オープンキャンパス（高校生向け）
平成 23 年 8 月 7 日 (日), 8 日 (月)11:00-11:50
103 講義室

## 通常とは別の処理方式紹介：

# ニューロ計算，進化的計算

情報工学科 元木達也

# 1 概要

## 通常のコンピュータ：

動作原理 ··· プログラム内蔵方式 (ノイマン型)

動作指示 ··· 数学的／論理的に 処理の手順 (プログラム) を与える

↑

通常、気まぐれな動きは無い。

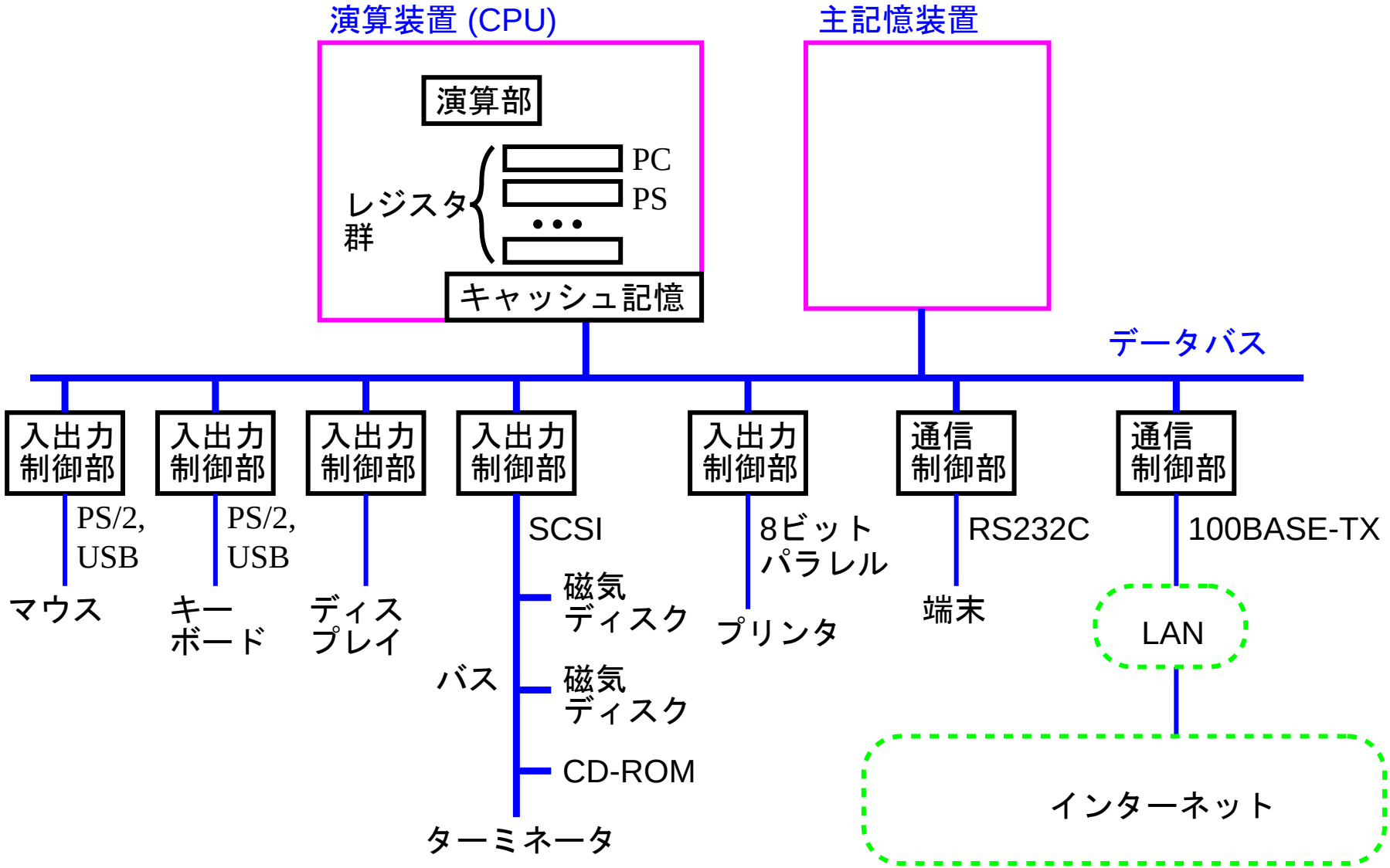
## 他の可能性：

- ニューロ計算 ··· 脳の情報処理をお手本にする。
- DNA計算 ···
- 進化的計算 ··· 生物の進化をお手本に、試行錯誤的に探索。
- 群知能 ··· 群をなす鳥、蟻等の社会的な振舞いを参考にする。
- .....

⇨ 以下、プログラム内蔵方式、通常の処理手順の与え方について復習した後で、ニューロ計算, 進化的計算について簡単に紹介する。

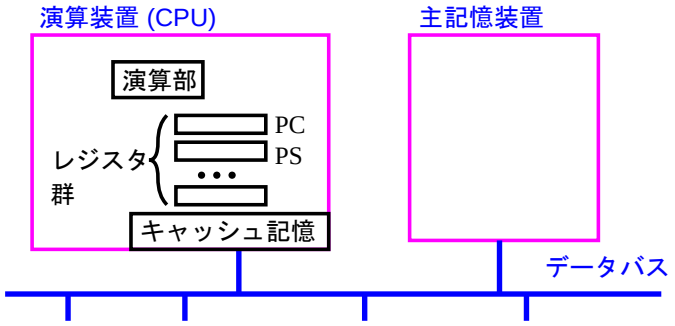
# ② プログラム内蔵方式

## 通常のコンピュータのハードウェア構成



## プログラム内蔵方式

現在の普通の計算機においては、
プログラム内蔵方式を採用。

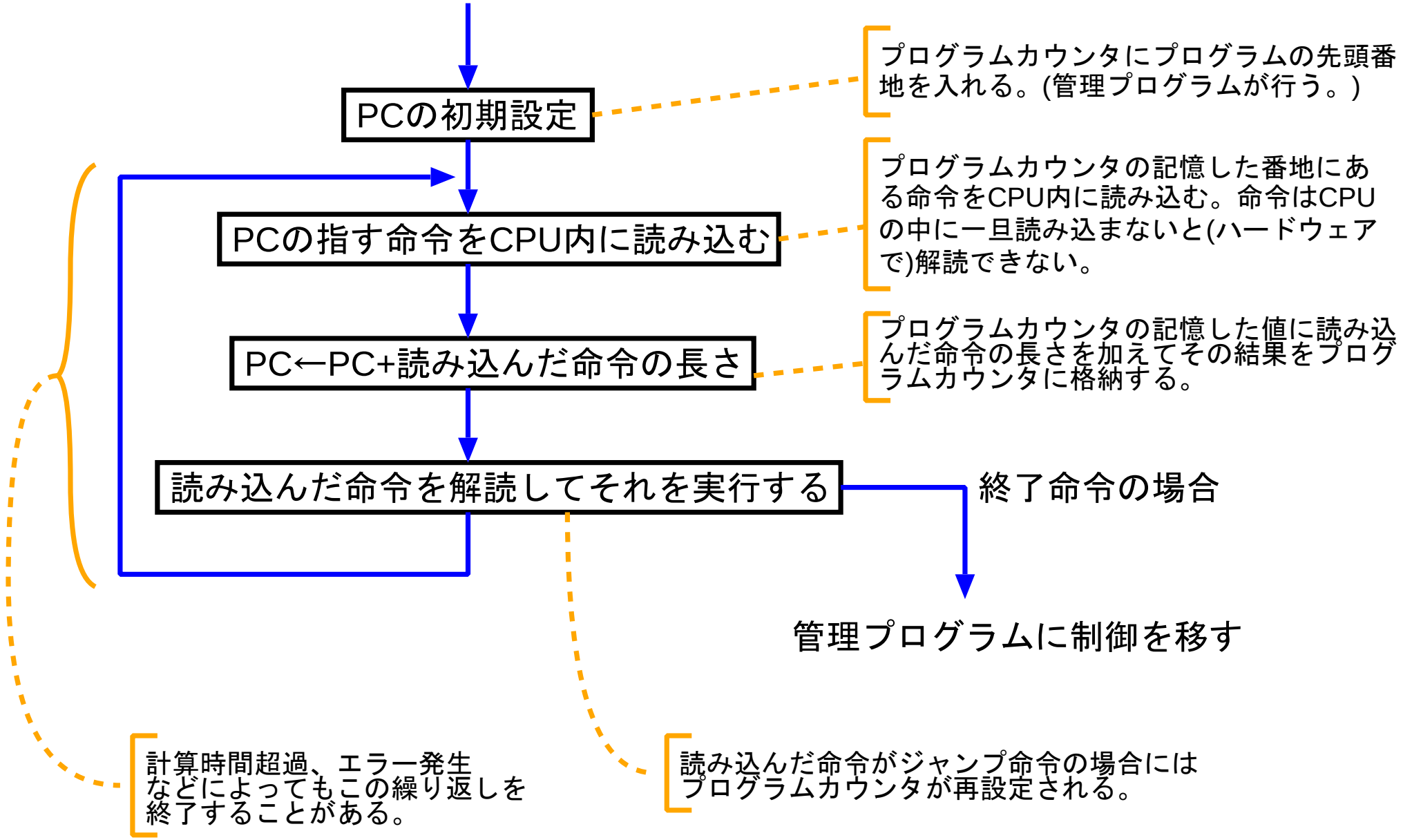



- プログラム (i.e. 処理手順) をデータと同様に記憶装置内に格納し、(プログラムをデータとして加工できる)
- プログラムを構成する機械語命令を逐次的に読み出しては実行していくことにより自動的に動作させる、

プログラム内蔵方式の計算機では 、

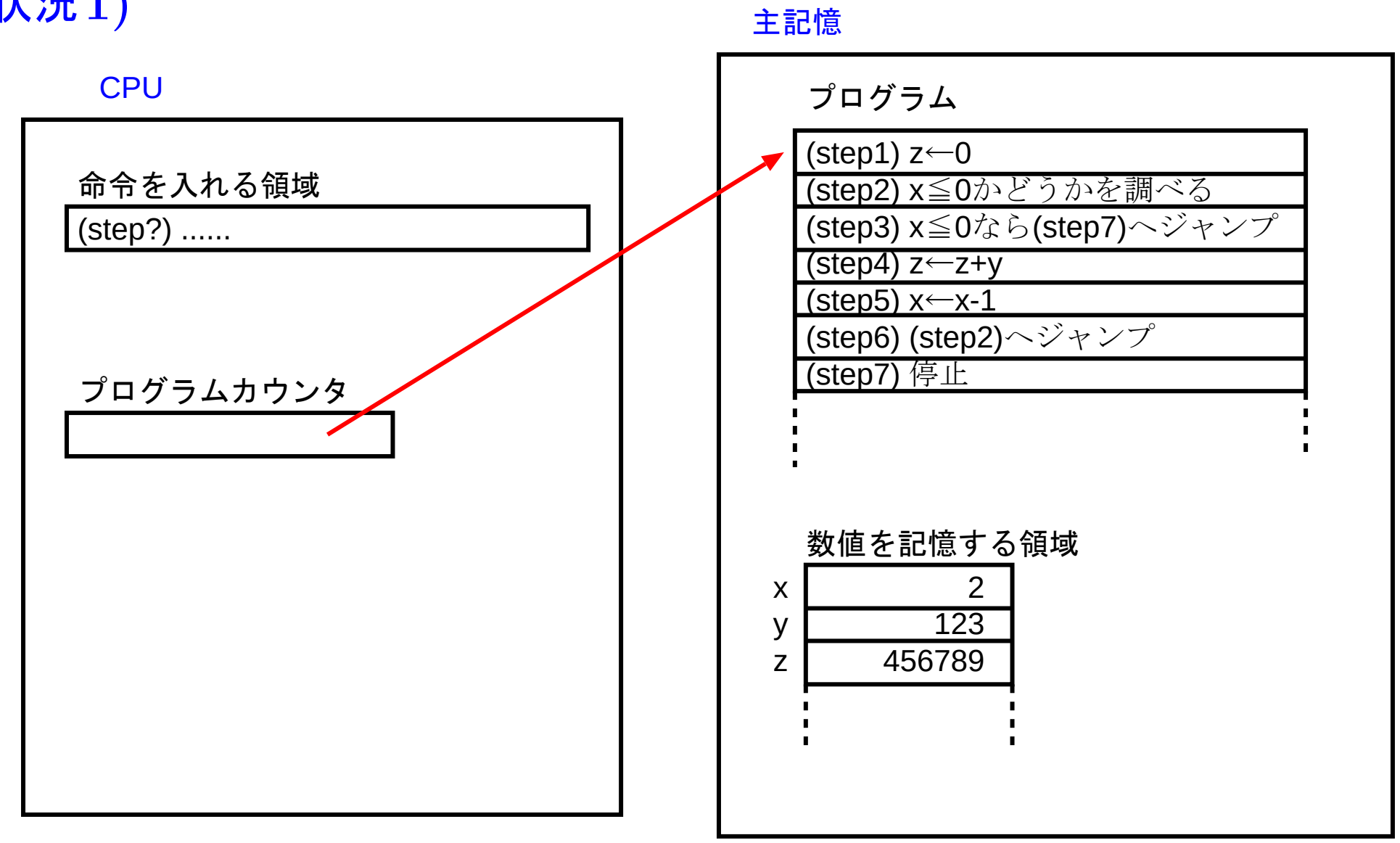
- プログラムカウンタと呼ばれる
  次に実行する命令の入った (主記憶上の) 番地
  を常に保持するレジスタ記憶を用意する。
- CPUはプログラムカウンタを用いてプログラム中の命令を逐次実行する。

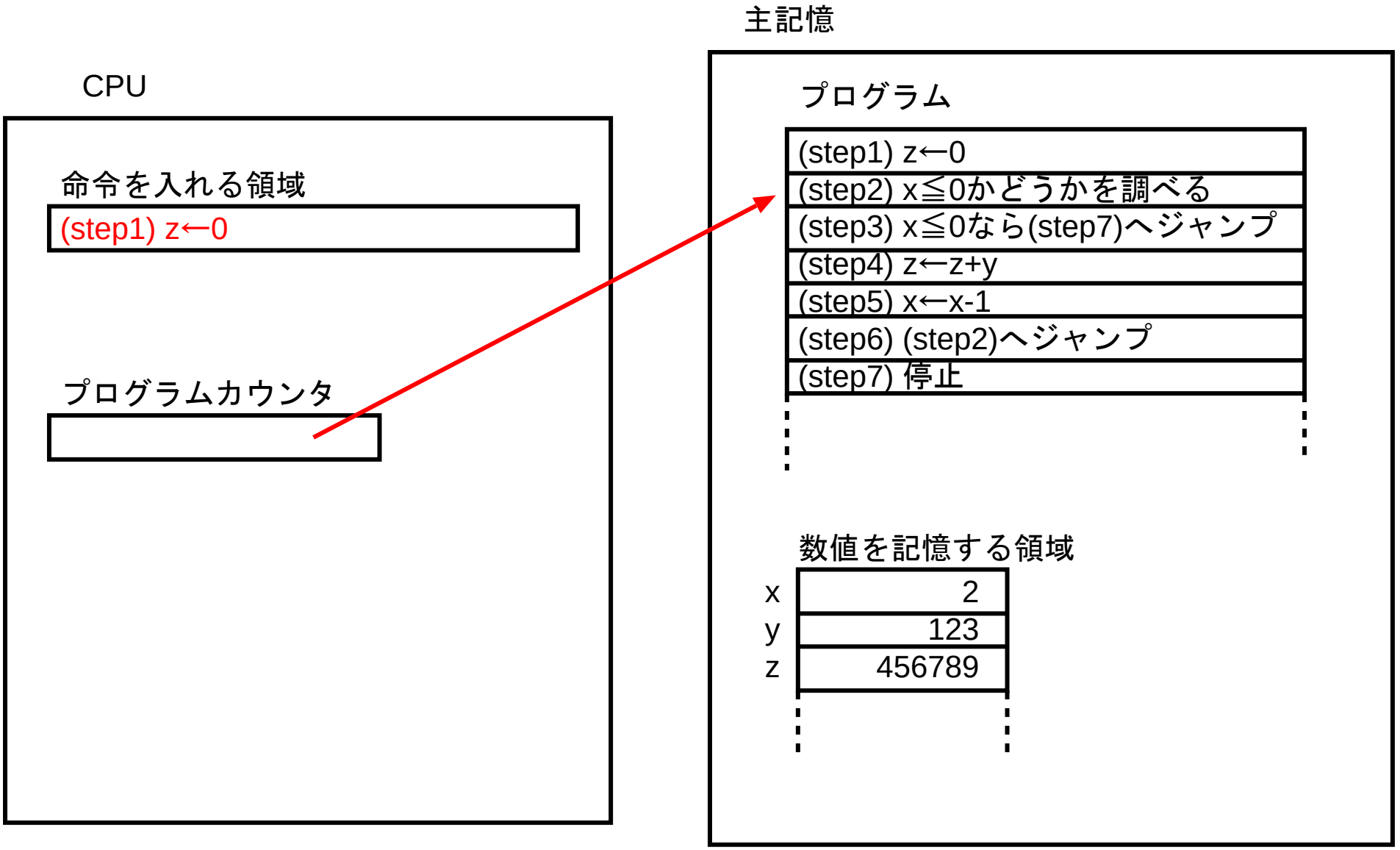
  $\longrightarrow$ 次頁の流れ図の動作がハードウェアで自動的に行われる。


PCの初期設定
PCの指す命令をCPU内に読み込む
PC←PC+読み込んだ命令の長さ
読み込んだ命令を解読してそれを実行する
終了命令の場合
管理プログラムに制御を移す
プログラムカウンタにプログラムの先頭番地を入れる。(管理プログラムが行う。)
プログラムカウンタの記憶した番地にある命令をCPU内に読み込む。命令はCPUの中に一旦読み込まないと(ハードウェアで)解読できない。
プログラムカウンタの記憶した値に読み込んだ命令の長さを加えてその結果をプログラムカウンタに格納する。
計算時間超過、エラー発生などによってもこの繰り返しを終了することがある。
読み込んだ命令がジャンプ命令の場合にはプログラムカウンタが再設定される。

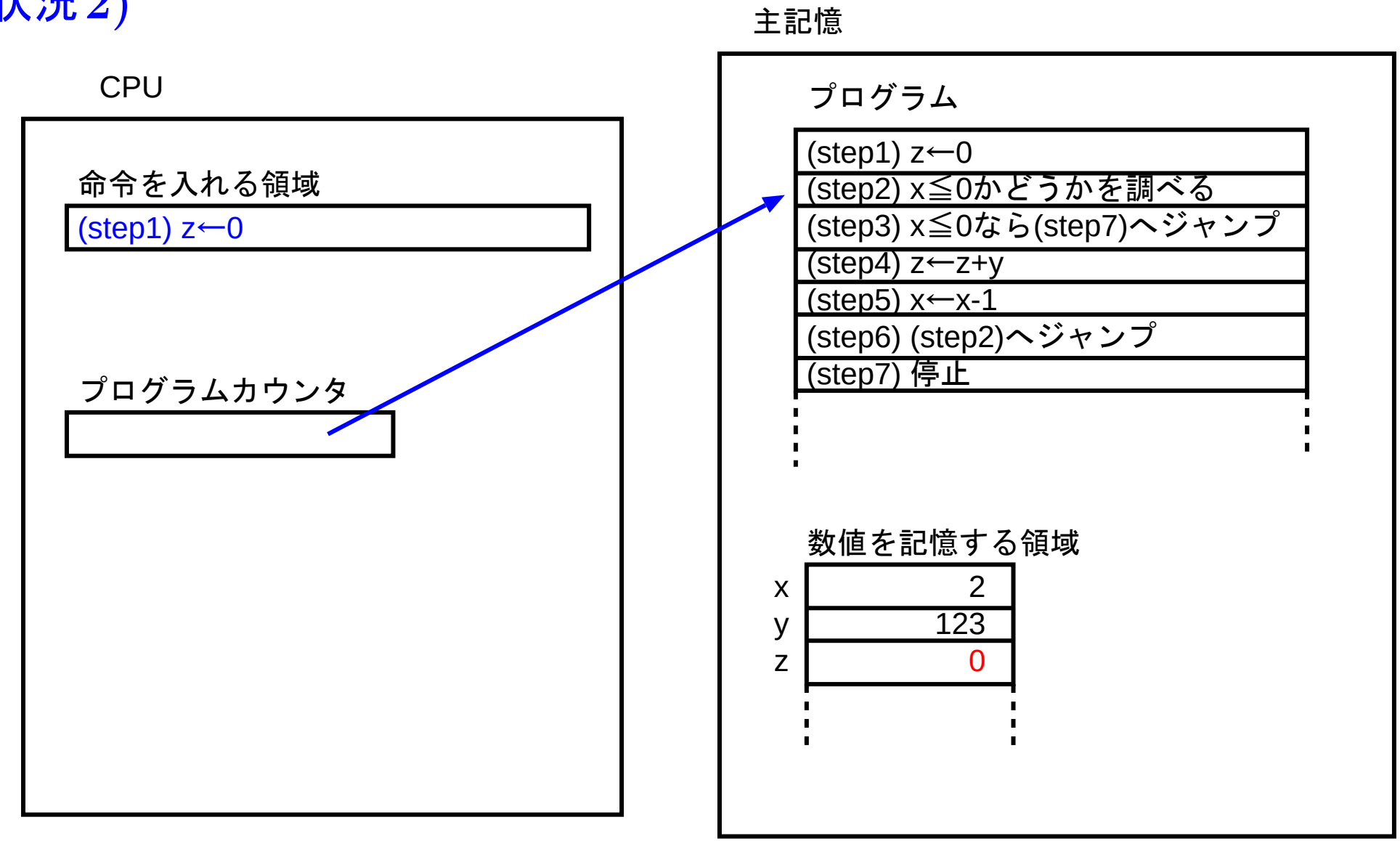
# 例2. 1 (プログラム内蔵方式計算機の動作)

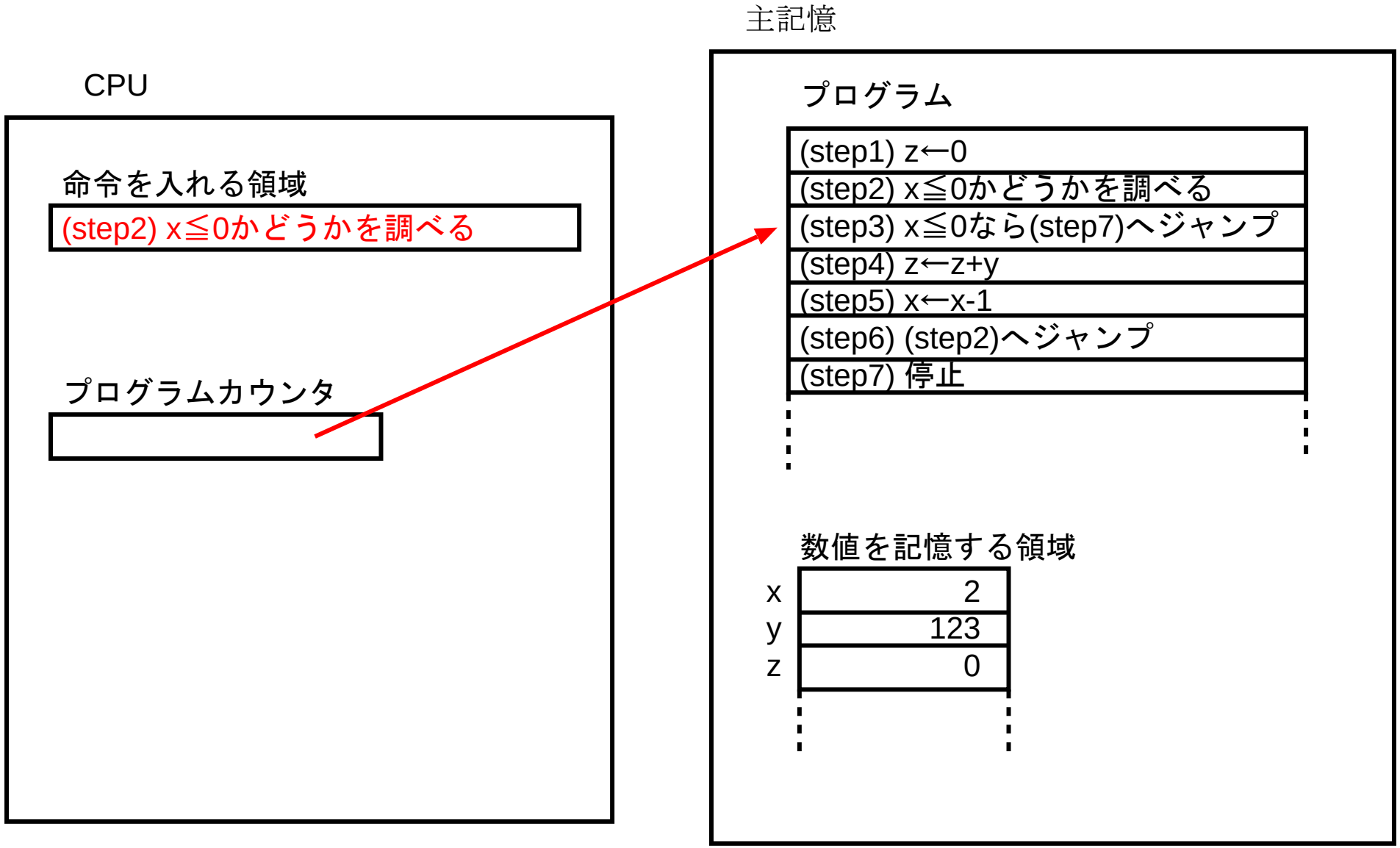
## (状況1)



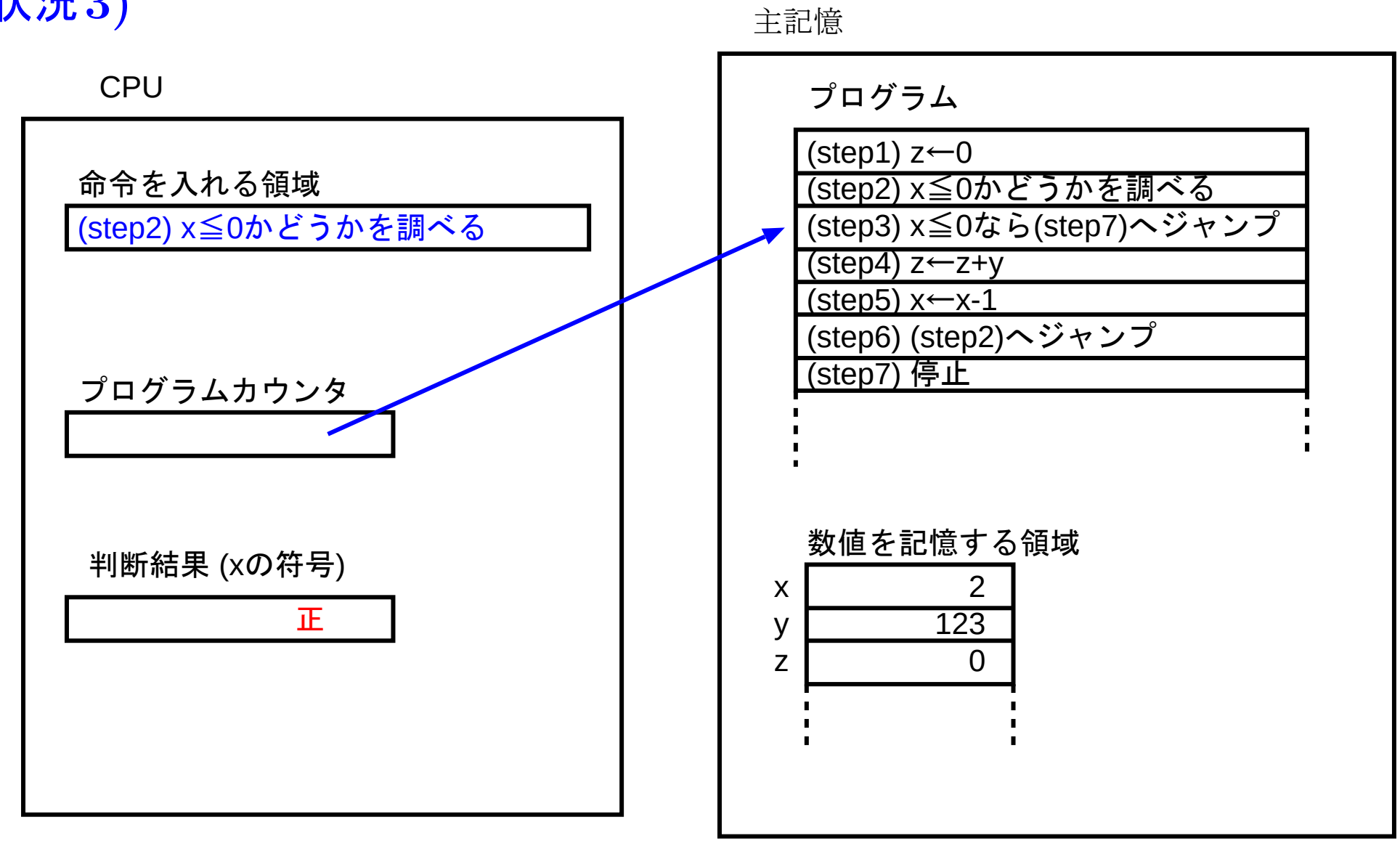

主記憶
CPU
プログラム
(step1) z←0
(step2) x≦0かどうかを調べる
(step3) x≦0なら(step7)へジャンプ
(step4) z←z+y
(step5) x←x-1
(step6) (step2)へジャンプ
(step7) 停止
命令を入れる領域
(step1) z←0
プログラムカウンタ
数値を記憶する領域
x 2
y 123
z 456789

# (状況2)

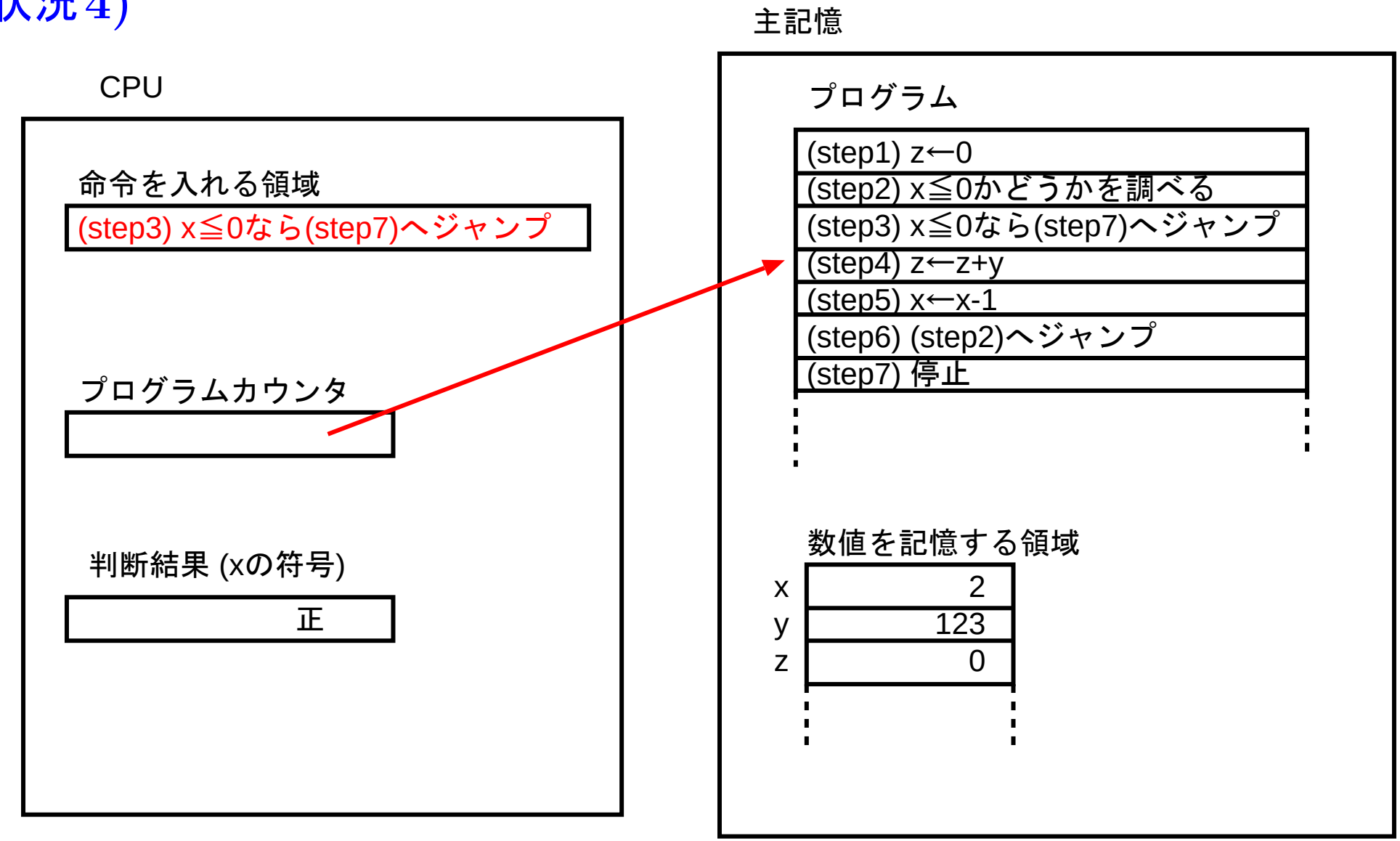

主記憶
CPU
プログラム
命令を入れる領域
(step1) z←0
(step2) x≦0かどうかを調べる
(step3) x≦0なら(step7)へジャンプ
(step4) z←z+y
(step5) x←x-1
(step6) (step2)へジャンプ
(step7) 停止
プログラムカウンタ
数値を記憶する領域
x
2
y
123
z
0

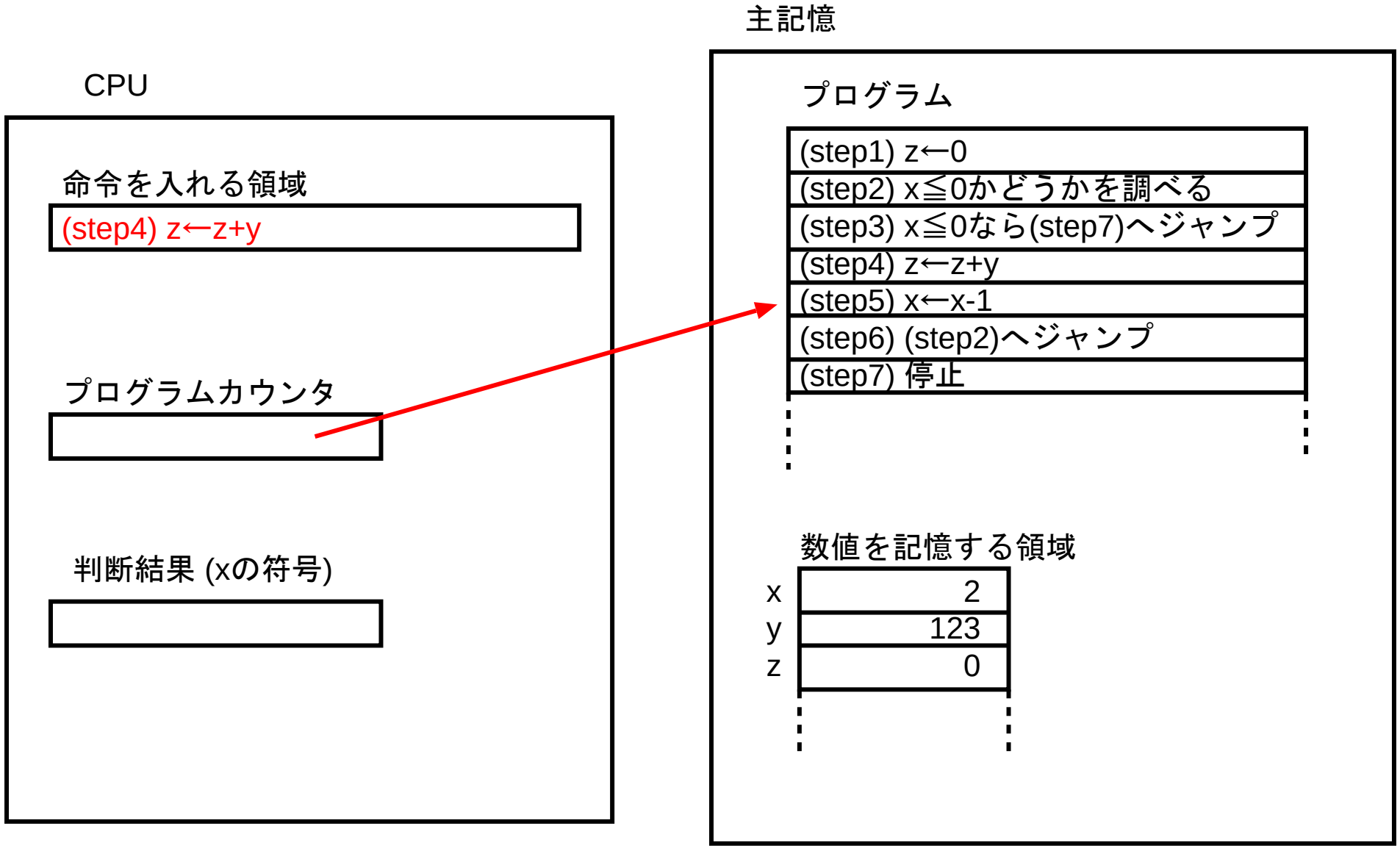

主記憶
CPU
プログラム
命令を入れる領域
(step2) x≦0かどうかを調べる
プログラムカウンタ
(step1) z←0
(step2) x≦0かどうかを調べる
(step3) x≦0なら(step7)へジャンプ
(step4) z←z+y
(step5) x←x-1
(step6) (step2)へジャンプ
(step7) 停止
数値を記憶する領域
x
y
z
2
123
0

# (状況3)

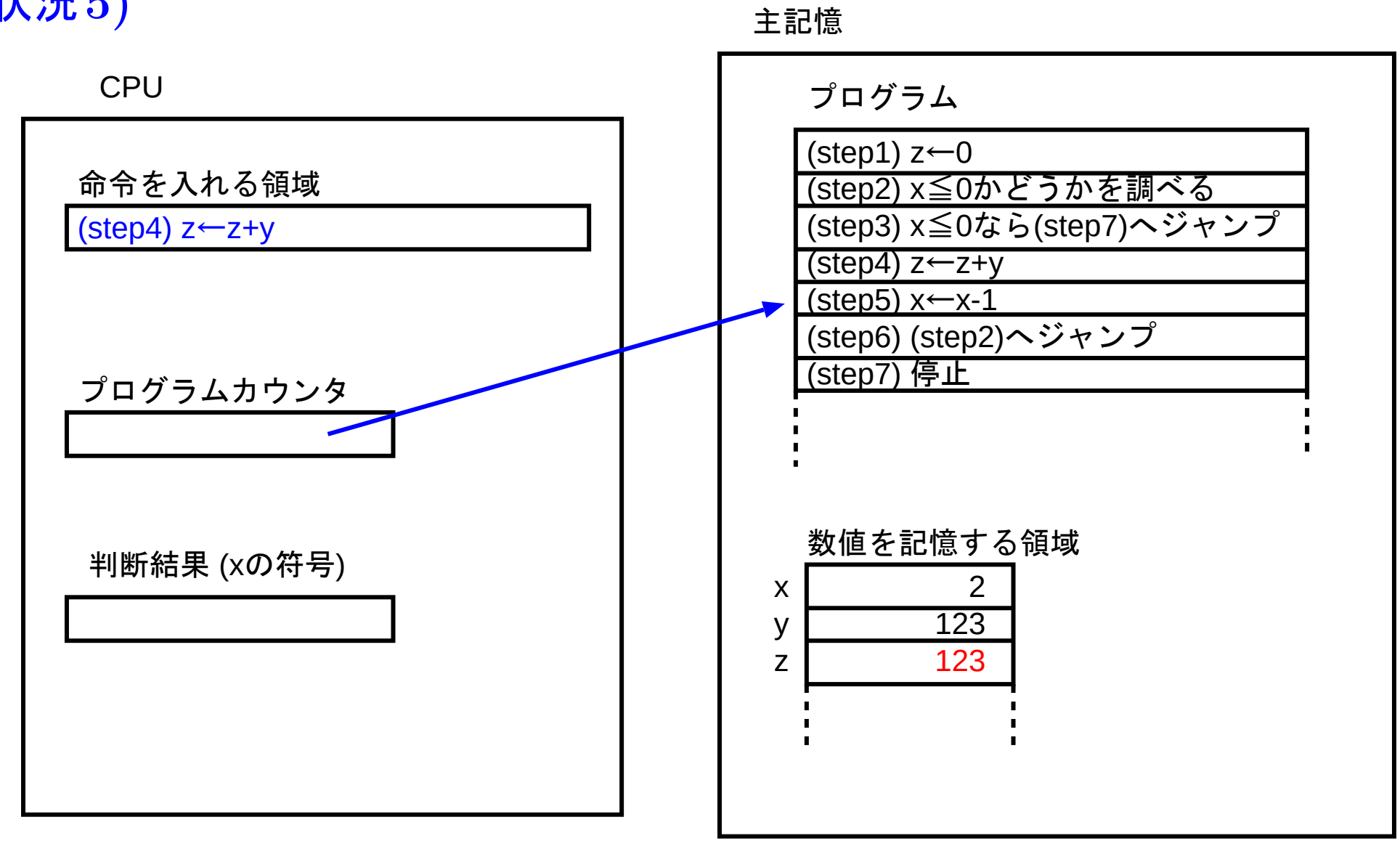

主記憶
CPU
命令を入れる領域
(step2) x≦0かどうかを調べる
プログラムカウンタ
判断結果 (xの符号)
正
プログラム
(step1) z←0
(step2) x≦0かどうかを調べる
(step3) x≦0なら(step7)へジャンプ
(step4) z←z+y
(step5) x←x-1
(step6) (step2)へジャンプ
(step7) 停止
数値を記憶する領域
x 2
y 123
z 0

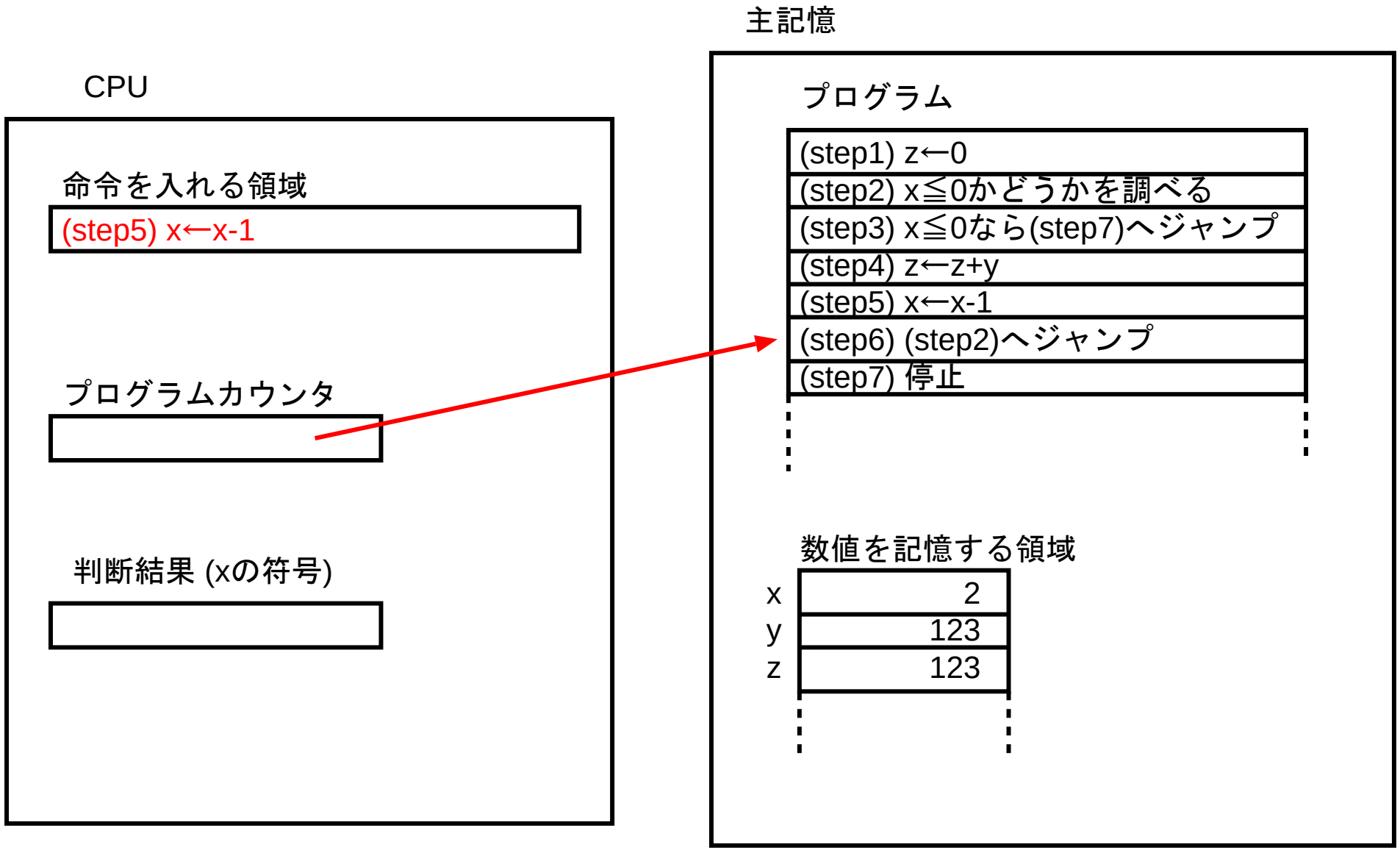
# (状況4)


主記憶
CPU
プログラム
(step1) z←0
(step2) x≦0かどうかを調べる
(step3) x≦0なら(step7)へジャンプ
(step4) z←z+y
(step5) x←x-1
(step6) (step2)へジャンプ
(step7) 停止
命令を入れる領域
(step3) x≦0なら(step7)へジャンプ
プログラムカウンタ
判断結果 (xの符号)
正
数値を記憶する領域
x
2
y
123
z
0

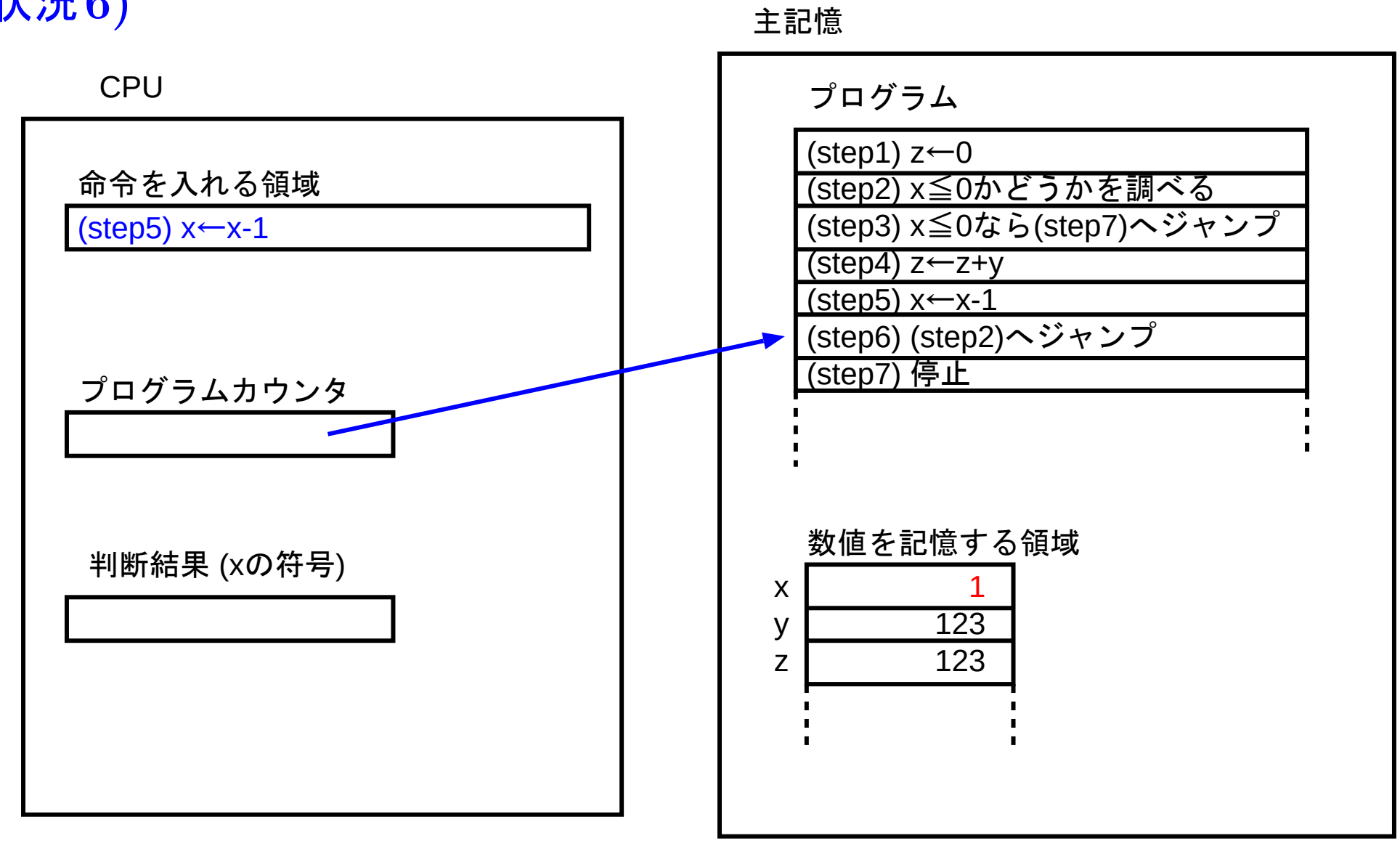

主記憶
CPU
プログラム
(step1) z←0
(step2) x≦0かどうかを調べる
(step3) x≦0なら(step7)へジャンプ
(step4) z←z+y
(step5) x←x-1
(step6) (step2)へジャンプ
(step7) 停止
命令を入れる領域
(step4) z←z+y
プログラムカウンタ
判断結果 (xの符号)
数値を記憶する領域
x 2
y 123
z 0

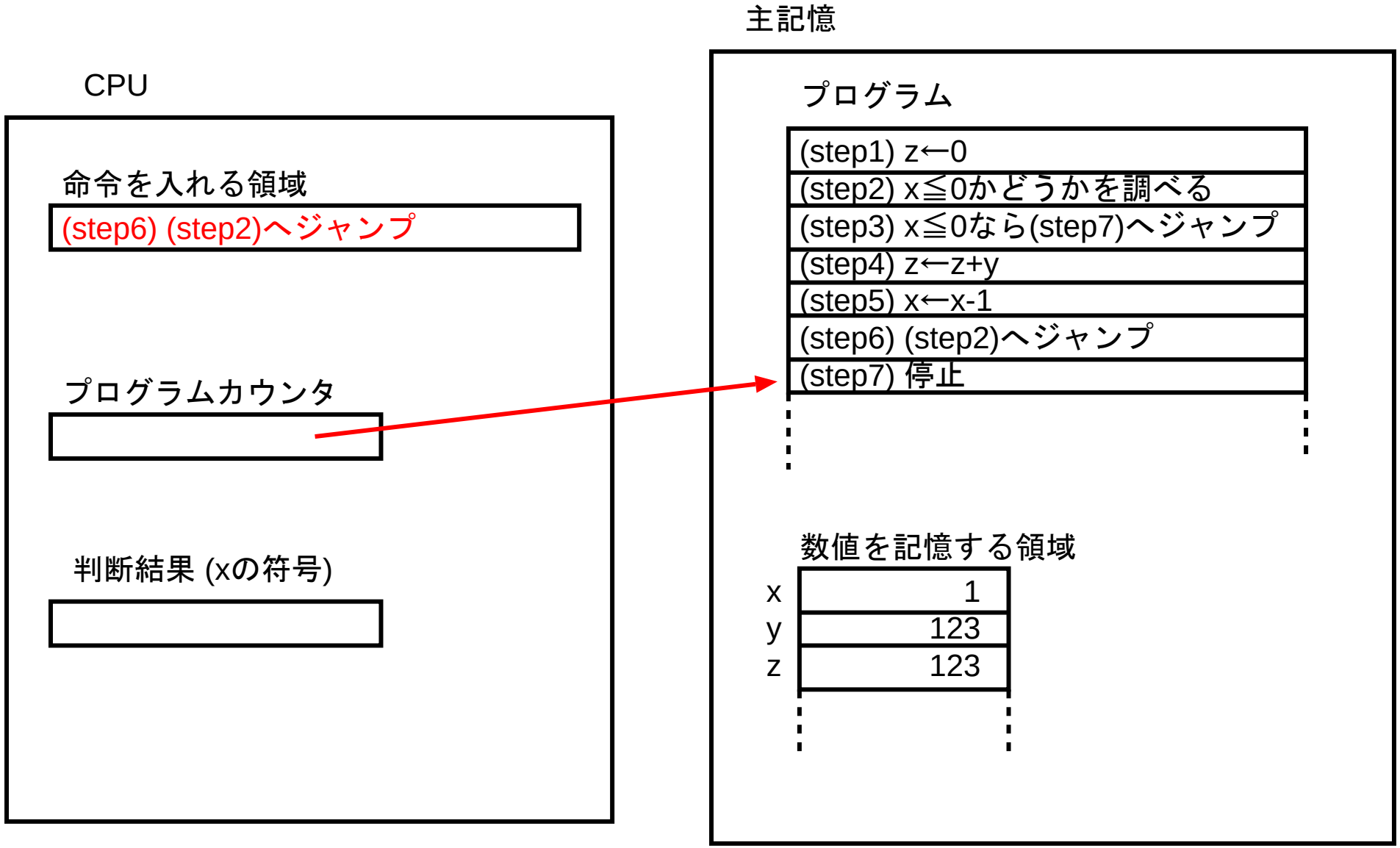
# (状況5)


主記憶
CPU
プログラム
(step1) z←0
(step2) x≦0かどうかを調べる
(step3) x≦0なら(step7)へジャンプ
(step4) z←z+y
(step5) x←x-1
(step6) (step2)へジャンプ
(step7) 停止
命令を入れる領域
(step4) z←z+y
プログラムカウンタ
判断結果 (xの符号)
数値を記憶する領域
x 2
y 123
z 123

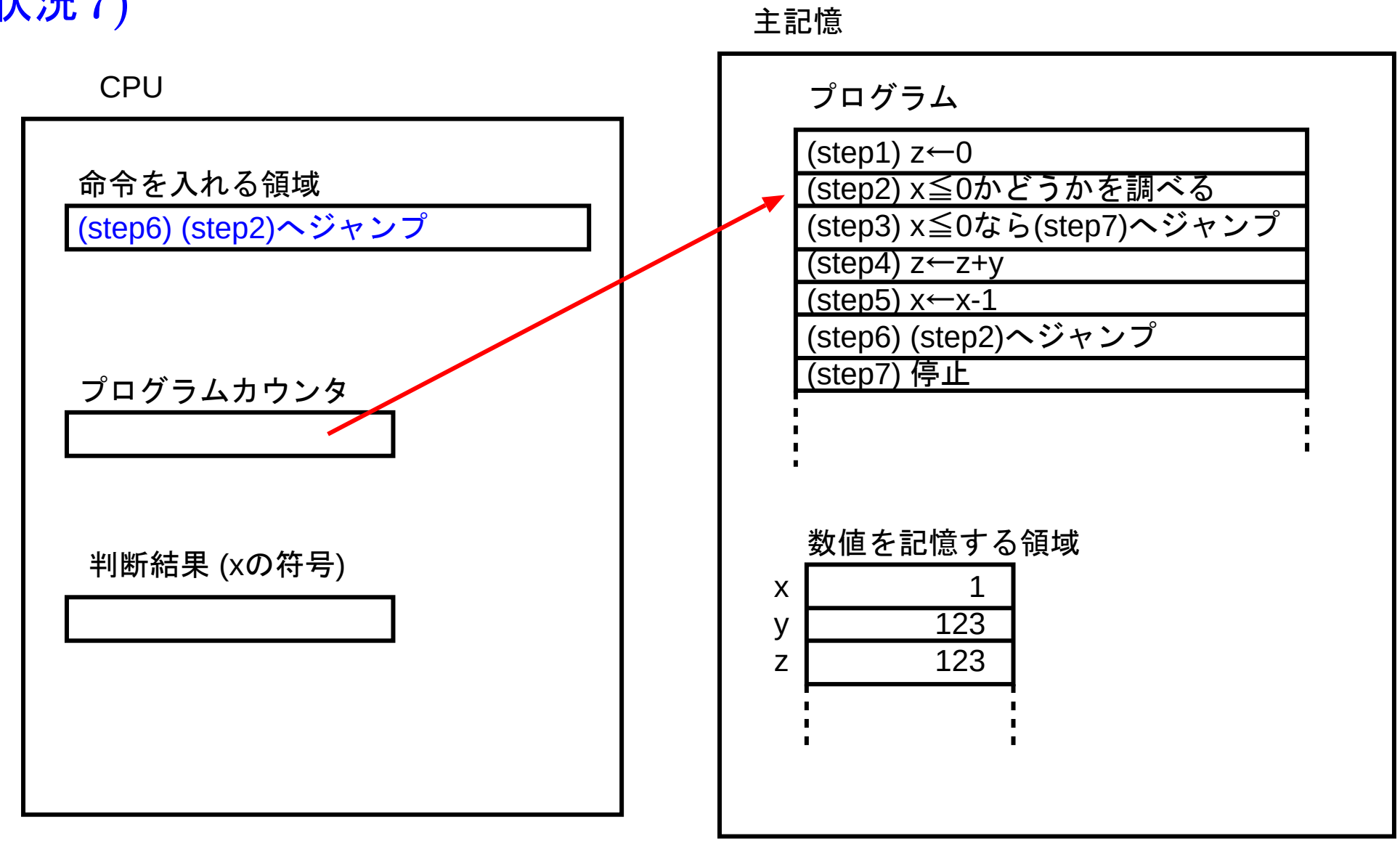

主記憶
CPU
プログラム
(step1) z←0
(step2) x≦0かどうかを調べる
(step3) x≦0なら(step7)へジャンプ
(step4) z←z+y
(step5) x←x-1
(step6) (step2)へジャンプ
(step7) 停止
命令を入れる領域
(step5) x←x-1
プログラムカウンタ
判断結果 (xの符号)
数値を記憶する領域
x 2
y 123
z 123

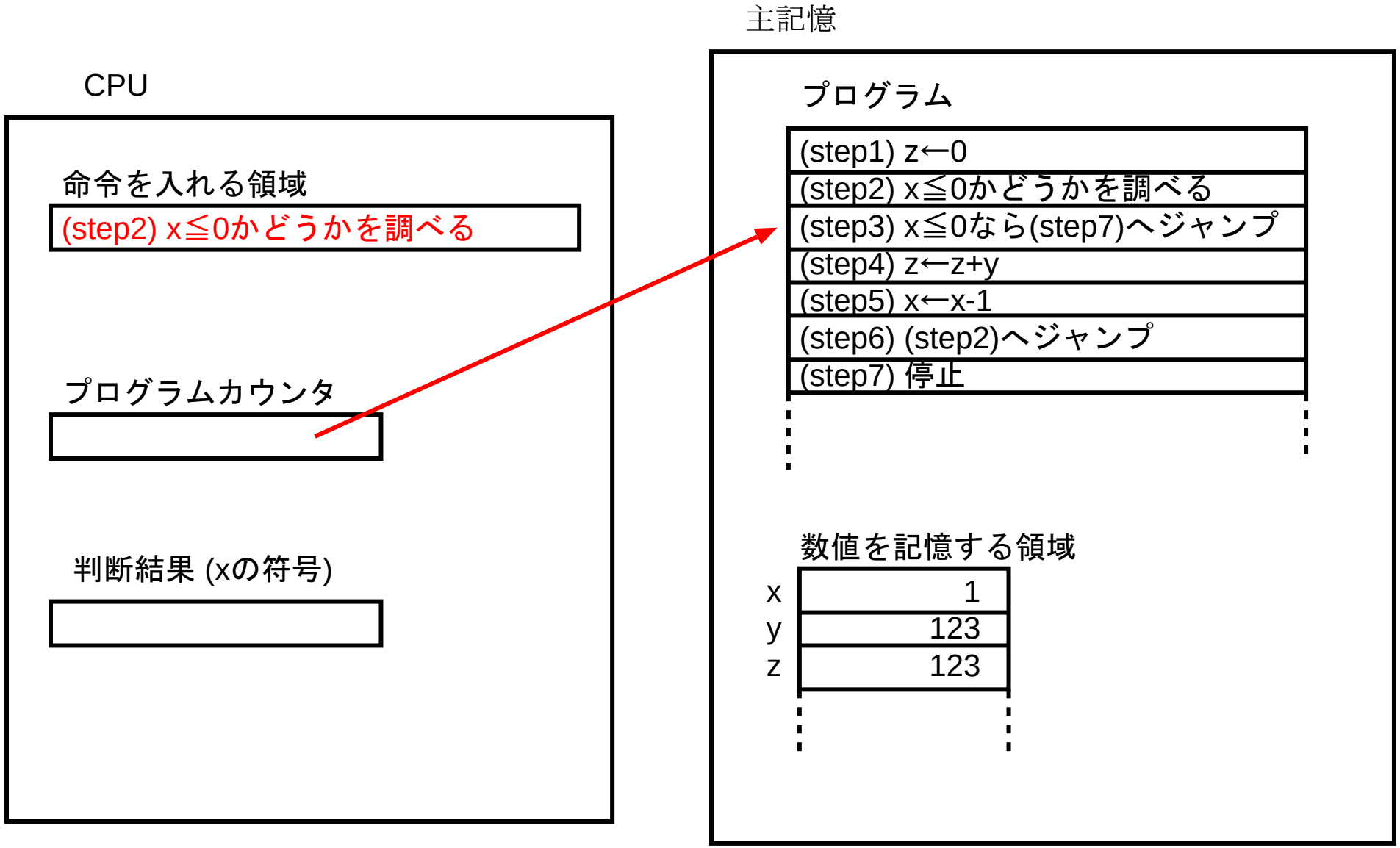
# (状況6)


主記憶
CPU
プログラム
(step1) z←0
(step2) x≦0かどうかを調べる
(step3) x≦0なら(step7)へジャンプ
(step4) z←z+y
(step5) x←x-1
(step6) (step2)へジャンプ
(step7) 停止
命令を入れる領域
(step5) x←x-1
プログラムカウンタ
判断結果 (xの符号)
数値を記憶する領域
x 1
y 123
z 123

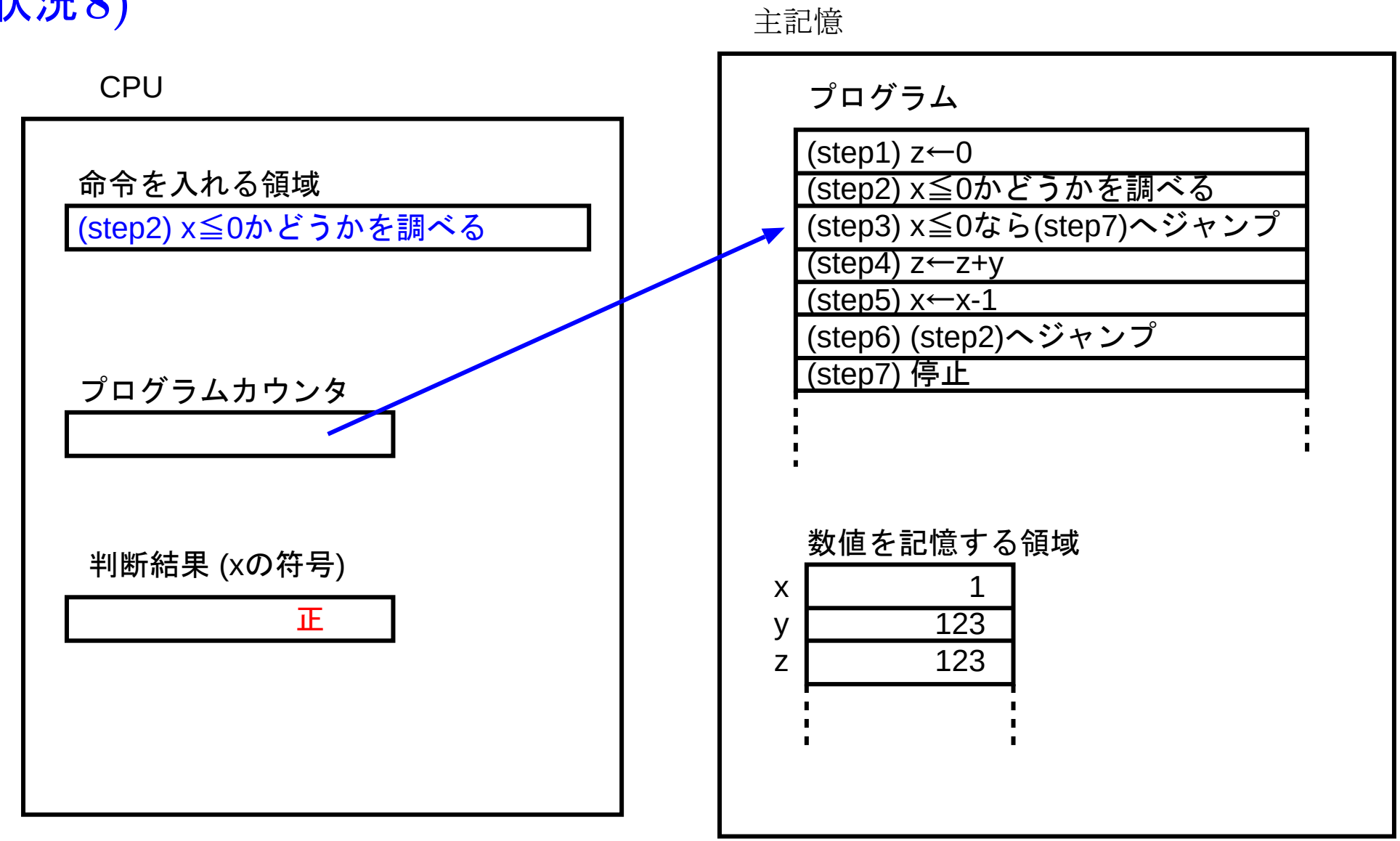

主記憶
CPU
プログラム
(step1) z←0
(step2) x≦0かどうかを調べる
(step3) x≦0なら(step7)へジャンプ
(step4) z←z+y
(step5) x←x-1
(step6) (step2)へジャンプ
(step7) 停止
命令を入れる領域
(step6) (step2)へジャンプ
プログラムカウンタ
判断結果 (xの符号)
数値を記憶する領域
x
y
z
1
123
123

# (状況7)

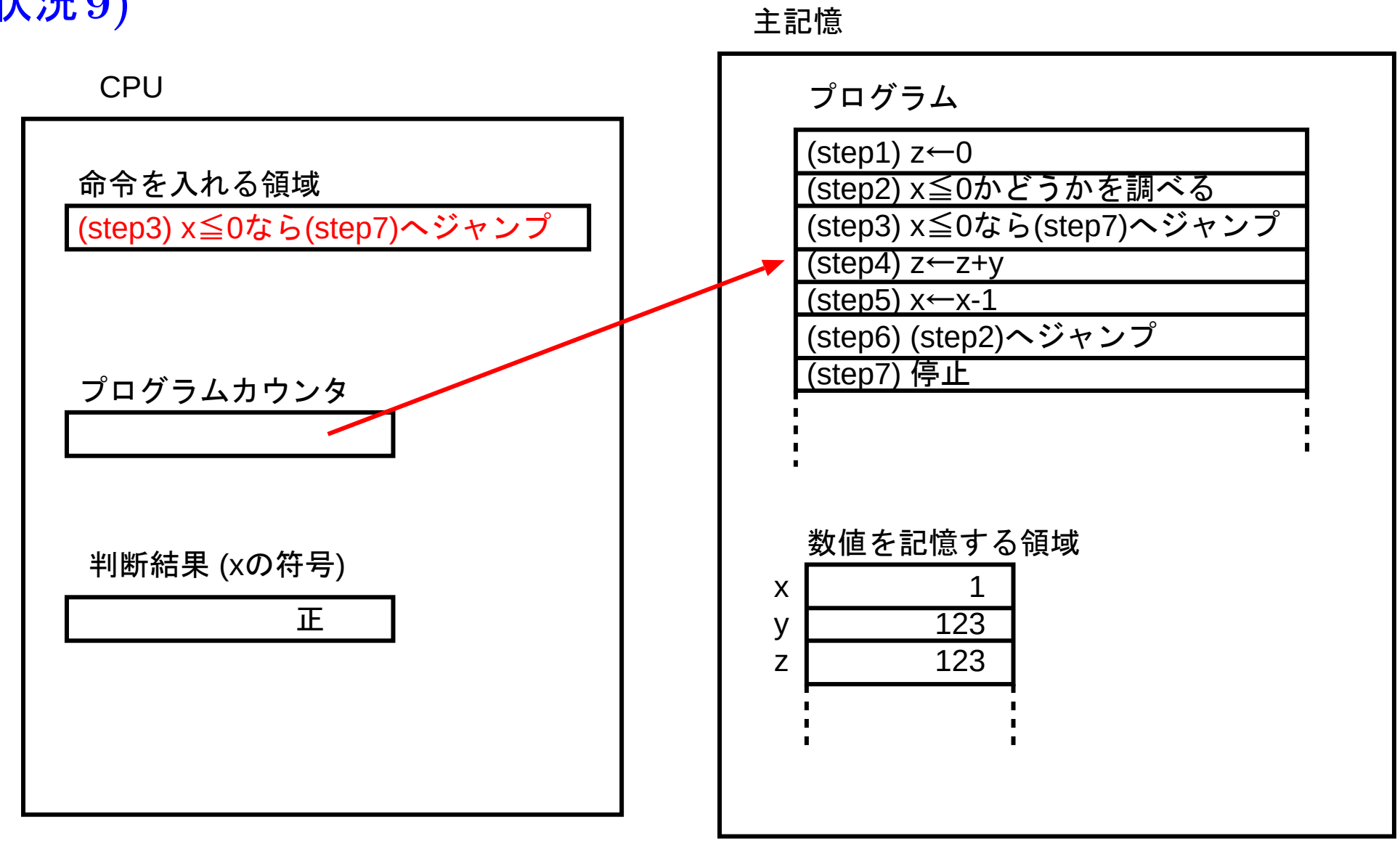

主記憶
CPU
プログラム
(step1) z←0
(step2) x≦0かどうかを調べる
(step3) x≦0なら(step7)へジャンプ
(step4) z←z+y
(step5) x←x-1
(step6) (step2)へジャンプ
(step7) 停止
命令を入れる領域
(step6) (step2)へジャンプ
プログラムカウンタ
判断結果 (xの符号)
数値を記憶する領域
x 1
y 123
z 123

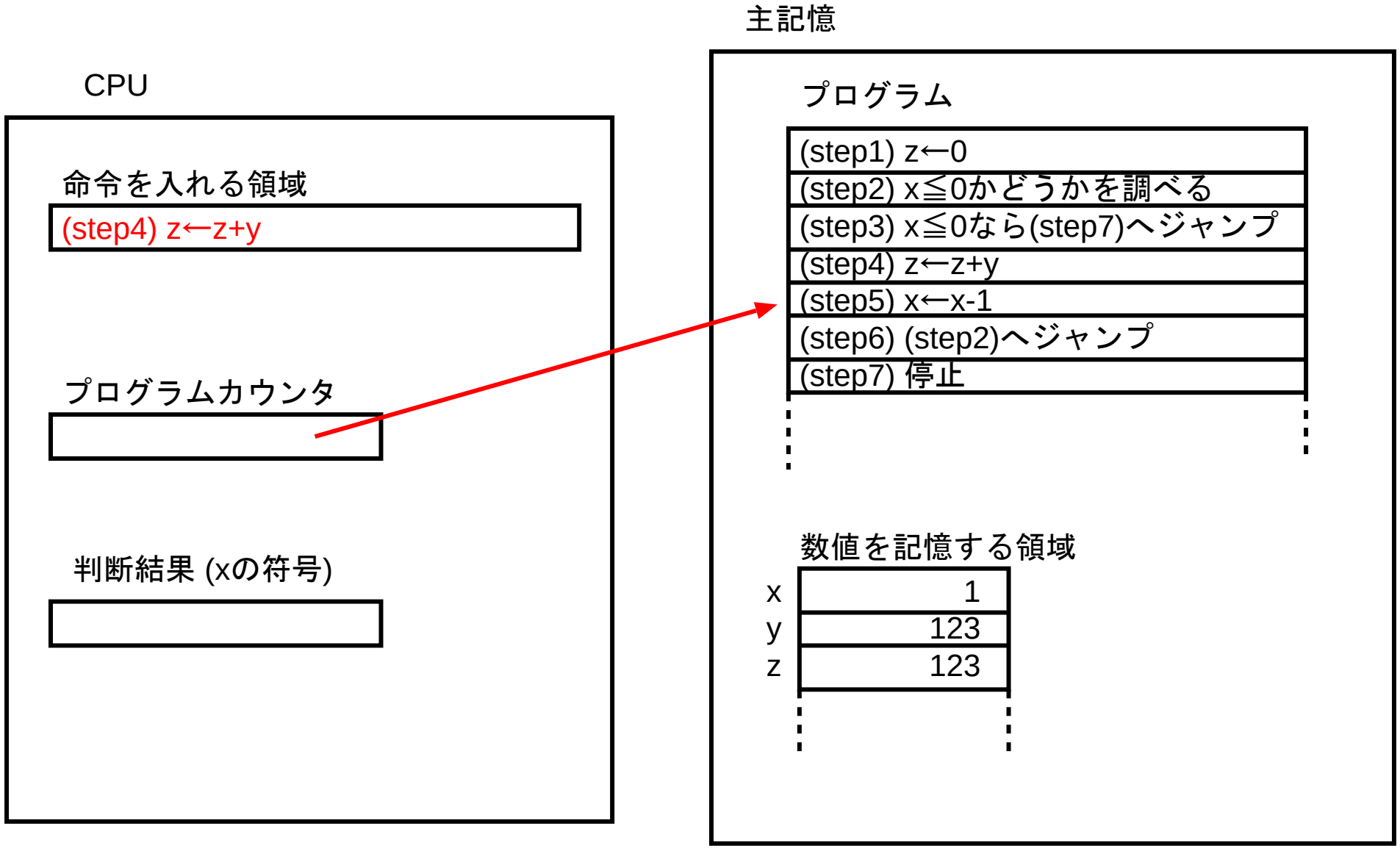

主記憶
CPU
プログラム
(step1) z←0
(step2) x≦0かどうかを調べる
(step3) x≦0なら(step7)へジャンプ
(step4) z←z+y
(step5) x←x-1
(step6) (step2)へジャンプ
(step7) 停止
命令を入れる領域
(step2) x≦0かどうかを調べる
プログラムカウンタ
判断結果 (xの符号)
数値を記憶する領域
x
y
z
1
123
123

# (状況8)

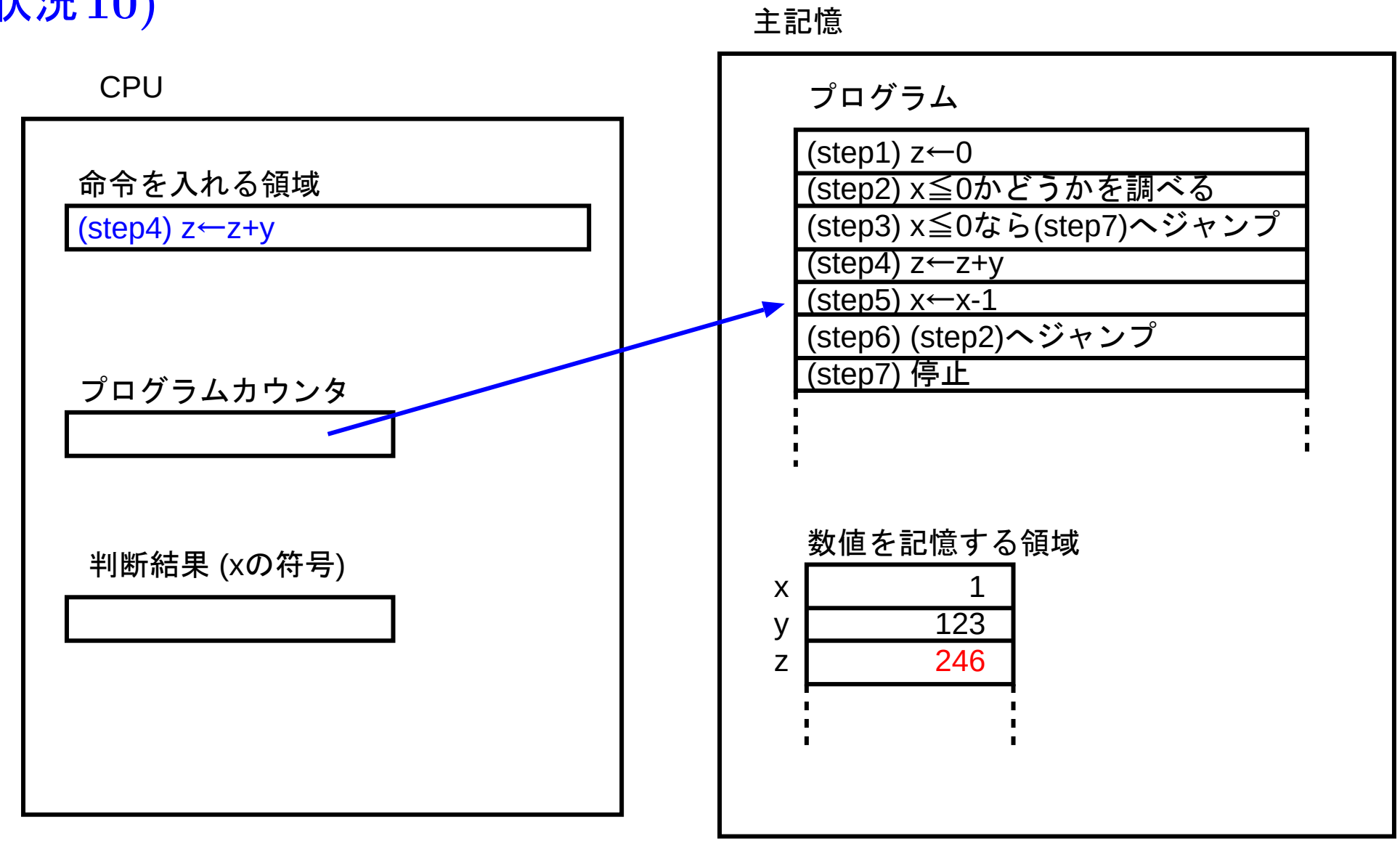

主記憶
CPU
命令を入れる領域
(step2) x≦0かどうかを調べる
プログラムカウンタ
判断結果 (xの符号)
正
プログラム
(step1) z←0
(step2) x≦0かどうかを調べる
(step3) x≦0なら(step7)へジャンプ
(step4) z←z+y
(step5) x←x-1
(step6) (step2)へジャンプ
(step7) 停止
数値を記憶する領域
x 1
y 123
z 123

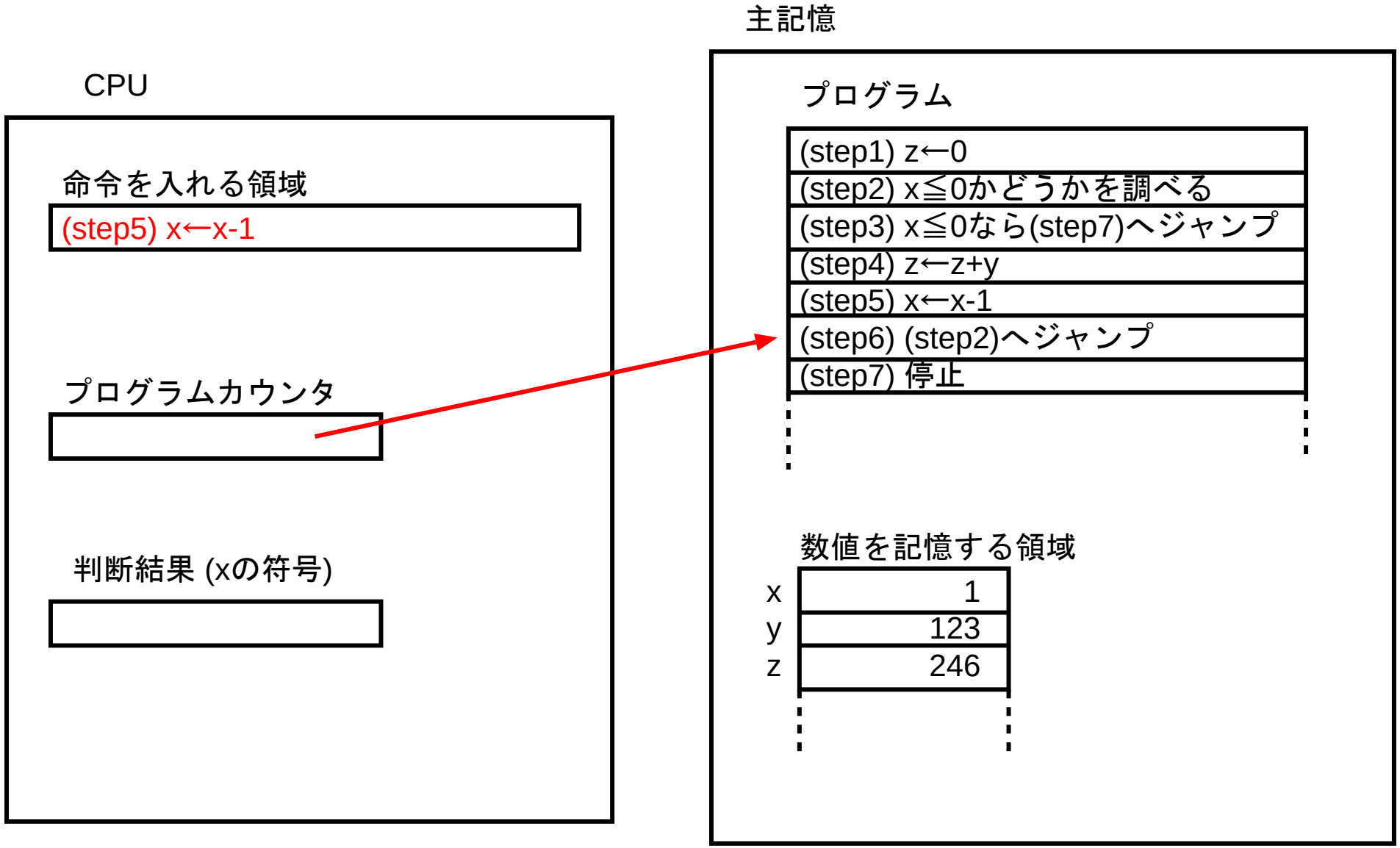
# (状況9)


主記憶
CPU
プログラム
(step1) z←0
(step2) x≦0かどうかを調べる
(step3) x≦0なら(step7)へジャンプ
(step4) z←z+y
(step5) x←x-1
(step6) (step2)へジャンプ
(step7) 停止
命令を入れる領域
(step3) x≦0なら(step7)へジャンプ
プログラムカウンタ
判断結果 (xの符号)
正
数値を記憶する領域
x 1
y 123
z 123

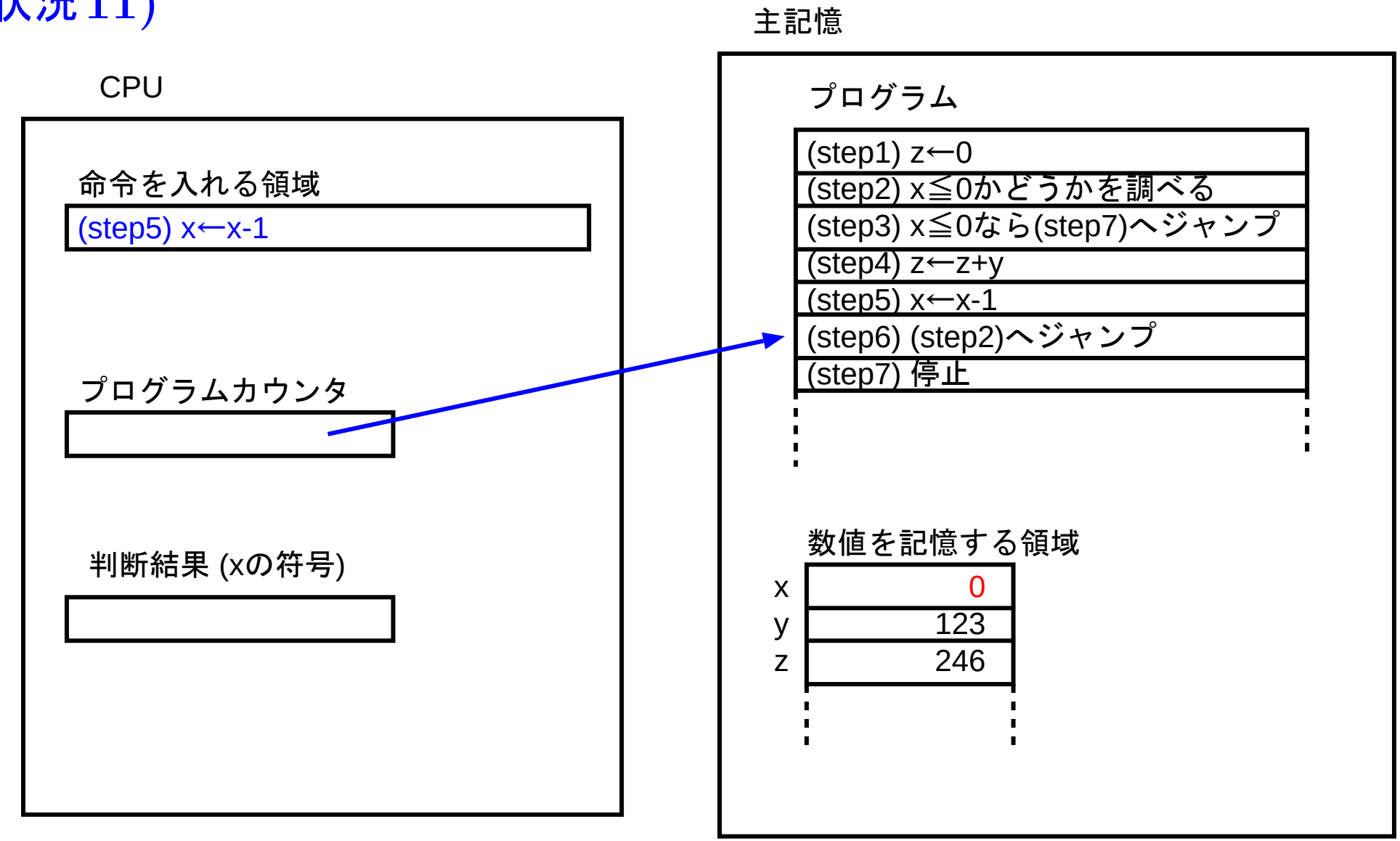

主記憶
CPU
プログラム
(step1) z←0
(step2) x≦0かどうかを調べる
(step3) x≦0なら(step7)へジャンプ
(step4) z←z+y
(step5) x←x-1
(step6) (step2)へジャンプ
(step7) 停止
命令を入れる領域
(step4) z←z+y
プログラムカウンタ
判断結果 (xの符号)
数値を記憶する領域
x
y
z
1
123
123

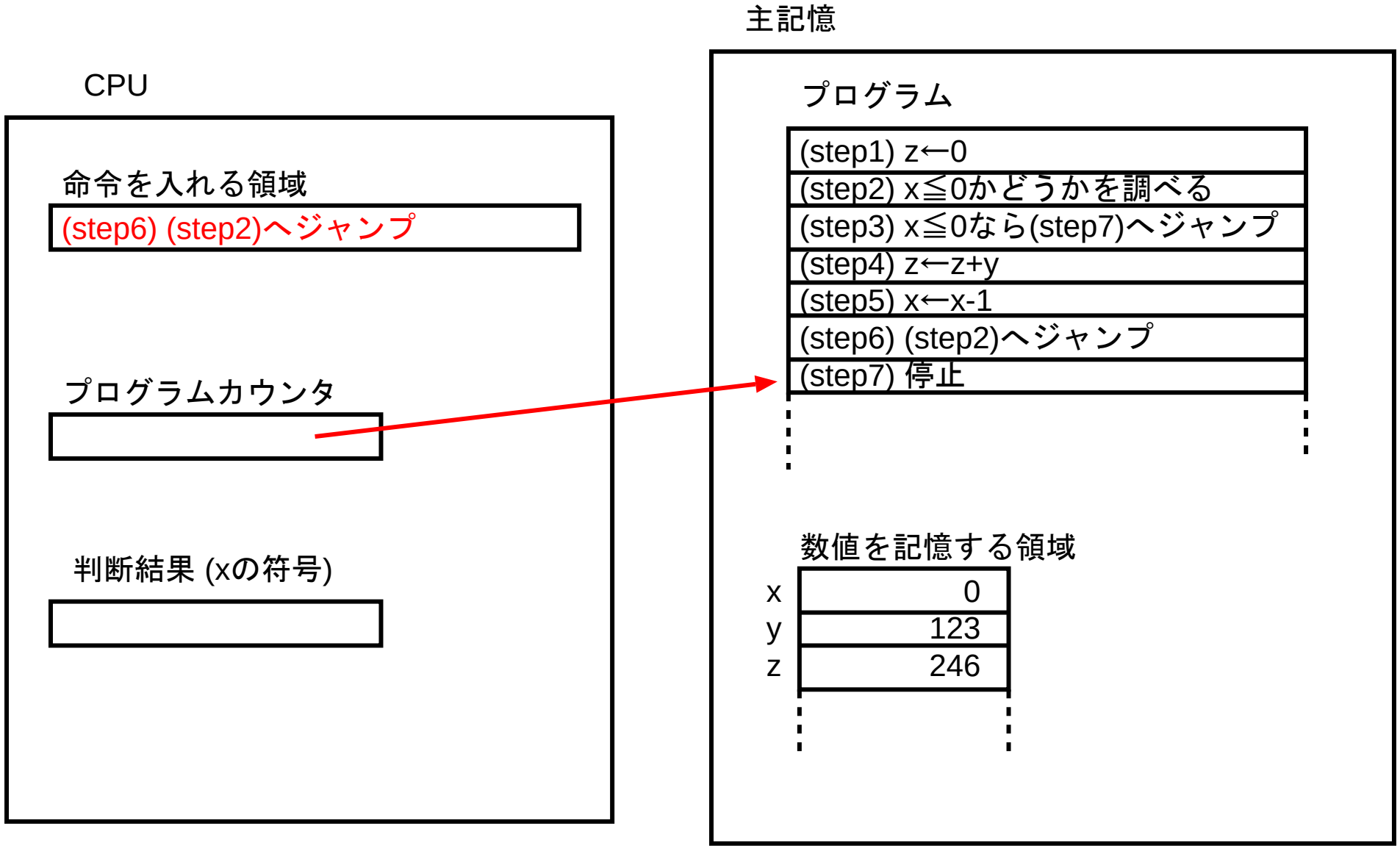
# (状況10)


主記憶
CPU
プログラム
(step1) z←0
(step2) x≦0かどうかを調べる
(step3) x≦0なら(step7)へジャンプ
(step4) z←z+y
(step5) x←x-1
(step6) (step2)へジャンプ
(step7) 停止
命令を入れる領域
(step4) z←z+y
プログラムカウンタ
判断結果 (xの符号)
数値を記憶する領域
x 1
y 123
z 246

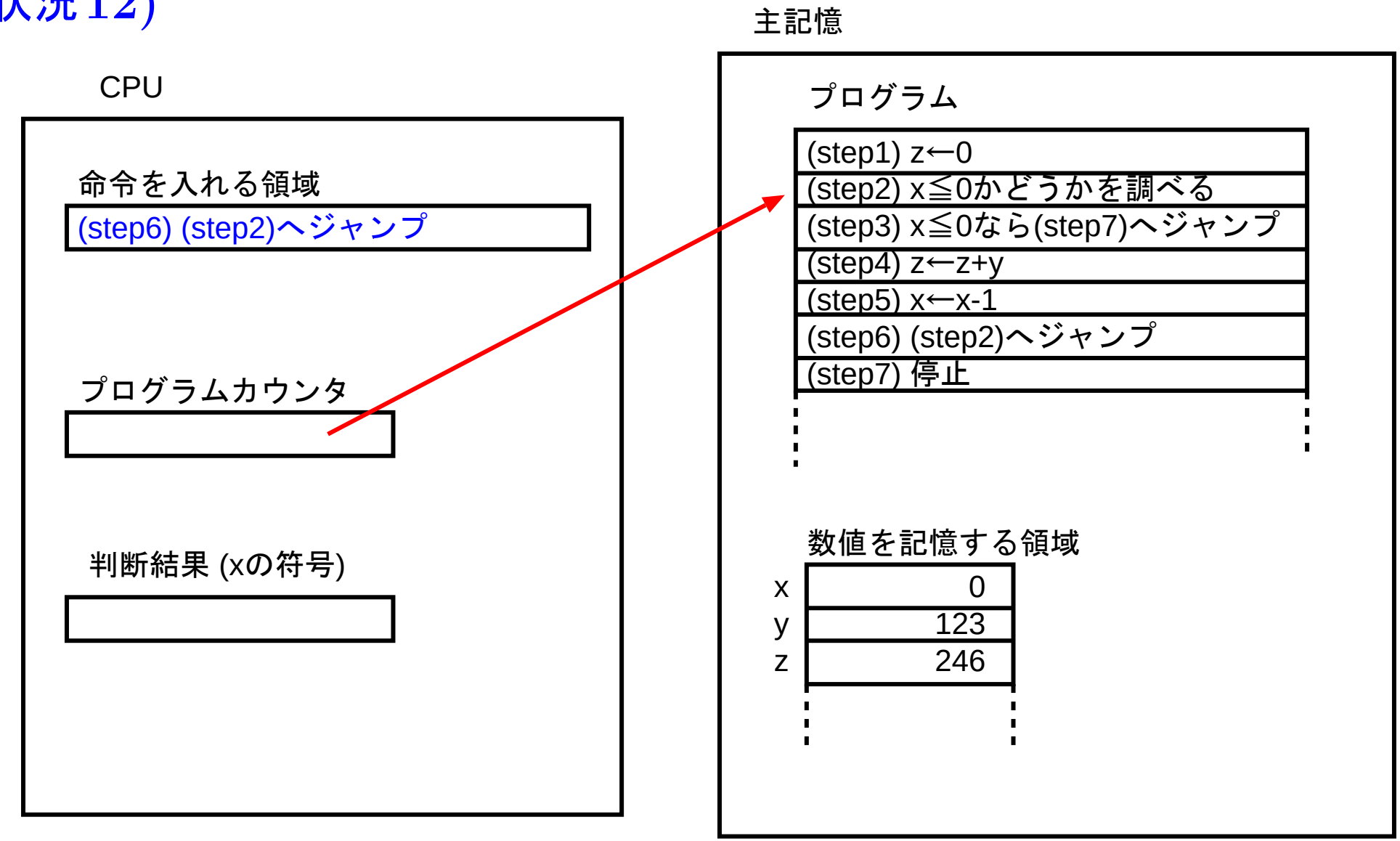

主記憶
CPU
プログラム
(step1) z←0
(step2) x≦0かどうかを調べる
(step3) x≦0なら(step7)へジャンプ
(step4) z←z+y
(step5) x←x-1
(step6) (step2)へジャンプ
(step7) 停止
命令を入れる領域
(step5) x←x-1
プログラムカウンタ
判断結果 (xの符号)
数値を記憶する領域
x 1
y 123
z 246

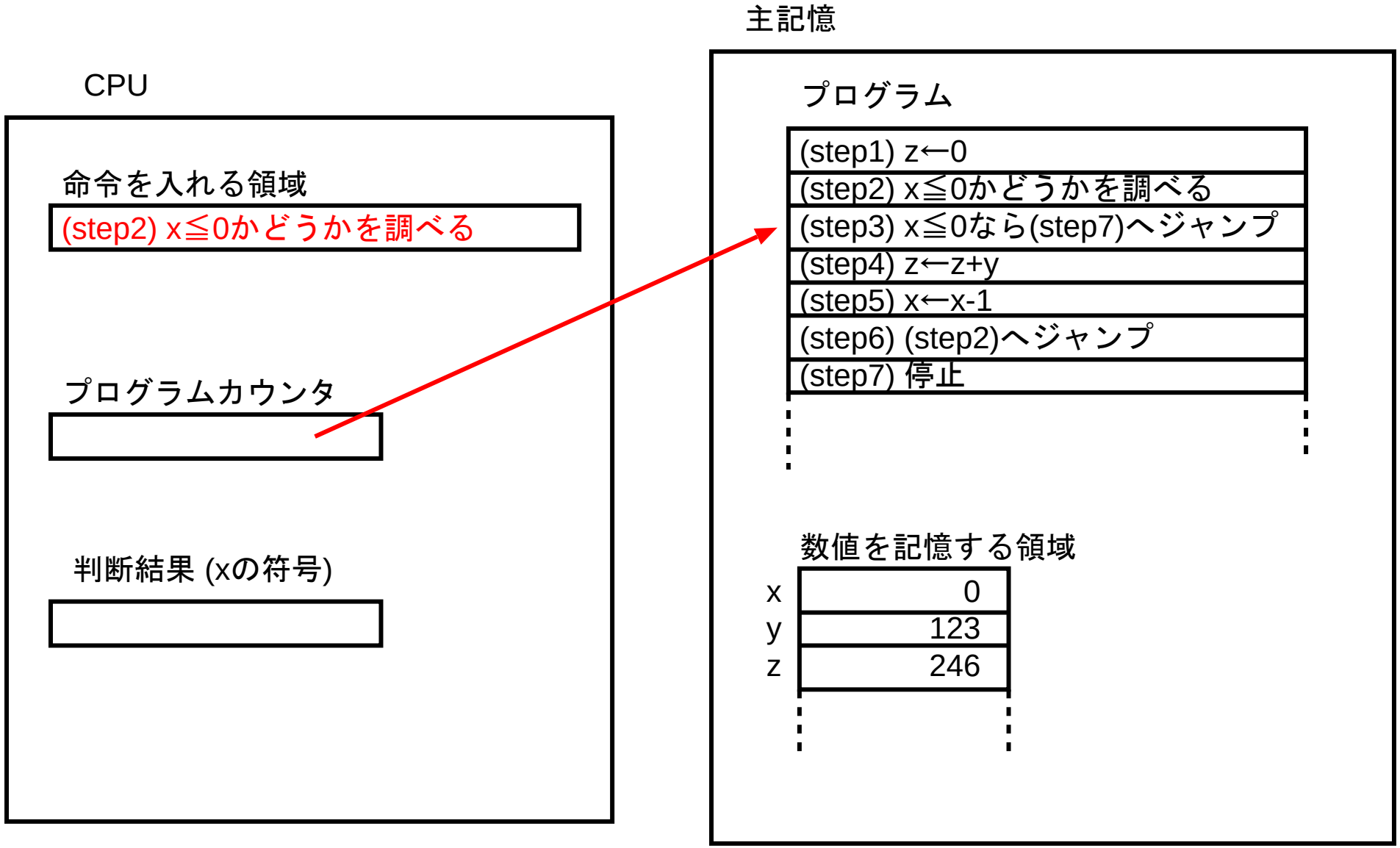
# (状況11)


主記憶
CPU
プログラム
(step1) z←0
(step2) x≦0かどうかを調べる
(step3) x≦0なら(step7)へジャンプ
(step4) z←z+y
(step5) x←x-1
(step6) (step2)へジャンプ
(step7) 停止
命令を入れる領域
(step5) x←x-1
プログラムカウンタ
判断結果 (xの符号)
数値を記憶する領域
x 0
y 123
z 246

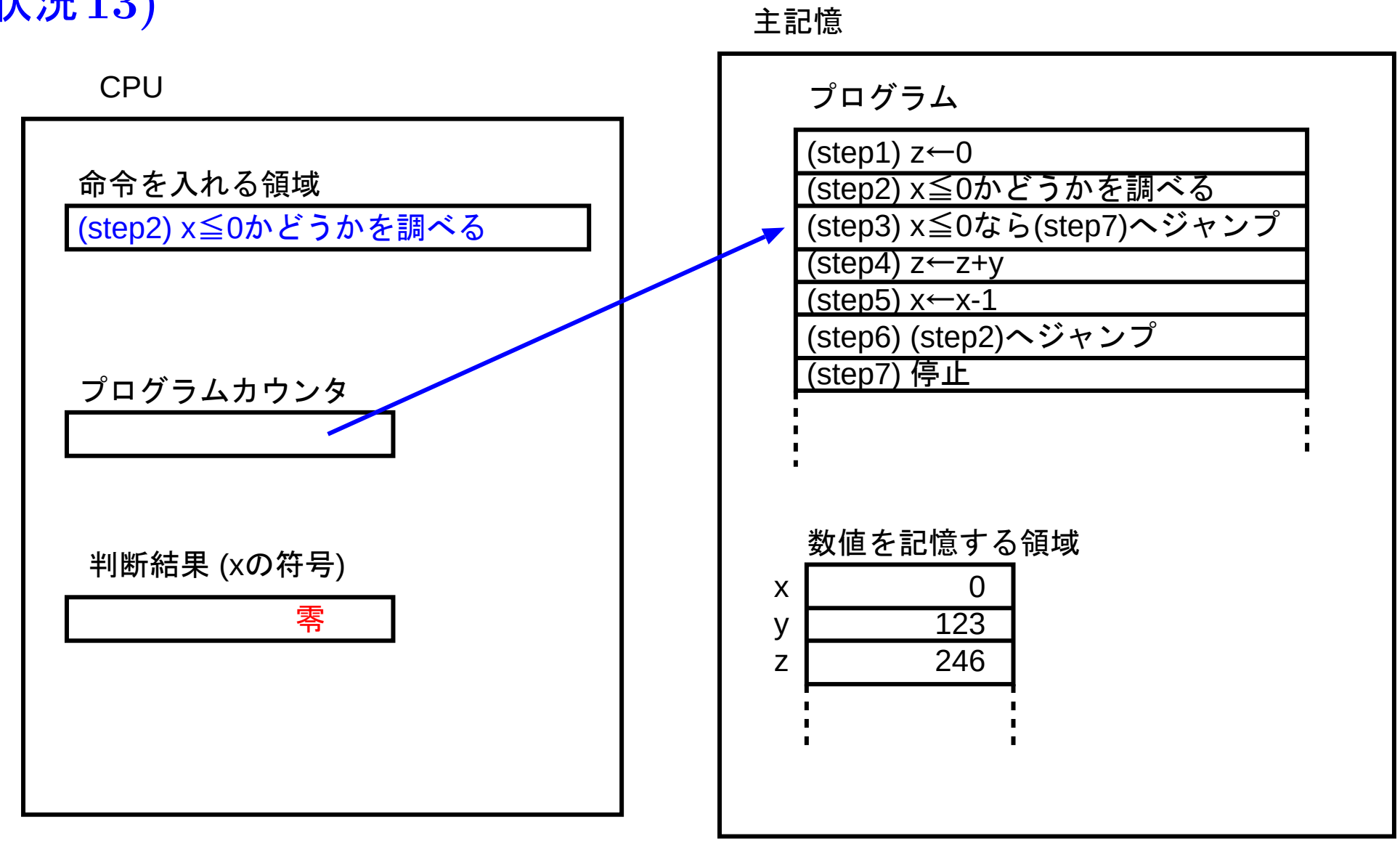

主記憶
CPU
プログラム
(step1) z←0
(step2) x≦0かどうかを調べる
(step3) x≦0なら(step7)へジャンプ
(step4) z←z+y
(step5) x←x-1
(step6) (step2)へジャンプ
(step7) 停止
命令を入れる領域
(step6) (step2)へジャンプ
プログラムカウンタ
判断結果 (xの符号)
数値を記憶する領域
x
y
z
0
123
246

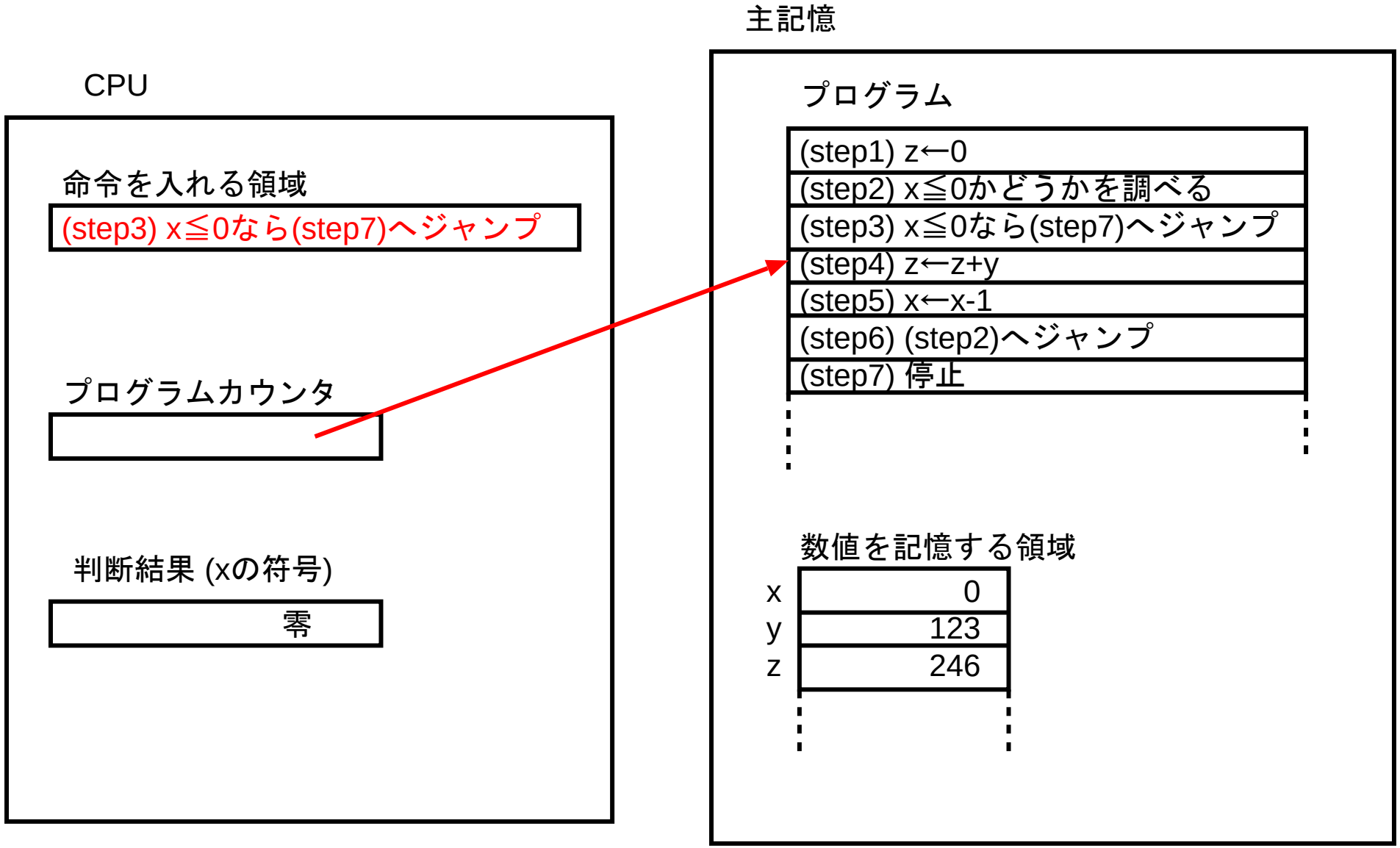
# (状況12)


主記憶
CPU
プログラム
(step1) z←0
(step2) x≦0かどうかを調べる
(step3) x≦0なら(step7)へジャンプ
(step4) z←z+y
(step5) x←x-1
(step6) (step2)へジャンプ
(step7) 停止
命令を入れる領域
(step6) (step2)へジャンプ
プログラムカウンタ
判断結果 (xの符号)
数値を記憶する領域
x 0
y 123
z 246

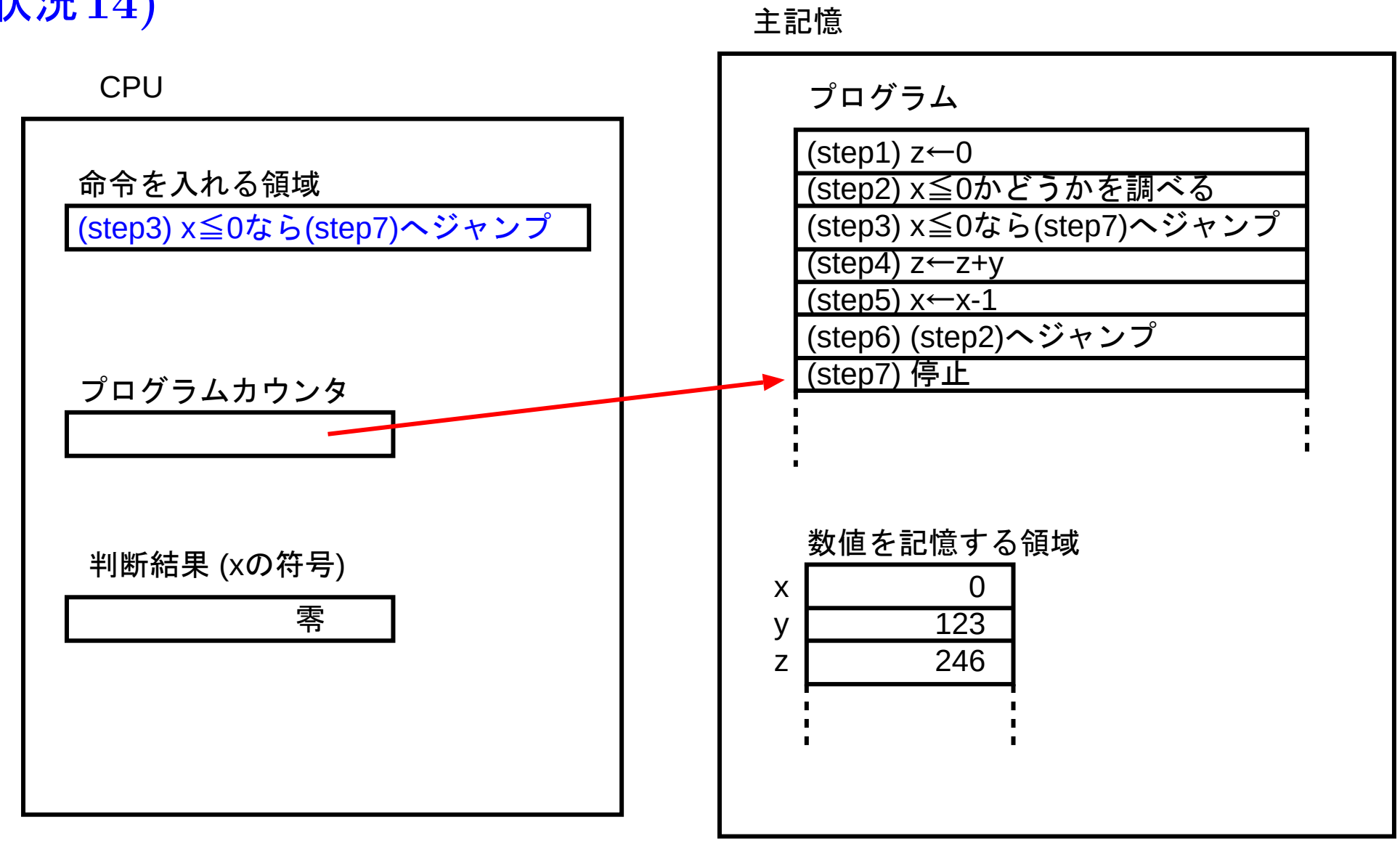

主記憶
CPU
プログラム
(step1) z←0
(step2) x≦0かどうかを調べる
(step3) x≦0なら(step7)へジャンプ
(step4) z←z+y
(step5) x←x-1
(step6) (step2)へジャンプ
(step7) 停止
命令を入れる領域
(step2) x≦0かどうかを調べる
プログラムカウンタ
判断結果 (xの符号)
数値を記憶する領域
x 0
y 123
z 246

# (状況13)


主記憶
CPU
プログラム
(step1) z←0
(step2) x≦0かどうかを調べる
(step3) x≦0なら(step7)へジャンプ
(step4) z←z+y
(step5) x←x-1
(step6) (step2)へジャンプ
(step7) 停止
命令を入れる領域
(step2) x≦0かどうかを調べる
プログラムカウンタ
判断結果 (xの符号)
零
数値を記憶する領域
x 0
y 123
z 246


主記憶
CPU
プログラム
(step1) z←0
(step2) x≦0かどうかを調べる
(step3) x≦0なら(step7)へジャンプ
(step4) z←z+y
(step5) x←x-1
(step6) (step2)へジャンプ
(step7) 停止
命令を入れる領域
(step3) x≦0なら(step7)へジャンプ
プログラムカウンタ
判断結果 (xの符号)
零
数値を記憶する領域
x
y
z
0
123
246

# (状況14)


主記憶
CPU
プログラム
(step1) z←0
(step2) x≦0かどうかを調べる
(step3) x≦0なら(step7)へジャンプ
(step4) z←z+y
(step5) x←x-1
(step6) (step2)へジャンプ
(step7) 停止
命令を入れる領域
(step3) x≦0なら(step7)へジャンプ
プログラムカウンタ
判断結果 (xの符号)
零
数値を記憶する領域
x 0
y 123
z 246

# 3 通常の処理手順の与え方

処理手順を記述する時は、通常、

気まぐれな動き（サイコロを振って 出た目に応じて動作を決める、等）

はさせない。

例3. 1 (最大公約数, ユークリッドの互除法)

2つの正整数の最大公約数を求めるためには、通常、次の数学的事実を利用したユークリッドの互除法と呼ばれる計算手順が用いられる。

命題3. 2　2つの正整数 a, b の最大公約数を gcd(a,b) と表すことにすれば、

(1) a<b なら gcd(a,b) = gcd(a,b−a)

(2) gcd(a,b) = gcd(「b ÷ a」の際の余り, a)

例えば 、156 と 540 の最大公約数 gcd(156, 540) の計算を次のように進めることができる。

$$
\begin{aligned}
\gcd(156,\,540) &= \gcd(\text{「}540 \div 156\text{」の際の余り},\,156) \\
&= \gcd(\quad 72,\quad 156) \\
&= \gcd(\text{「}156 \div 72\text{」の際の余り},\,72) \\
&= \gcd(\quad 12,\quad 72) \\
&= \gcd(\text{「}72 \div 12\text{」の際の余り},\,12) \\
&= \gcd(\quad 0,\quad 12) \\
&= 12
\end{aligned}
$$

# 一般の gcd(a, b) を求める作業 を C言語風に書くと

```
x = a;          // a という場所のデータを x という場所にコピー
y = b;

while (x != 0) {
  next_x = y%x;    //「(yという場所のデータ)
                   //   ÷(xという場所のデータ)」の際の余り
  y      = x;
  x      = next_x;
}

ans = y;
            //  これで ans=gcd(a,b) となっているはず
```

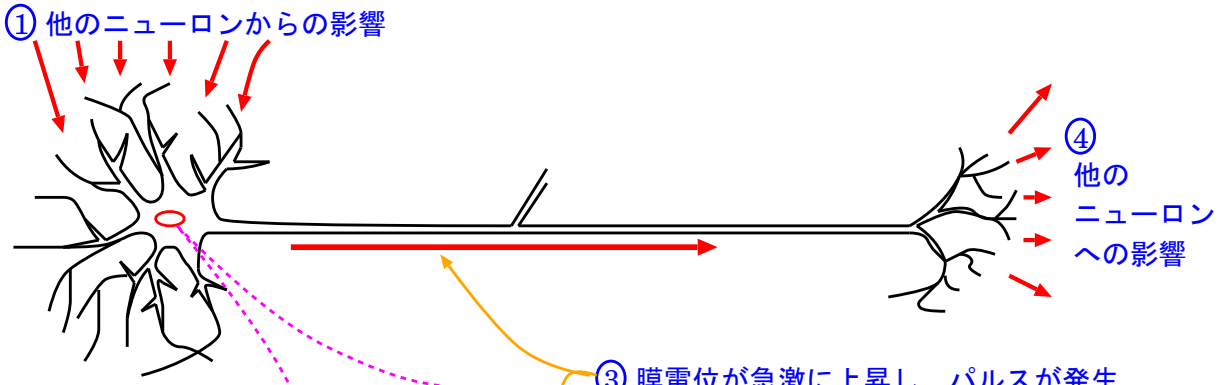
# 4 ニューロ計算

脳の情報処理をお手本にする。

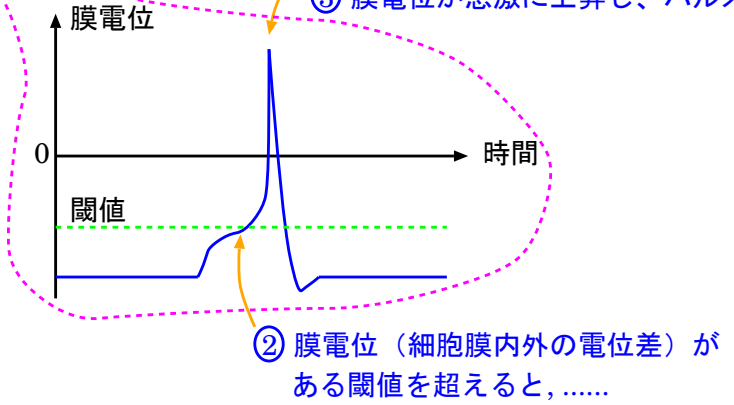
$\longrightarrow$ 基本素子となっているのはニューロン (神経細胞)

## 本物のニューロン：

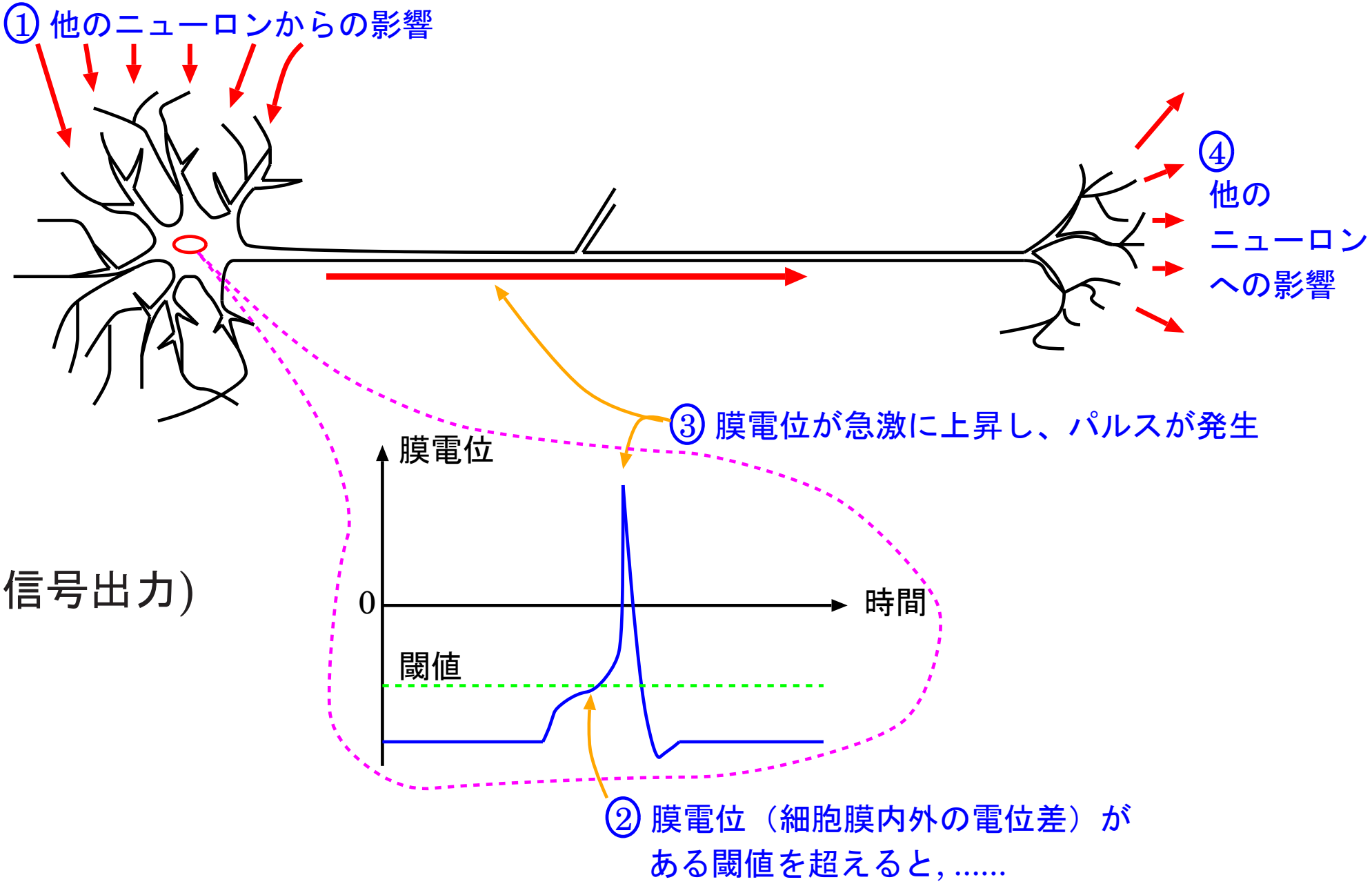
(形状)




(信号出力)

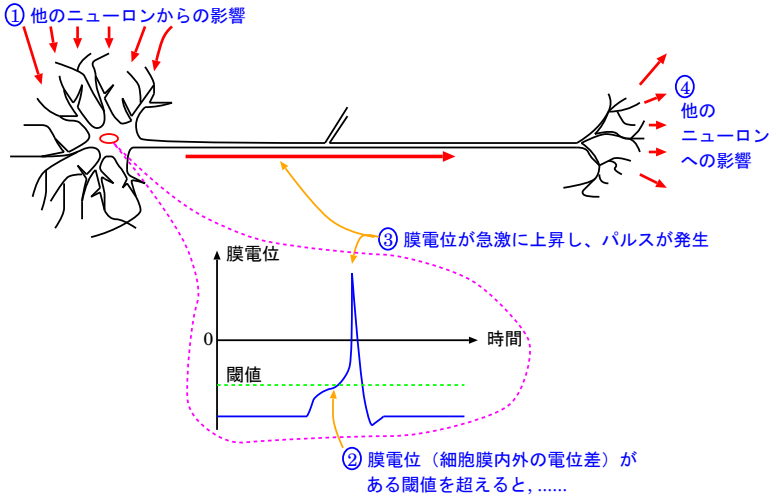


(形状)

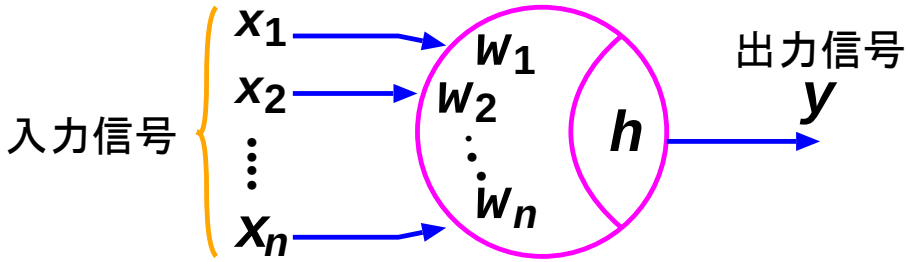

① 他のニューロンからの影響
④ 他のニューロンへの影響
③ 膜電位が急激に上昇し、パルスが発生
膜電位
0
時間
閾値
② 膜電位（細胞膜内外の電位差）がある閾値を超えると, .....

(信号出力)

# 本物のニューロン



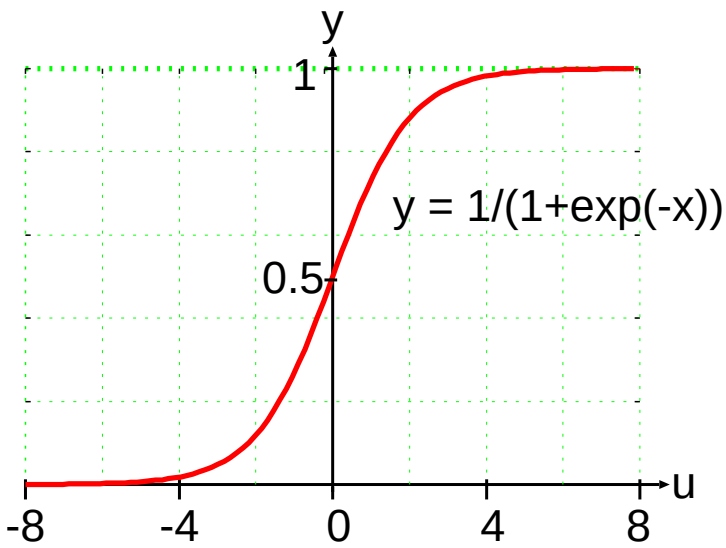

## 擬似ニューロン素子：




出力 y は例えば次の様に定める。

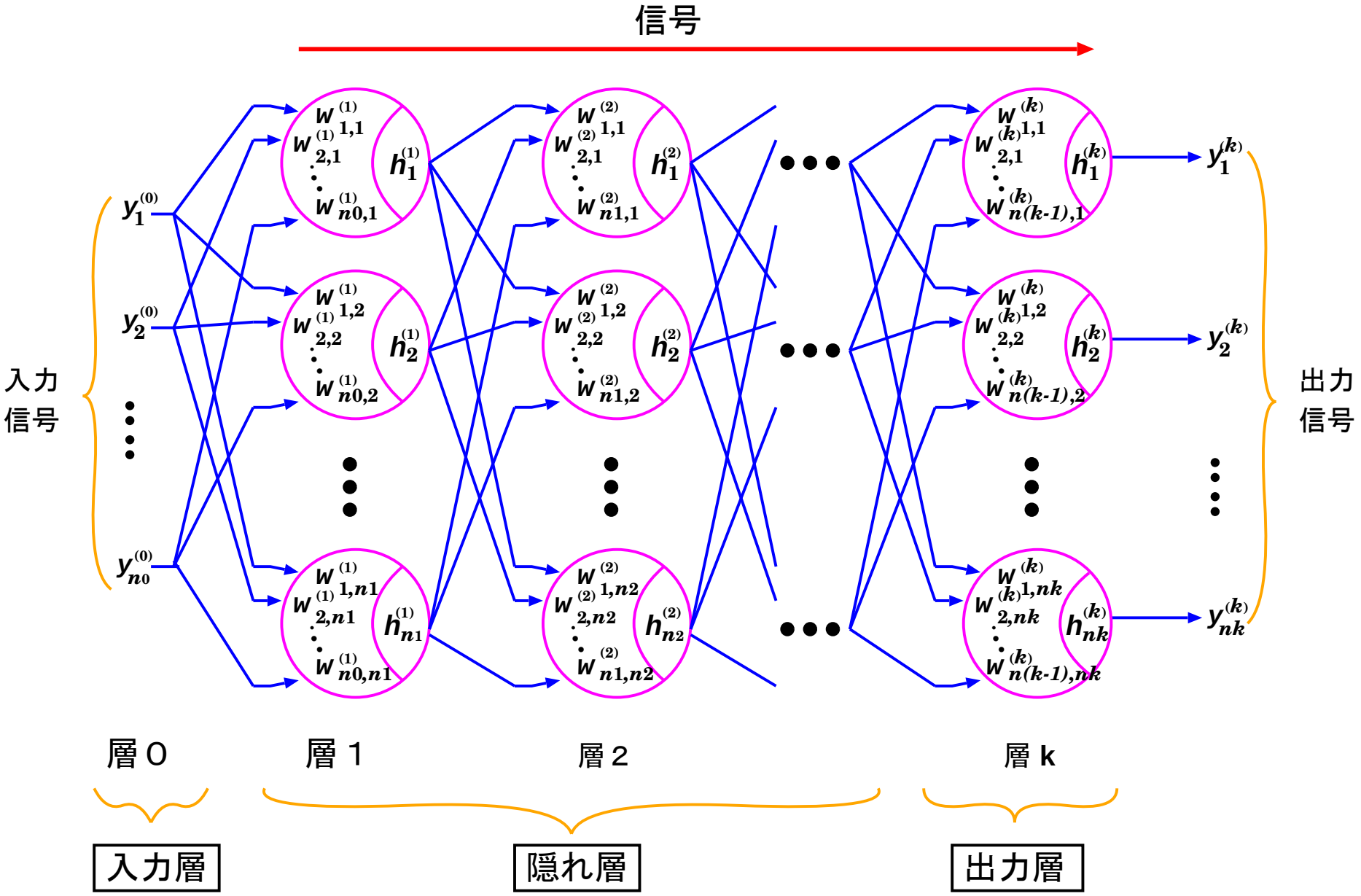
$$u = \sum_{i=1}^{n} x_i w_i - h$$

$$y = \frac{1}{1 + e^{-u}}$$

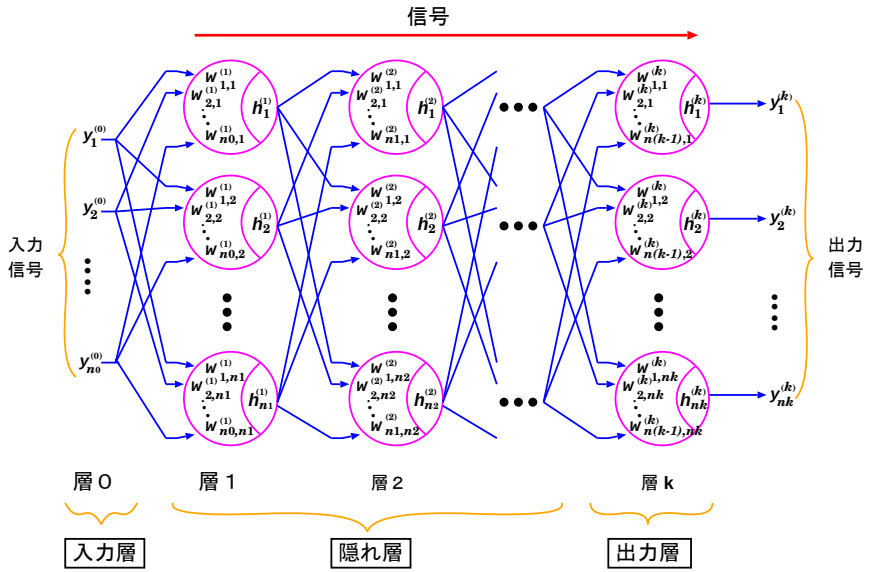


⇨ 擬似ニューロンを組み合わせてネットワークを作る。

例えば、階層的なもの



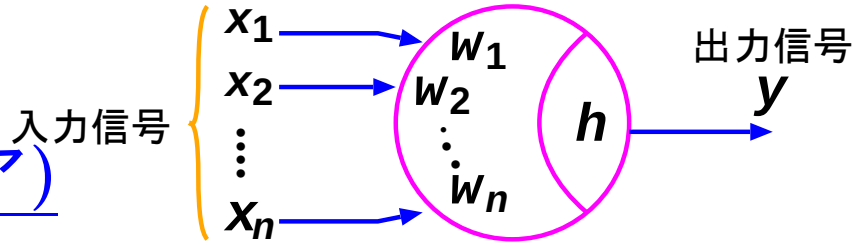
⇨ 擬似ニューロンを組み合わせて
ネットワークを作る。




そして、
ネットワークに何らかの(知的)判断作業を行ってもらえる様に、
内部のパラメータ群をうまく調整 する。

- プログラム内蔵方式のコンピュータにおいて処理手順を構築する作業(プログラミング)に相当。

- ネットワークが目標とする動作をする様に
内部のパラメータを少しずつ変化 させる、
ことを繰り返す。 ⟶ 学習あるいは訓練という。




## 補足 (パラメータ調整の直観的なアイデア)

右図の場合には、$\alpha$ を微小正定数として

入力信号 $x_1, \ldots, x_n$ の色々な与え方に対して、

(1) 実際の出力 $y$ を調べ、

(2) $y$ が目標とする出力 $t$ と違っていたら、

$$w_j \longleftarrow w_j + \alpha \times x_j \times (t - y)$$
$$h \longleftarrow h - \alpha \times (t - y)$$

とする

という訓練作業を繰り返す。

例えば $t=1, y=0$ の場合は 、

$x_j > 0$ の所で $w_j$ が増加、

$h$ が減少、

従って、

$\sum_{i=1}^{n} x_i w_i - h$ が増加

して $y$ が増加する傾向が強くなる。

## 補足 (パラメータ調整の一般的方法)

$$\sum_{\text{c: 入力事例}} \{(\text{cに対する目標の出力}) - (\text{cに対する実際の出力})\}^2$$

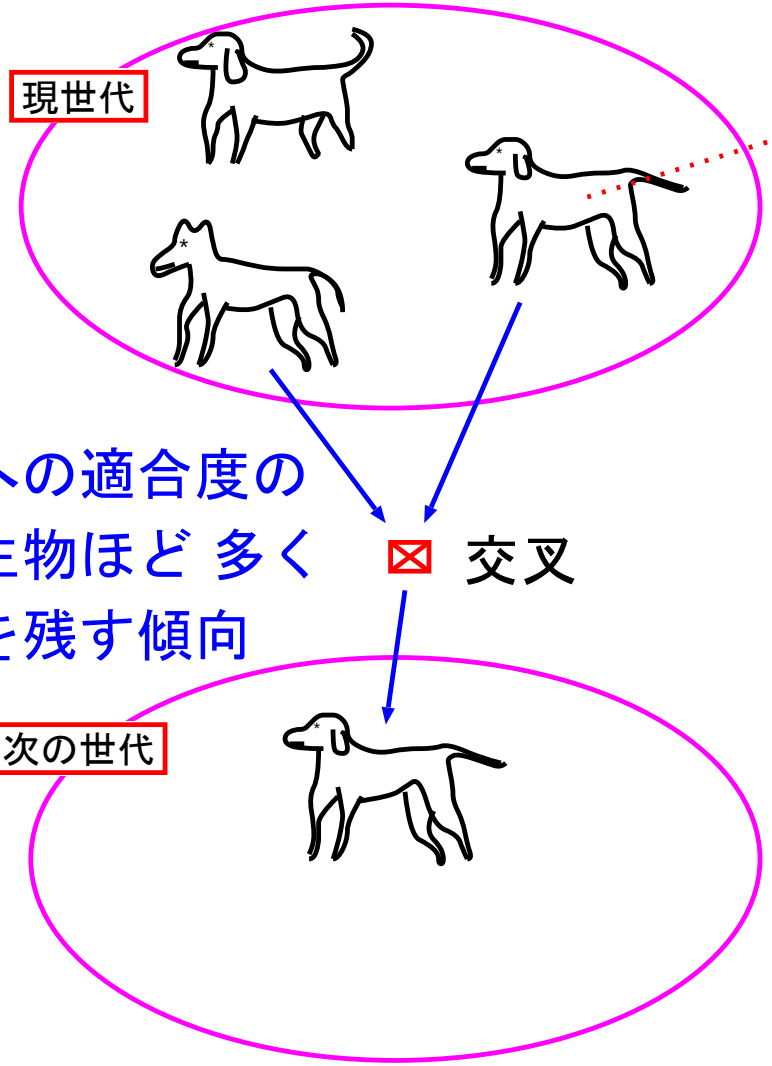
を最小にするためのパラメータ群変更の方向を、

数学的に見つける 。 $\longrightarrow\longrightarrow$ 更新式

ここで必要になるのが、
(偏) 微分等に関する考え方／知見。

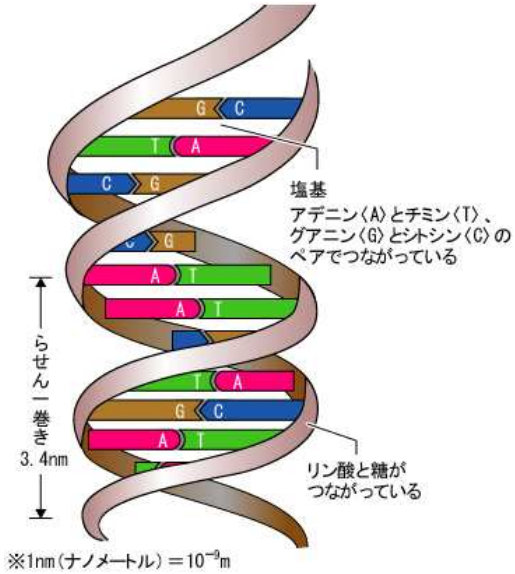
# 5 進化的計算

生物集団の世代交代／進化




遺伝情報

染色体 …… DNA .. 二重らせん構造




次世代に
受け継がれる

⇨ 生物集団の世代交代／進化をお手本にして、
求めたいもの(解)を試行錯誤的に探索する。

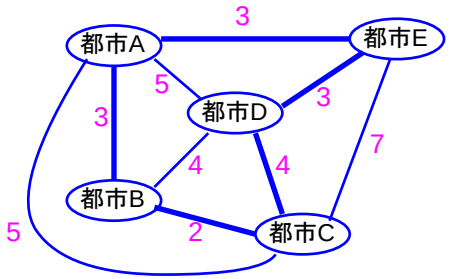
---

問題：○○の中で△△を最大にするものを見つけたい。
(最小)

例えば、

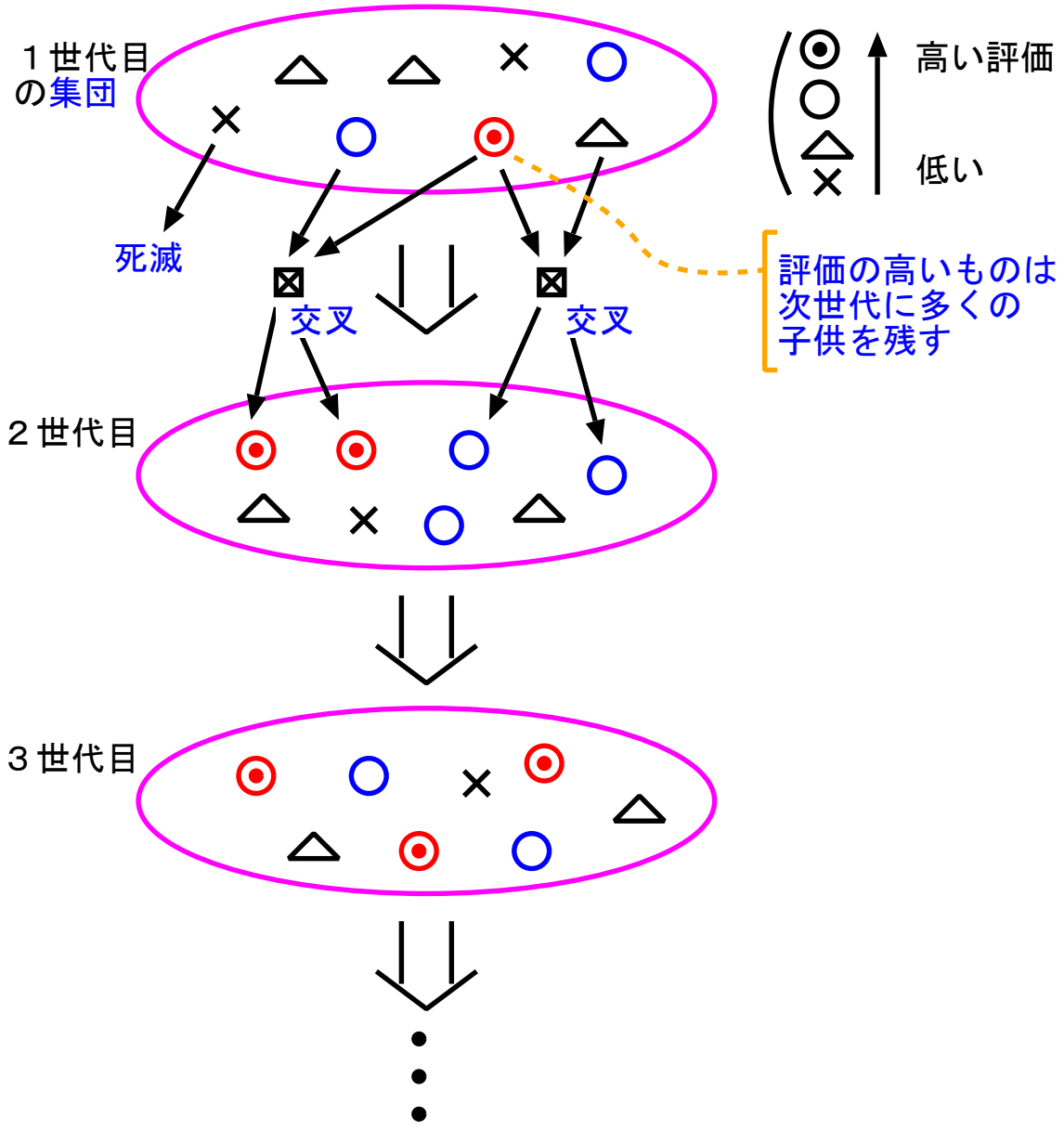
- $-1 \leqq x \leqq 2$ の範囲で、
  $f(x)=x\ \sin(10\pi x)+1$ を最大にする $x$ (の近似値) を探す。

- 与えられた都市を丁度1回ずつ経由して元の場所に戻る経路の内、
  最短のものを探す。



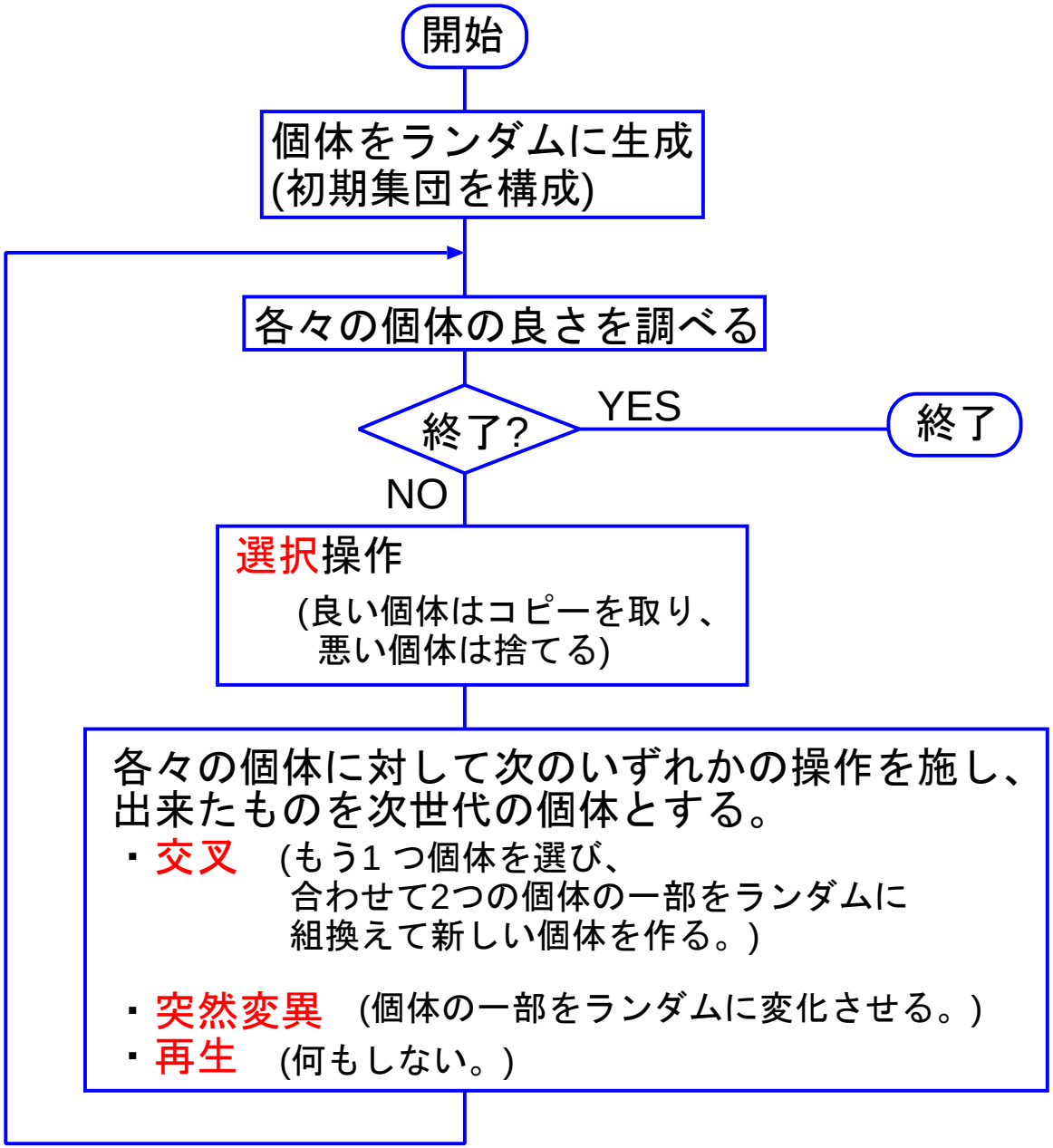
# 生物集団の世代交代／進化をお手本にして解を試行錯誤的に探索

- 具体的な解の候補を集団中の個体と見なす。
- 各時点で、複数の個体(解の候補)に生存競争させる
- 低い評価の個体は高い確率で死滅させる。(選択・淘汰)
- 高い評価の個体を変形(交叉、突然変異)して次世代の個体を生成する。

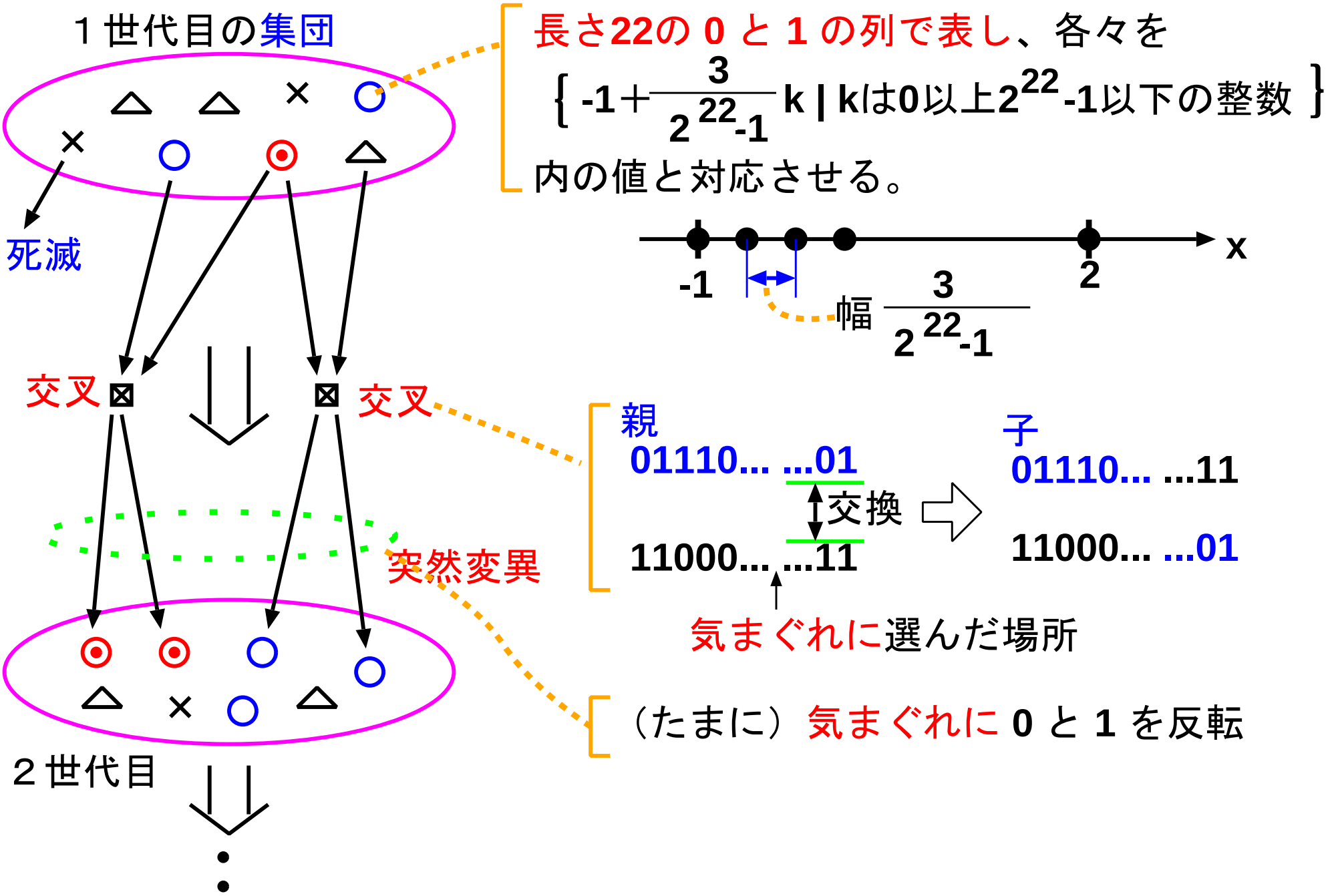



この様な探索をコンピュータ内で行う

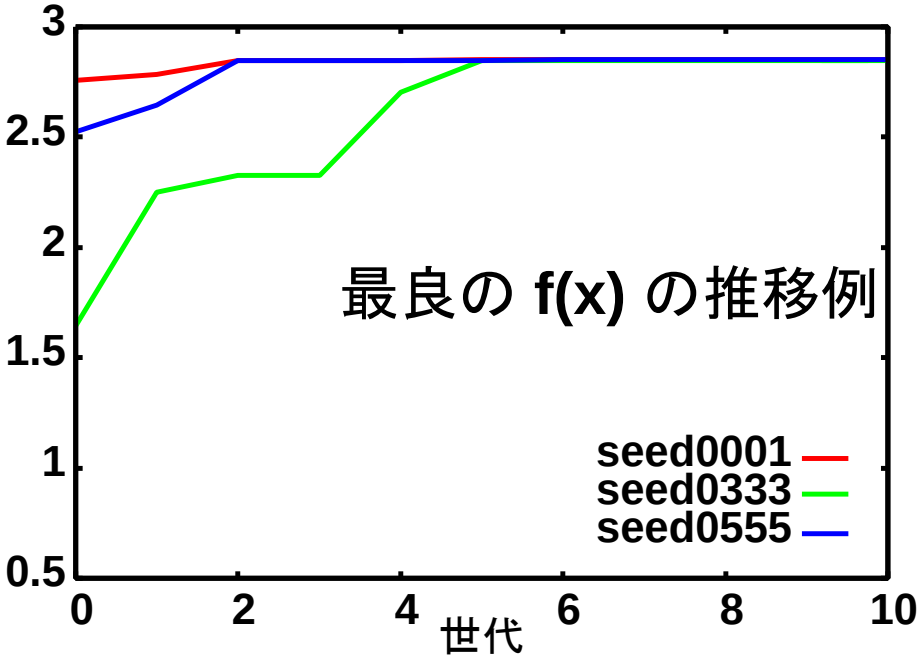
# 進化計算の処理の流れ

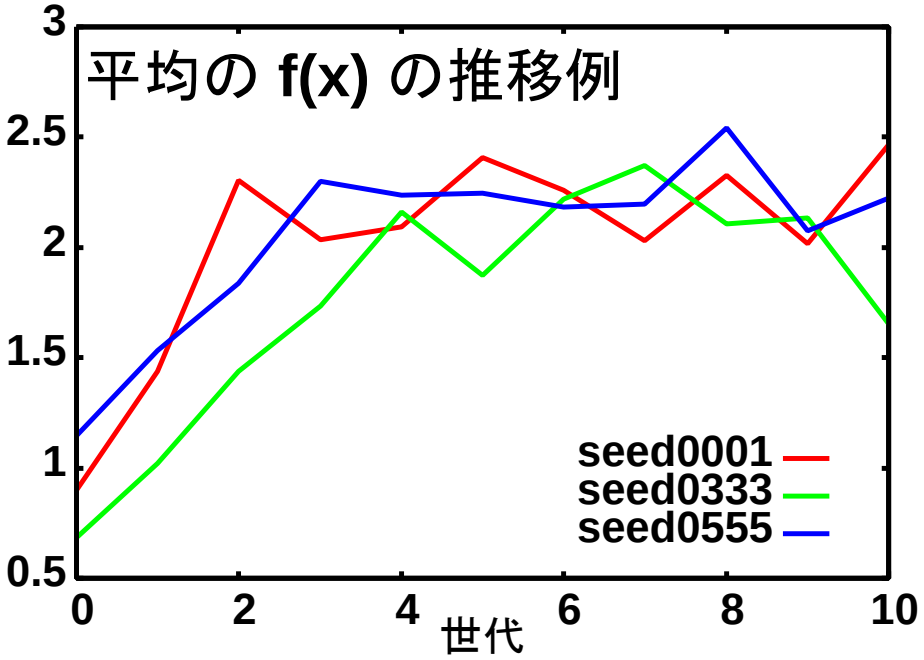

開始
個体をランダムに生成
(初期集団を構成)
各々の個体の良さを調べる
終了?
YES
終了
NO
選択操作
(良い個体はコピーを取り、
悪い個体は捨てる)
各々の個体に対して次のいずれかの操作を施し、
出来たものを次世代の個体とする。
・交叉 (もう1 つ個体を選び、
合わせて2つの個体の一部をランダムに
組換えて新しい個体を作る。)
・突然変異 (個体の一部をランダムに変化させる。)
・再生 (何もしない。)

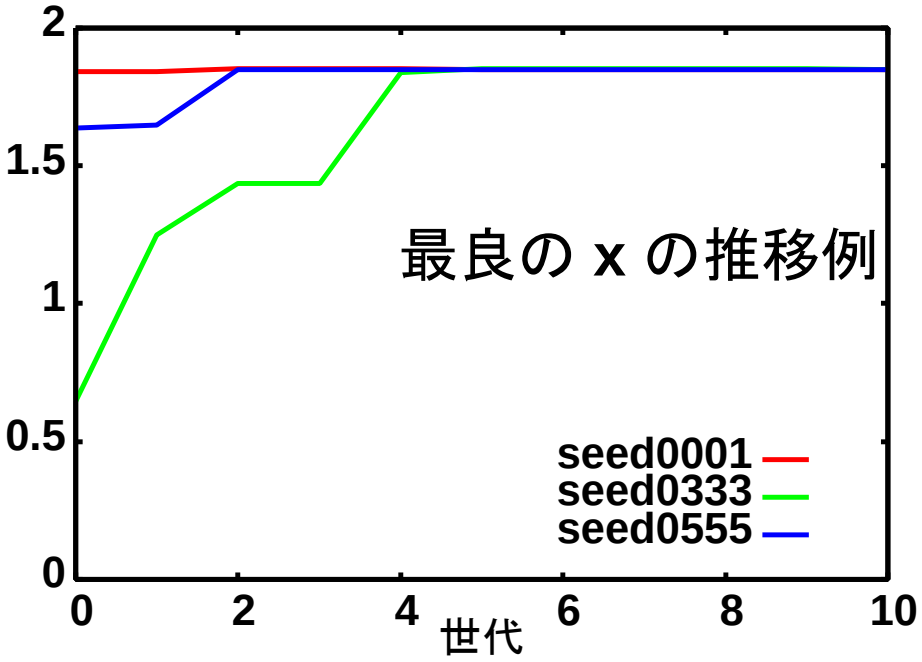
# 例 ($f(x)=x\sin(10\pi x)+1$ を最大にする x を探す, $-1 \leqq x \leqq 2$)

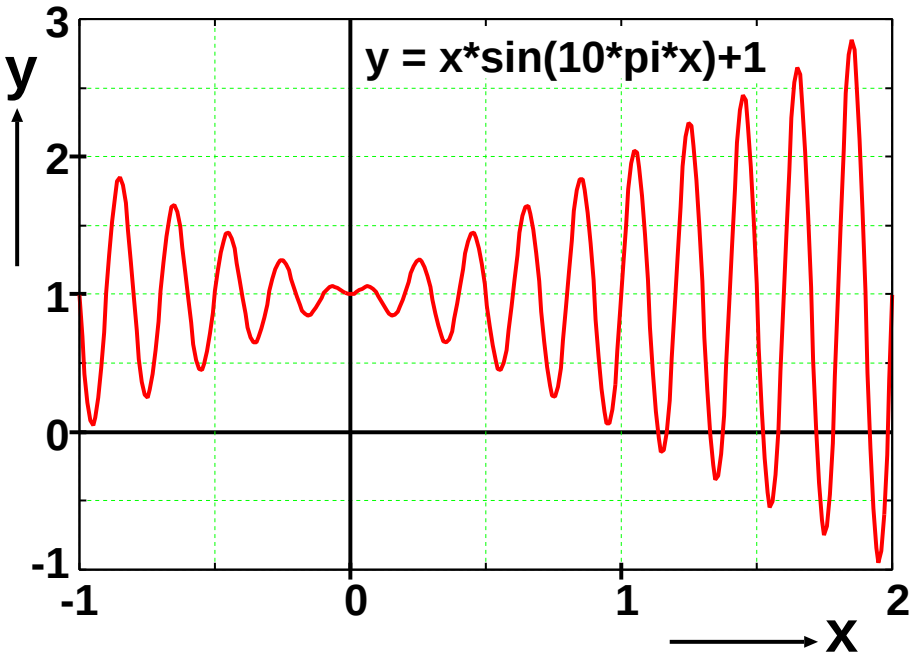


# 探索結果:


最良の f(x) の推移例
seed0001
seed0333
seed0555
世代


平均の f(x) の推移例
seed0001
seed0333
seed0555
世代


最良の x の推移例
seed0001
seed0333
seed0555
世代


y = x*sin(10*pi*x)+1
y
x

# 進化計算の利点

- 応用範囲が広い。

（理由 求めたい解の形について何も分からなくても、望ましい振舞をするかどうかの評価が出来れば適用可能。）

- 我々の思いもつかない解が見つかることがある。

⇨ PCを並列に動かして同時に数百万個の候補解を探索することによって、特許レベルのものを発見しようとする研究者もいます。

# 6 まとめ

現在当然の様に使われている**コンピュータの方式**も、
それが**唯一のものではない**。

基本原理や各々の応用分野(情報系以外も含む)において

- **新しい可能性**の探求、
- 従来方式の**改良**、

  ・・・・・・・・・・・・・・・・・・・・

さらには、

- **新しい応用分野**でのコンピュータ利用、

が試みられている。

⇒ 情報工学科

その前に、
現在の技術を充分に理解する必要もある