2015.04.08

情報科学 I

(ねらい)

- 計算機の動作原理を理解する.
- プログラミングの基礎を習得する.
- その他

プログラミング → 言語

- 高級言語 …… C, Java 等
- 低級(?)言語 …… 機械語 あるいは アセンブリ言語 (記号言語)

参

- 線形 …… 比例
- 非線形

2015.04.08

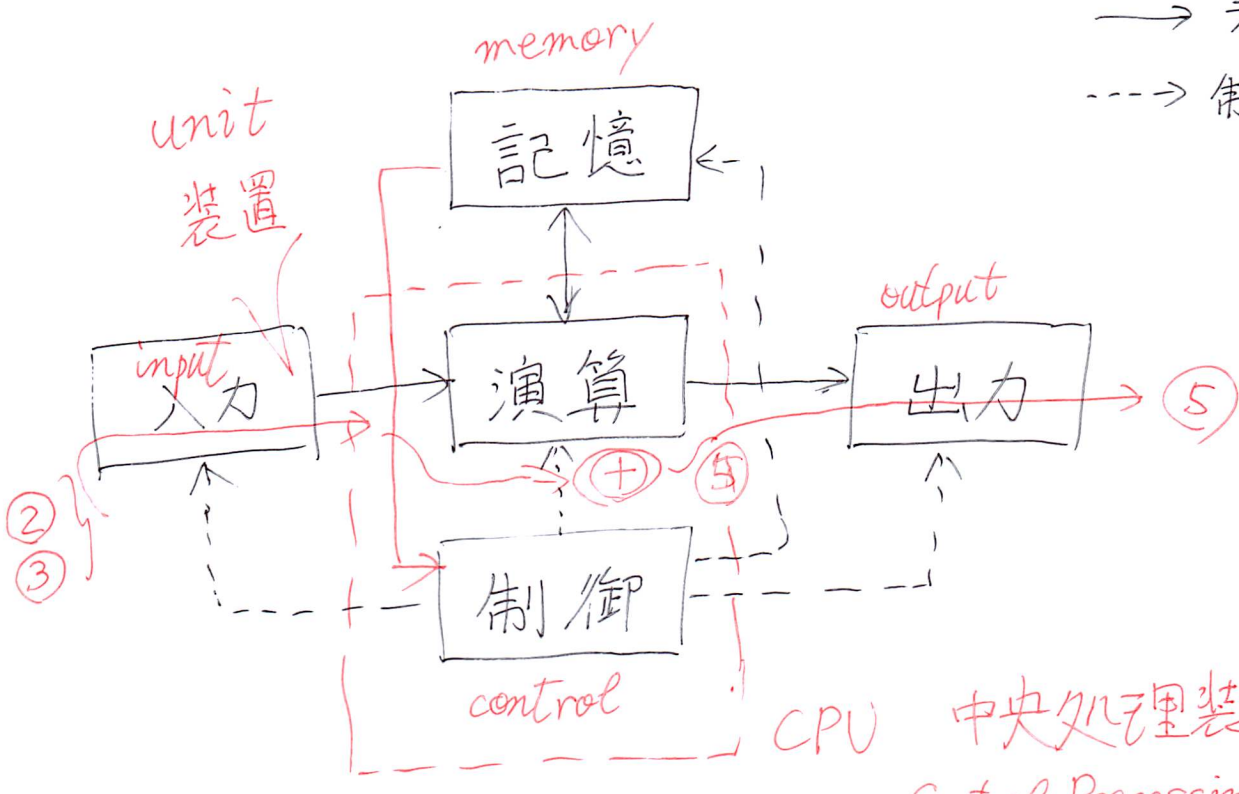
電子

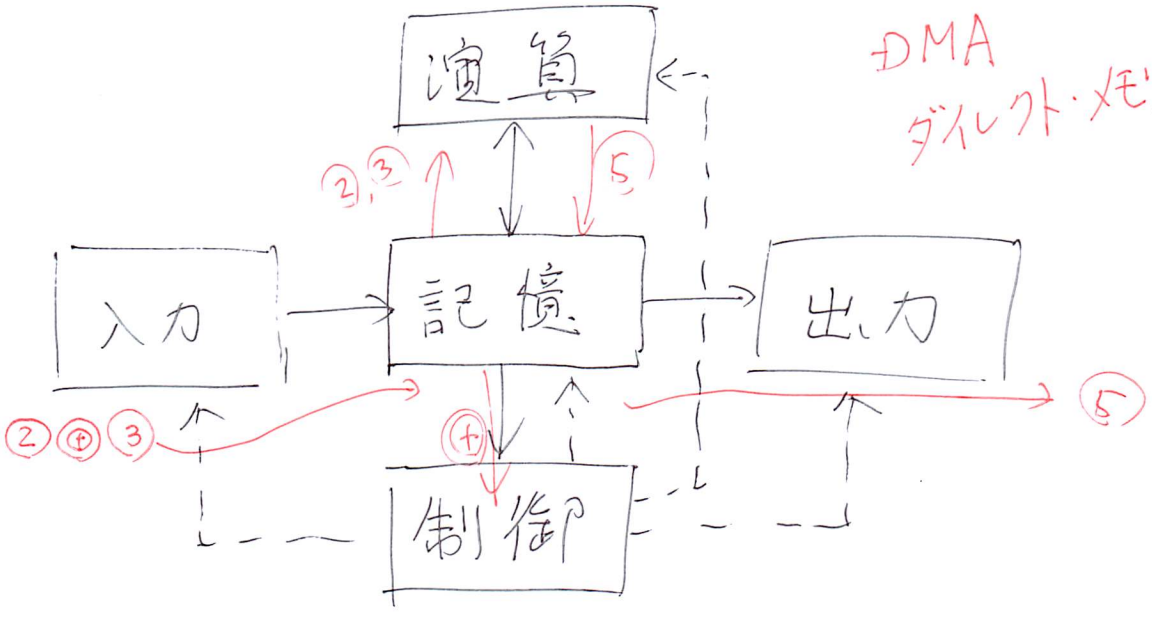
計算機の基本構成

(⟷ 双方向の流れ)

⟶ データの流れ

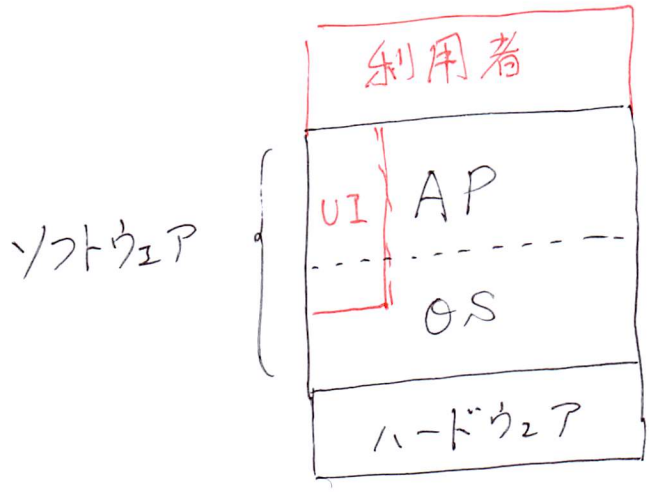
⇢ 制御の流れ (信号)





2015.04.08

ハードウェアとソフトウェア



ユーザー ／ word / excel

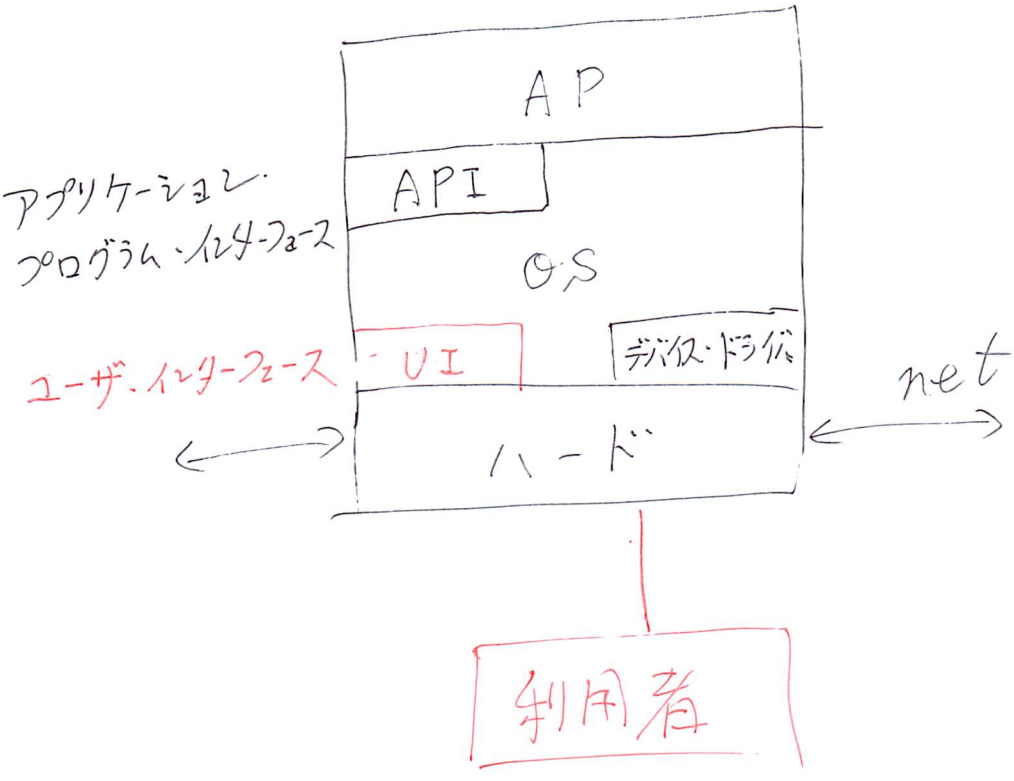
アプリケーション・プログラム

オペレーティング・システム

Windows 7 8.1 10

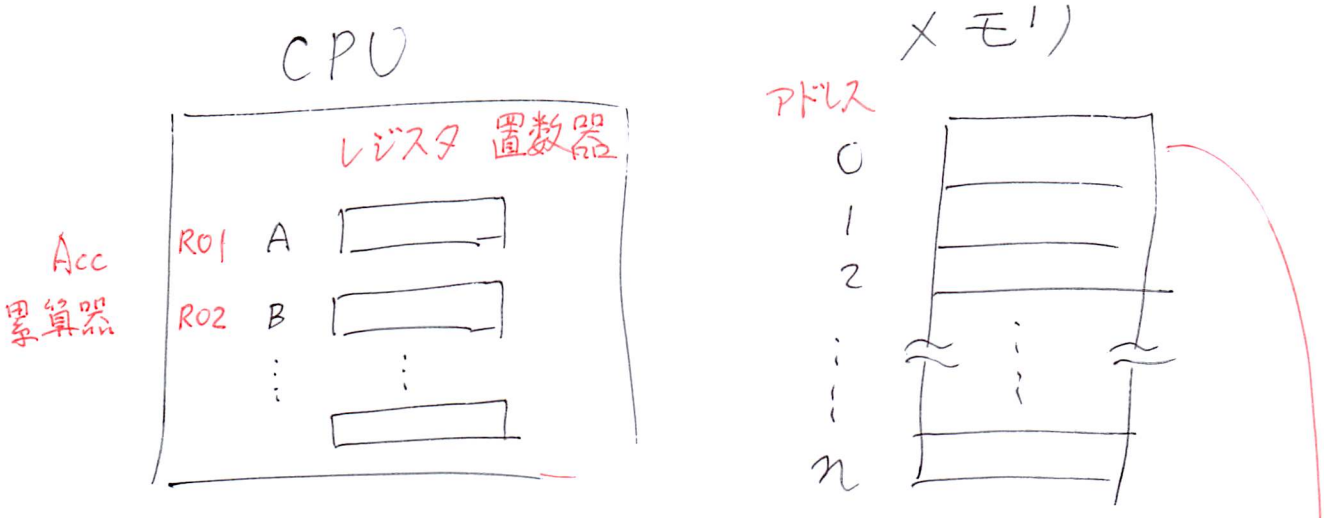
Linux Unix

OSX

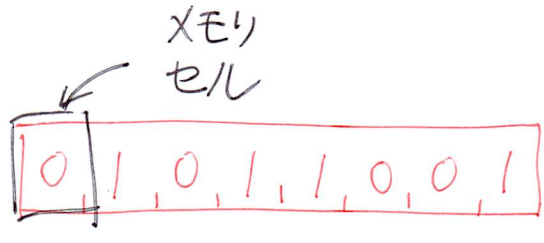


2015.04.08

- データ表現
- データの在存場所 --- メモリ, CPU



	CPU	メモリ
読書の速さ	速	遅
容量	小	大
コスト	高	低



データは, 0 または 1 (の組合せ)で表現される)

参 機械語のプログラムも ----。

情報の単位 bit (ビット)

binary digit

2進の 指

2015.04.10

計算機内部の情報は すべて

0と1の組合せ(並び)で表現される.

- 数
- 文字
- 論理(正しい, 誤り)
- 図形
- 音声
- 動画, 写真

情報交換が可能なように, ルールを決める。

'A'という文字 --- 01000001

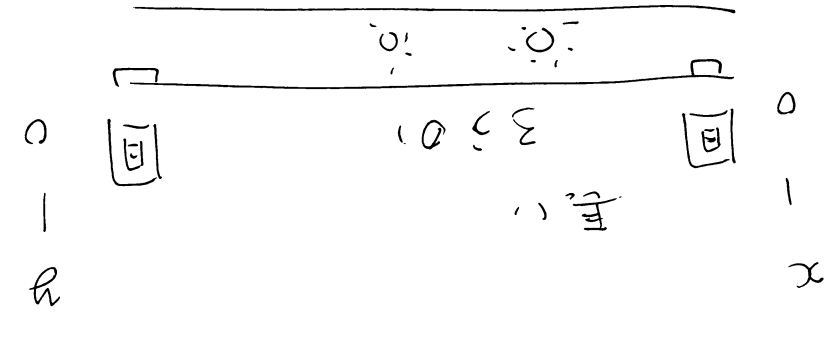
文字コード

2015.04.10

2種類の区別できるもの

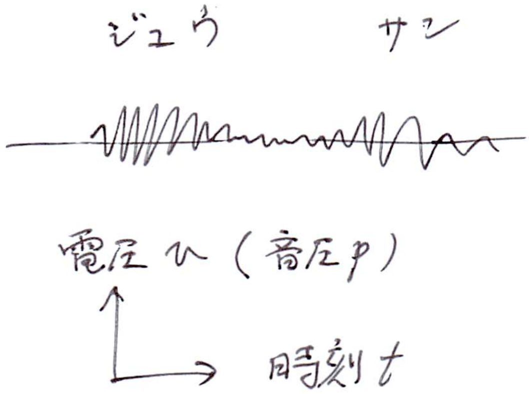
0	1	数または符号（記号）
○	×	
H	L	
High	Low	
高い	低い	
高い電圧	低い電圧	
5V	0V	⟶ 1.4V 以上 / 以下
ON	OFF	
電流が流れる	電流が流れない	

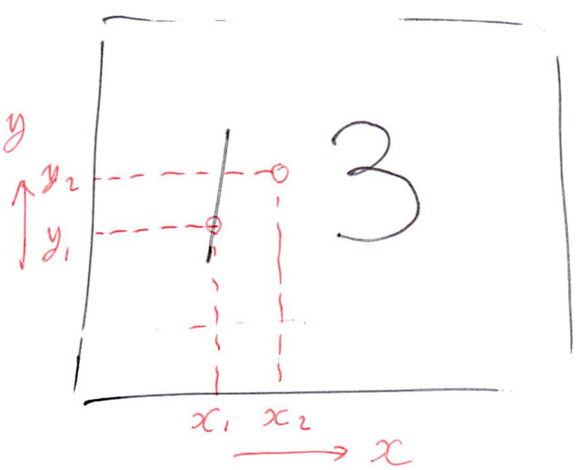
TTL ロジック



2015.04.15

音をマイクで電気信号へ変換して、オシロスコープで見る





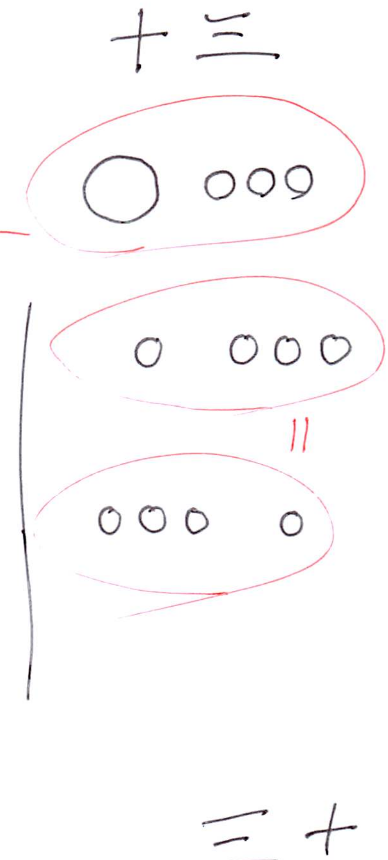
インクの濃度 $d(x, y)$

位取り

13

バラが 3コ

10のかたまりが 1コ



31

13 ≠ 31

103

2015.04.15

ローマ数字

I --- 1
V --- 5
X --- 10
L --- 50
C --- 100

II, Ⅱ --- 2

Ⅲ

IV --- 4 = 5 - 1
VI --- 6 = 5 + 1

XC --- 90 = 100 - 10

漢数字

一, 二, 三, ⋯, 九
十
百
千
万
億
兆
京

十 一

$$44100 = 441 \times 100$$
$$= 21^2 \times 10^2$$
$$= (3 \times 7)^2 \times (2 \times 5)^2$$
$$= 2^2 \cdot 3^2 \cdot 5^2 \cdot 7^2$$

2015.04.15

- 2進数 --- 2種類の符号を用いて表わされた数
- 十 〃 --- 10 〃
- n 〃 --- n

十進数で用いる符号

0, 1, 2, 3, 4, 5, 6, 7, 8, 9

2進数 〃

0, 1

数の表現の例

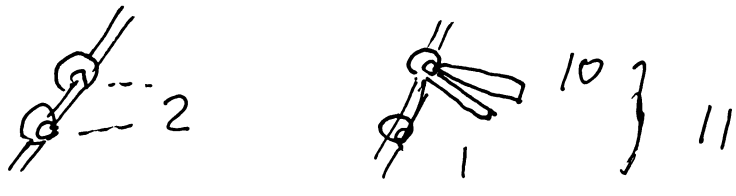
- 牛の数 --- 小石の数　　対応付け（一対一対応）

○○

- かたまり

きい
小石 10個 --- 大石 1個

- なわの結び目



- 岩はだへきざむ

2015.04.15

情報の表現

数の表現

2進数（n進数）　基数が 2　（n）
10

- 情報交換用符号と人類の「ことば」の歴史
- 「書きことば」と「話しことば」
  - 空間の関数，時間の関数
- 位取り記法　（発見 --- 「0の発見」）
- 情報処理における性能
  - コスト，変換の容易さ

2進 ＋進変換 --- 2進数表現を＋進数表現へ変換
＋進2進変換

(例)　2進表記の
$(1010)_2$ を　＋進表記へ変換する

各桁の重み（加重）を考えながら加算する.

1 0 1 0
8 4 2 1 ---- 加重
2^3 2^2 2^1 2^0

基数　指数

1 × 3

2018.04.15

$(1010)_2 = 1\times2^3+(0\times2^2)+1\times2^1+(0\times2^0)$

$= 1\times8+(0\times4)+1\times2+(0\times1)$

$= 8+(0)+2+(0)$

$= 10$

$(1010)_2 = (1\times)2^3+(1\times)2^1$

$= 8+2$

$= 10$

問. 二進表記を十進表記へ変換せよ

① 16 8 4 2 1
10010 = 16 + 2 = 18

② 16 8 4 2 1 / 32 16 8 4 2 1
11111 = 100000 − 1 = 32 − 1 = 31
2^5-1

③ 10110 = 32 + 8 + 4 + 1 = 45
16 4 1
32 8 2

2048 − 33
100001

11111011111

2015.04.17

十進 2進 変換

(原理)　　n進数 k桁の数を考える

$$x = a_k n^k + a_{k-1} n^{k-1} + \cdots + a_2 n^2 + a_1 n^1 + a_0 n^0 \quad (n^0 = 1)$$

$$\therefore \quad x/n = a_k n^{k-1} + a_{k-1} n^{k-2} + \cdots + a_2 n^1 + a_1 n^0 \quad \cdots \text{余り: } a_0$$

($n^0 = 1$)

さらに，nで割る → これを（商を）

$$x/n^2 = a_k n^{k-2} + \quad \cdots \quad + a_2 n^0 \cdots \text{余り: } a_1$$

続ける

$\cdots$ 余り: a_2

$\vdots$

a_k $\cdots$ 余り a_{k-1}

0 $\cdots$ 余り: a_k

「商が0」になるまで繰り返す．

$$\begin{array}{r} 104 \cdots 0 \\ 2\overline{)\,208} \cdots 1 \\ 2\overline{)\,417} \end{array}$$

私の身長は百六十cmです．

#　160cmです．

2015.04.16(木)

(例)　417 を　2進表記へ変換せよ.

417
208 --- 1
104 --- 0
52 --- 0
26 --- 0
13 --- 0
6 --- 1
3 --- 0
1 --- 1
0 --- 1

最下位

最上位

商が0になるまで

$(110100001)_2$
256 64 16 4 1
128 32 8 2

$= 256 + 128 + 32 + 1$
$= 417$

8ヶ
16進表記

問. つぎの十進表記された数を二進表記へ変換せよ。

① 99 $= 64 + 35 = 64 + 32 + 3$
$= 2^6 + 2^5 + 2^1 + 2^0$
$= 0(1100011)_2$
(6 3)$_{16}$

② 2015 $= (011111011111)_2$
(7 d f)$_{16}$

③ 511 $= 512 - 1$
$= 2^9 - 1$
$= (1000000000)_2 - 1$ (9コ)
$= (111111111)_2$ (9コ)
$= (1\ 1111\ 1111)_2$

16進表記 $= (1FF)_{16}$

2015.04.17

十進	2進	8進	16進
99	1 4 3 00 110 011	143	63
2015	0 111 111 011 111	3737	7DF
511	111 111 111	777	1FF

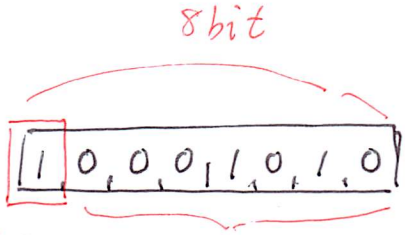
2015.04.22

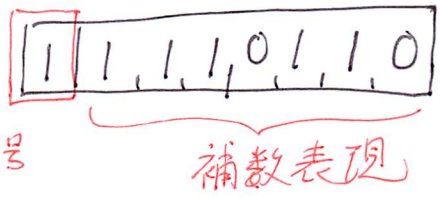
負数の表現　(2通りを紹介)

(ア) 符号と絶対値

(イ) 符号と補数(表現)

(例)　−10





(参) $(1110110)_2$

$= 64+32+16+4+2$

$= 118$

118は，10の補数

では「何について」のか？

$128 = 2^7$

補数とは，

「yは，xの(nについての)補数である」

(例)「3は，7の(10　〃　)補数　〃　」

$$3 = 10 - 7$$

あるいは

$$7 + 3 = 10$$

$$x + y = n$$

←「きりの良い数」が適当

＋整数の場合は，10とか100

(別の例)

78の補数は？　---　答. 22

2015.04.22

「応用のことを考えて決めた」らしい

(元) 正数の加算

(例) 減算の実現」--- ハードウェアの実装

(具体的な例)

```
  23
- 10
----
  13
```

.....--- 減算をできる

(例) 負数を含む加算 --- 負数も扱える

```
  23
+)-10
----
  13
```

$$23 + (-10) = 23 - 10$$

負数の加算　　正数(のみ)の減算

実装

2015.04.22

```
  1 1   ← 桁上がり
    1 1
+)  0 1
-------
  1 0 0
```

```
    1
   78
+  13
-----
   91
```

(例)

23 + (−10)

= $(10111)_2$ + (?)$_2$

=

```
   23
+) -10
-------
  -33 ?
```

```
  0 0 0 1 0 1 1 1
+ 1 0 0 0 1 0 1 0   絶
-----------------
  1 0 1 0 0 0 0 1   ?
```

```
   23
+) -10
-------
   13
```

```
  0 0 0 1 0 1 1 1
  1 1 1 1 0 1 1 0   補
-----------------
1 0 0 0 0 1 1 0 1
↓
捨てる
```

2進数における補数（負数）

「きりの良い数」 --- 2のべき乗 --- $2^{8-1} = 128$

```
  128
-) 10
------
  118
```

```
  1 0 0 0 0 0 0 0
- 0 0 0 0 1 0 1 0
---------------
  1 1 1 1 0 1 1 0
  ↑   7bit
符号(−)
```

2015.04.22

1補数と2補数

参 9補数と(10)補数　代表

99　　100
999　　1000
⋮　　⋮

「きりの良い数」の選び方の問題

10補数　9補数

10^n or 10^n-1

2^n or 2^n-1

2補数　1補数

$$\begin{array}{r} 99 \\ -)\ 23 \\ \hline 76 \end{array}$$

$$\begin{array}{r} 100 \\ -)\ 23 \\ -)\ 1 \\ \hline 77 \end{array}$$

$$\begin{array}{r} \overset{2}{\cancel{3}}\overset{1}{\cancel{0}} \\ -)\ 23 \\ \hline 68 \end{array}$$

2015.04.24

ⓐ 23の99についての補数をもとめる。

$a = 99 - 23 = 76$

ⓑ 23の100についての補数をbとすると、

$b = 100 - 23 = 77$

$b = a + 1$　　9補数に「1を加える」と10補数になる

2015. 04. 24

2進表現のもとで,

$10 = (1010)_2$ の 2補数 x を求める.

その前に, 10 の 1補数 y を求めておくと,

$$x = y + 1$$

である.

~~+000~~

1111 1111

→ 0001010

$y =$ 1110101 ⤵ 反転 complement

$x = y+1 =$ 1110110 ⤵ +1 increment

1	1110110

↑ 符号 (−)

$= (F6)_{16}$

0xF6

0F6H

~~2補~~

問. 変換表の空欄を埋めよ。

ただし, 2進表現の負数は 2補数を用いよ。

また, 2進表現は符号を含めて 8bit とする。

+進数	2進数
−24	1 1 1 0 1 0 0 0
−64	1 1 0 0 0 0 0 0
−39	1 1 0 1 1 0 0 1
−128	1 0 0 0 0 0 0 0

符号

$24 = (0001\ 1000)_2$

1110 0111

1110 1000

000 0000

111 1111

1 0000000

= 128

2015. 04. 24

元に戻すには、

$10 = (00001010)_2$

反転 ↓ 1111 0101 ↑ 反転

+1 ↓ 1111 0110 ↑ −1

同じ操作

反転 ↓ 0000 1001

+1 ↓ 00001010

最大と最小

基数 base

255	+127	1111 1111
254		1111 1110
⋮	⋮	⋮
	0	1000 0000
	−1	0111 1111
2	⋮	0000 0010
1		0000 0001
0	−128	0000 0000

(2^8) 256通りの数

$= (1\,0000\,0000)_2 - 1$

$= 2^8 - 1$

$= 256 - 1$

$= 255$

符号

+127	0	111 1111
+126	0	111 1110
⋮		⋮
+1	0	000 0001
+0	0	000 0000
−1	1	111 1111
−2	1	111 1110
⋮		⋮
−127	1	000 0001
−128	1	000 0000

256通りの数

$2^7 - 1 = 128 - 1$

$= 127$

2015.04.24

文字の表現 --- 文字コード

~~ASCCII~~ ASCII Code

A	0100 0001	a	0110 0001
B	0100 0010	b	0110 0010
C	0100 0011	c	
⋮			
O	0100 1111	o	
P	0101 0000	p	
Q	0101 0001	q	
⋮		⋮	
Z	0101 1010	z	

0	0011 0000
1	0011 0001
2	0011 0010
⋮	
9	0011 1001

$2^{10} = 1024 \quad (= 1K)$

$\fallingdotseq 1000 \quad (= 1k)$

$2^{30} = 1G = (2^{10})^3 = (1K)^3$

$= 1,073,741,824$

$2^{40} = 1,099,5\cdots$

$1T = (1K)^4$

補助単位

$k = 10^3$

$M = 10^6$

$G = 10^9$

$T = 10^{12}$

1 BYTE = 8 bit

2018.05.01

制御のしくみ

○ プログラム内蔵方式 ⟷ ワイヤード方式
(配線)
Stored Program

⇓

自身で制御プログラムを書き変えることが可能

コンパイラの技術

高級言語のプログラムを機械語へ変換する

⇓ それでも,

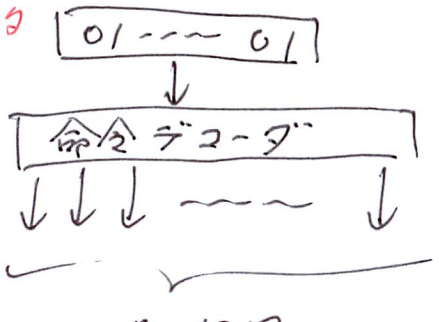
別のアイデア ---- ハーバード・アーキテクチャ

データ用メモリ
プログラム用メモリ
) 別々に実装

プログラム（機械語）

命令レジスタ

マイクロプログラム方式



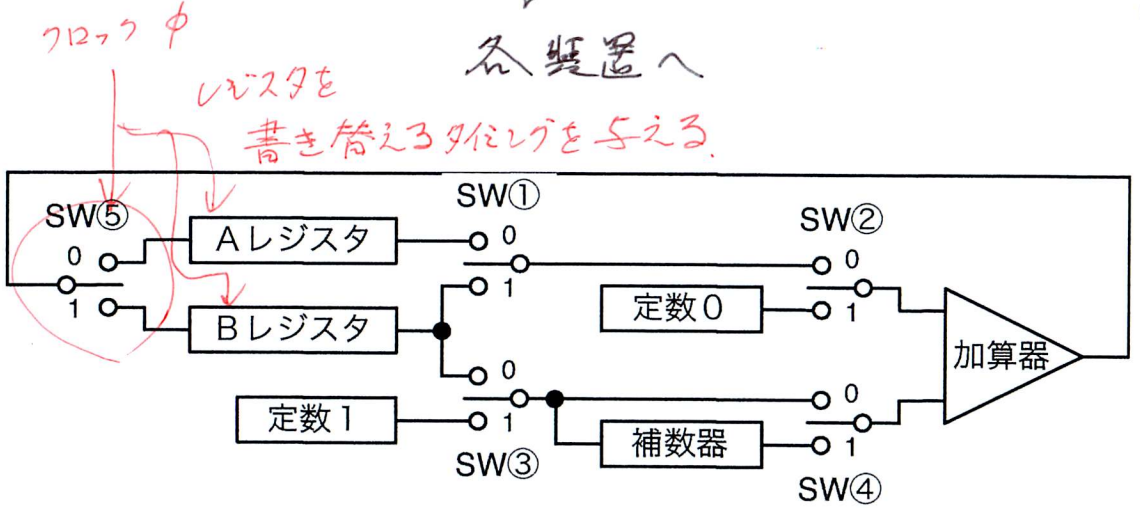
機械語命令 → 制御信号

変換

（解読 decode）

⇩

encode



x : don't care

スイッチ

①	②	③	④	⑤	
0	0	0	0	0	Aレジ ← Bレジ　　x1000　（ここに，xは0でも1でも良い）
0	0	0	1	0	Aレジ ← Aレジ + Bレジ　　加算
0	0	1	0	0	Aレジ ← Aレジ − Bレジ　　減算
?					Aレジ ← Aレジ + 1
?					Bレジ ← Aレジ
1	0	0	1	0	Bレジ ← − Aレジ
					Aレジ ← 0　（Bレジ − Bレジ）

Bレジ ← 0　　10011

Bレジ ← Aレジ + Bレジ　　00001

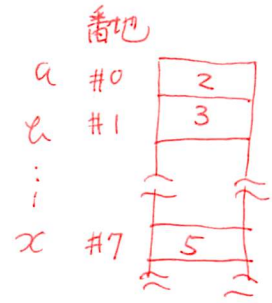
Bレジ ← − Bレジ　　x1011

2015.05.08.(金)

○ メモリ、アクセス（読み書き）

- データは メモリに 置く
- レジスタは 一時的な 置き場所（作業用）

$x = a + b$



○ マシン・サイクル ― （処理）計算の実行には、有限の時間がかかる。

- 電子回路における信号伝達遅れ
- 同期回路と非同期回路

  同期信号（クロックともいう）を用いるか用いないか。

- フェッチ・サイクルと実行サイクル

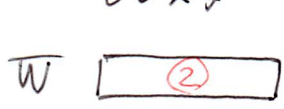
○ 命令の種類（命令セット）

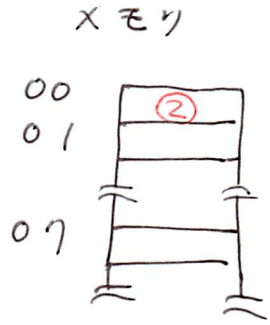
- 転送命令（コピー命令） MOVE（ムーブ）, LOAD（ロード）, STOR（ストア）
- 演算、
- 制御、
- その他

○ 条件分岐と繰り返し （高級言語）

2015.05.08

[方式1]



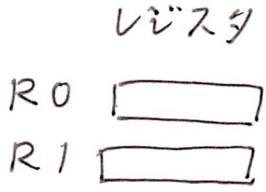


~~{~~

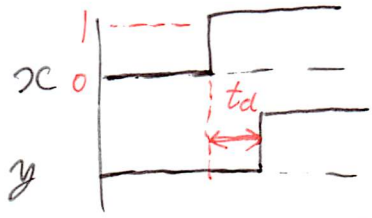
(#0)

{ Wレジ ← (00番地)
  Wレジ ← Wレジ + (01番地) --- 演算命令
(07番地) ← Wレジ

[方式2]

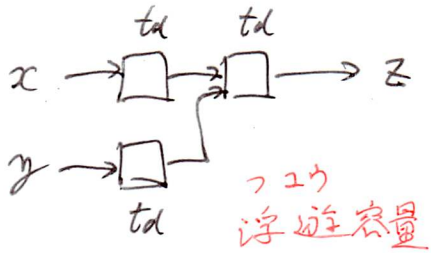


{ R0 ← (#0)
  R1 ← (#1)
  R0 ← R0 + R1 --- 演算命令
(#7) ← R0

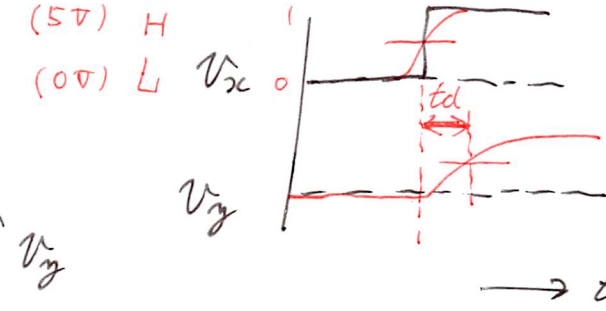


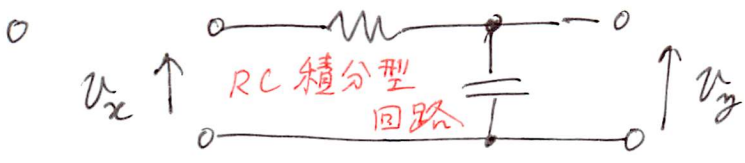
td：(伝達) 遅延時間

→ 時刻 t



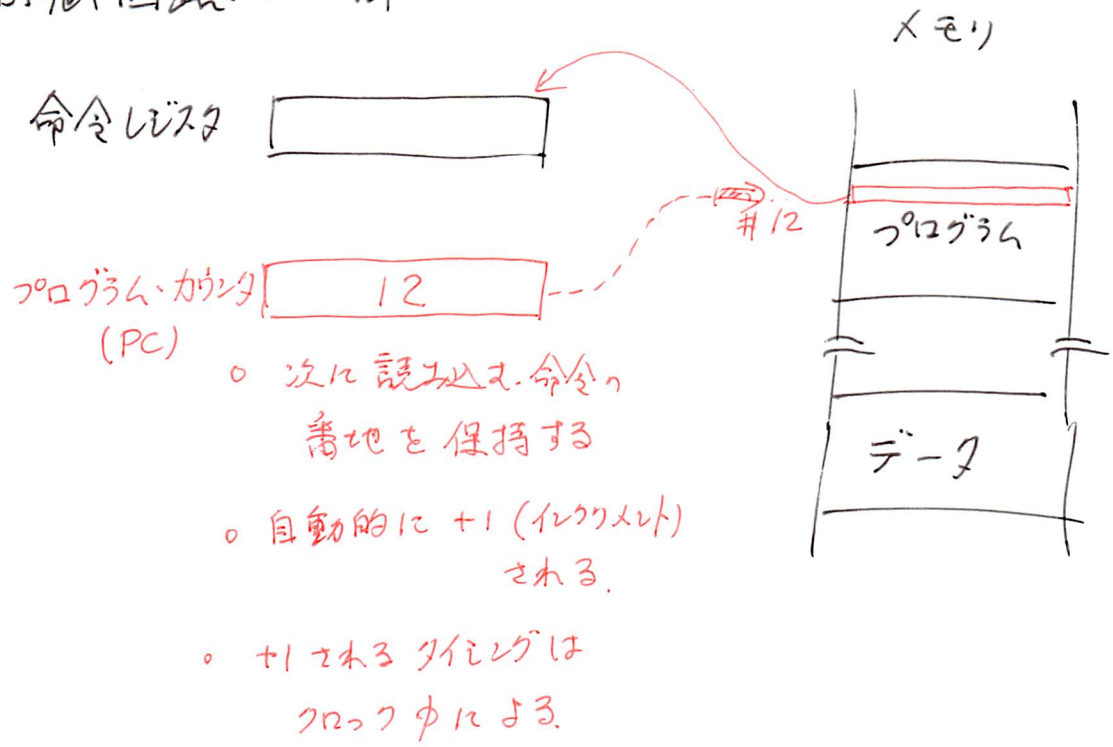
つユウ浮遊容量



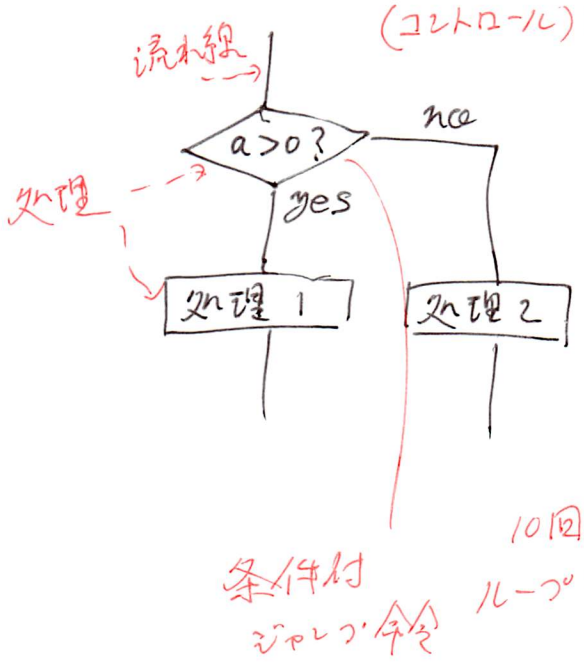


2015.05.08

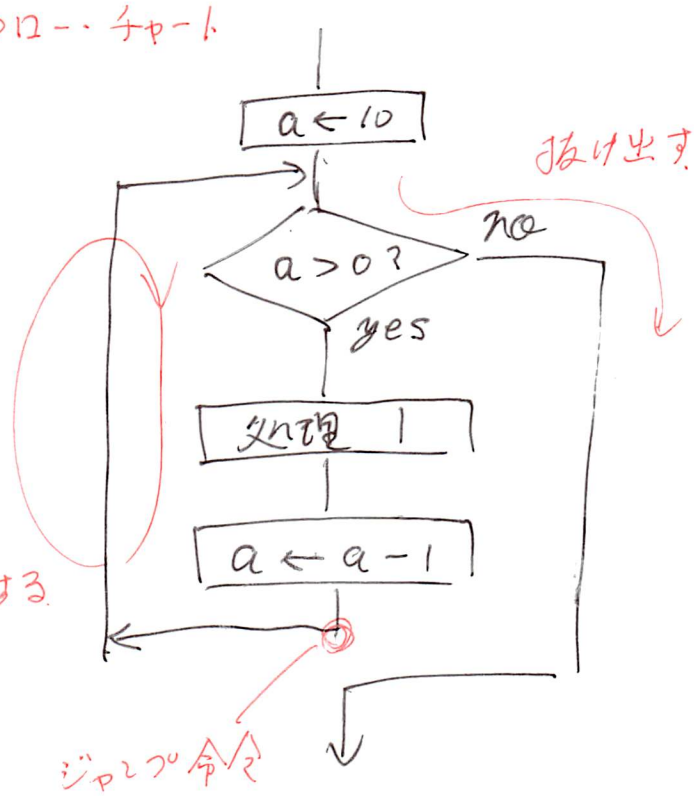
○ 制御回路の一部



条件分岐の例



繰り返しの例



2015.05.13

実際にプログラムを実行する.

- 開発環境を利用する.
  - 導入 と 設定
    インストール
- プログラミング言語の文法に従う.

Java , PIC のアセンブリ言語

$x = a + b$ ← 入力 / ← 出力

右辺の値を評価して
左辺の変数へ代入する.

等しい(状態)
⇕
操作

```
MOVF    A,W     ; w ← a
ADDWF   B       ; w ← w + b
MOVWF   X       ; x ← w
```

書式(フォーマット)

op-code

ラベル : 命令コード オペランド ; コメント
オペコード

オペランド: 複数可
- 引数
- 引き数

,(カンマ)で区切る

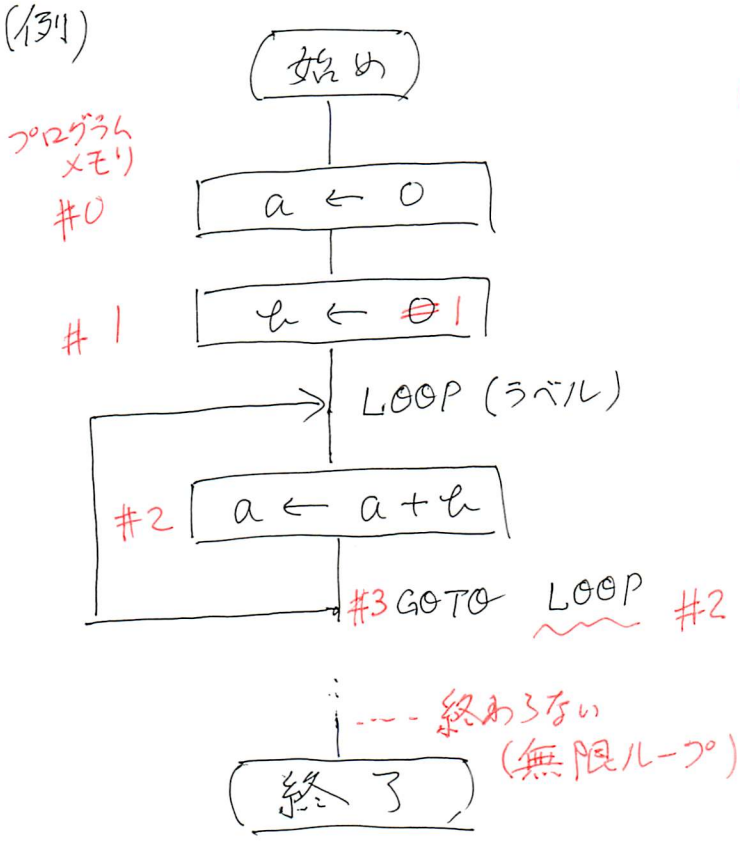
コメント: 説明, メモ書き

2015.05.20 (水)

繰り返しと制御命令

GOTO 命令

(例)

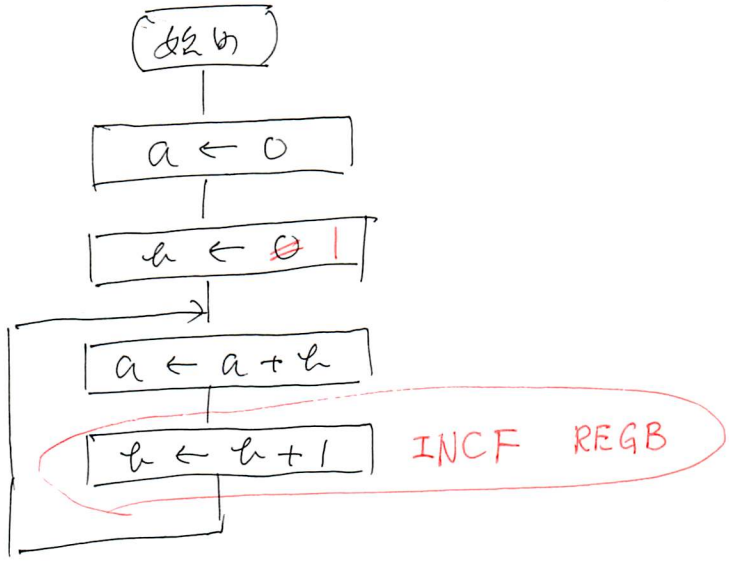



```
LOOP:
        MOVF    a, w    ; a←a+b
        ADDWF   b       ;  〃
~~LOOP~~
        MOVWF   a       ;  〃

        GOTO    LOOP

        END
```

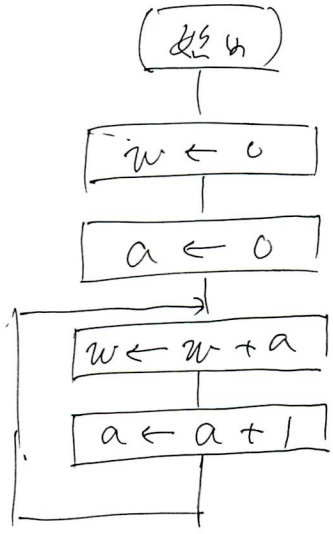
データメモリ (汎用ファイル・レジスタ)

a	#20	0
b	#21	1

(解答例)

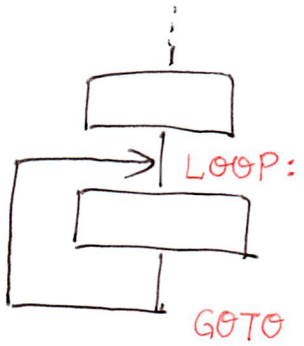


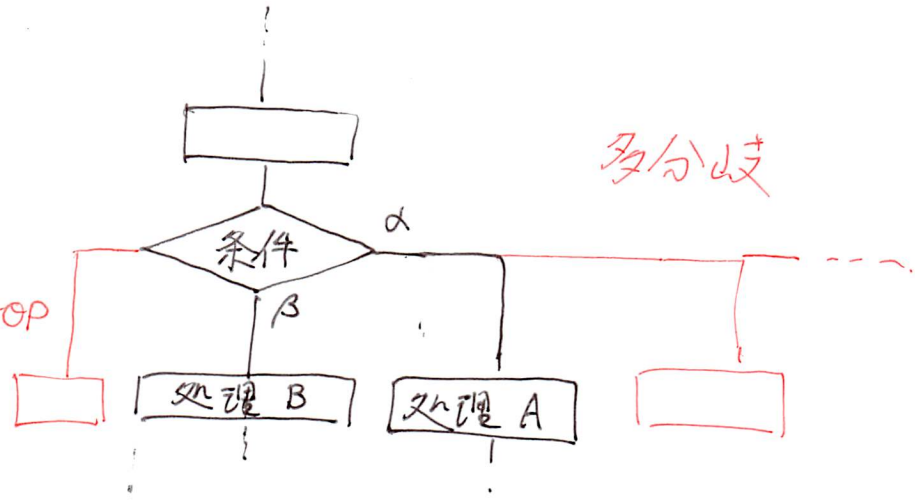
(別解)

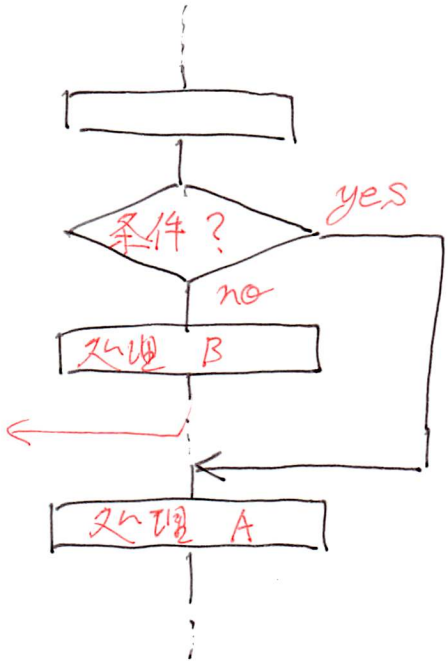


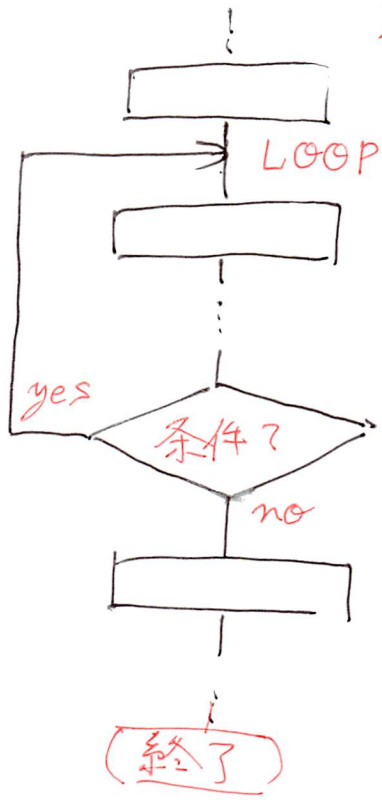
2015.05.22 (金

分岐と条件付分岐









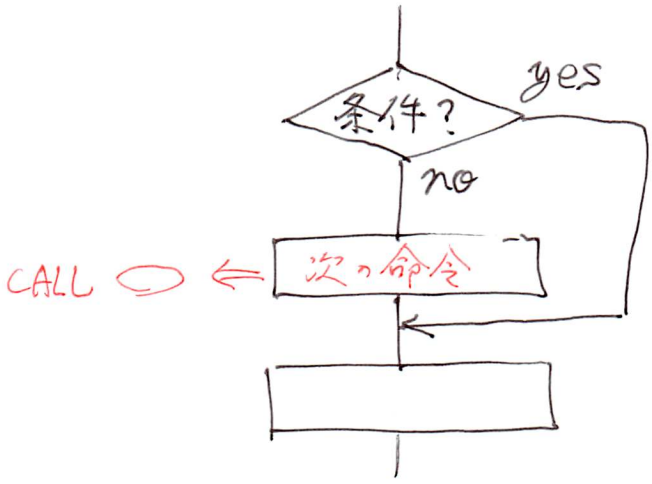
条件付分岐命令の例

```
JMP   Z, LOOP    ; 直前の命令の実行結果が0ならば，
                   ラベル LOOP へ ジャンプする．
                   （条件付ジャンプ命令）
JZ    LOOP

JMP   C, ---     キャリー（桁上げ）が発生したら，
JC    ---
```

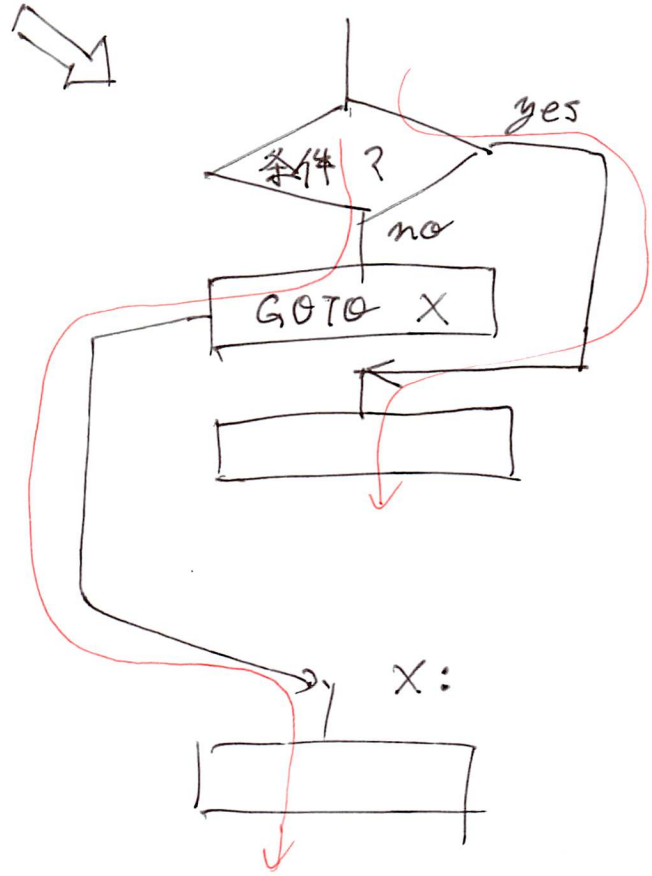
2015.05.22(金)

条件付スキップ命令 と GOTO (or JMP) 命令を組み合わす.

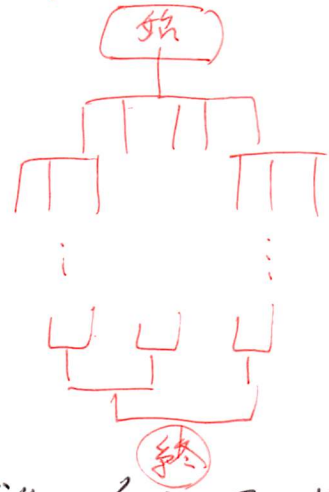
無条件分岐



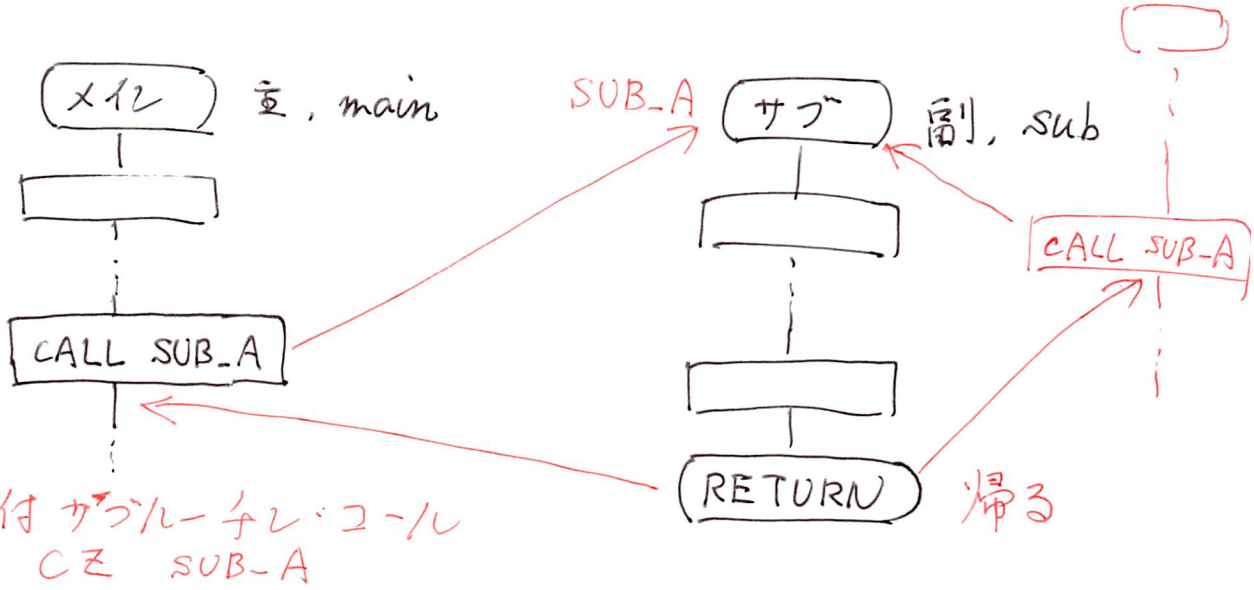
次の命令をスキップ(ひとつ飛ばし)する



分岐 ⟺ 合流



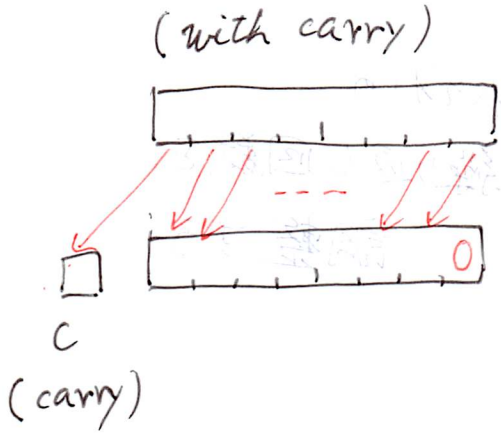
サブルーチン・コール命令



2015.06.12(金)

「ニーモニックの文字列にこだわらないこと.」

(例) 左シフト命令 が存在する



(解) 自分自身を加える ---- モデル計算機であった

$$\begin{cases} A \leftarrow A + A \\ A \leftarrow A \times 2 \\ A \leftarrow A\text{の左シフト} \end{cases}$$

ADD A,A

PICでは,2ステップで実現する

(例)

```
MOVF   AREG,W      ; W ← AREG
ADDWF  AREG,F      ; AREG ← AREG + W.
```

⑨ 6/16(火) 16:30— 阿部 追試

ソフトウェア・タイマ　　2015.06.24 (水)

- 待ち合わせ時間を作る

① 短かい場合

ムダな命令を実行する

NOP (non-operation)等を利用

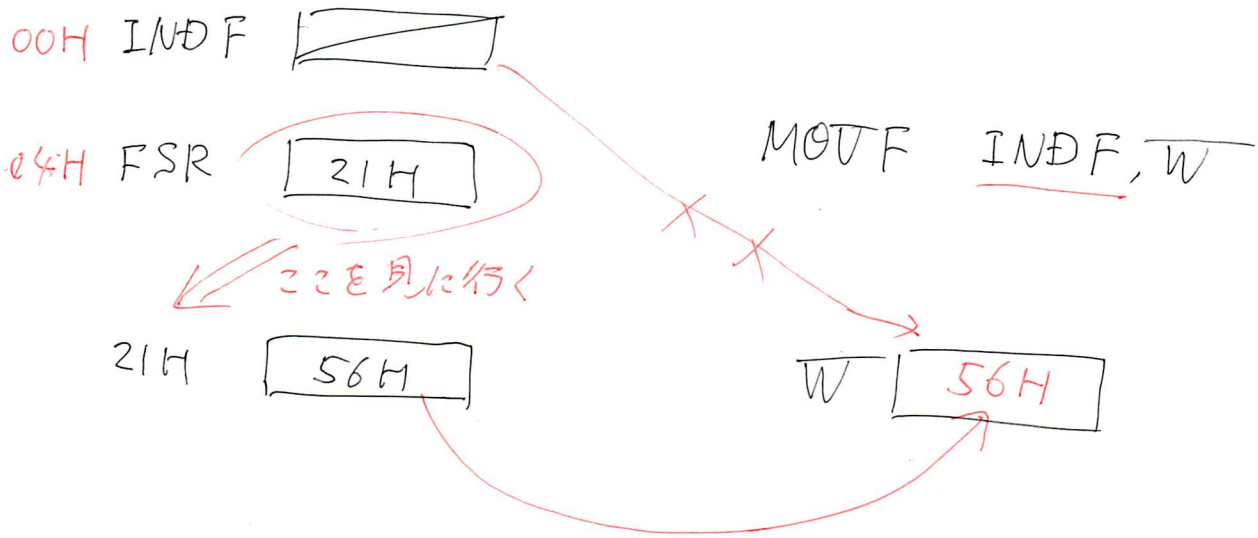
② 少し長い場合

- ループを利用する ---- プログラム・メモリは有限(数kB)
- call 命令を活用する ⟶ テキスト参照

③ 長い場合

- 多重ループを利用する

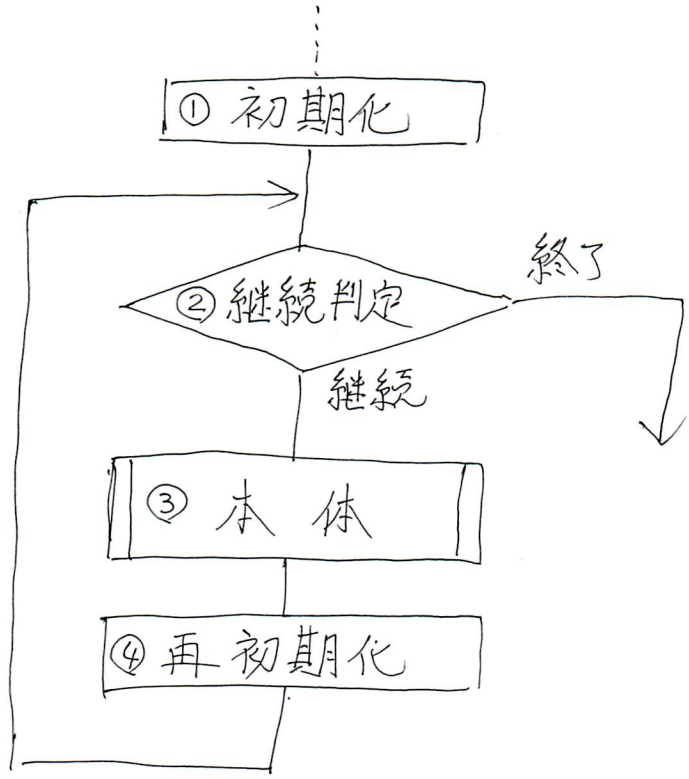
2重，3重



PICにおける (INDFとFSRを用いた)
間接アドレッシング.

2015.07.10 (金)

繰り返しの 4 要素



Java 言語では

for (①; ②; ④) ③

(例) 1～10の整数の和

```
int sum = 0;
int i;
for (i=1; i<=10; i=i+1)
    sum = sum + i; //③
    ⋮
```

⇒ System.out.println(sum);

参 System.out.println(55);

~~2014~~
2015.09.02 (水)

アルゴリズム

問題を解く手順(手続き)を記述したもの

(例) 12個の重りの問題

---- 12個の大きさの等しい重り(鉄球)がある。
この内、1個だけが重さが異なる。
(その前に、簡単な例) 「てんびんばかり」を用いて、これを探せ。

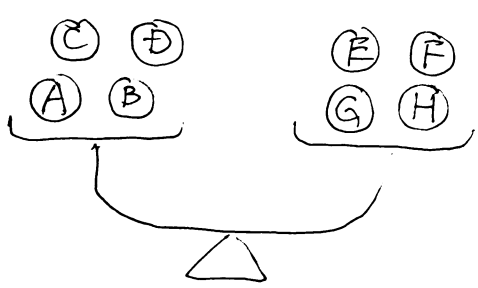
8個の重りの問題

-- 8個の大きさの重りがある。
この内、1個だけが他のものより重い。
(他の7個は重さが等しい。)
てんびんばかりを用いて、これを探せ。

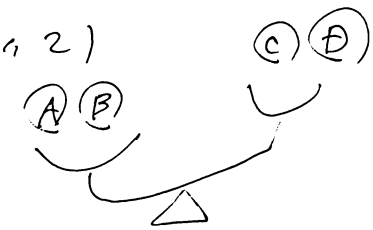
(解) —— 手順を記述する。

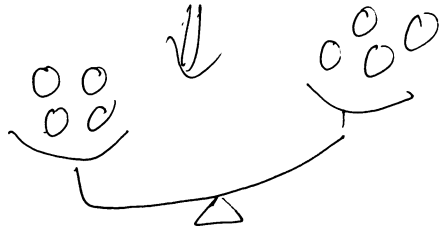
Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ Ⓕ Ⓖ Ⓗ

(手順1)

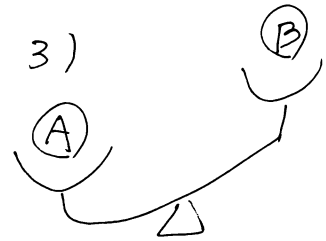


(〃2)



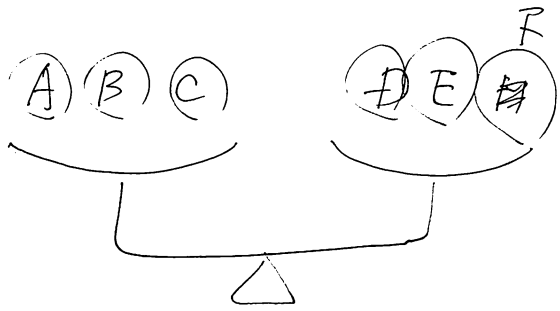


(〃3)



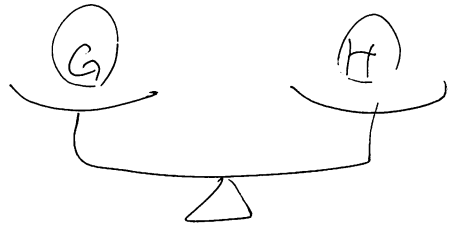
Ⓐが重い

手順①



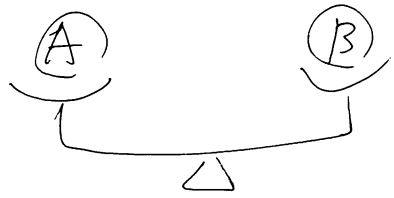
- つり合ったら　②へ
- 左が下がったら　②'へ
- 右が下がったら　(左右を取りかえて)②'へ

手順②



下がった方が重い

手順②'



- つり合ったら　C
- 左が下がったら　A
- 右　〃　B

アルゴリズムの検討　（理論的な）

(例)　8個の重りの問題の場合

{ 解の可能性は　8通り
{ てんびんの状態は　3つ

1回の操作（手順）で可能性を 1/3 にできる.

$3^2 = 9 > 8 \Rightarrow \log_3 8 < 2$

なので，2回の操作で，可能性を1つに
しぼり込むことができる。

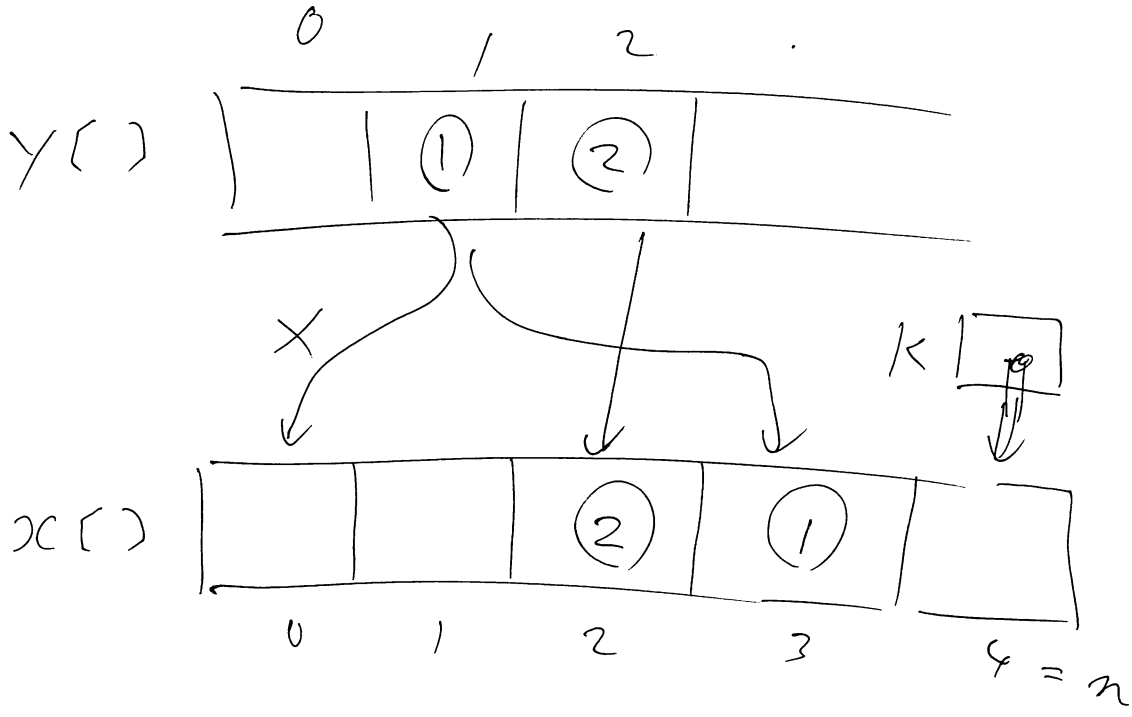
(例)　n個のデータの並べ換え（ソーティング）

{ n! 通りの可能性
{ 1回の操作（比較と交換）で 1/2 に できる

$\log_2 n!$ 回の操作で、解ける可能性がある.

$\Longrightarrow$ クイック・ソート 等

バブル・ソートでは　n^2 回の操作をする.


0
1
2
y[ ]
①
②
x
k
x[ ]
②
①
0
1
2
3
4 = n