コンピュータアーキテクチャII

第1回

- お久しぶりです。充実した夏休みを過ごせましたでしょうか？後期も引き続きよろしくお願いします。
- 進め方は前期と同じですので、ガイダンスは特に行いません。
- 基本的には評価は前期同様毎回の課題、試験の2つで評価を行うことになるかと思います。

それでは早速授業に入っていきましょう

- 第１回は前期の復習をしたいと思います。
- 重要な部分をピックアップし、再掲しました。
  前期に学んだことを思い出してみてください。
  少し分量が多いかもしれませんが、新しい内容は全くないので、自分のペースで復習してみてください。
- 全て前期のスライドの再掲になります。ところどころ空欄を設けています。Formsで回答してください。
- 一応Formsはクイズになっているため、送信ボタンを押すと正解か不正解かが出ます。ただ、ちょっとした入力の誤差もあるので参考程度にしてください。（成績には関係ありません。どれくらい身についているかの参考にします）

それではがんばっていきましょう！

コンピュータの構成要素

空欄1~7をFormsに入力しましょう。

1入力装置

データを入力する。
例：マウス、キーボード、バーコードリーダー、カード読み取り機…

2主記憶装置(メモリ)

計算に必要なデータを記憶する
（数値、命令…）

3補助記憶装置

＿＿＿＿装置は記憶容量が小さいのでこちらにも記憶し、必要に応じて＿＿＿＿装置に移動させる

7中央処理装置（CPU）

4制御装置

計算に伴うすべての動きの制御（指示）を行う

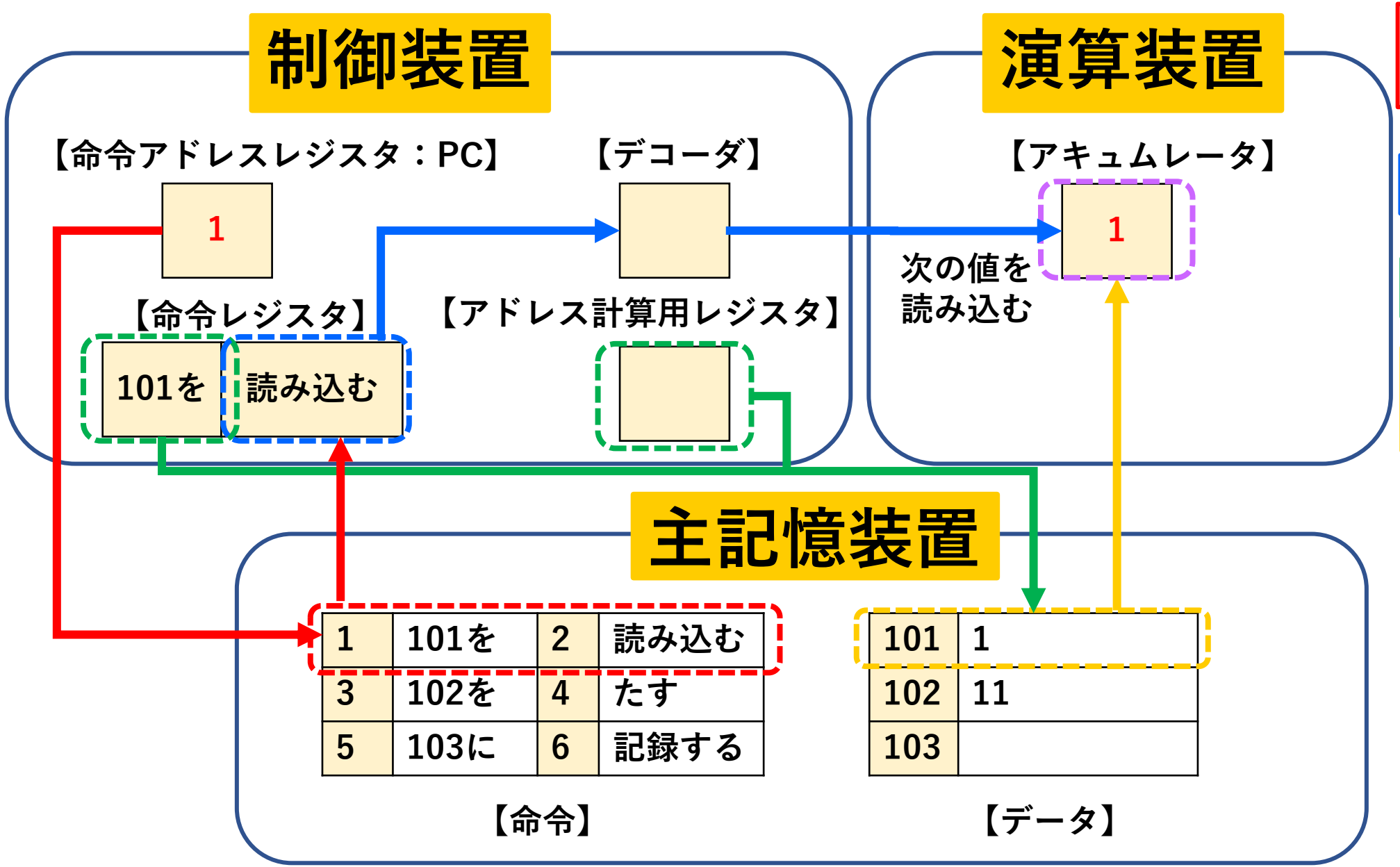
5演算装置

全ての算術計算と論理演算を行う

6出力装置

データを出力する。
例：プリンタ、液晶ディスプレイ…

空欄8~13をFormsに入力しましょう。



①8命令の取り出し
（9フェッチ）

②命令の10デコード
or解読

③11アドレス計算

④12データの
取り出し

⑤13命令の実行

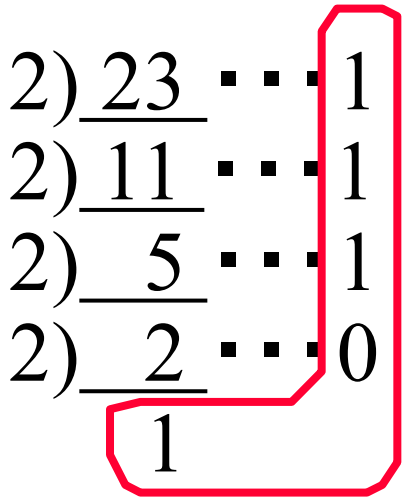
⇩

①に戻る

逐次制御

10進数→2進数

- 23_{10}を2進数で表現すると？？



下から順に並べる

10111_2

基数変換

【練習問題】

55_{10}を2進数で表現すると？

もう一度練習問題を解いて解答をForms14に入力してください。

小数の場合は？

- 2進数→10進数
  - 例：$10.101_2 = 2^1 \times 1 + 2^0 \times 0 + 2^{-1} \times 1 + 2^{-2} \times 0 + 2^{-3} \times 1 = 2.625_{10}$

$\frac{1}{2^1}$　$\frac{1}{2^2}$　$\frac{1}{2^3}$

- 10進数→2進数
  - 例：13.75_{10}

$13_{10}=1101_2$

1101.11_2

$$\begin{array}{r} 0.75 \\ \times \quad 2 \\ \hline 1.50 \end{array}$$

$$\begin{array}{r} 0.5 \\ \times \quad 2 \\ \hline 1.0 \end{array}$$

$0.75_{10} = 0.11_2$　順に並べる

固定小数点数の引き算はどのように行うのでしょうか？

思い出してから次のスライドへ進みましょう。

固定小数点数の引き算はどのように行うのでしょうか？

- 2 の補数を負の値として、負の値を足すことで行う。

$1101\ 0110_2$の1の補数及び2の補数を求めよ

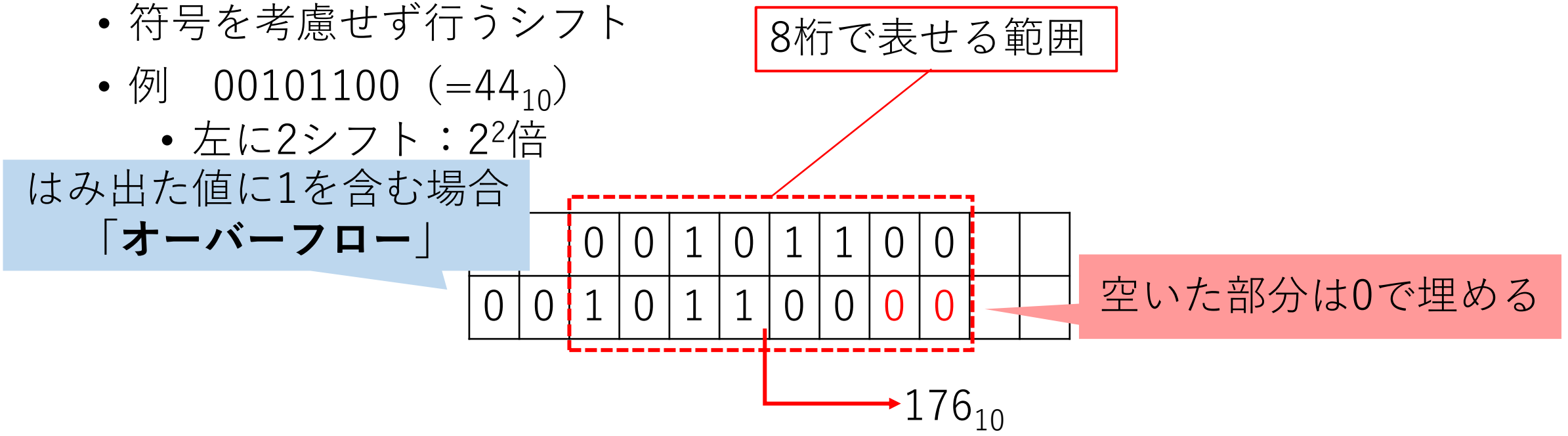
1の補数：15 1111

2の補数：16 10000

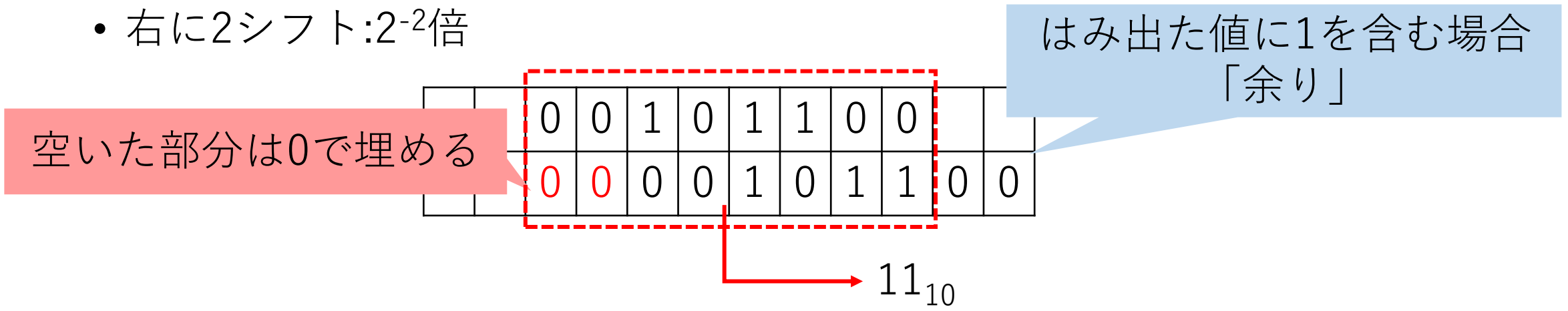
それぞれ求めてFormsに入力しましょう。

論理シフト

- 符号を考慮せず行うシフト
- 例　00101100（$=44_{10}$）
  - 左に2シフト：2^2倍



  - 右に2シフト:2^{-2}倍



例題

再度例題を解き、Forms17,18に解答を2進数で入力しましょう。

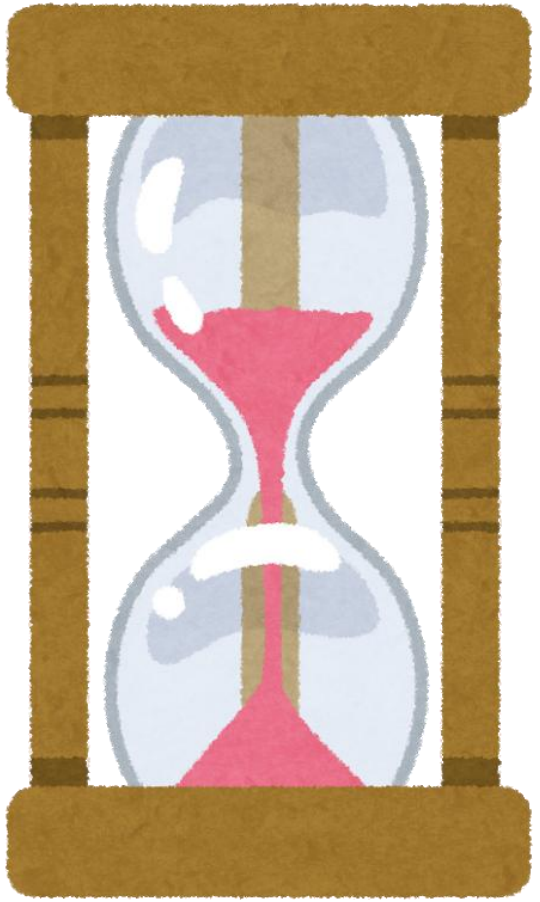
01100011 $(=99_{10})$を次の通りに論理シフトしなさい。

- 左に2シフト：2^2倍→17へ入力

		0	1	1	0	0	0	1	1		
		1	0	0	0	1	1	0	0		

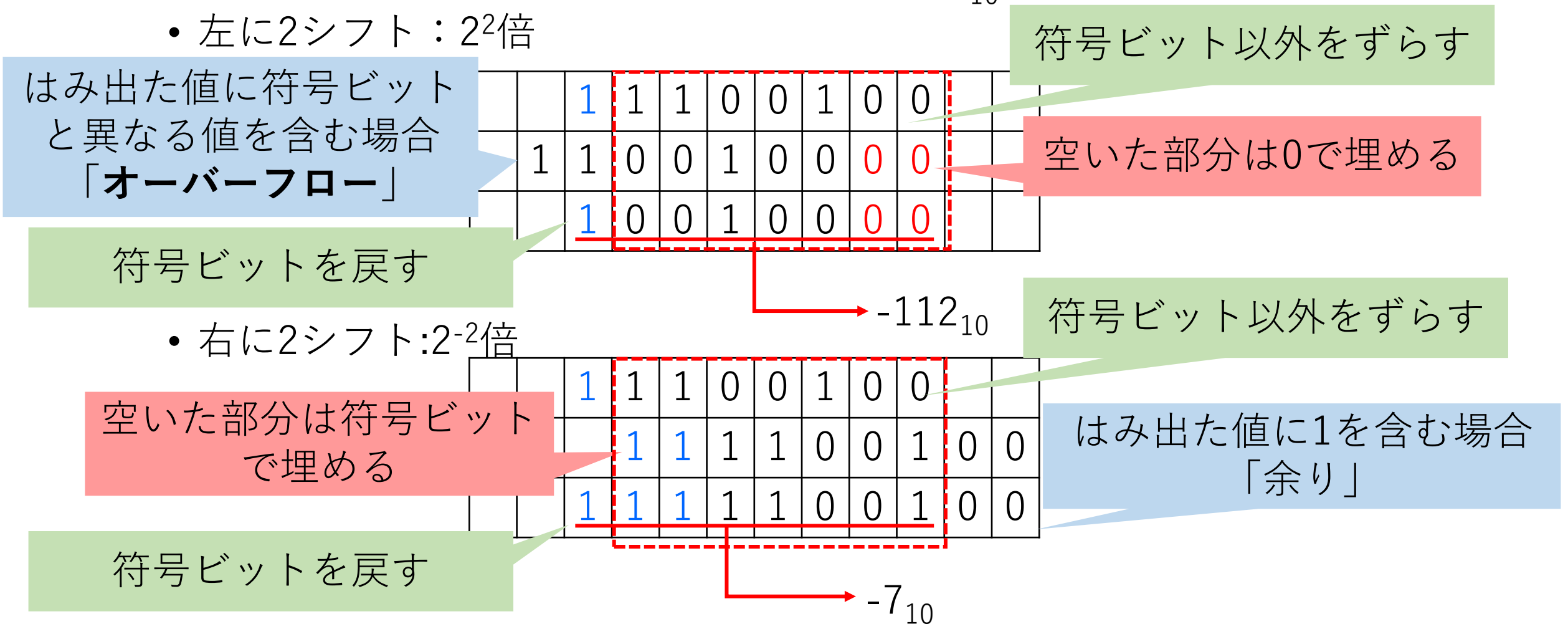
- 右に2シフト:2^{-2}倍→18へ入力

		0	1	1	0	0	0	1	1		
		0	0	0	1	1	0	0	0		



算術シフト

- 符号を考慮して行うシフト
- 例　1(=符号ビット0:正 1:負）1100100（=-28_{10}）
  - 左に2シフト：2^2倍



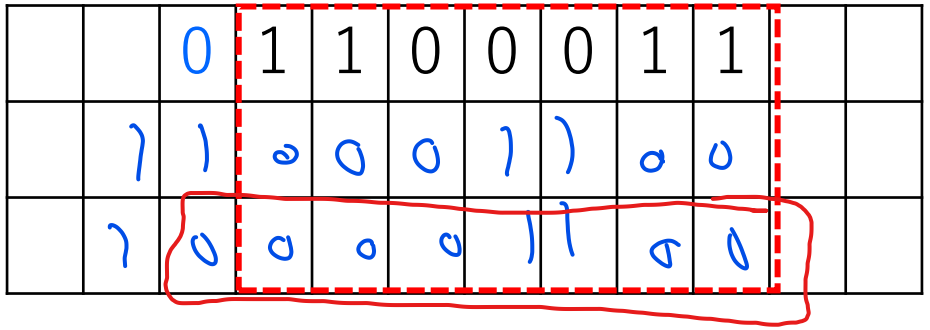
  - 右に2シフト:2^{-2}倍

例題

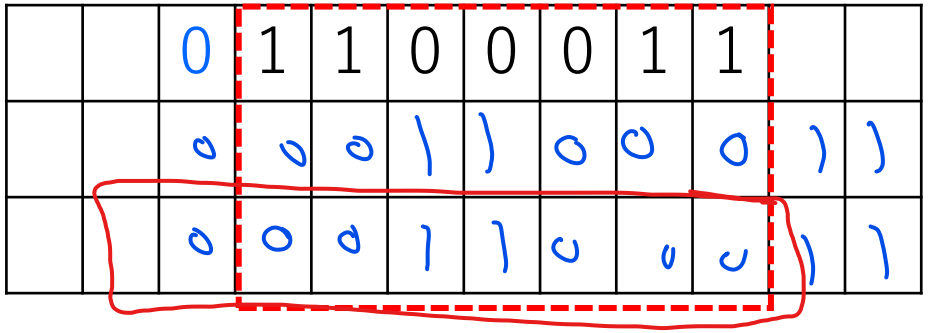
再度例題を解き、Formsの19,20に解答を2進数で入力しましょう。

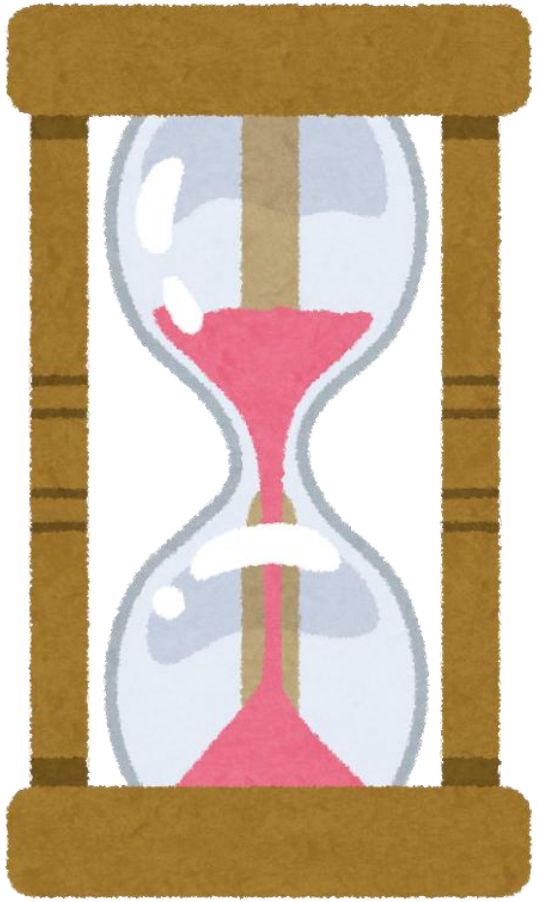
0(=符号ビット0:正 1:負）1100011 （$=99_{10}$)を次の通りに算術シフトしなさい。

- 左に2シフト：2^2倍→19に入力



- 右に2シフト:2^{-2}倍→20に入力

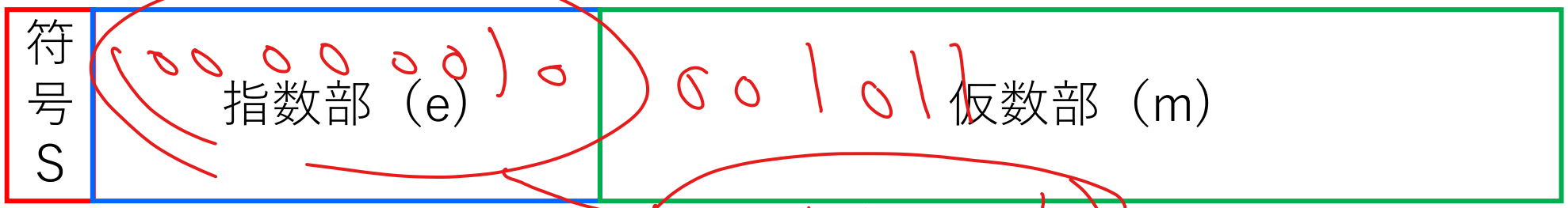




浮動小数点数の表現方法

- 例： 1.001011×2^{2}を32ビットで表すには…

ここに書かれていた部分を省略しました。単精度で32ビットで表したものをFormsの21に入力しましょう。



IEEE(Institute of Electrical and Electronics Engineers:米国電気電子技術協会、通称アイトリプルイー）の浮動小数点形式が標準形式として発表されています

[単精度]S: 1 bit　e:8bit　m:23bit

[倍精度]S: 1 bit　e:11bit　m:52bit

※単精度：通常の32ビットで表現
倍精度：仮数部のビット数を増やして数字の精度を上げたもの（64ビット）

命令の形について

- 命令ってどんな形になっているのでしょうか？

  先ほどの例では…

101を	読み込む

  場所　＋　命令の内容　でした

  わかりやすくするために場所を先にしましたが、実際は

読み込む	101を

  のように　命令の内容　＋　場所　となっています

命令部　　**アドレス部**

何の命令化を表す部分を
「**オペコード**」と言います

各アドレスの示すデータを
「**オペランド**」と言います

命令の形式

- 命令部：CPUの設計時に表現方法が決められている（CPU固有）
  - $2^{ビット数}$種類の処理を指定可能

例：3ビットの場合
2^3種類＝８種類

000	加算
001	減算
010	読み込む
011	書き込む
100	…
101	
110	
111	

命令の形式

- アドレス部

一つの命令に含まれるアドレスの数によって

0アドレス命令、　1アドレス命令、　2アドレス命令、　3アドレス命令…という形式になる。

例：「101の値と102の値を足して103に書き込む」という命令

0アドレス命令

（1番上をアキュムレータに）読み込む
（1番上とアキュムレータを）たす
（アキュムレータを一番下に）書き込む

暗黙の場所

1アドレス命令

読み込む	101を
たす	102を
書き込む	103に

2アドレス命令

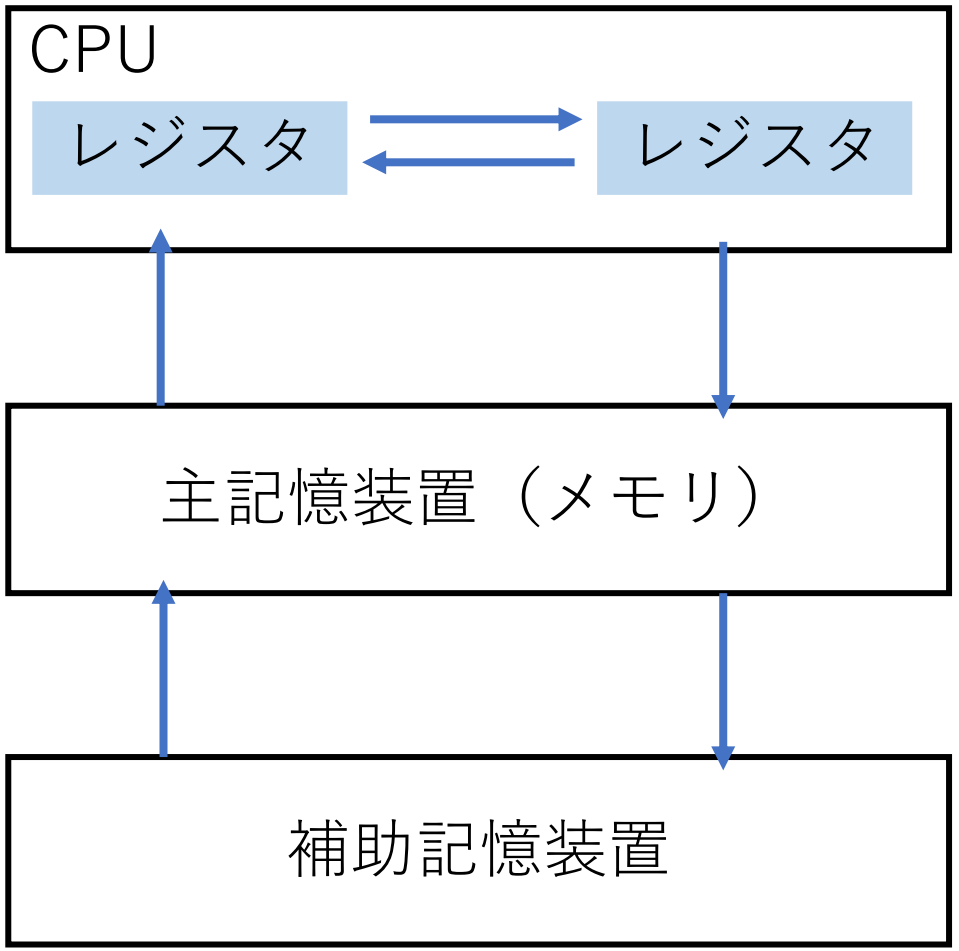
移動する	101を	103に
たす	102を	103に

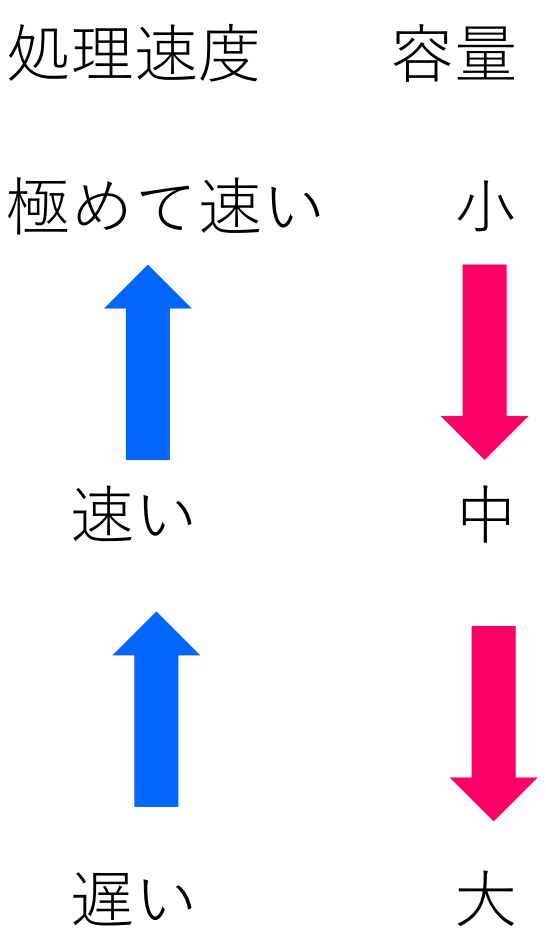
3アドレス命令

たす	101と	102を	103に

「場所」ってどこのこと？

- レジスタの場合とメモリの場合がある





必要なものを
必要な時に
上の階層に
取り出して使う！

空欄22,23をFormsに入力しましょう

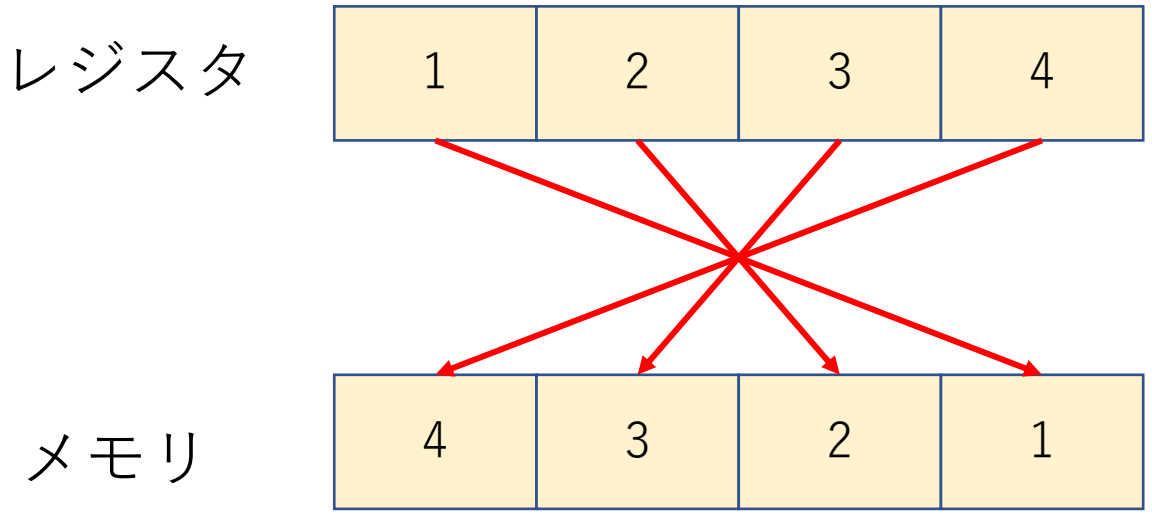
レジスタ→メモリ

- レジスタとメモリの間のデータ移動は1バイトずつ行います（1バイトずつメモリのアドレスが振られている）

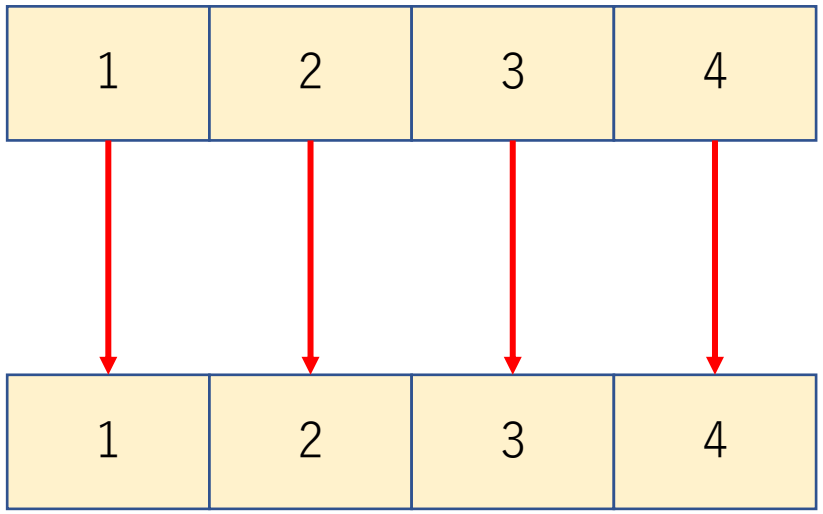
  でも…1バイトずつ移動したものはどう並べる？

  →2パターンあります。（=**バイトオーダ**）

  いずれも命令時は一番最初のアドレスを指定します



22リトルエンディアン
（データを後ろから順に格納）



23ビッグエンディアン
（データを順番通りに格納）

命令の種類

- 計算：基本的にレジスタに対して行う
  - 算術演算
  - 論理演算
- コピー（データ転送）：メモリ⇔レジスタ、レジスタ⇔レジスタ
  - メモリ→レジスタ：ロード命令
  - レジスタ→メモリ：ストア命令
- 実行制御（分岐、比較）
  - 実行するアドレスを変える
  - 条件によって次のアドレスを変える
- その他
  - 入出力制御命令，例外処理のためのOS呼び出し等

直接アドレス指定方式

- アドレス部の値がそのまま有効アドレスになっている方式

読み込む	101を

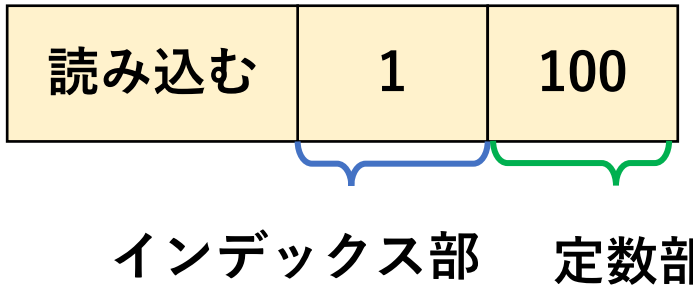
101	1
102	11
103	

読み込まれる値はいくつでしょう？Forms24に入力しましょう。

処理なし
（これまでの例で挙げていた方式）
【デメリット】
命令長が長くなる。
プログラムを置く場所が変わるとプログラムを変えなければいけない。

インデックスアドレス指定方式（指標アドレス指定方式）

- インデックスレジスタの値を定数部の値に加算する（=インデックス修飾）ことで実行アドレスを求める



101	1
102	11
103	

読み込まれる値はいくつでしょう？Forms25に入力してください。

0	1
1	2
2	3

→1,2

インデックスレジスタ

【メリット】
連続したデータに同じ処理を繰り返すとき便利。
（インデックスレジスタの値を増やしながら同じ命令を繰り返すことが可能）

ベースアドレス指定方式

- ベースレジスタの値をアドレス部の値に加算することで実行アドレスを求める

読み込む	**1**

101	**1**
102	**11**
103	

読み込まれる値はいくつでしょう？Forms26に入力してください。

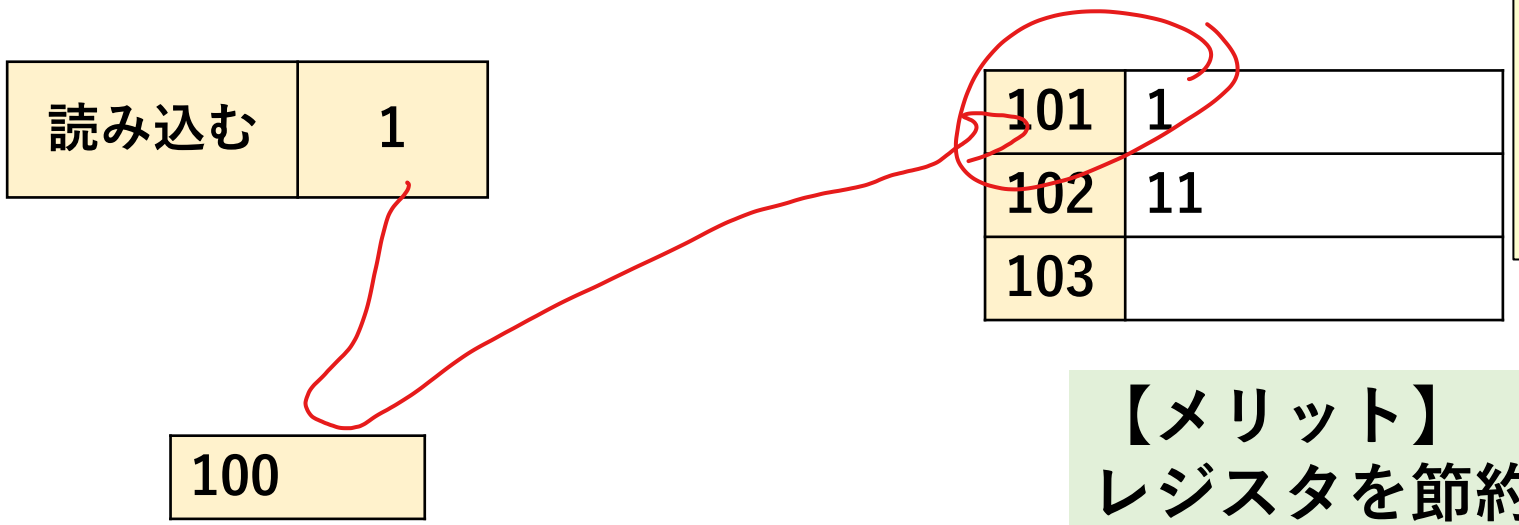
100

ベースレジスタ
←プログラムの先頭のアドレスを記憶している

【メリット】
プログラムのアドレス部を変えずにプログラムを置く場所を変えることが可能

相対アドレス指定方式

- プログラムカウンタの値ををアドレス部の値に加算することで実行アドレスを求める



ベースアドレス指定方式と仕組みは同じ。
ベースレジスタの代わりにプログラムカウンタを使う。

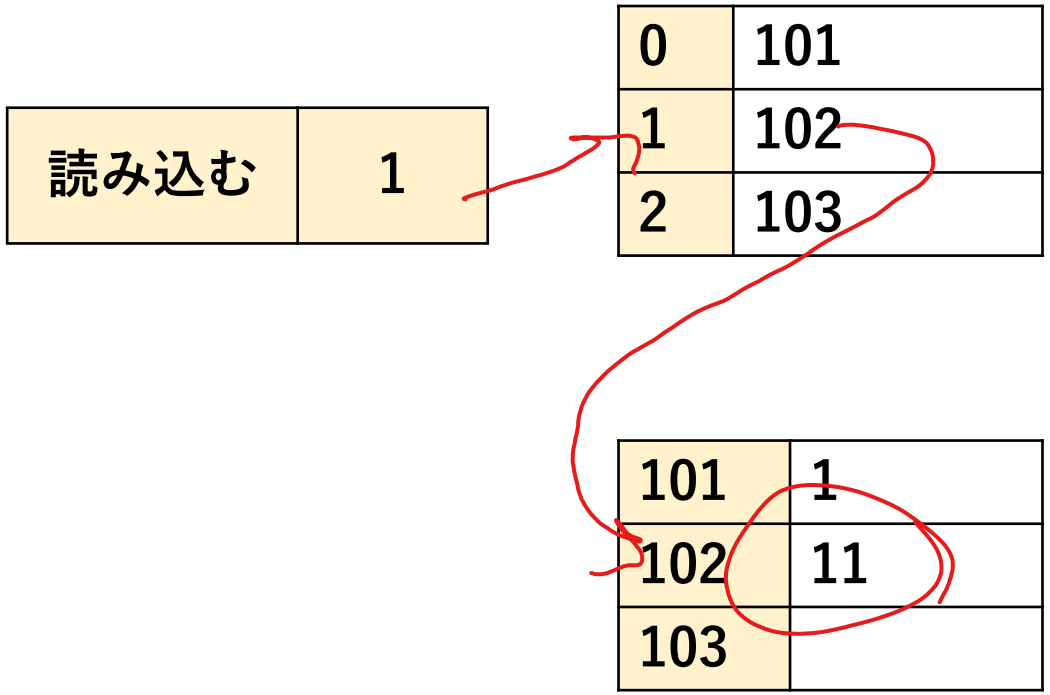
読み込まれる値はいくつでしょう？Forms27に入力してください。

【メリット】
レジスタを節約できる。
ベースレジスタを持たなかったり、汎用レジスタの数が少ない
ミニコンピュータやマイコンなどが使っている。

間接アドレス指定方式

- アドレス部に指定された領域に実行アド

読み込まれる値はいくつでしょう？Forms28に入力してください。



プログラムを変えなくてもここのデータを変えればよい。

【メリット】

メモリを使えばよいからレジスタを多く持たなくても使える。

【デメリット】

メモリに複数回アクセスするから速度は遅い。

最近はあまり使われない。（レジスタは十分にあるから）

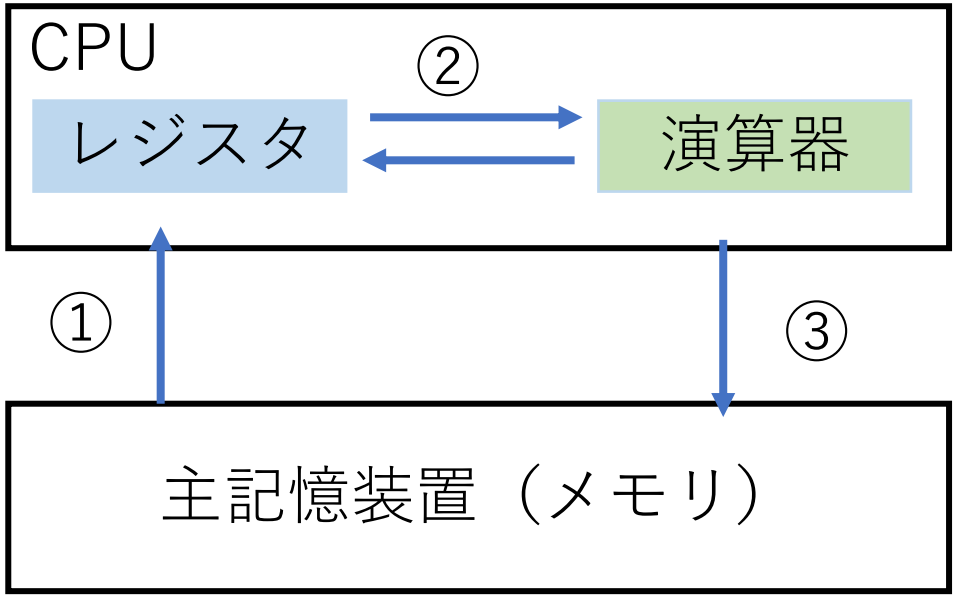
即値アドレス指定方式

- アドレス部の値をそのままオペランドとして用いる方式

アキュムレータ

定数を指定するときなどに使われる。
【メリット】
メモリにデータを記憶する容量を節約できる。
メモリにアクセスしないから処理速度は速い。
【デメリット】
値を変えるときはプログラムを変える必要がある。

コンピュータの命令実行の流れ



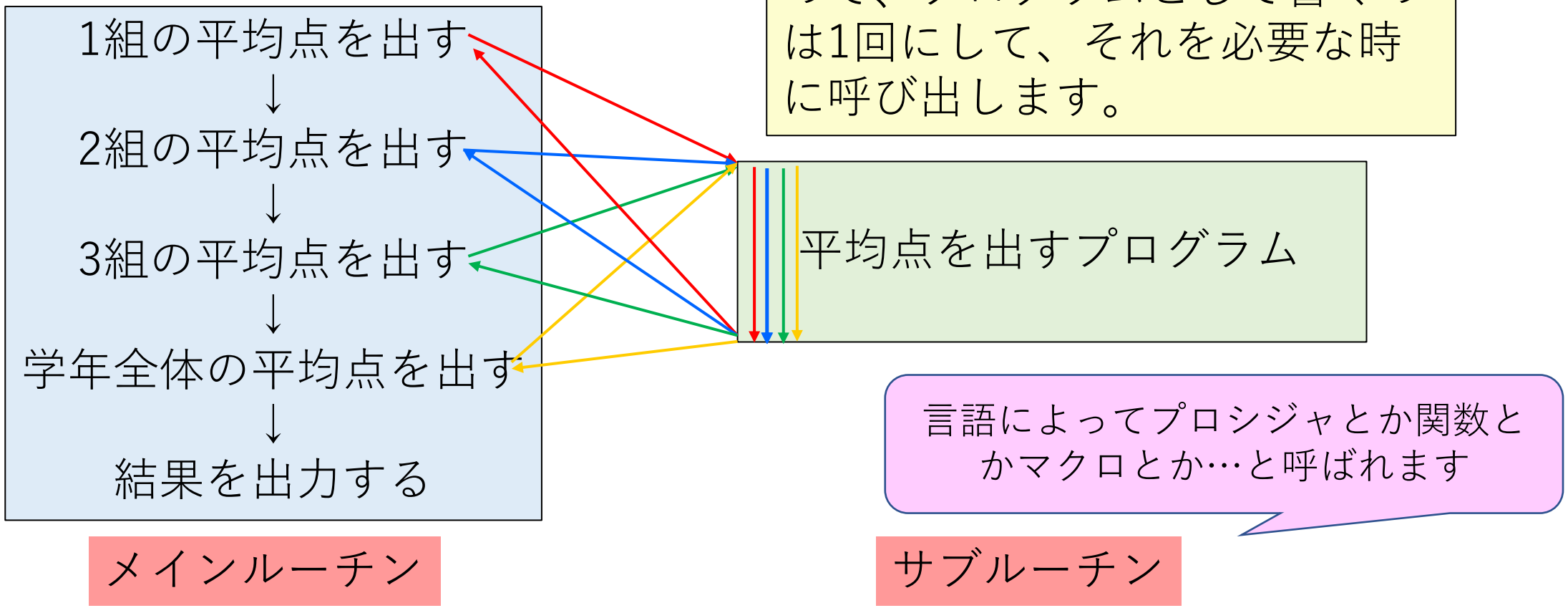
①メモリからデータをレジスタに持ってきて
　＝ロード命令
②それを取り出して何回か演算して
　＝計算
③その結果をメモリに移す
　＝ストア命令
というのが基本的な流れです

①～③の順番とかをコントロール
＝実行制御命令

サブルーチン呼び出しとは

- プログラム上でよく使われる処理を別の場所に作っておき、必要な時に呼び出して実行する命令のこと

例：試験の平均点を出すプログラム



引数の渡し方1:29値渡し

空欄29をFormsに入力しましょう。

____渡し:値をコピーしてを渡すイメージ。（サブルーチン中で更新があってもメインルーチンの値は更新されない）

メインルーチン

```
x=1
y=2
z=3
mean(x,y,z)
```

サブルーチン

```
mean(x,y,z){
x=x+y x=1+2
y=x+z y=3+3
m=y/3 m=6/3
}
```

メインルーチンの
x=1, y=2, z=3

x,y,z が格納されている場所

サブルーチンが用いる変数

101	1
102	2
103	3
104	
105	

値をコピー

1001	1
1002	2
1003	3
1004	
1005	return

引数の渡し方2:30参照渡し

空欄30に入る用語をFormsに記入しましょう。

____渡し:メインルーチンの値を参照して処理する。（サブルーチン中で更新があったらメインルーチンの値も更新される）

メインルーチン

x=1→3
y=2→6
z=3
mean(x,y,z)

サブルーチン

mean(x,y,z){
x=x+y x=1+2=3
y=x+z y=3+3=6
m=y/3 m=6/3
}

メインルーチンの
x=3, y=6, z=3

x,y,z が格納されている場所

サブルーチンが用いる変数

101	1
102	2
103	3
104	
105	

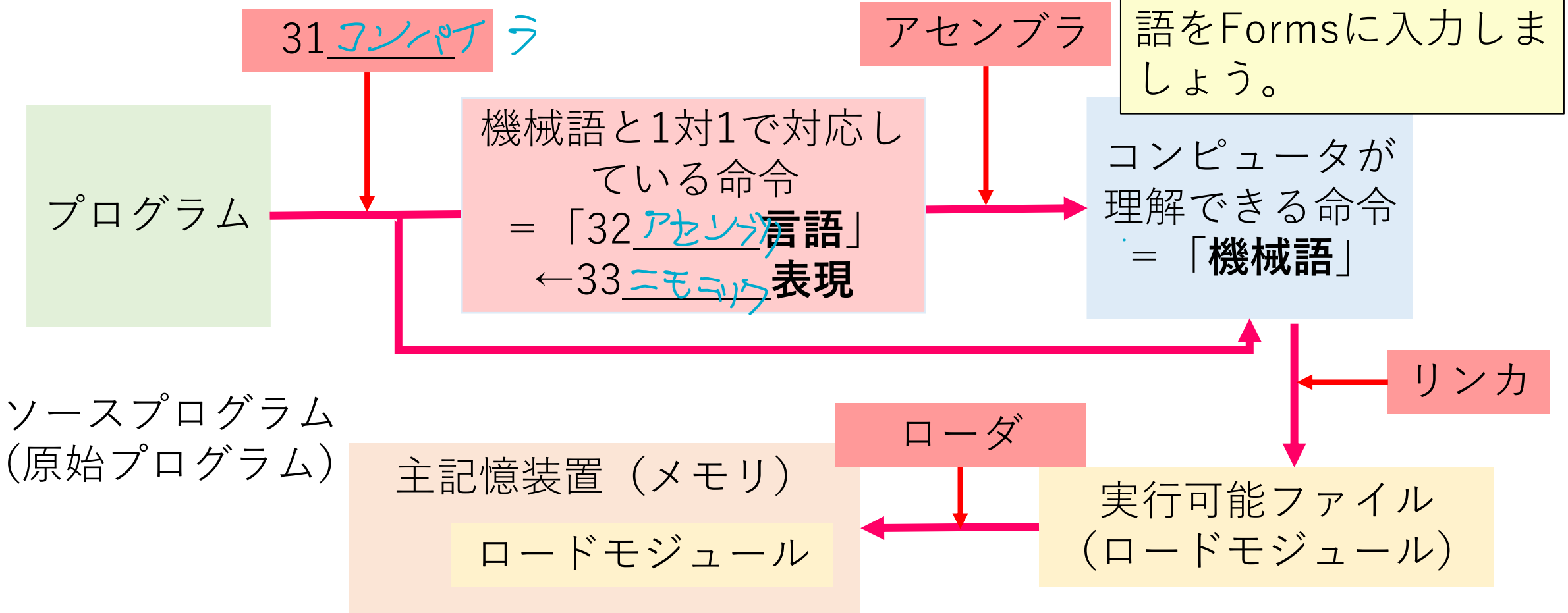
番地をコピー

1001	101の値
1002	102の値
1003	103の値
1004	
1005	return

コンパイラ方式でのプログラム実行手順

ロードモジュールを主記憶装置に読み込ませる作業を「**ロード**」といい、「**ローダ**」とよばれるプログラムが行います。

空欄31~33に入る用語をFormsに入力しましょう。



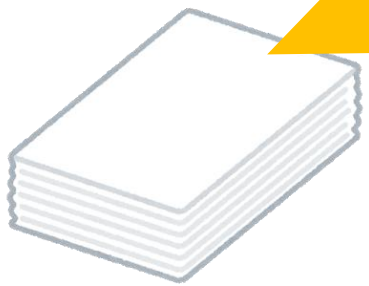
スループット

システムが単位時間あたりに処理できる仕事量のこと。
大きいほど性能がよい。

例えば…
　100人分の書類をコピーしたい

10分

完成！！



1分あたり10人分
コピーできた！！

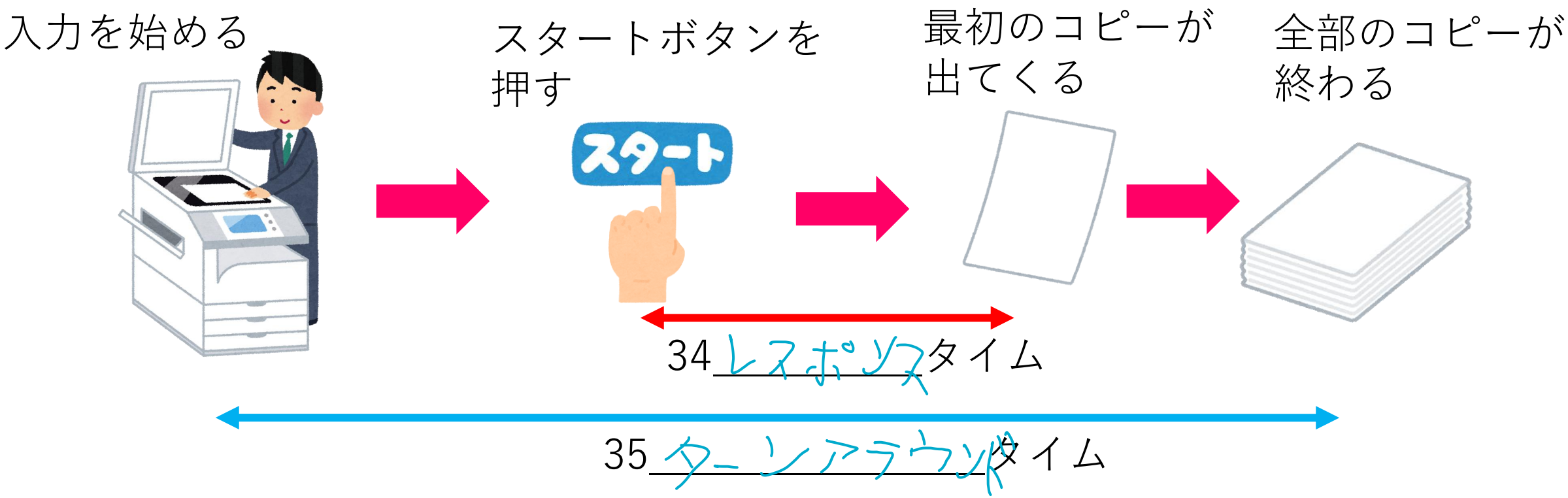
・単位時間当たりなので、必ず「～あたり…」という形式
・CPUやメモリ、処理内容等様々な要因が作用

ターンアランドタイム/レスポンスタイム

・ターンアラウンドタイム
　処理開始の指示を出してから全ての結果が得られるまでの時間

・レスポンスタイム
　処理を依頼し終えてから実際何か応答が返されてくるまでの時間

空欄34,35に入る用語をFormsに入力しましょう。



クロック周波数とは何のことだったでしょう？単位は？

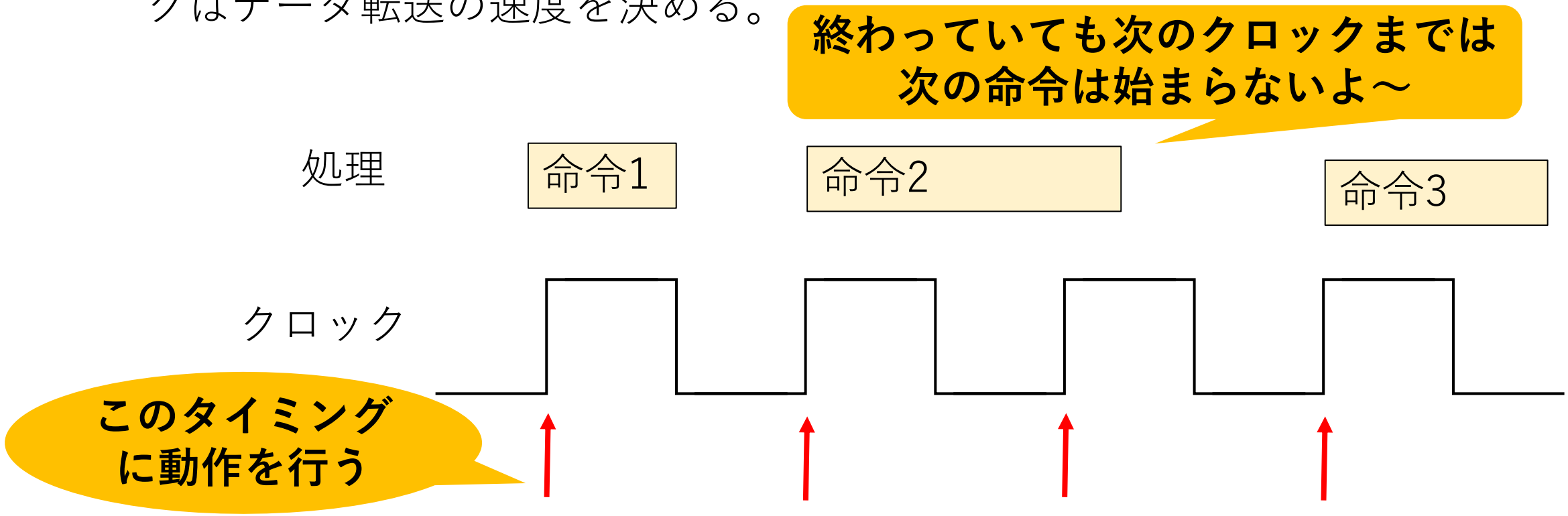
頭の中に思い浮かべてから次へ進みましょう

クロック周波数

クロックが１秒間に繰り返される回数のこと。単位はHz。

※クロック

コンピュータ内の動作のタイミングをとるために規則正しくクロック信号を発生させる回路。クロックの信号が速ければコンピュータの動作も速くなる。CPUのクロックは処理の速度を決め、システムバスのクロックはデータ転送の速度を決める。



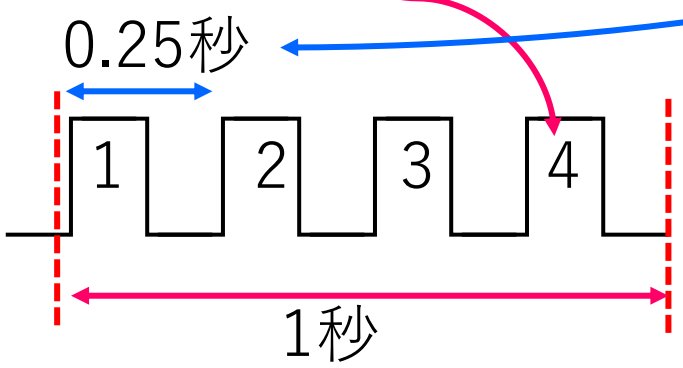
クロックサイクル時間

クロック周波数=4HzのCPUが1クロックに要する時間は何秒でしょうか？

１秒間に繰り返されるクロックの回数がクロック周波数なので、クロック1回は1/4=0.25秒です。

これを「**クロックサイクル時間**」といいます。

$$\frac{\text{クロック数}}{\text{1秒}} = \text{クロック周波数} \Leftrightarrow \frac{\text{1秒}}{\text{クロック数}} = \text{クロックサイクル時間}$$



CPIって何のことだったでしょう？

頭の中に思い浮かべてから次へ進みましょう

CPI(Clock cycles Per Instruction)

１命令あたり何クロックサイクル必要かを表します。

この値と前出のクロックサイクル時間を使うと命令の実行時間を求められます。

命令の実行時間＝クロックサイクル時間×CPI

MIPSって何のことだったでしょう？

頭の中に思い浮かべてから次へ進みましょう

MIPS/MFLOPS

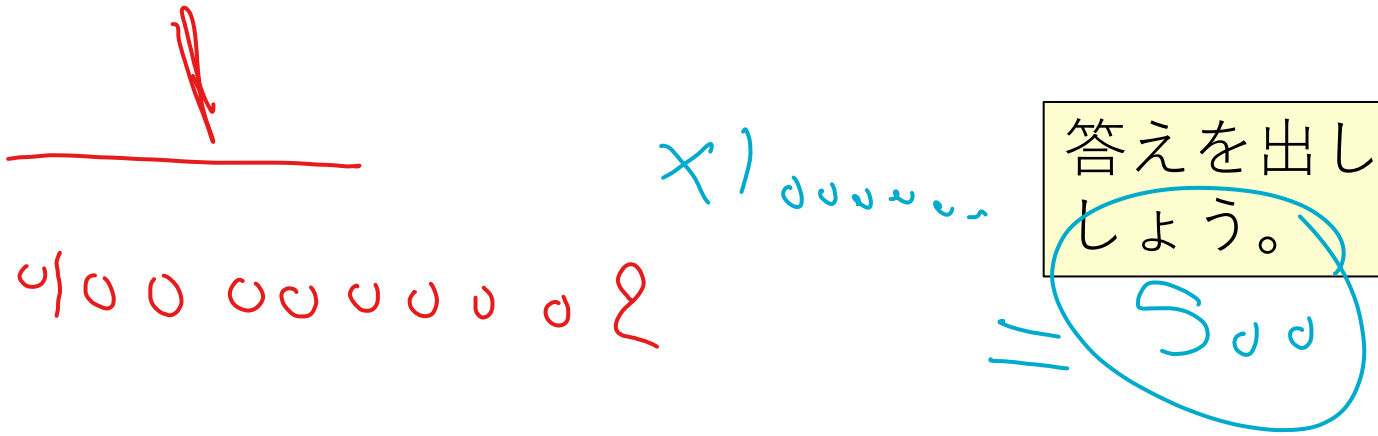
MIPS(Million Instructions Per Second)

　１秒間に実行できる命令の数を百万単位で表す

MFLOPS(Million Floating-point Operations Per Second)

　１秒間に実行できる浮動小数点演算の回数を百万単位で表す

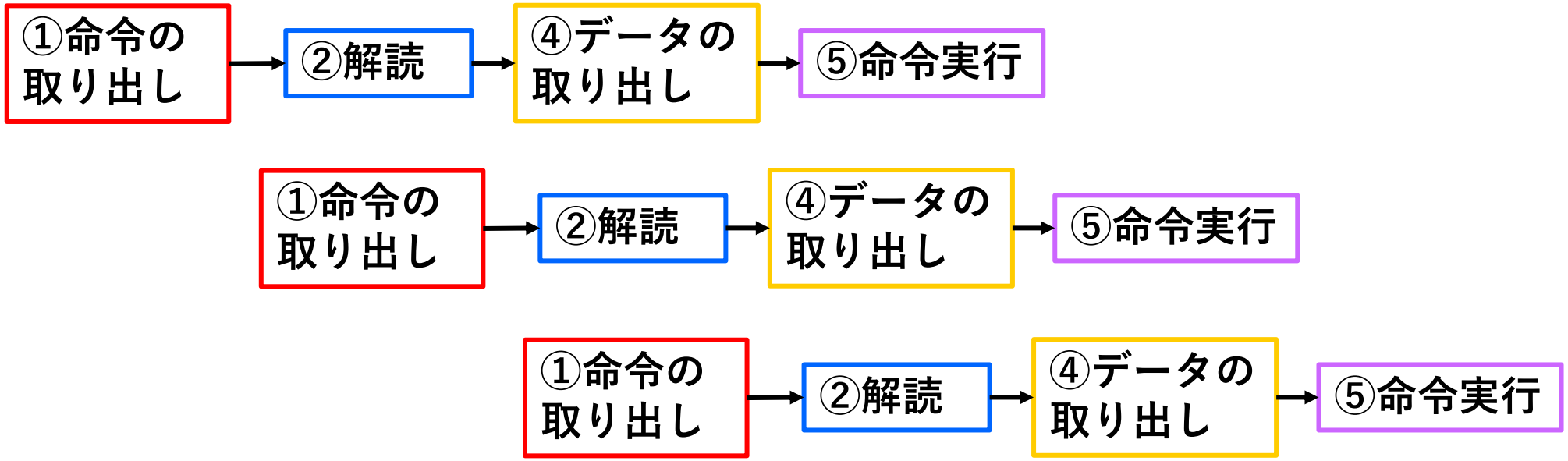
例：１つの命令に2ナノ秒(2×10^{-9}秒)かかるCPUのMIPSは？



答えを出してForms36に入力しましょう。

パイプライン処理（美容室の例で学びましたね）

複数の命令をずらしながら
並行して行うことを「**パイプライン処理**」と呼びます。

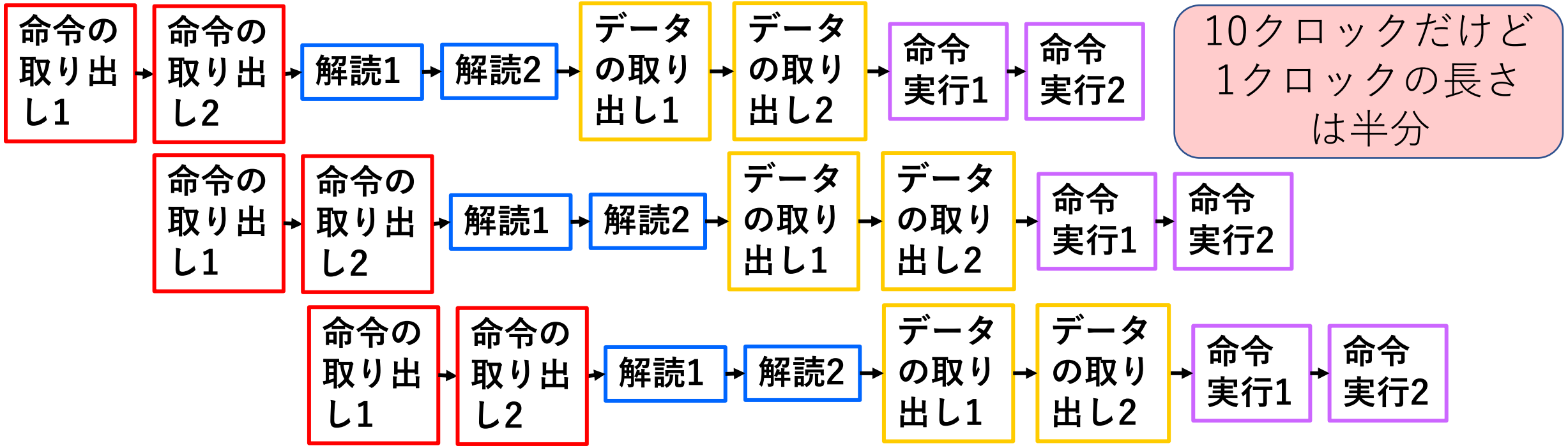


1　　2　　3　　4　　5　　6

スーパーパイプライン

更に手順を細かく分けること（美容師の例だとカラーやカットを2人で分けて行う）「**スーパーパイプライン**」といいます。

各手順のボリュームを小さくすることで**クロック周波数自体を上げる**方法です。

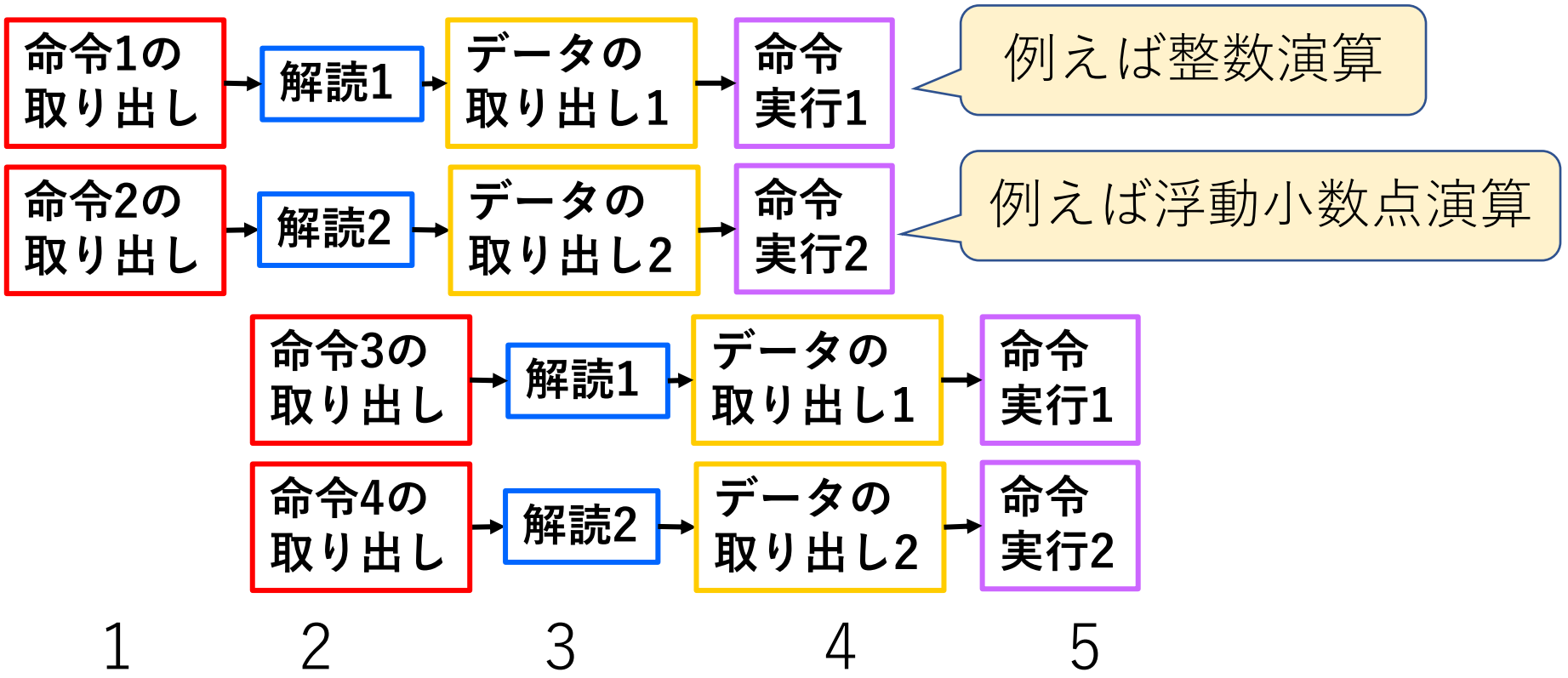


スーパースカラ

更に、パイプライン処理を行う回路を複数持たせることを「**スーパースカラ**」といいます。

命令の実行に使う回路は複数種類あります。それらを効率的に使うため、

複数の命令を同時にフェッチ・デコードし、**異なる演算回路を用いて**同時に命令を実行することで多くの命令を速く実行します。

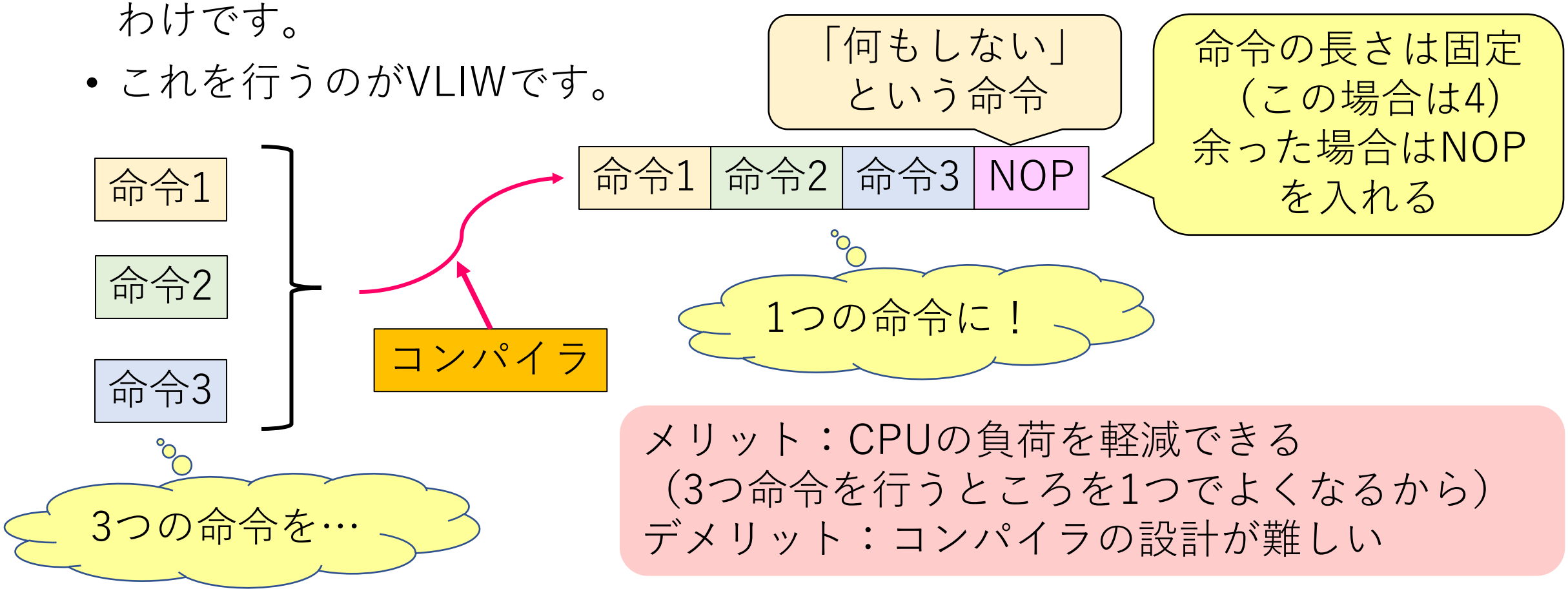


VLIWって何のことだったでしょう？

頭の中に思い浮かべてから次へ進みましょう

VLIW(Very Long Instruction Word)

- スーパースカラでは同時に実行できない命令がありました。（同じ回路を使う命令、前のデータを使う命令…）
- ということは、同時に実行できる命令をまとめればとても効率よくできるわけです。
- これを行うのがVLIWです。



メリット：CPUの負荷を軽減できる
（3つ命令を行うところを1つでよくなるから）
デメリット：コンパイラの設計が難しい

CISC

- **<u>複雑で高機能</u>**な命令を持つCPU

- 命令の種類は多い→解読に時間がかかる

- 複雑な処理を一つの命令でこなせる
  →命令読み出し回数は少ない・一つの命令に時間がかかる

- 命令の長さはバラバラ→パイプラインは難しい

RISC

- **単純で簡単**な命令を持つCPU

- 命令の種類は少ない→解読にかかる時間は短い

- 簡単な命令を組み合わせることで複雑な処理を可能にする
  →一つの命令は速い・命令読み出し回数は多い

- 命令の長さは同じ→パイプラインが可能

今日はここまで。

- 重要な部分をかいつまんで前期の振り返りを実施しましたがいかがでしたでしょうか？
- 来週からは新しい内容に入ります。
- また半年間、一緒に頑張っていきましょう！