピー)**



演算

- **`add R1,R2,R3`**
  - R1 + R2 を R3 へ格納



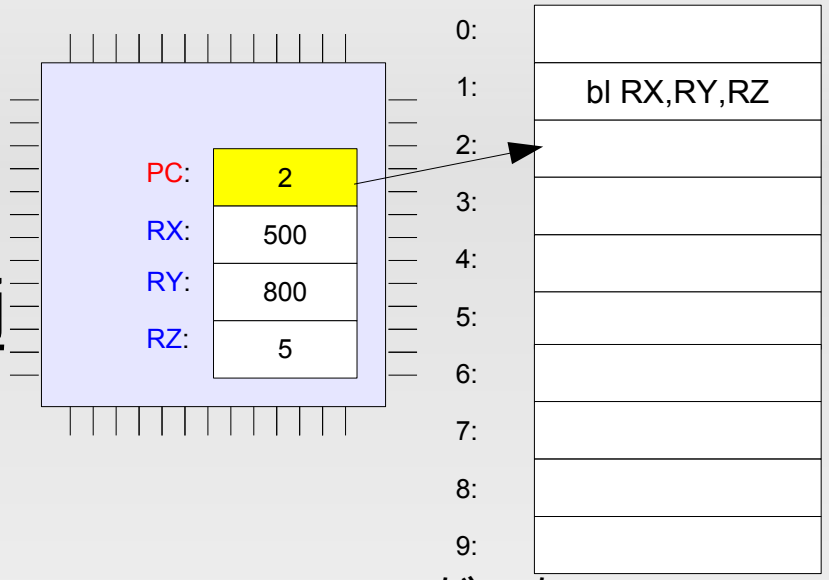
その他の演算

- sub (R1 – R2), mul (R1 * R2), … (算術演算)
- lt (R1 < R2), eq (R1 = R2), … (比較演算)
- and (R1 and R2), or (R1 or R2), … (論理演算)
- …

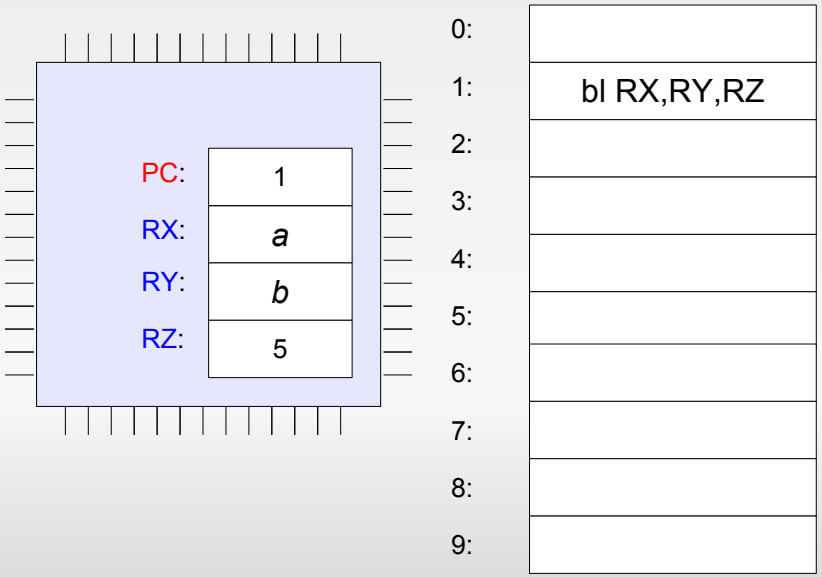
分岐

- **`bl R1,R2,R3`**
- R1 < R2 ならば PC を R3 に，そうでなければ (普通どおり) 次の命令の番地に

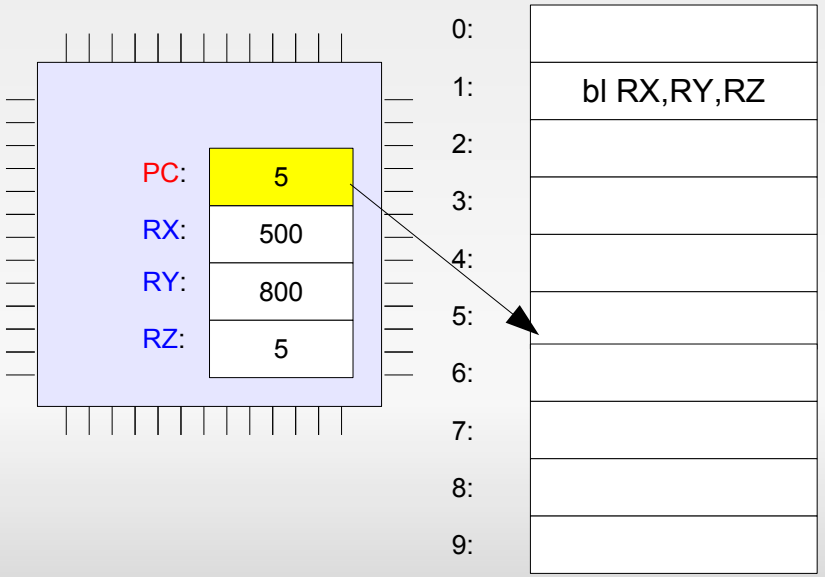
after (a >= b だったら)



before



after (a< b だったら)



分岐の意義

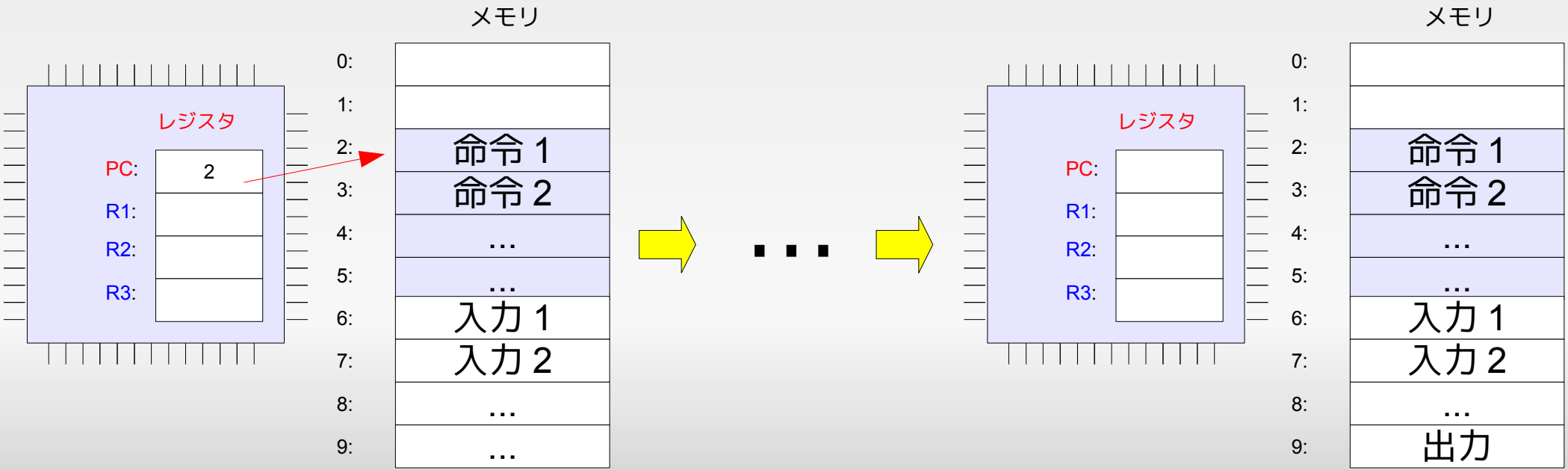
- 場合分け
  - 例 :x の正負によって以降の計算方法を変える
- 繰り返し
  - 同じ命令を何度も繰り返して実行する

halt 命令

- 計算の終了を示す命令
- 現実の CPU では計算を終了したからと言って CPU を本当に止めてしまう分けではないが，「計算の終わり」を示すための何らかの命令が必要ということ

「○○を計算するプログラム」とは

- 入力データがメモリにおかれた初期状態から，指定された番地に PC を設定し，あとはひたすら halt 命令まで実行し続けると
  - 必ず有限回の命令で停止 (halt) し，
  - メモリ上の決まった位置に「答え」が書かれている
- ような命令列の事



いくつかの現実の数字

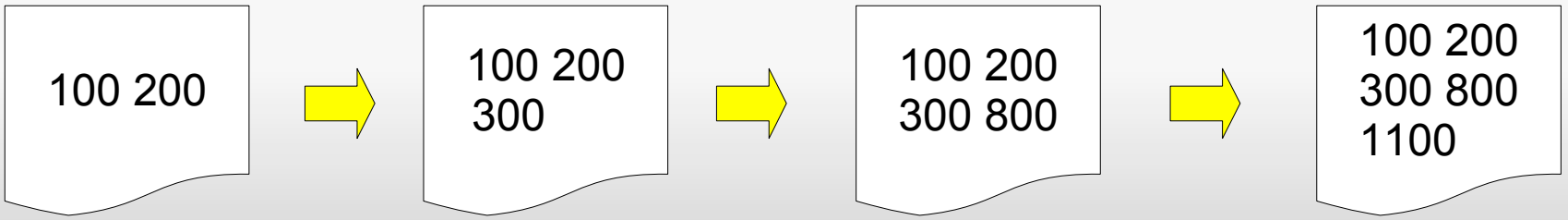
- 今時の普通のコンピュータ (目安)
  - 命令実行速度 : 1GHz-3GHz (1 サイクル 1 命令として 10 億～ 30 億命令 / 秒)
  - メモリの大きさ : 1GB-100GB (10 億～ 1000 億バイト)
  - ディスクの大きさ : 100GB-1TB (1000 億～ 1 兆バイト)
  - CPU がメモリを読む速度　1GB/sec-100GB/sec (1 億～ 100 億バイト / 秒)
  - ネットワークからデータを受け取れる速度　100 万～ 10 億バイト　(1MB/sec - 1GB/sec)

例題

- 入力 :
  - X が 100 番地　　(以降の記法 : x@100)
  - Y が 101 番地　　(y@101)
- 出力 :
  - 3x + 4y を 102 番地

どう考えるか？

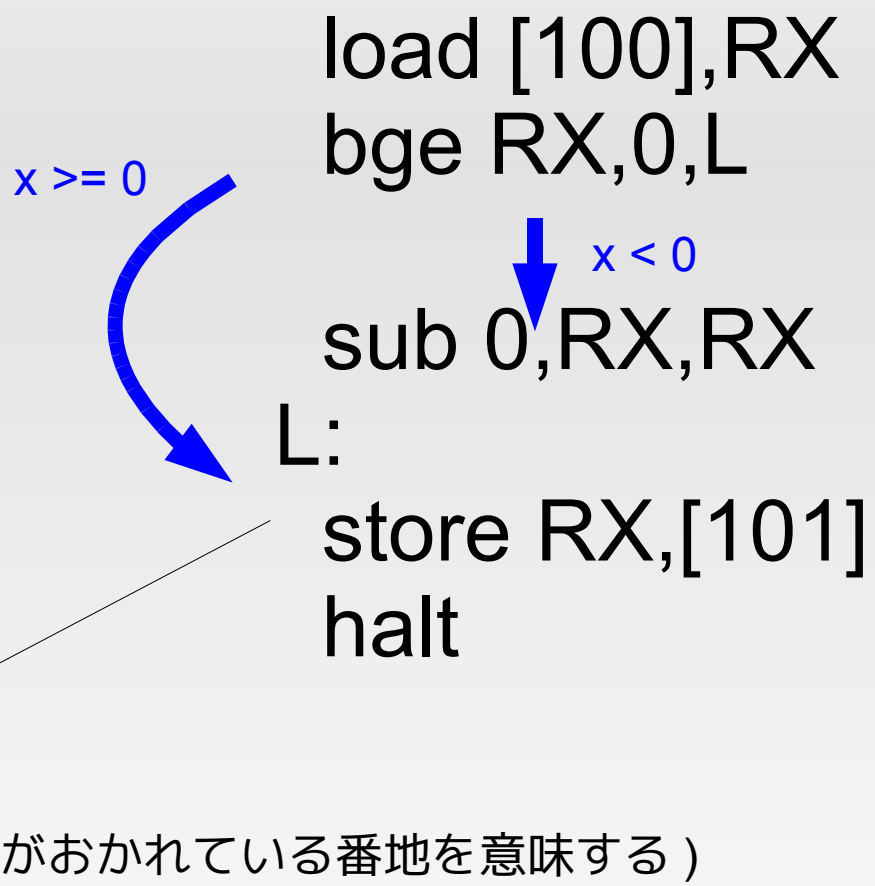
- 方眼紙と鉛筆と消しゴムが渡されたと想像
  - 紙がメモリ
- 方眼紙の余白はいくら使ってもよい (余白じゃなくてもよい . 完璧な消しゴムあり)
- 走り出したら手は出せない
  - あくまで「完璧なマニュアルを書く」のが目的
  - 各ステップは紙の上にかかれた数字に簡単な演算を施して , 紙の余白に結果を書く , ということしかしてはいけない



3x (x')	load [100],RX mul 3,RX,RY store RY,[1000]
4y (y')	load [101],RX mul 4,RX,RY store RY,[1001]
x' + y'	load [1000],RX load [1001],RY add RX,RY,RZ store RZ,[102] halt

|x| を計算するプログラム

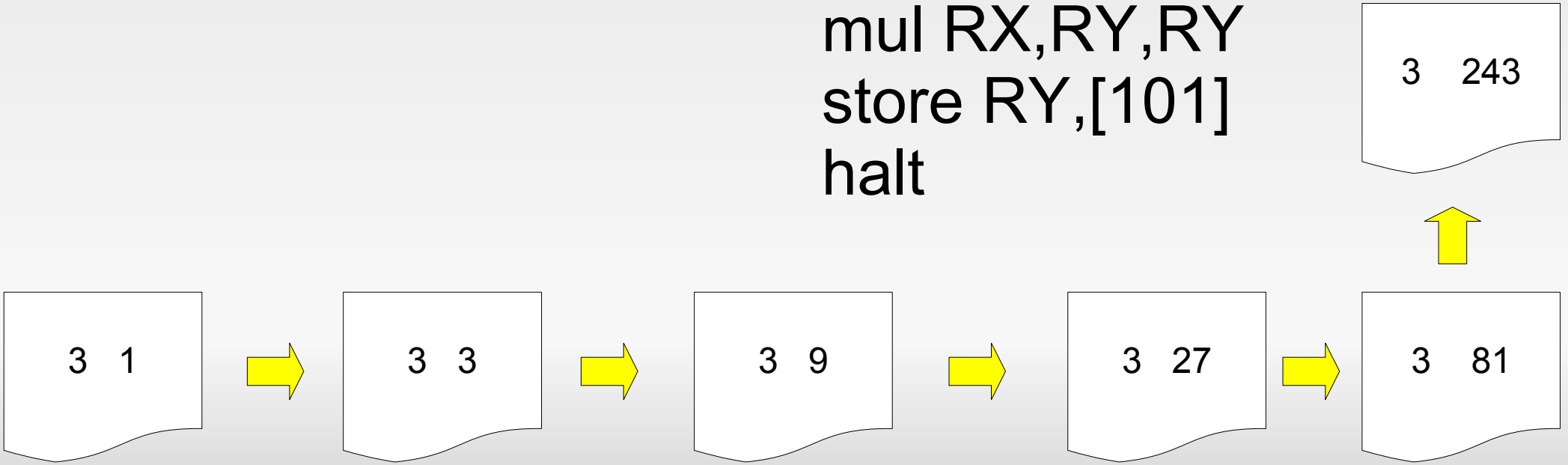
- 入力 :
  - x @ 100
- 出力
  - |x| @ 101



x^5を計算するプログラム

- 入力
  - x @ 100
- 出力
  - x^5 @ 101

```
load [100],RX
add 0,1,RY
mul RX,RY,RY
mul RX,RY,RY
mul RX,RY,RY
mul RX,RY,RY
store RY,[101]
halt
```



x^n を計算するプログラム

- 入力
  - x @ 100
  - n @ 101
- 出力
  - x^n @ 102

以下は答えではない

- えーと ,x^3 ならこれで ,x^7 ならこっち , とか

```
load [100],RX
add 0,1,RY
mul RX,RY,RY
mul RX,RY,RY
mul RX,RY,RY
store RY,[101]
halt
```

```
load [100],RX
add 0,1,RY
mul RX,RY,RY
mul RX,RY,RY
mul RX,RY,RY
mul RX,RY,RY
mul RX,RY,RY
mul RX,RY,RY
mul RX,RY,RY
store RY,[101]
halt
```

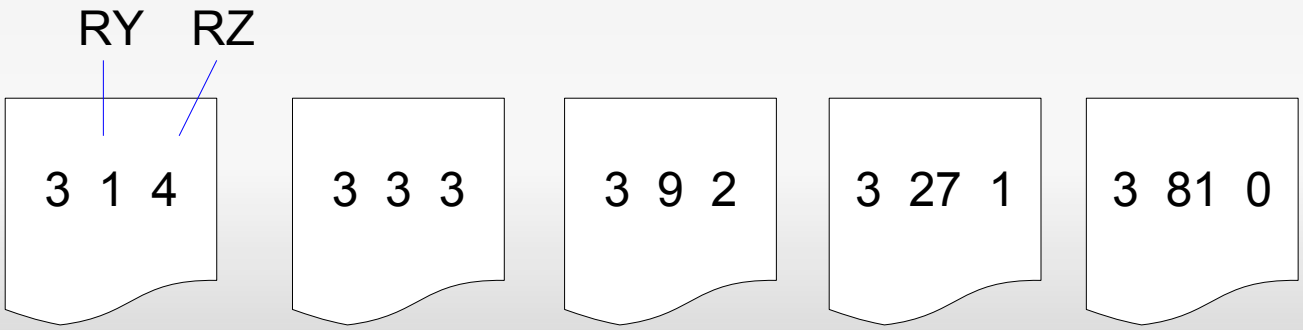
求められているのはあくまで , 入力 (n) に応じて , x の何乗でも計算できる「ひとつの」プログラム

答え：

- あと何回かけるのかもメモリ (またはレジスタ) に書いておく
- その値が 0 になるまで「繰り返し」(ジャンプを使って)

```
  load [100],RX
  load [101],RZ
  add 0,1,RY
L1:
  ble RZ,0,L2

  mul RX,RY,RY
  sub RZ,1,RZ
  b L1
L2:
  store RY,[102]
  halt
```



「方眼紙と鉛筆」とのアナロジー

- 人間だったら「あと何回かけるのか」を無意識に覚えながら計算している
- 計算機は「次に何するか」を絶対に「レジスタ・メモリに書いてあること」からしか決められない
- ⇒ ルール：「脳内記憶」使用禁止 (覚えておきたいことは紙に書く)
- 途中まで掛け算したところで記憶が飛んでも続きができなくてはいけない

(飛躍するが)
他にも「できそう」なこと

- 要するに「どうすれば (いかなる入力に対しても) 常に答えが求まるか」が分かっているような問題は , それを「プログラム」に翻訳する事は原理的にはできそう
  - 連立一次方程式をとく (ガウスの消去法)
  - ベクトルの直交化 (シュミットの直交化)
  - この盤面は「詰み」ですか (将棋)?
  - この手牌はテンパってますか？