

コンピュータ工学 I

Rev. 2016.11.28

コンピュータの基本構成とCPU

✦ 内容

- ① CPUの構成要素
- ② 命令サイクル
- ③ アセンブリ言語
- ④ アドレッシング方式
- ⑤ CPUの高速化
- ⑥ CPUの性能評価

コンピュータの構成装置

❖ 中央処理装置(CPU)

主記憶装置から命令を読み込み、実行を行う。

❖ 主記憶装置

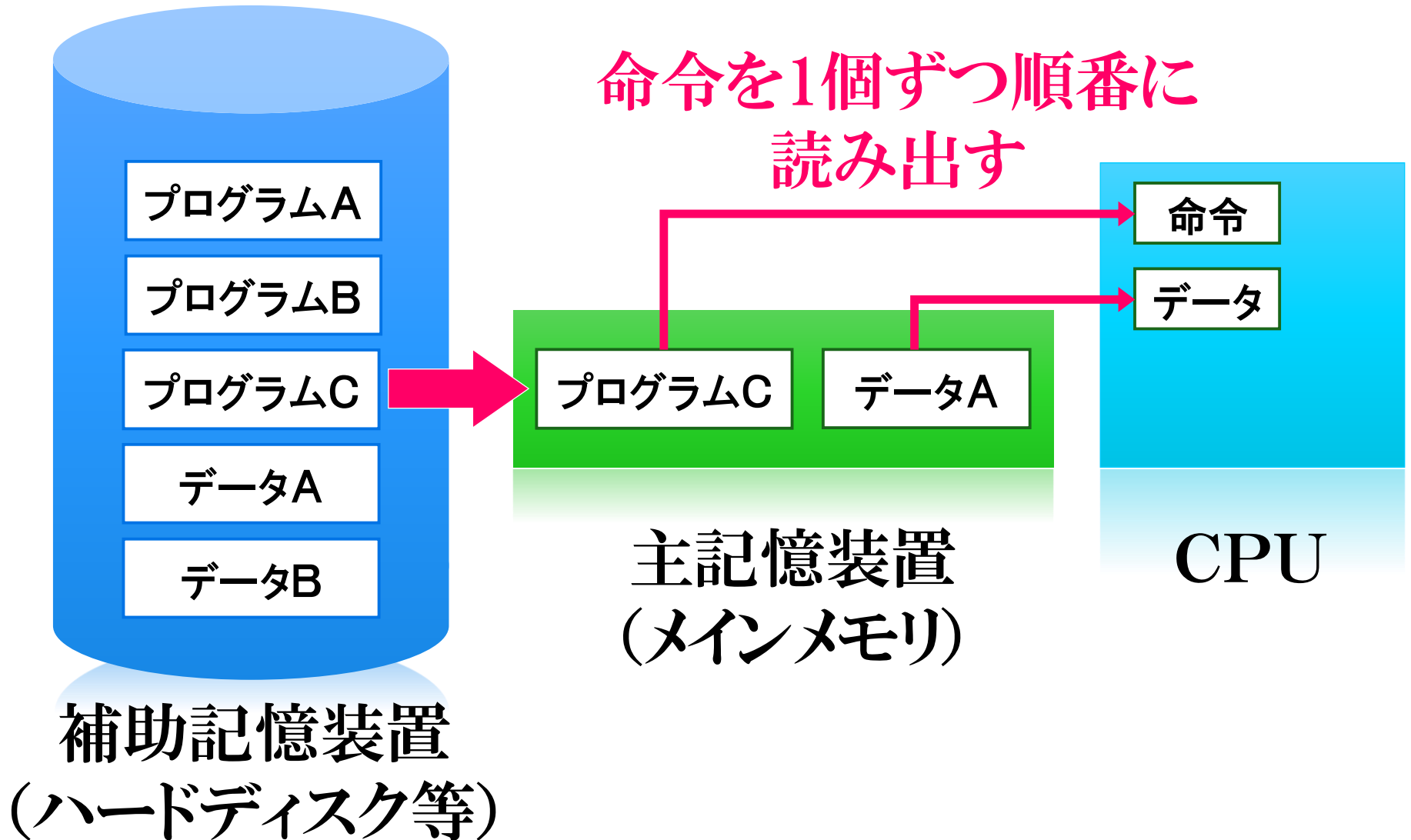
CPUで実行するプログラム(命令の集合)やデータを記憶する。

❖ 補助記憶装置

❖ 入力装置

❖ 出力装置

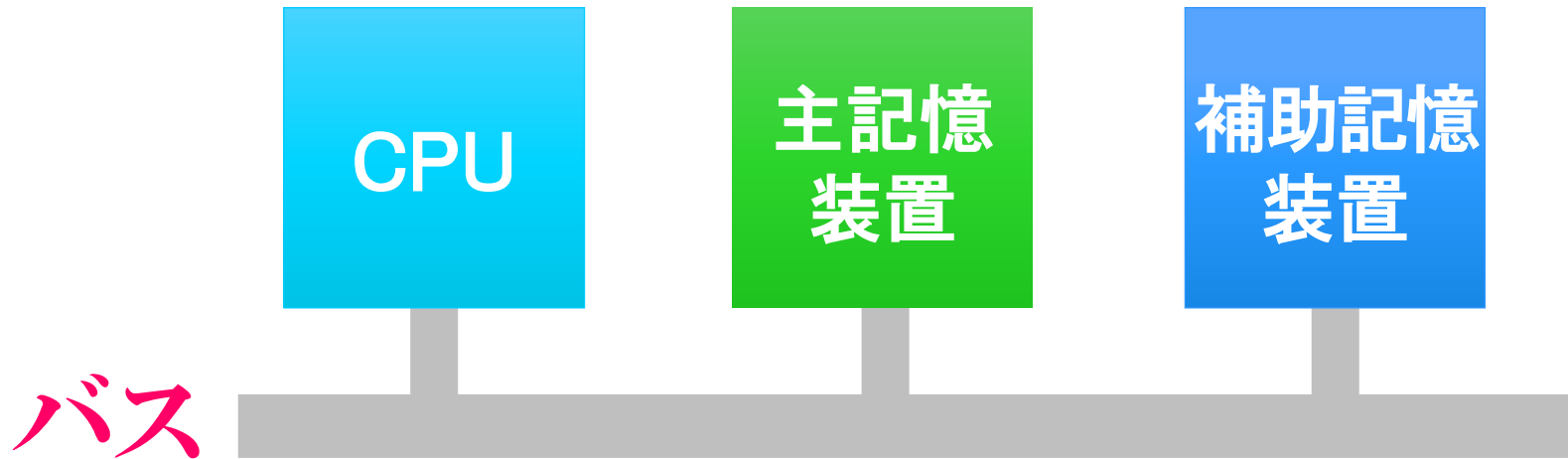
プログラムの実行



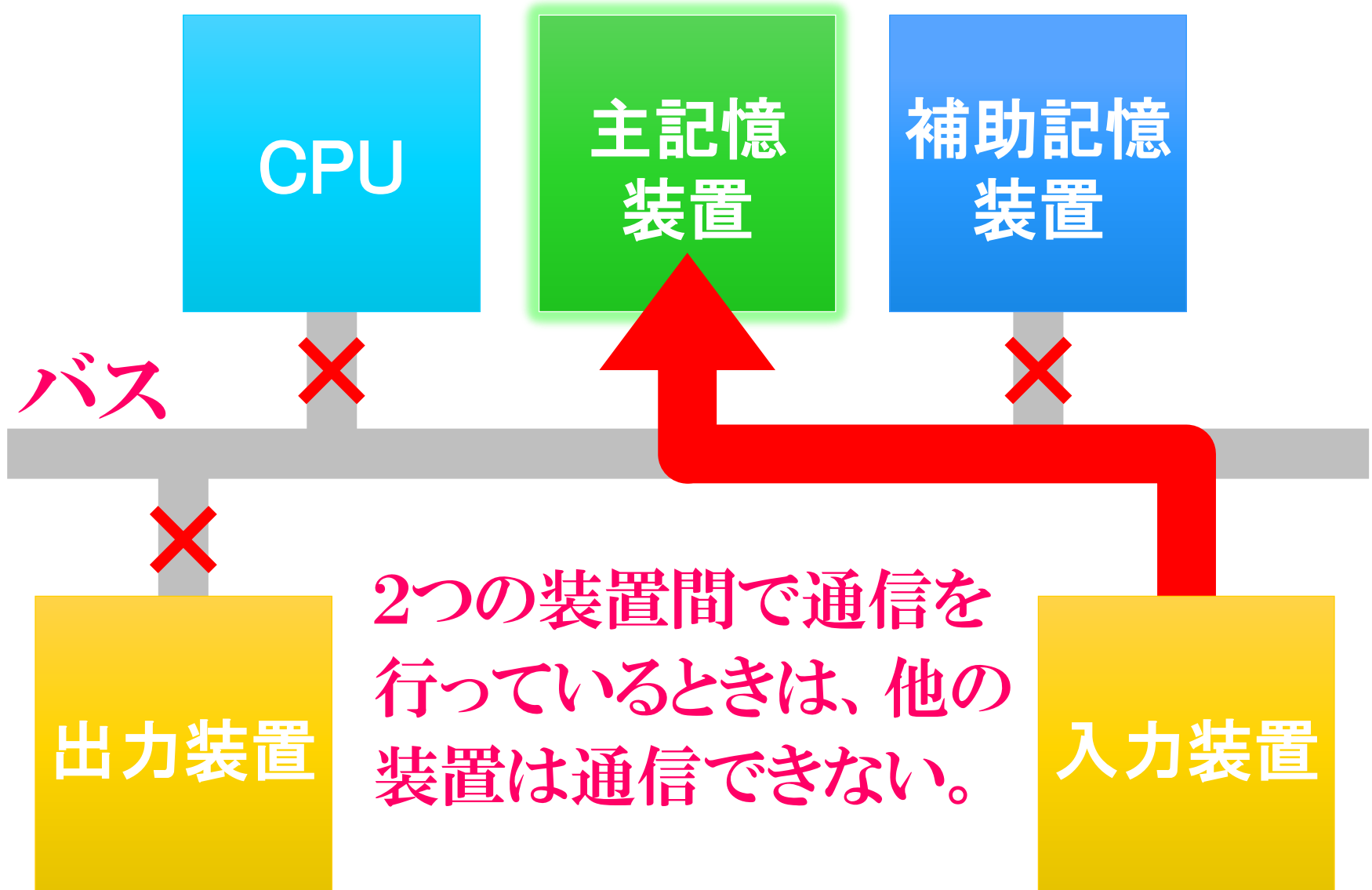
バス方式

バス

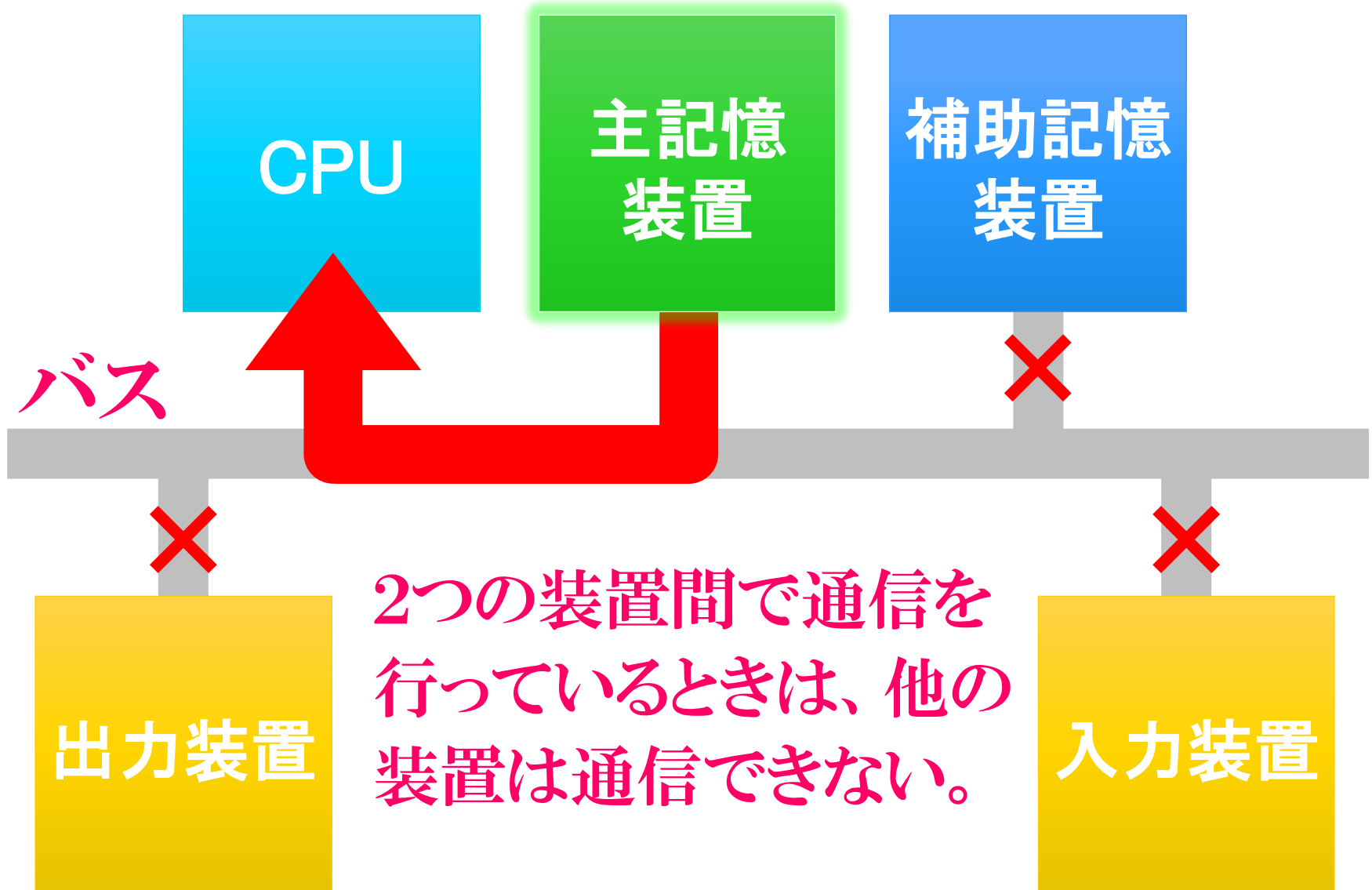
コンピュータ内の各装置がデータ通信に用いる共通の通信経路



バスによる装置間の通信



バスによる装置間の通信



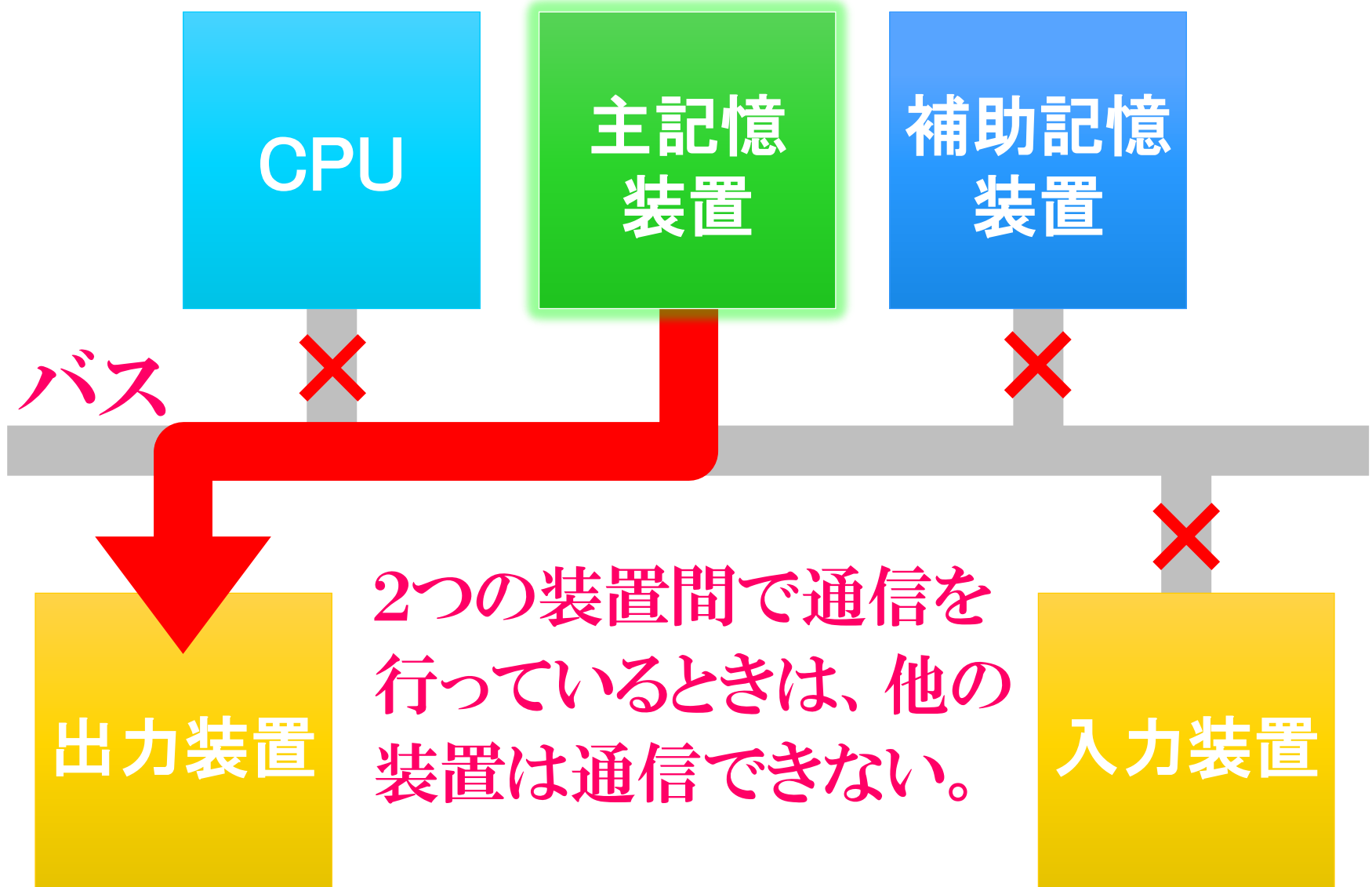
バスによる装置間の通信



バスによる装置間の通信



バスによる装置間の通信



バスの種類

重要!

❖ データバス

データを送る。

❖ アドレスバス

主記憶装置のアドレス(番地)を送る。

❖ コントロールバス

データの読み書きの制御信号を送る。

主記憶装置

アドレス(番地)

| | |
|-----|----|
| ⋮ | ⋮ |
| 100 | 2E |
| 101 | 93 |
| 102 | 00 |
| 103 | 3C |
| 104 | 56 |
| 105 | 2A |
| ⋮ | ⋮ |

記憶場所

数値化された命令やデータを記憶

主記憶装置内のデータを読み書きするときは、対象のデータのアドレスを指定する必要がある。

機械語とアセンブリ言語

| CPUの機能 | 命令番号 | 命令語 |
|--------|------|------|
| データ転送 | 10 | LD |
| 加算 | 24 | ADDA |
| 減算 | 25 | SUBA |
| 論理積 | 34 | AND |
| 論理和 | 35 | OR |

機械語 アセンブリ言語

CPUの機能1つ1つに命令番号が与えられている。

CPUの構成要素

重要!

♣レジスタ

| 略称 | 名称 |
|---------|-------------|
| MDR | 記憶データ用レジスタ |
| MAR | 記憶番地指定用レジスタ |
| PC | プログラムカウンタ |
| IR | 命令レジスタ |
| GR0~GR7 | 汎用レジスタ |
| FR | フラグレジスタ |

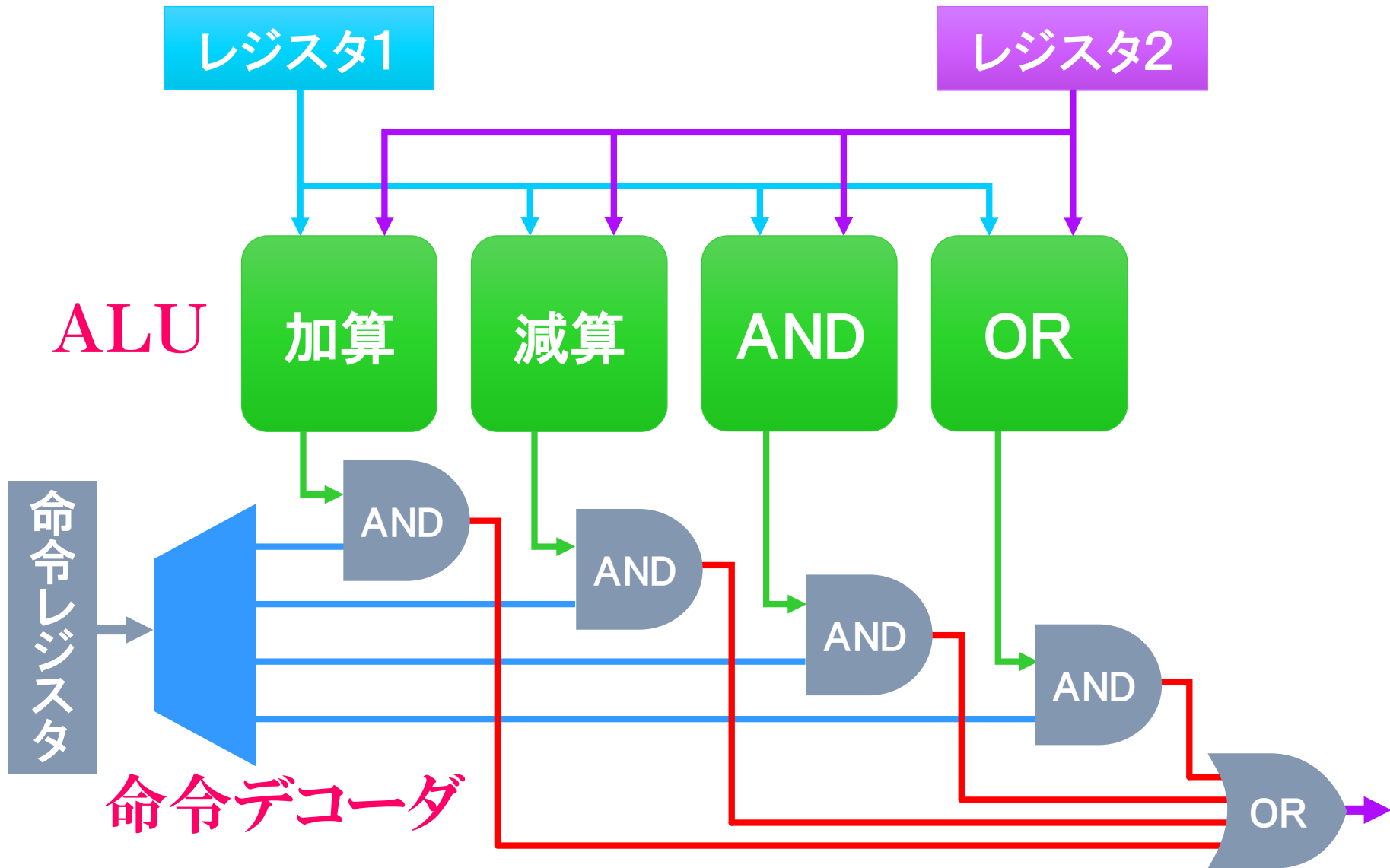
CPUの構成要素

重要!

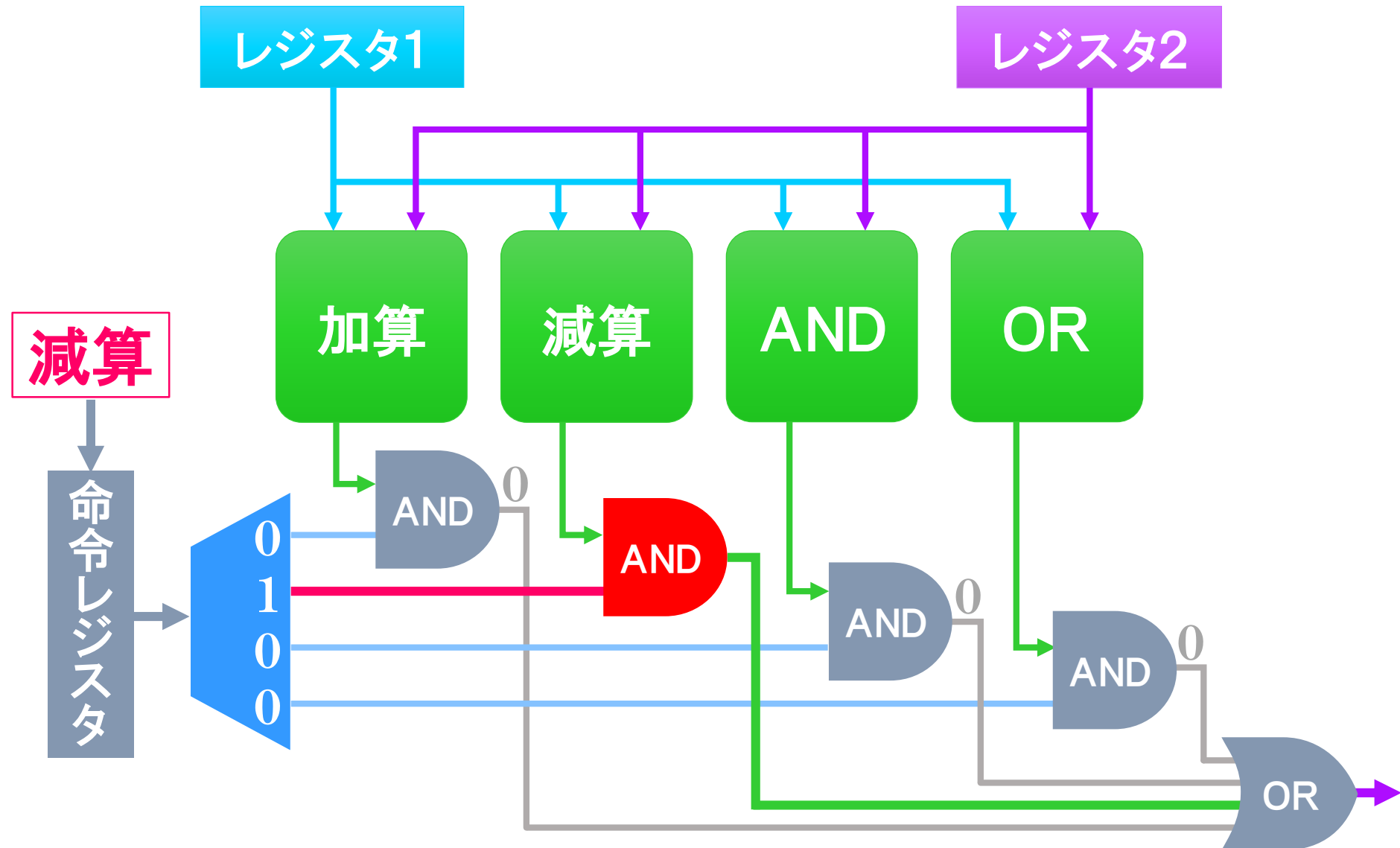
♣ その他装置

| 略称・名称 | 働き |
|------------|-------------------|
| ALU | 算術演算や論理演算を行う |
| 命令デコーダ | 命令番号を解読し、各装置を制御する |
| コントロールユニット | コントロールバスに制御信号を送る |

命令デコーダとALU



減算を行う場合



命令サイクル

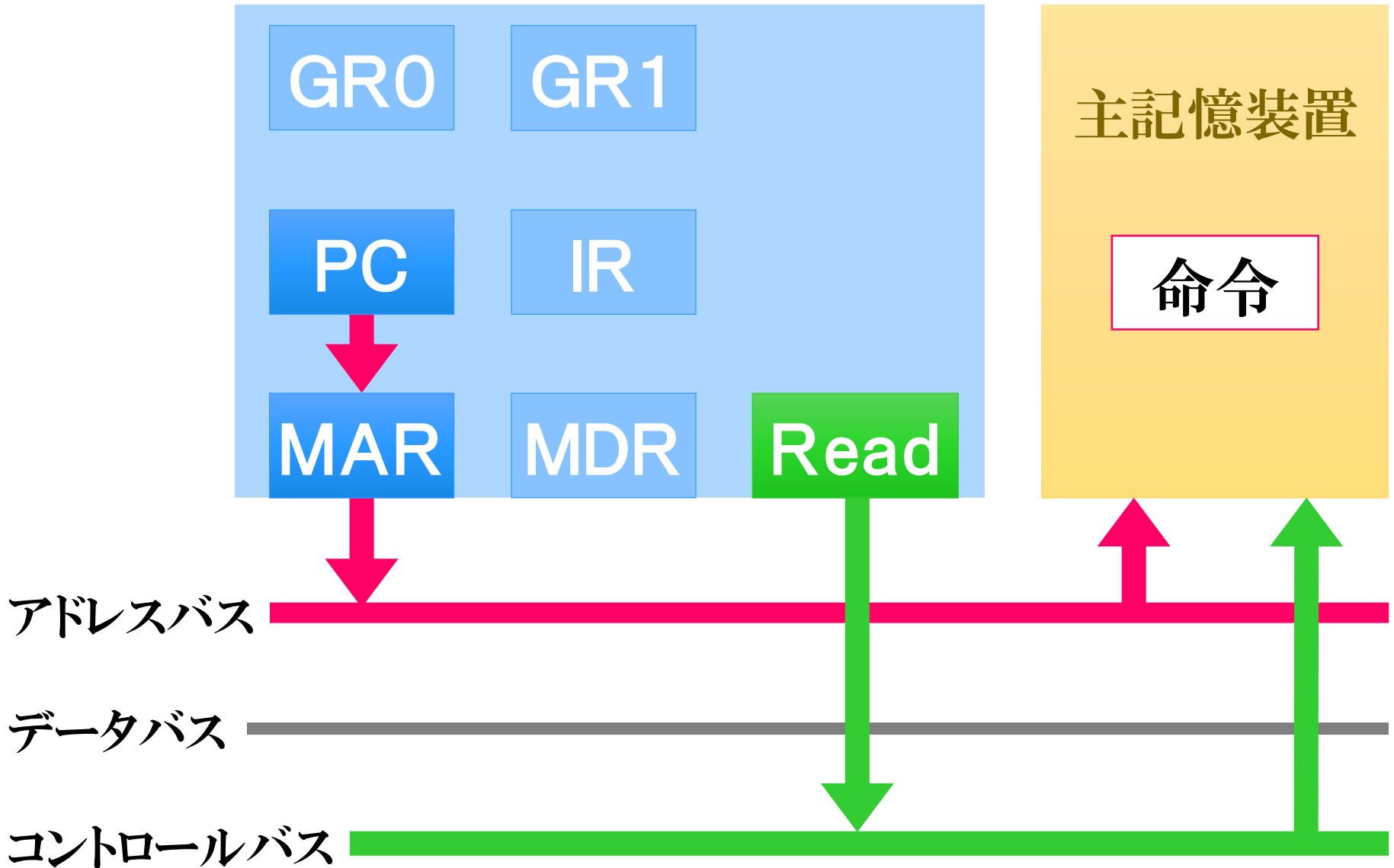
重要!

1つの命令を実行する手順

- ① 命令読み出し(fetch)
- ② 命令解読(decode)
- ③ 命令実行(execute)
- ④ 結果書き込み(writeback)

命令読み出し

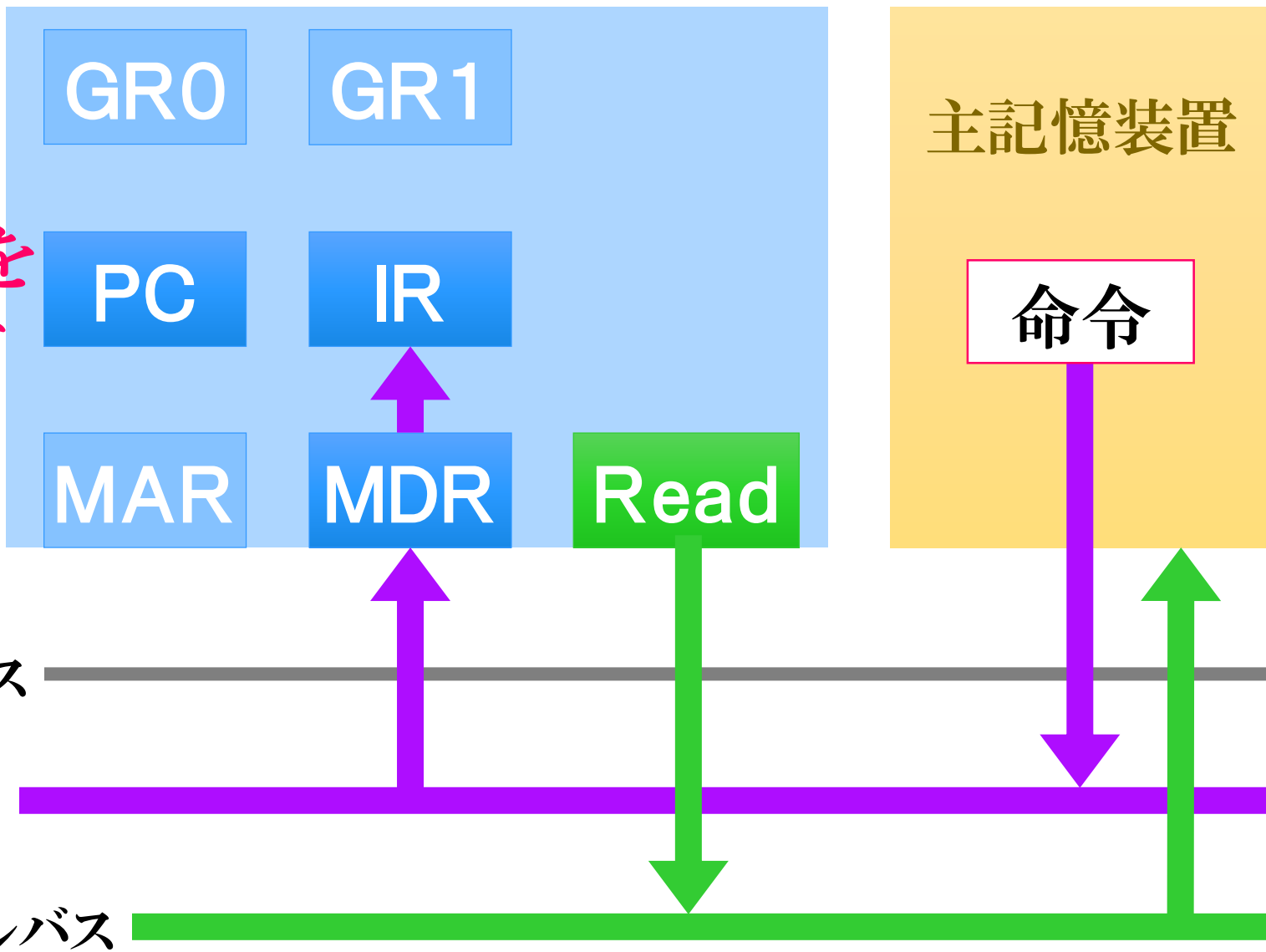
重要!



命令読み出し

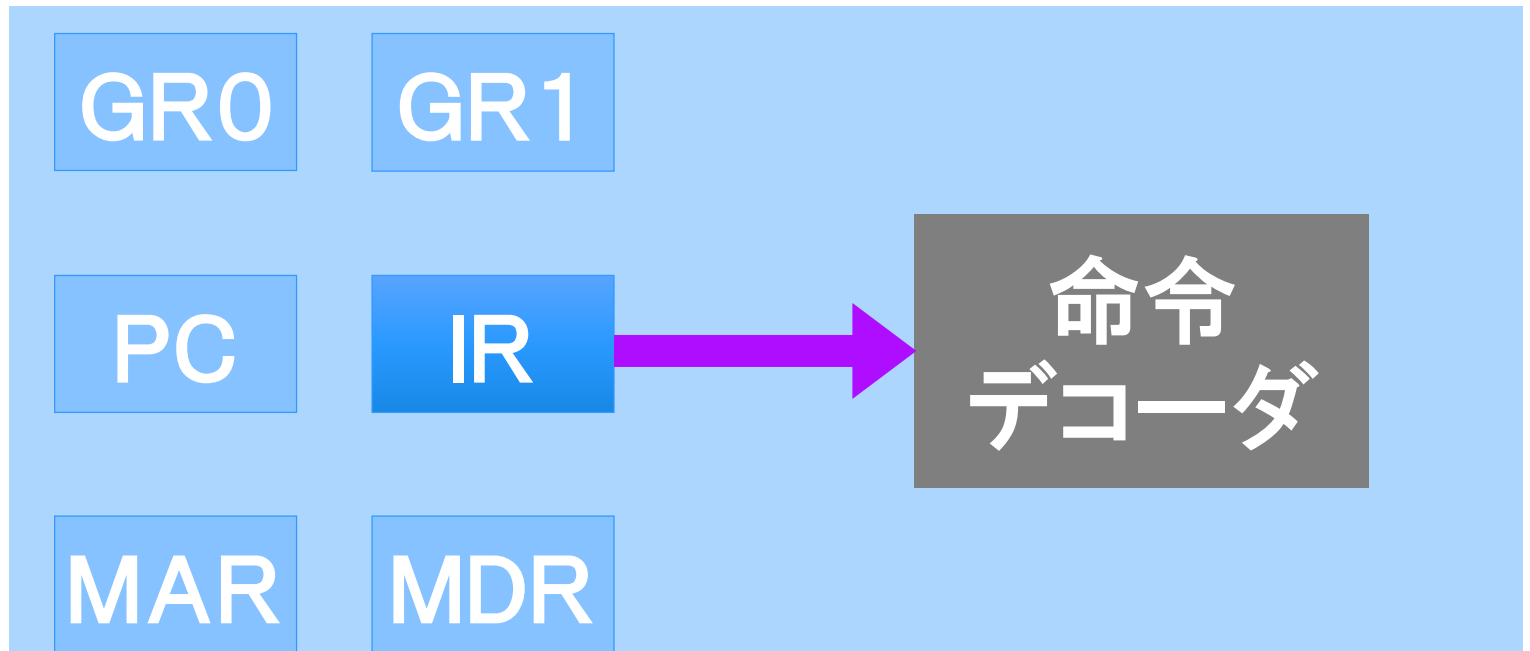
重要!

PCの値を
1増やす



命令解読

重要!



$GR1 \leftarrow GR1 + GR0$

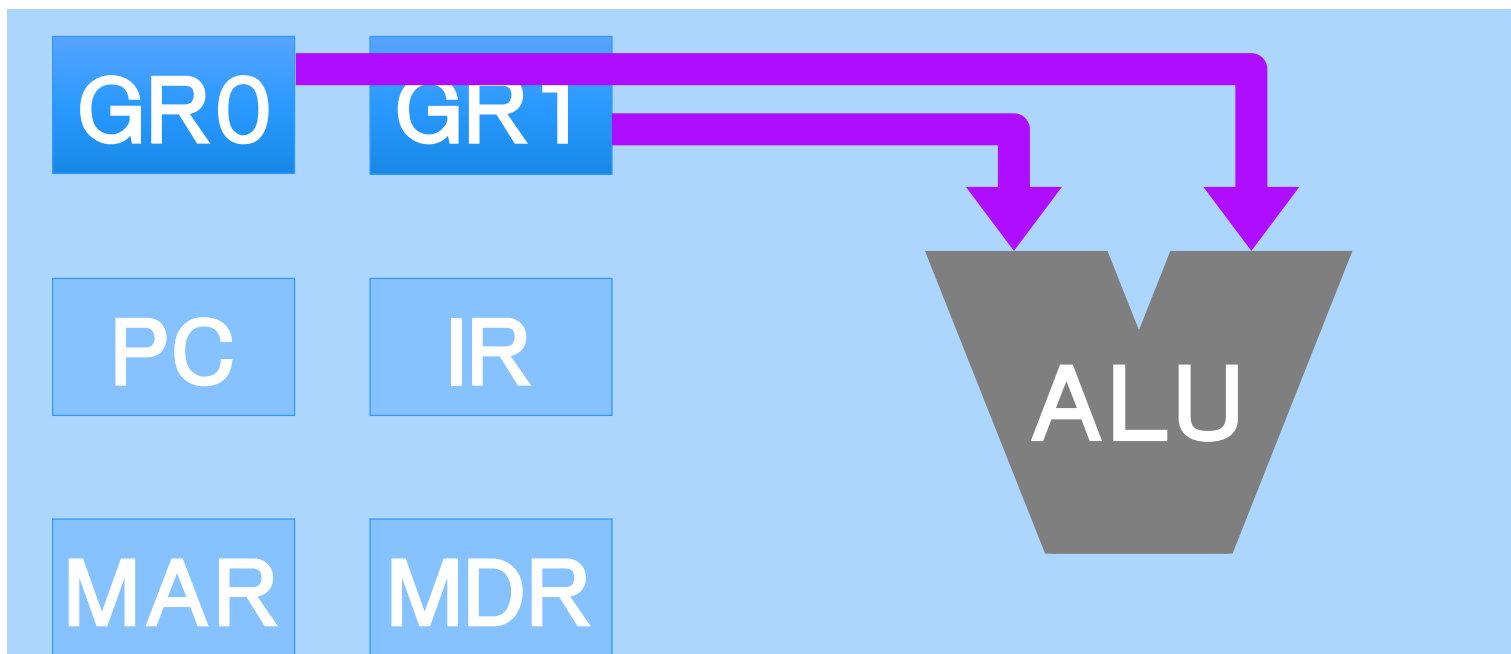
アドレスバス

データバス

コントロールバス

命令実行

重要!



$GR1 \leftarrow GR1 + GR0$

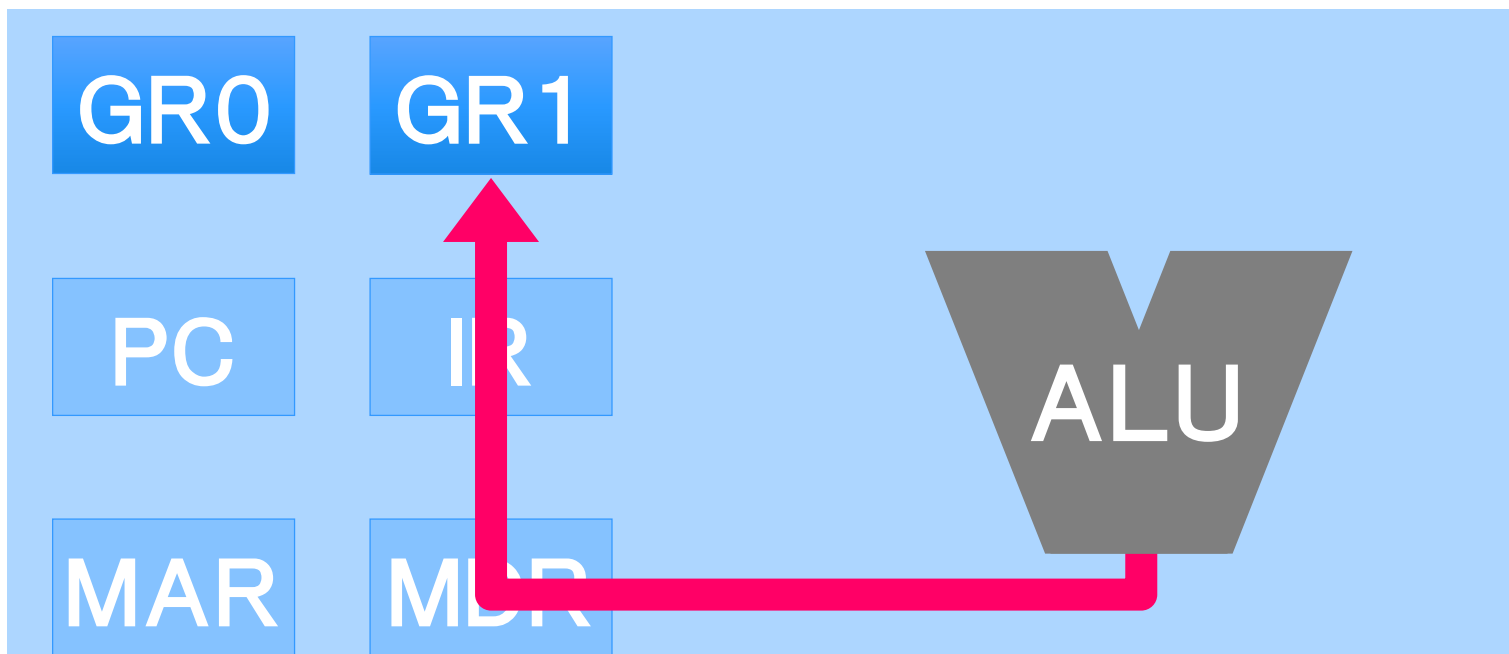
アドレスバス

データバス

コントロールバス

結果書き込み

重要!



$$GR1 \leftarrow GR1 + GR0$$

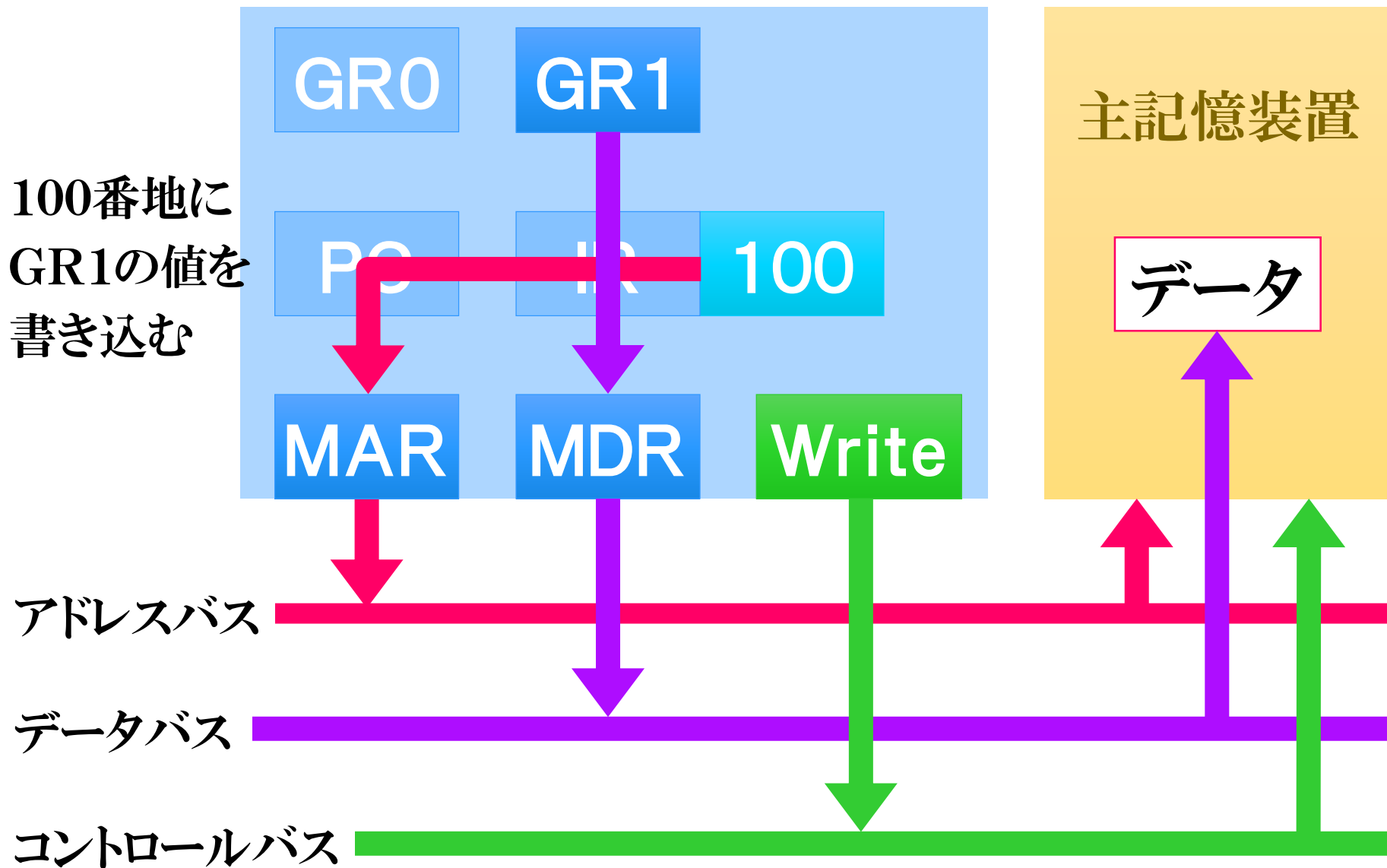
アドレスバス

データバス

コントロールバス

主記憶装置への書き込み

重要!



アセンブリ言語

CASL II を取り扱う。

❖ データ転送命令

❖ 算術演算命令(加算、減算)

❖ 分岐命令

❖ 算術比較命令

命令の記述

機械語 1 0 1 2 0 0 8 0 (16)

アセンブリ言語

LD

GR1,

80,

GR2

オペコード

命令の種類

オペランド

命令の対象

(レジスタ, アドレス)

重要!

LD (Load)

レジスタ、または、主記憶装置内の値を、指定したレジスタに読み込む。

重要!

LD r1, r2

レジスタ**r1**に、レジスタ**r2**の値を読み込む

LD r1, adr

レジスタ**r1**に、**adr**番地の値を読み込む

ST (Store)

レジスタの値を、主記憶装置内の指定したアドレスに書き込む。

重要!

```
ST r1, adr
```

レジスタ`r1`の値を、`adr`番地に書き込む

LAD (Load Address)

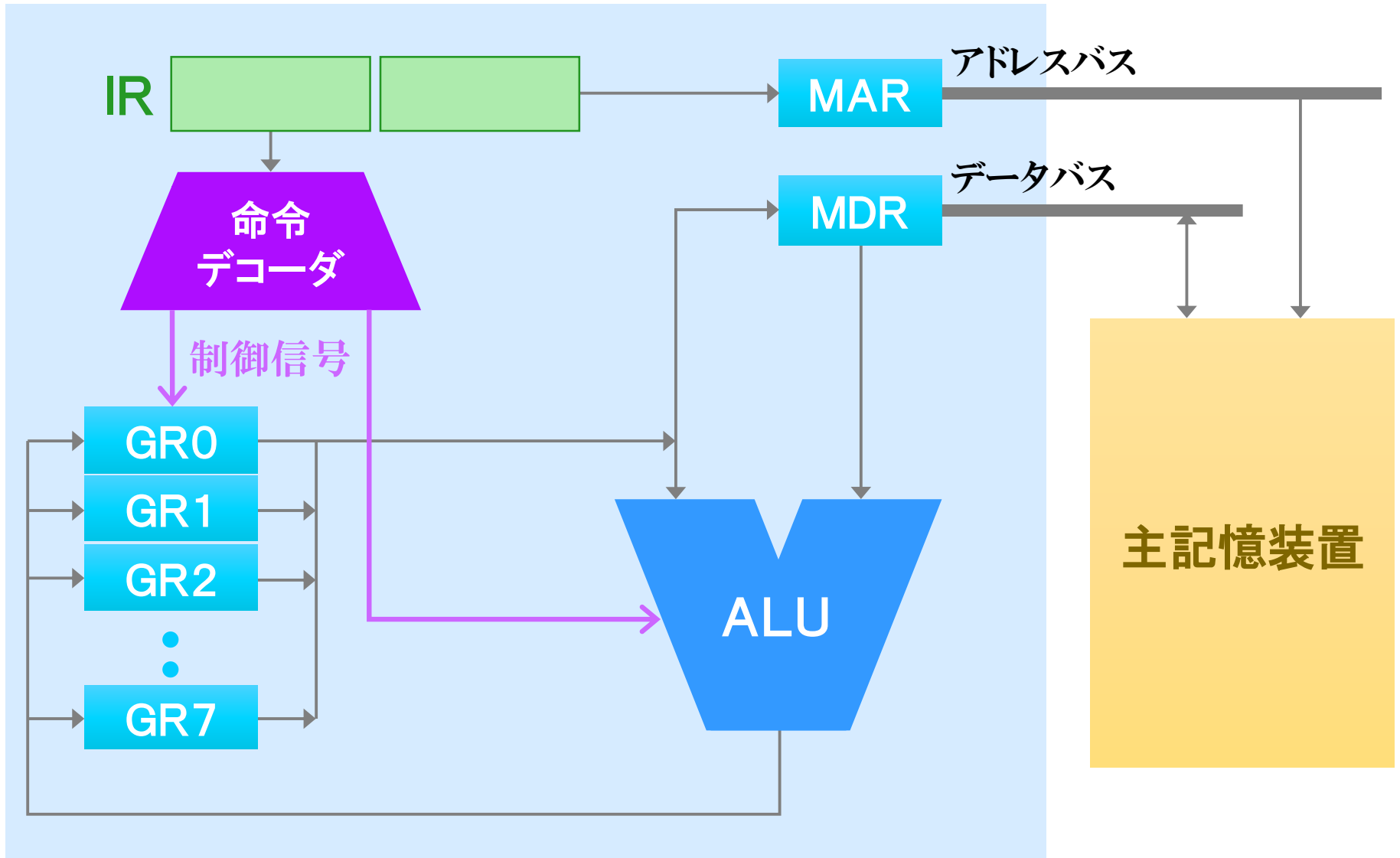
レジスタに定数(アドレス値)を書き込む。

重要!

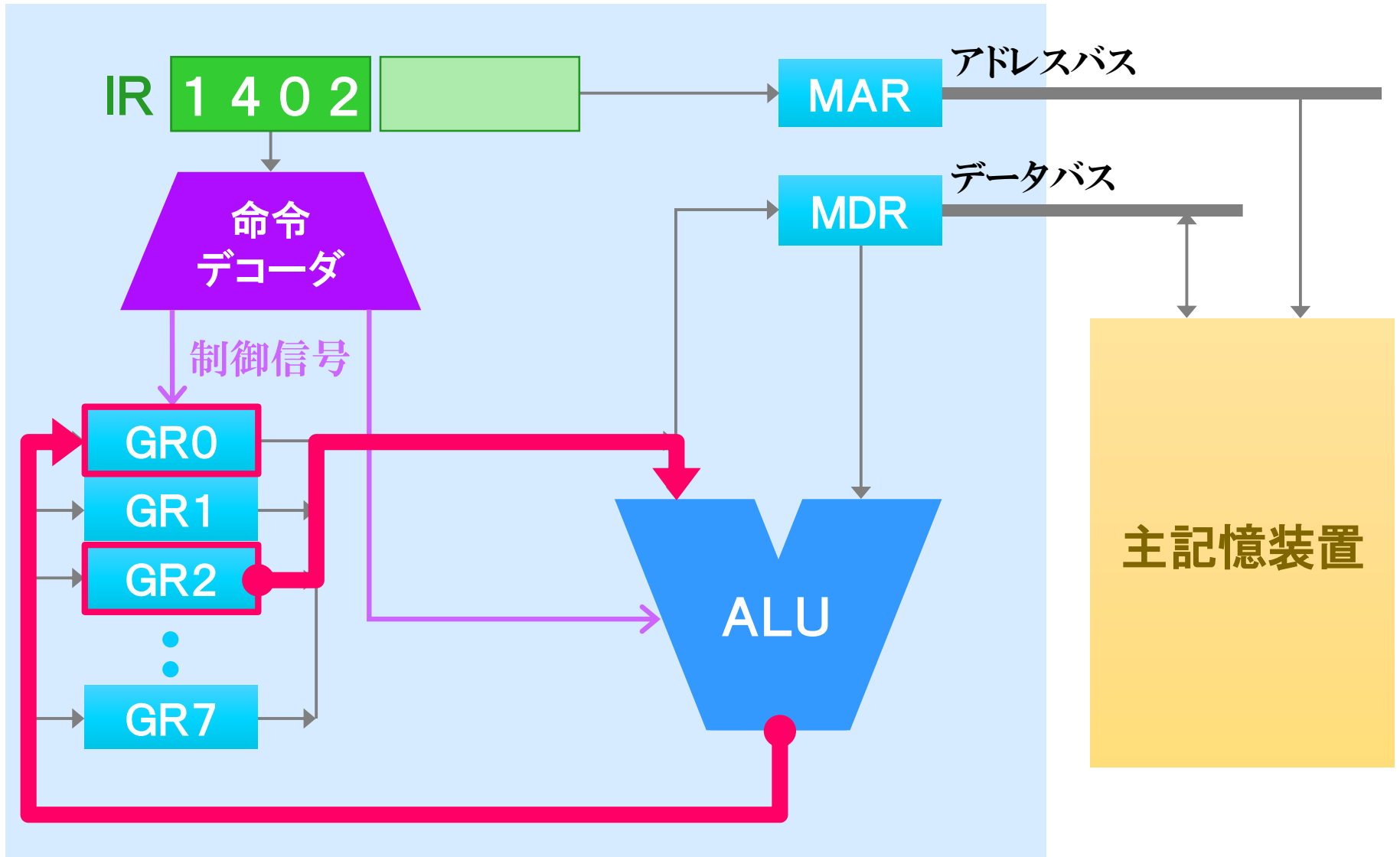
```
LAD r1, adr
```

レジスタ**r1**の値を、**adr**にする

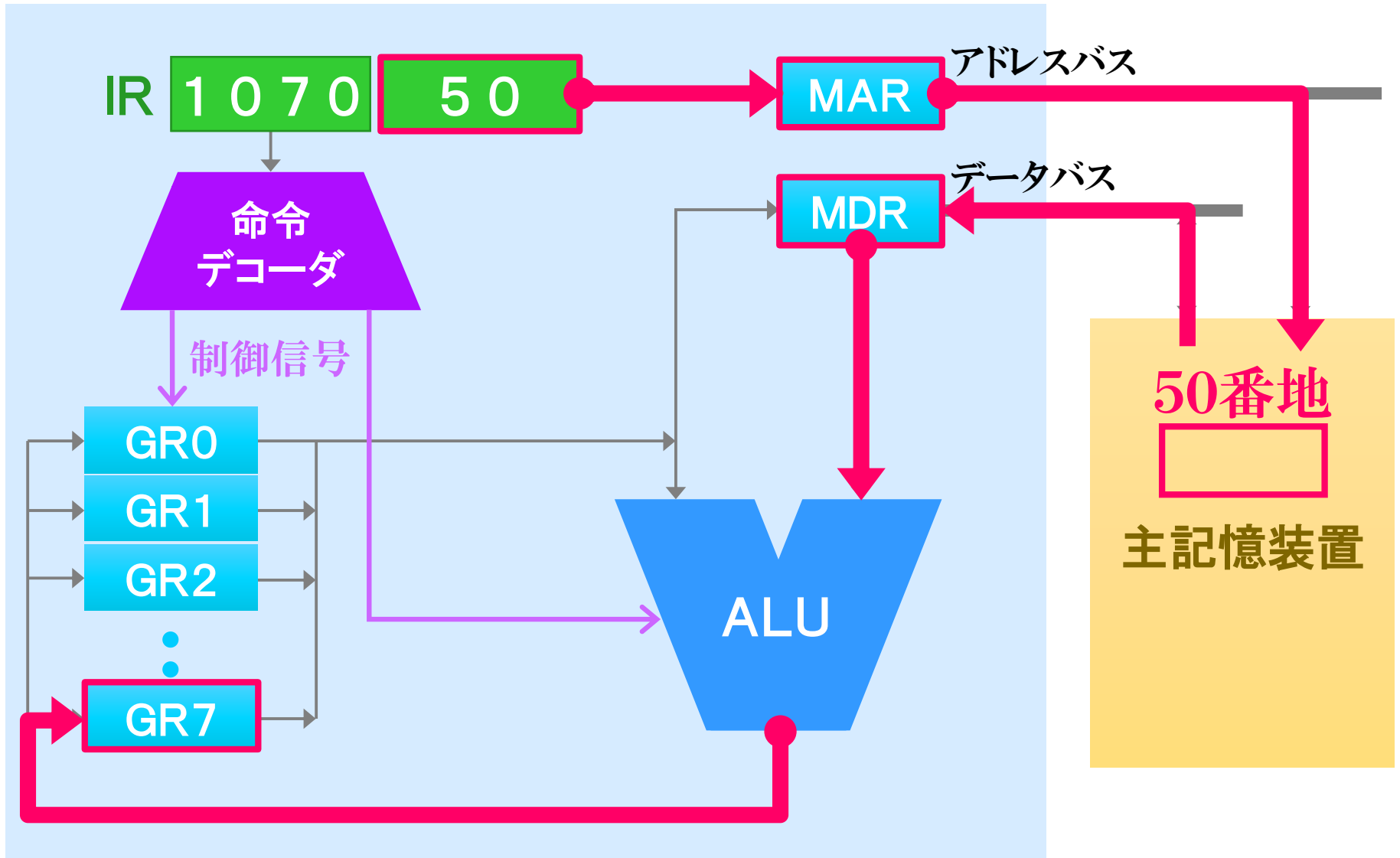
CPUの概略図



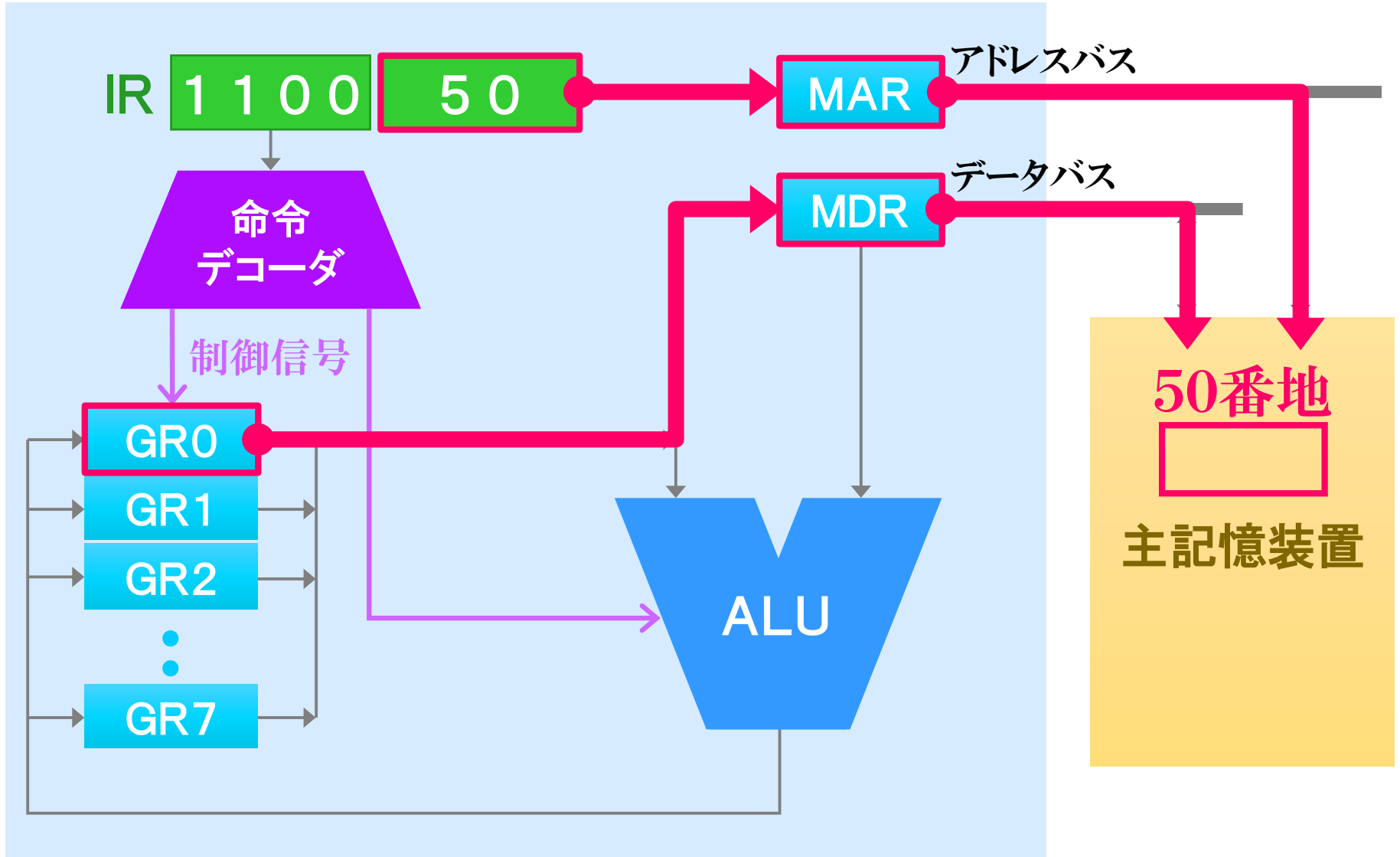
LD GR0, GR2



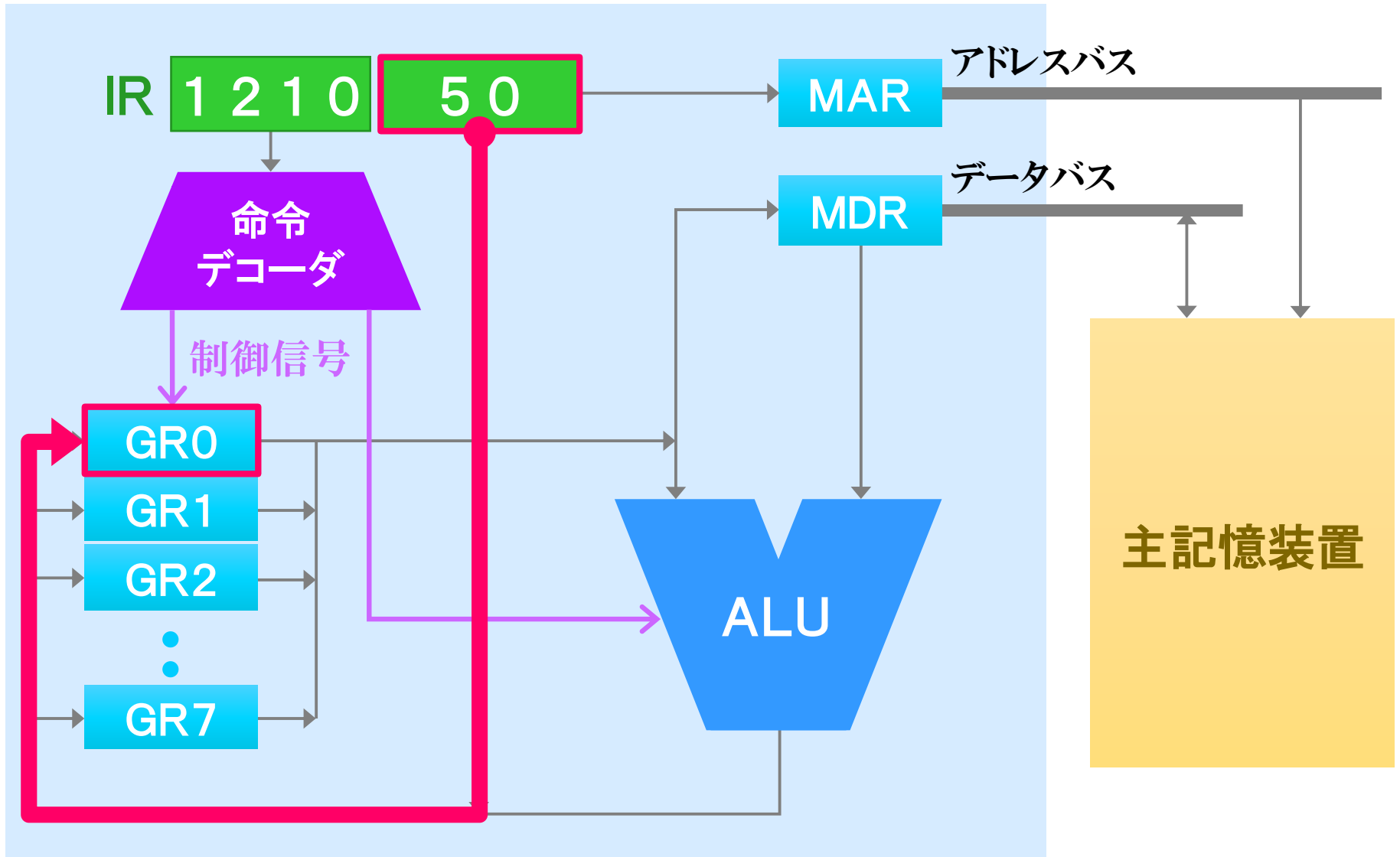
LD GR7, 50



ST GR0, 50



LAD GR1, 50



ADDA (Add Arithmetic)

2つの値の加算結果をレジスタに書き込む。

重要!

ADDA r1, r2

レジスタ**r1**と**r2**の加算結果を**r1**へ書き込む

ADDA r1, adr

レジスタ**r1**と**adr**番地の値の加算結果を**r1**へ書き込む

SUBA (Subtract Arithmetic)

2つの値の減算結果をレジスタに書き込む。

重要!

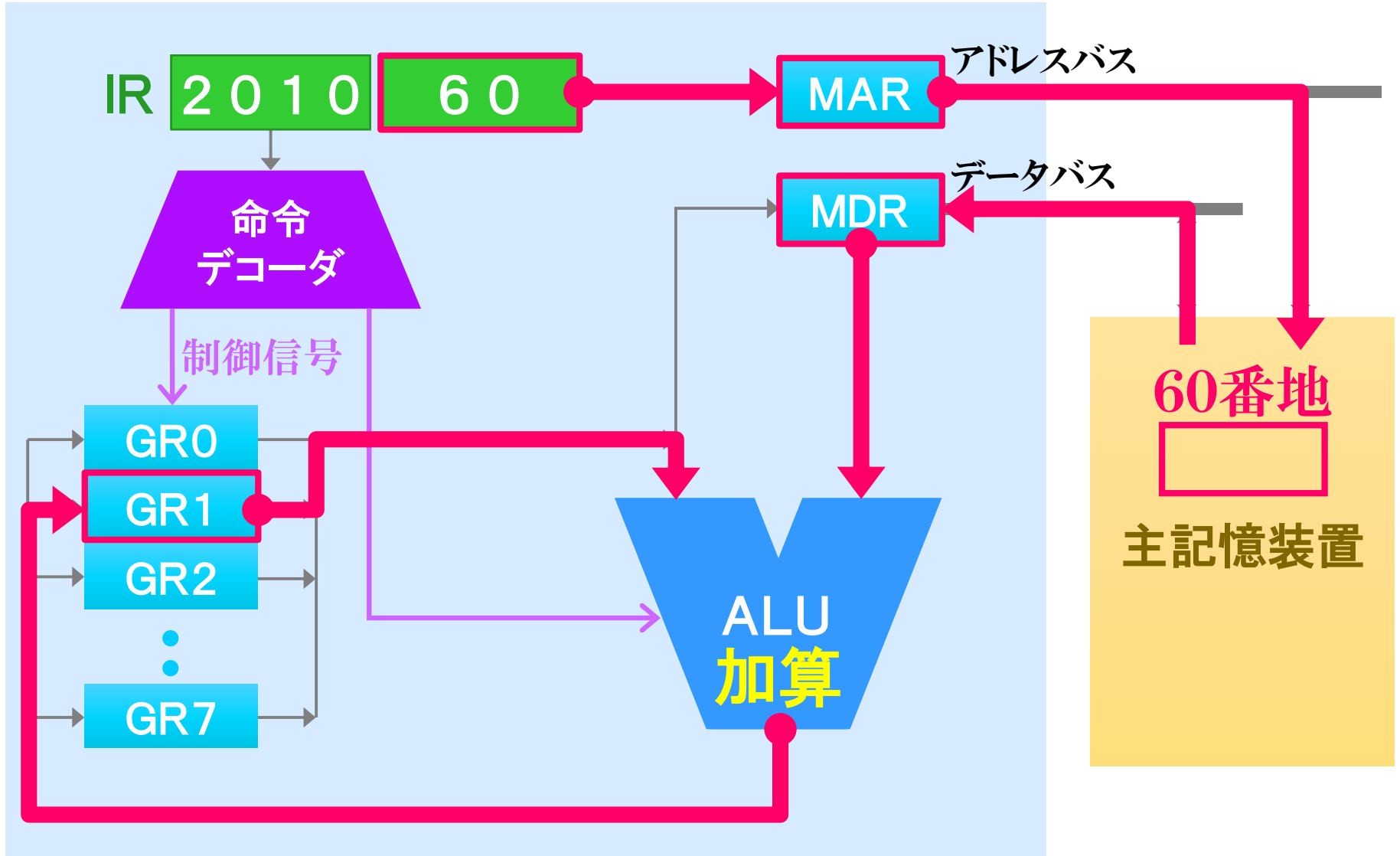
SUBA r1, r2

レジスタ**r1**と**r2**の減算結果を**r1**へ書き込む

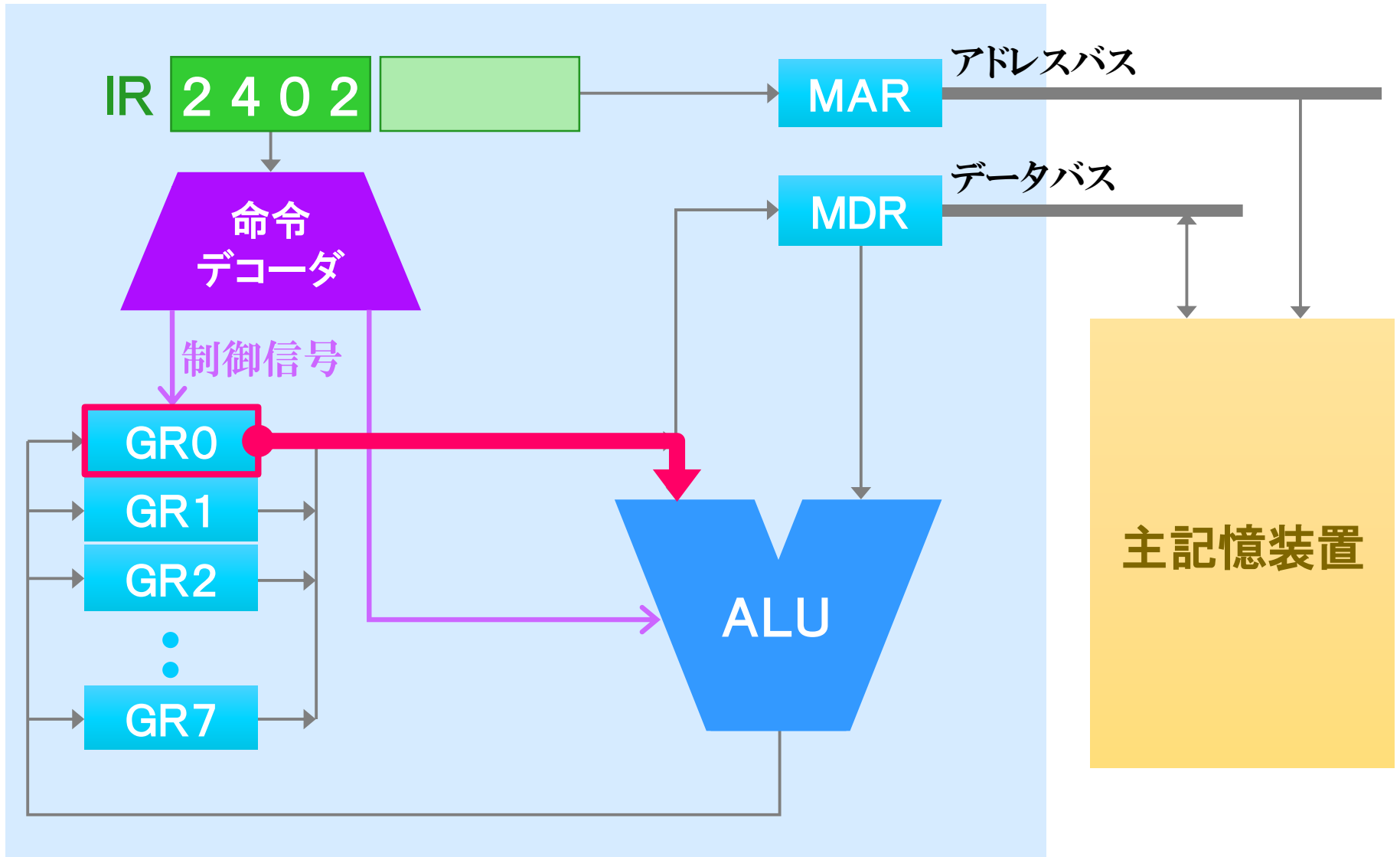
SUBA r1, adr

レジスタ**r1**と**adr**番地の値の減算結果を**r1**へ書き込む

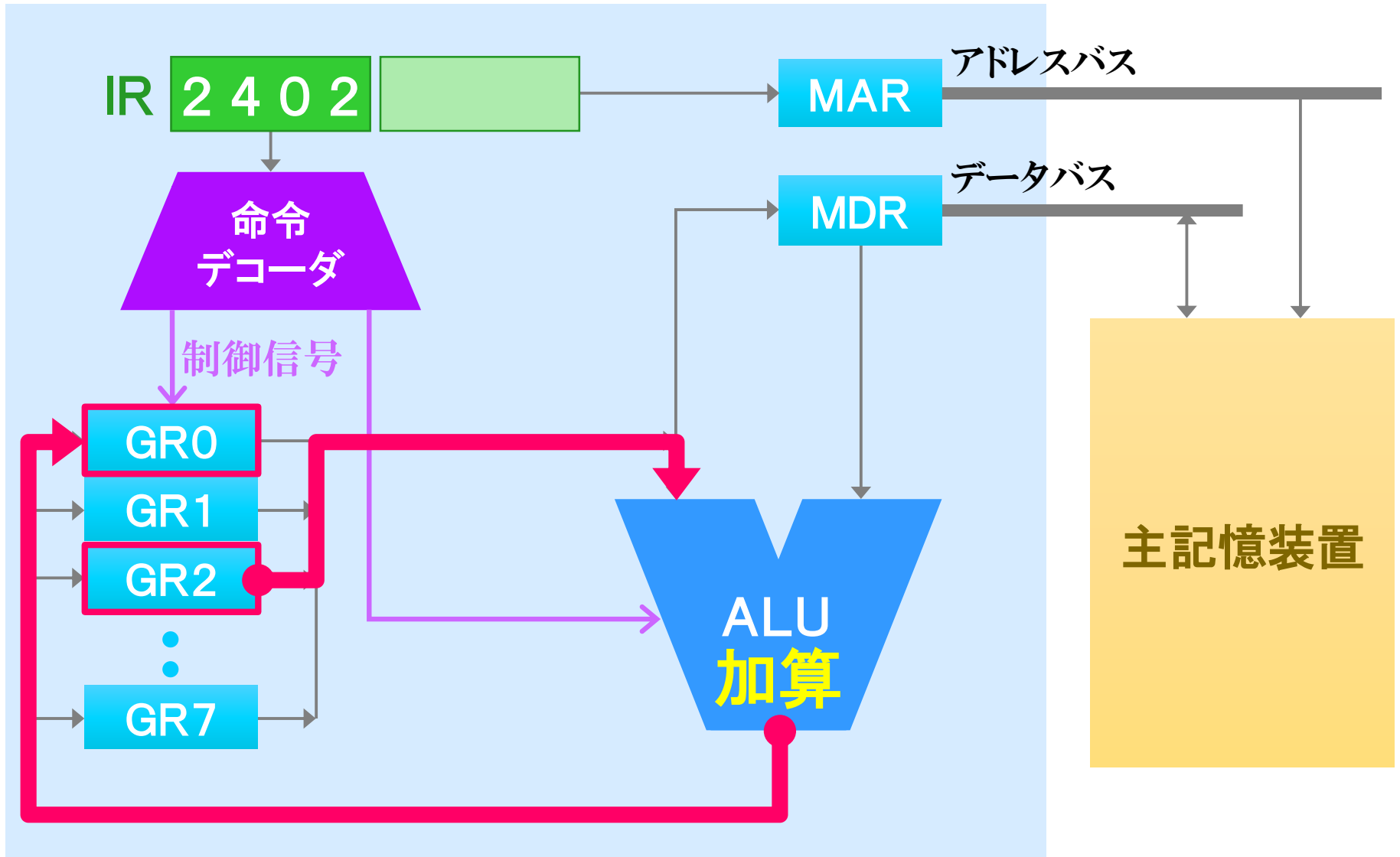
ADDA GR1, 60



ADDA GR0, GR2



ADDA GR0, GR2



実効アドレスと指標レジスタ

LD r1, adr

レジスタr1に、adr番地の値を読み込む

実効アドレス

処理対象のアドレス

重要!

指標レジスタ

汎用レジスタの中の1つ

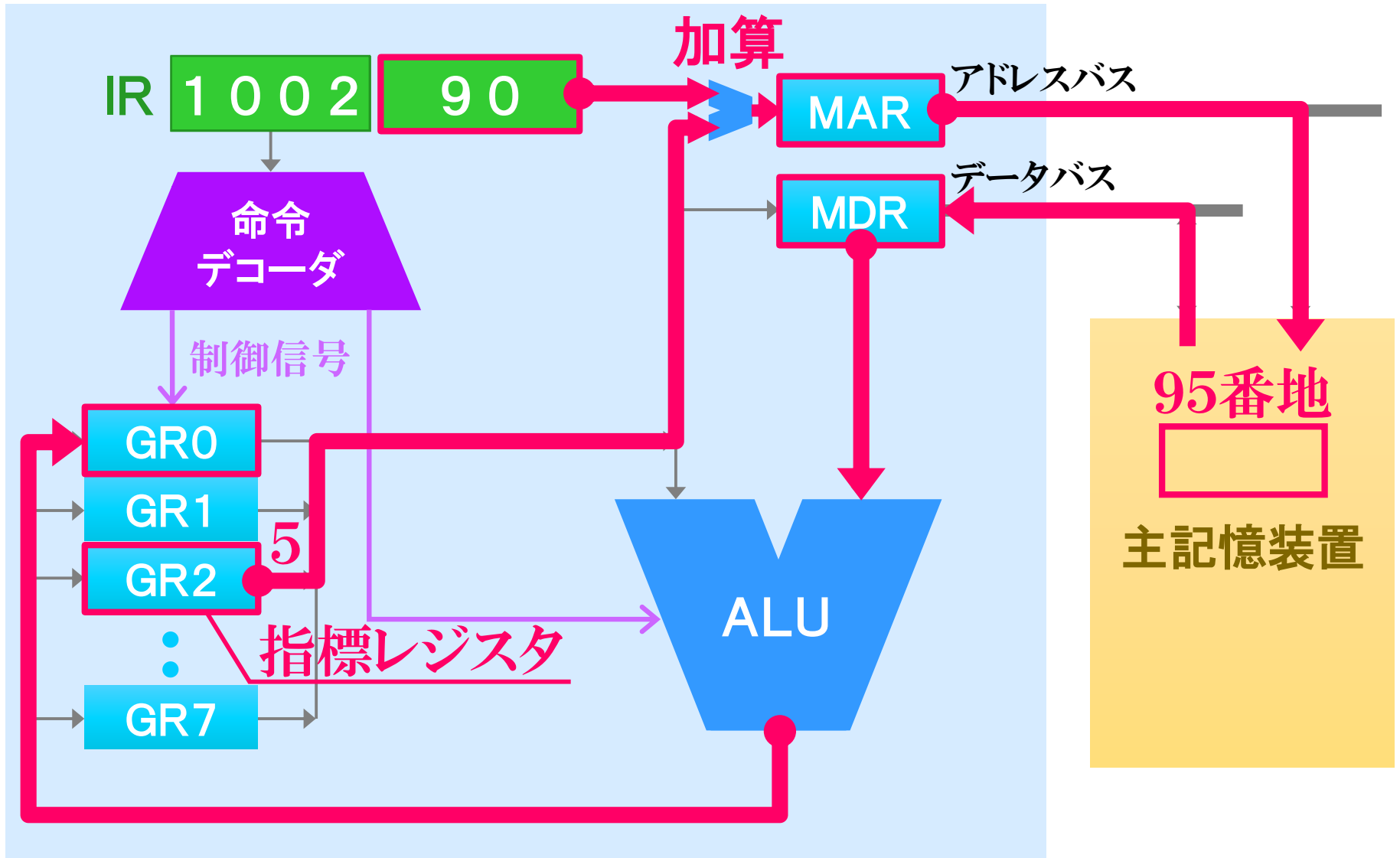
LD r1, adr, x

レジスタr1に、(adr+レジスタx)番地の値

を読み込む

実効アドレス

LD GR0, 90, GR2



指標レジスタを用いた命令記述

重要!

LD $r1, \text{adr}, x$

レジスタ $r1$ に、 $(\text{adr} + \text{レジスタ}x)$ 番地の値を読み込む

ST $r1, \text{adr}, x$

レジスタ $r1$ の値を、 $(\text{adr} + \text{レジスタ}x)$ 番地に書き込む

LAD $r1, \text{adr}, x$

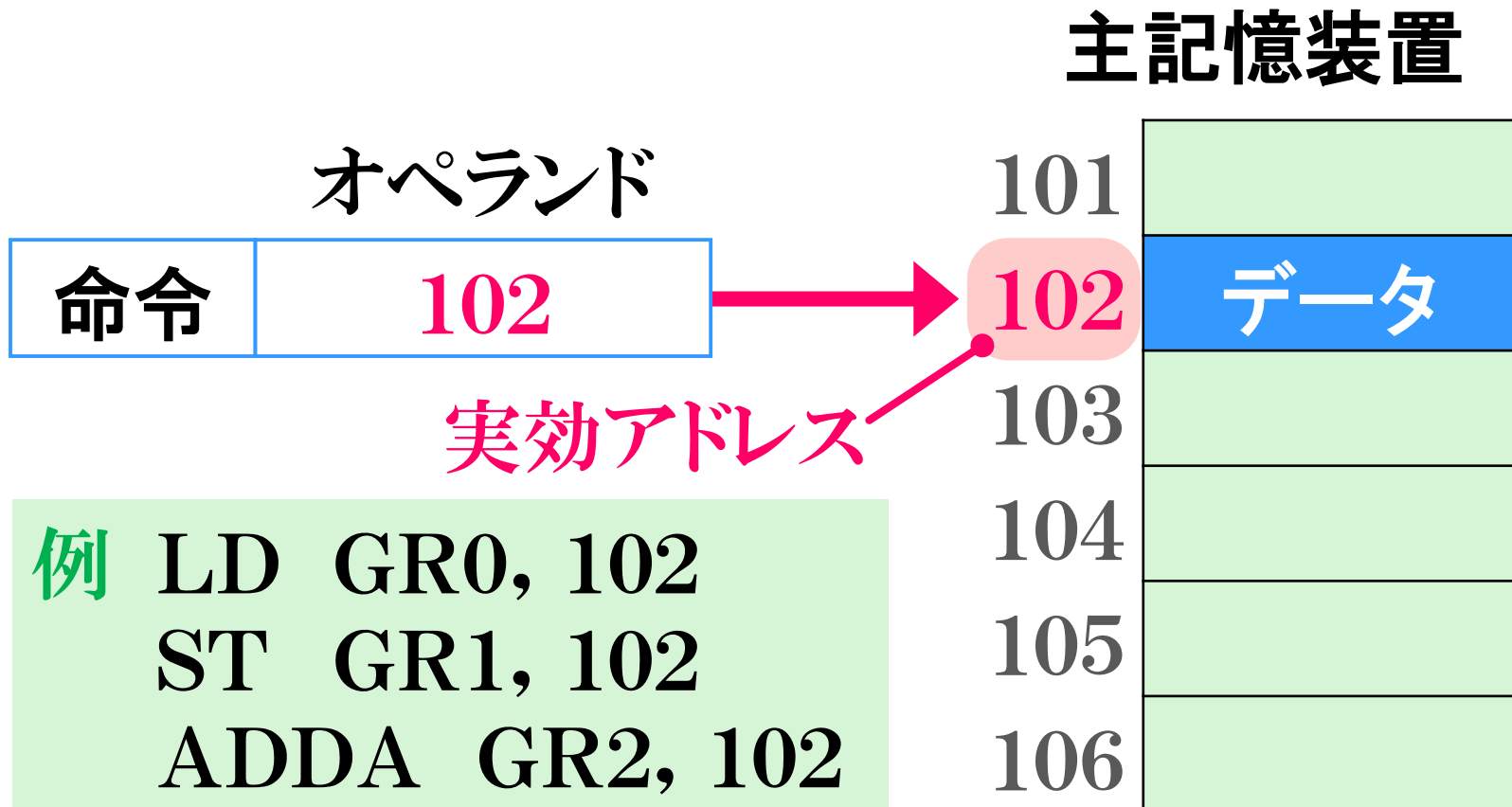
レジスタ $r1$ の値を、 $(\text{adr} + \text{レジスタ}x)$ にする

アドレス指定方式

- ✦ 直接アドレス指定
- ✦ 間接アドレス指定
- ✦ 相対アドレス指定
- ✦ 指標アドレス指定
- ✦ 即値アドレス指定

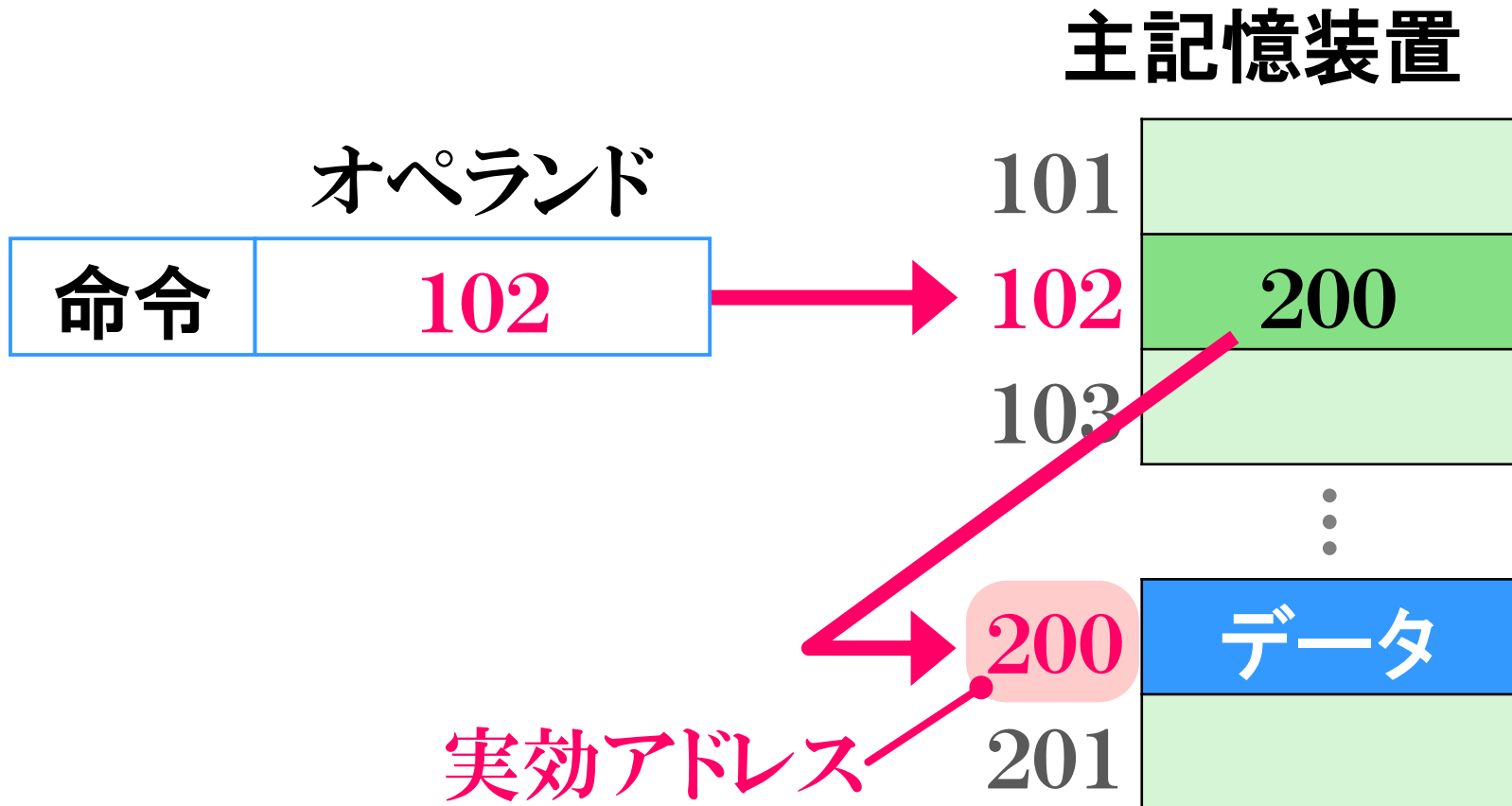
直接アドレス指定

オペランドのアドレス部の値を実効アドレスとする方式



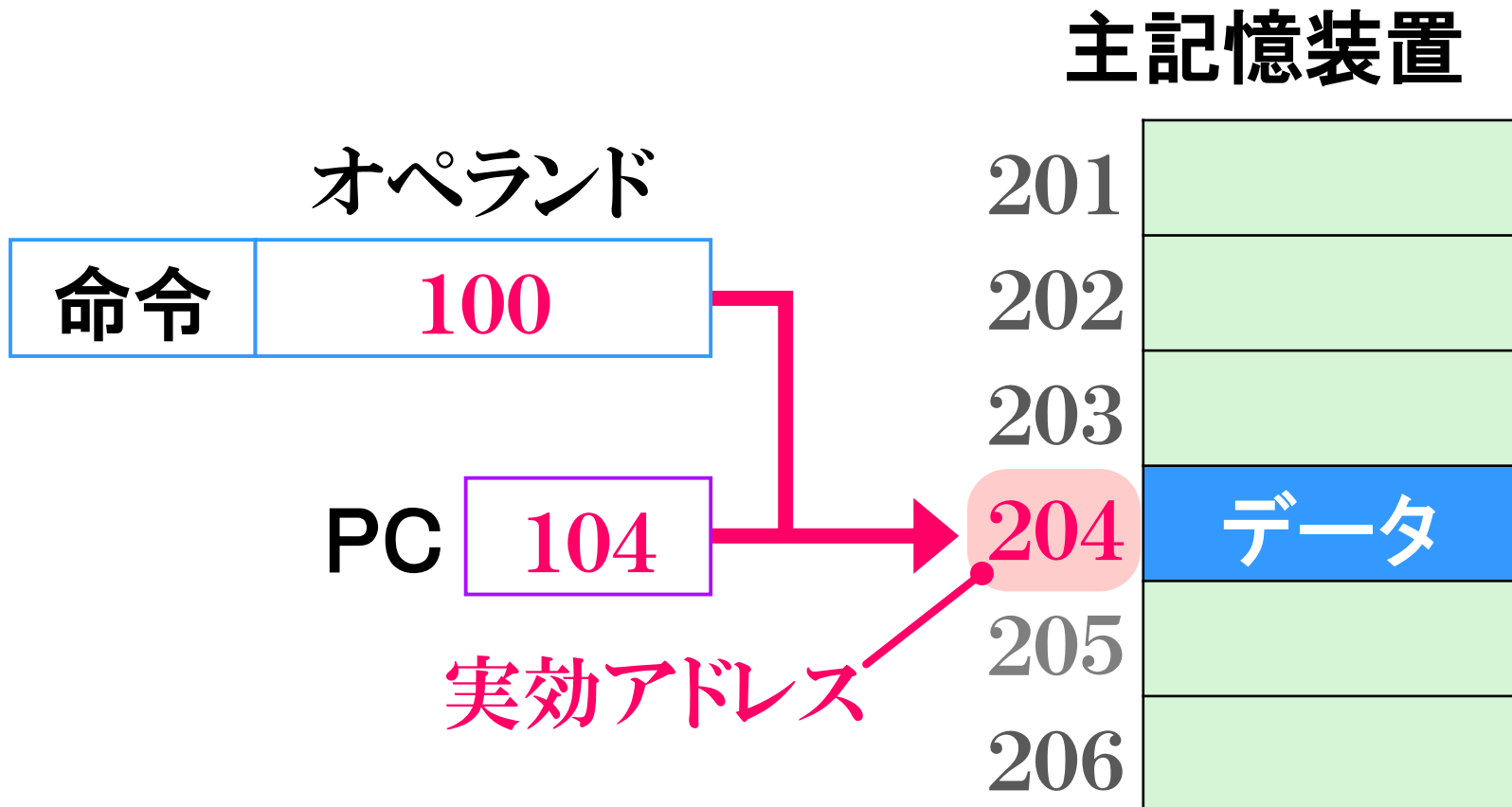
間接アドレス指定

オペランドのアドレス部に、実効アドレスを格納しているアドレスを格納する方式



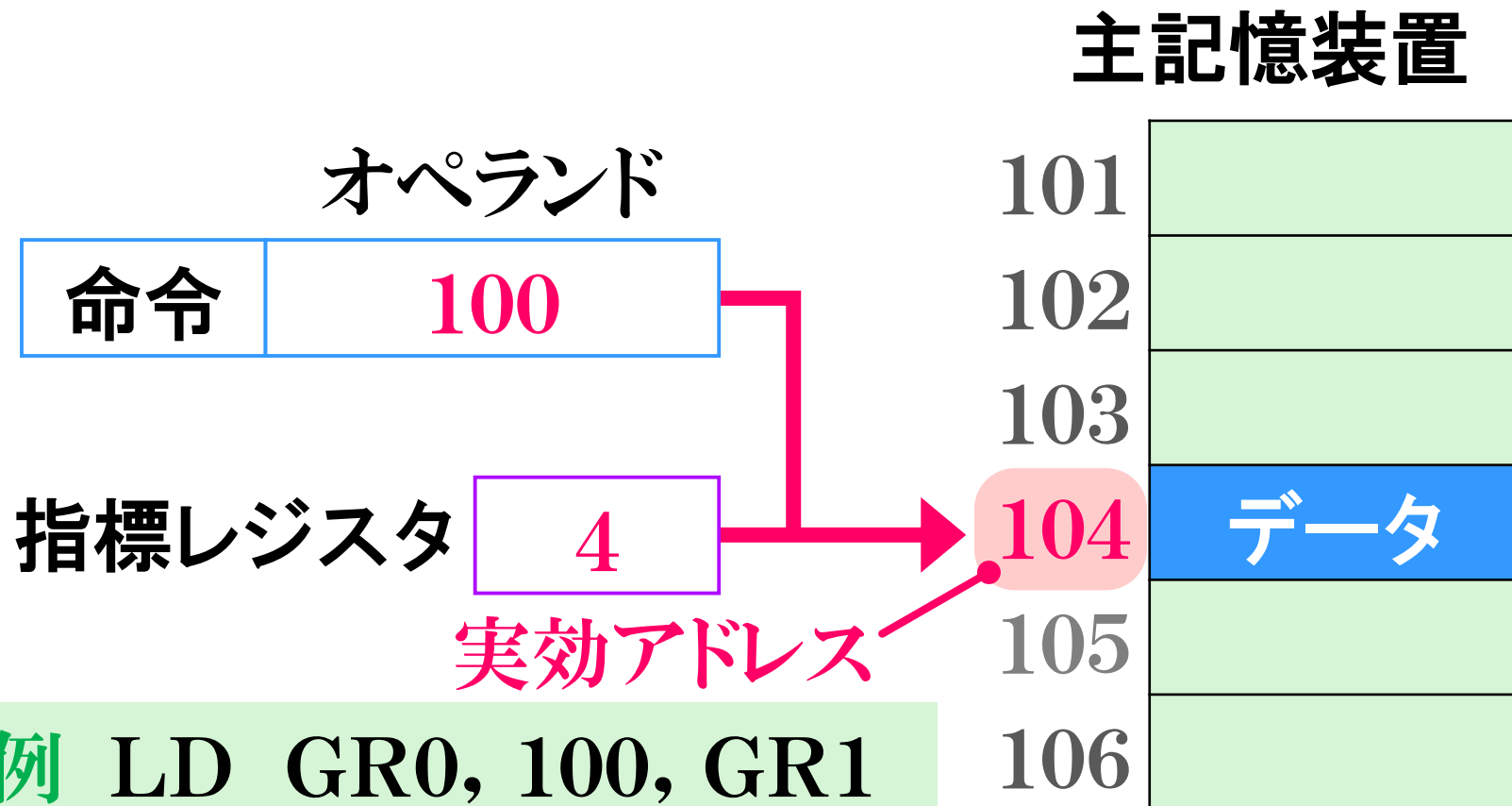
相対アドレス指定

オペランドのアドレス部の値に、PCの値を加算したものを実効アドレスとする方式



指標アドレス指定

オペランドのアドレス部の値に、指標レジスタの値を加算したものを実効アドレスとする方式



即値アドレス指定

オペランドのアドレス部にデータそのものを格納する方式

オペランド

| | |
|----|-----|
| 命令 | データ |
|----|-----|

データ = 実効アドレス

例 LAD GR1, 30

主記憶装置

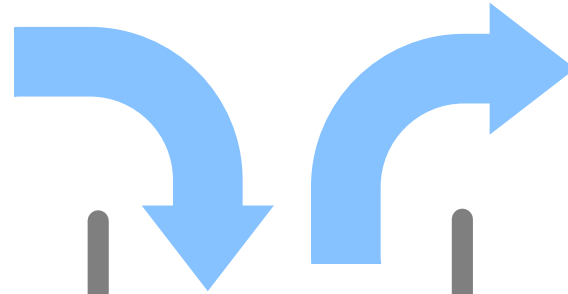
| | |
|-----|--|
| 101 | |
| 102 | |
| 103 | |
| 104 | |
| 105 | |
| 106 | |

スタック

データを一時的に蓄えておくための主記憶
装置上の記憶場所

後入れ先出し方式 (LIFO)

Push
データを入れる



Pop
データを取り出す

PUSH/POP

PUSH adr

adrをスタックへ入れる

PUSH adr, x

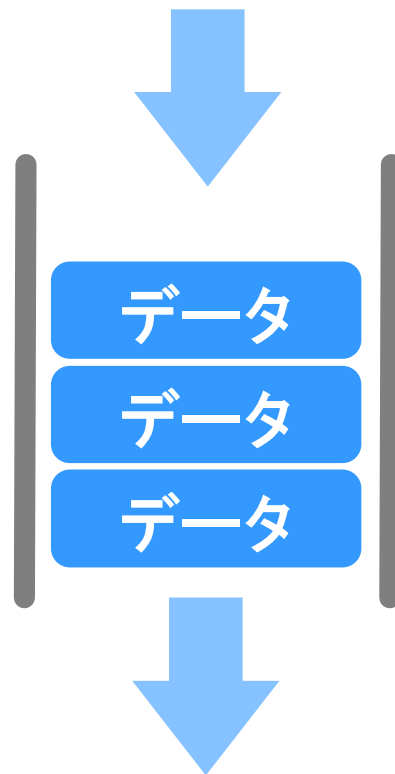
(adr+レジスタx)をスタックへ入れる

POP r1

スタックから取り出した値をレジスタr1へ書き込む

キュー

データを一時的に蓄えておく方式の一つ。
先入れ先出し方式 (FIFO)。待ち行列。



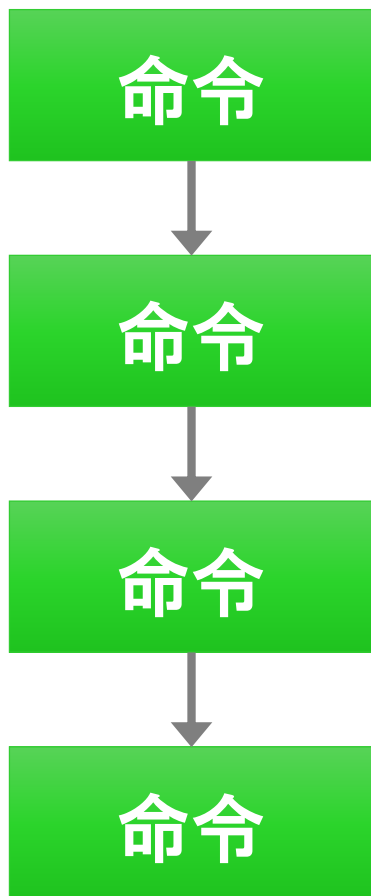
Enqueue

データを入れる

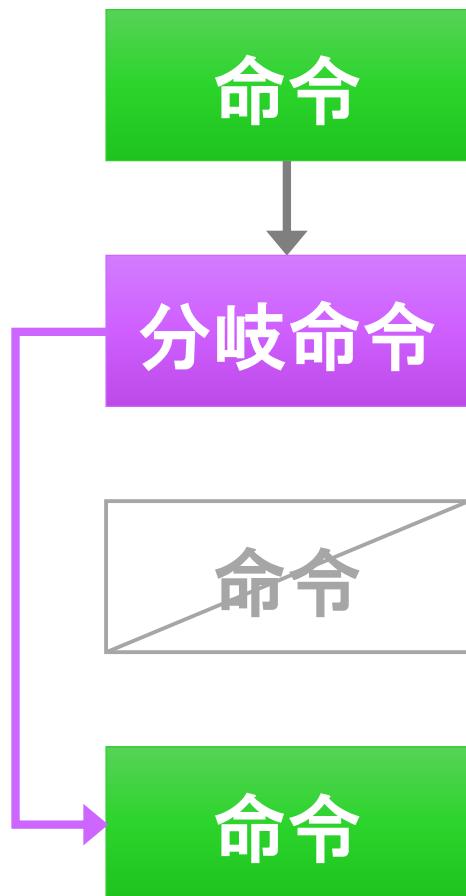
Dequeue

データを取り出す

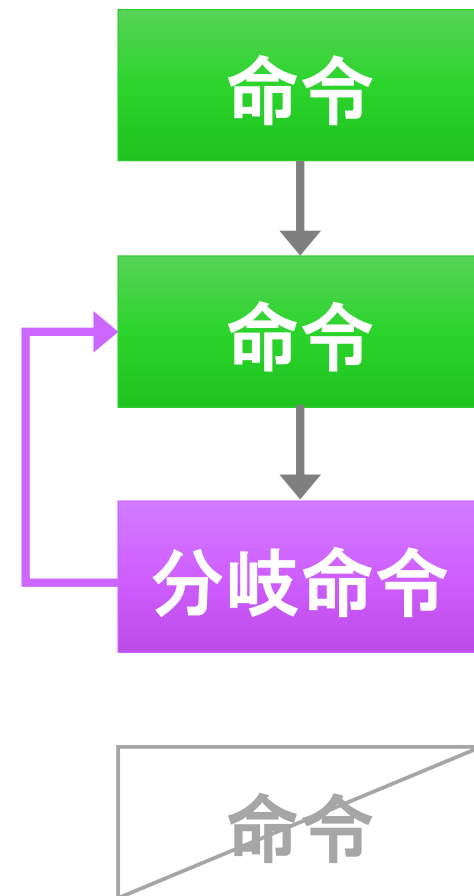
逐次実行と分岐



逐次実行



ジャンプ



繰り返し

分岐命令の種類

✿ 無条件分岐

✿ 条件分岐

✿ 負分岐

✿ 正分岐

✿ 零分岐

✿ 非零分岐

無条件分岐

プログラムの実行位置を移動する。

重要!

JUMP adr

次の実行を、**adr番地**の命令へ移動する。

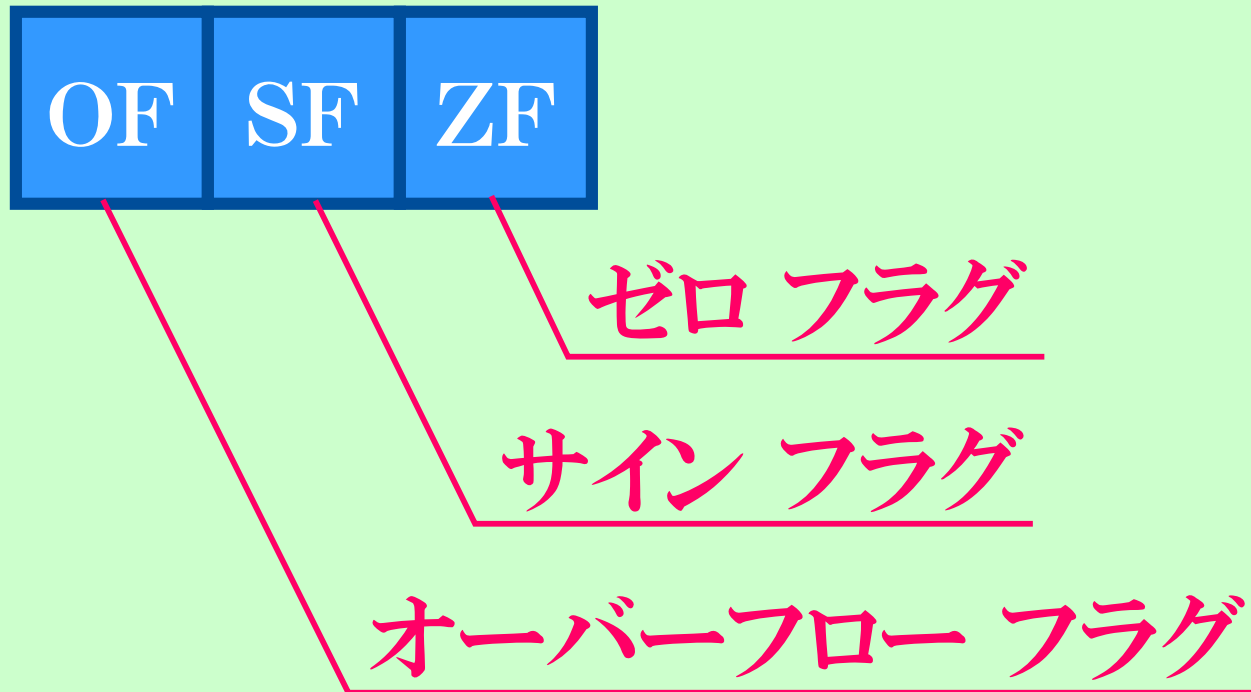


PC (プログラムカウンタ) の値を**adr**にする。

フラグレジスタ(FR)

重要!

演算の結果やデータ転送の値の状態を示すレジスタ。3つのフラグ(各1bit)から成る。



フラグ

重要!

❖ オーバーフローフラグ(OF)

値が $\begin{cases} \text{有効bitを越えたとき} & \rightarrow 1 \\ \text{有効bitを越えないとき} & \rightarrow 0 \end{cases}$

❖ サインフラグ(SF)

値が $\begin{cases} \text{負のとき} & \rightarrow 1 \\ \text{0または正のとき} & \rightarrow 0 \end{cases}$

❖ ゼロフラグ(ZF)

値が $\begin{cases} \text{0のとき} & \rightarrow 1 \\ \text{0以外するとき} & \rightarrow 0 \end{cases}$

フラグの値

| | 値 | | |
|----------------|-----|---|-----|
| | 正の数 | 0 | 負の数 |
| サインフラグ (SF) | 0 | 0 | 1 |
| ゼロフラグ (ZF) | 0 | 1 | 0 |

算術比較命令

2つの値の減算結果からFRを設定する。
減算結果の値は残さない。

重要!

CPA r1, r2

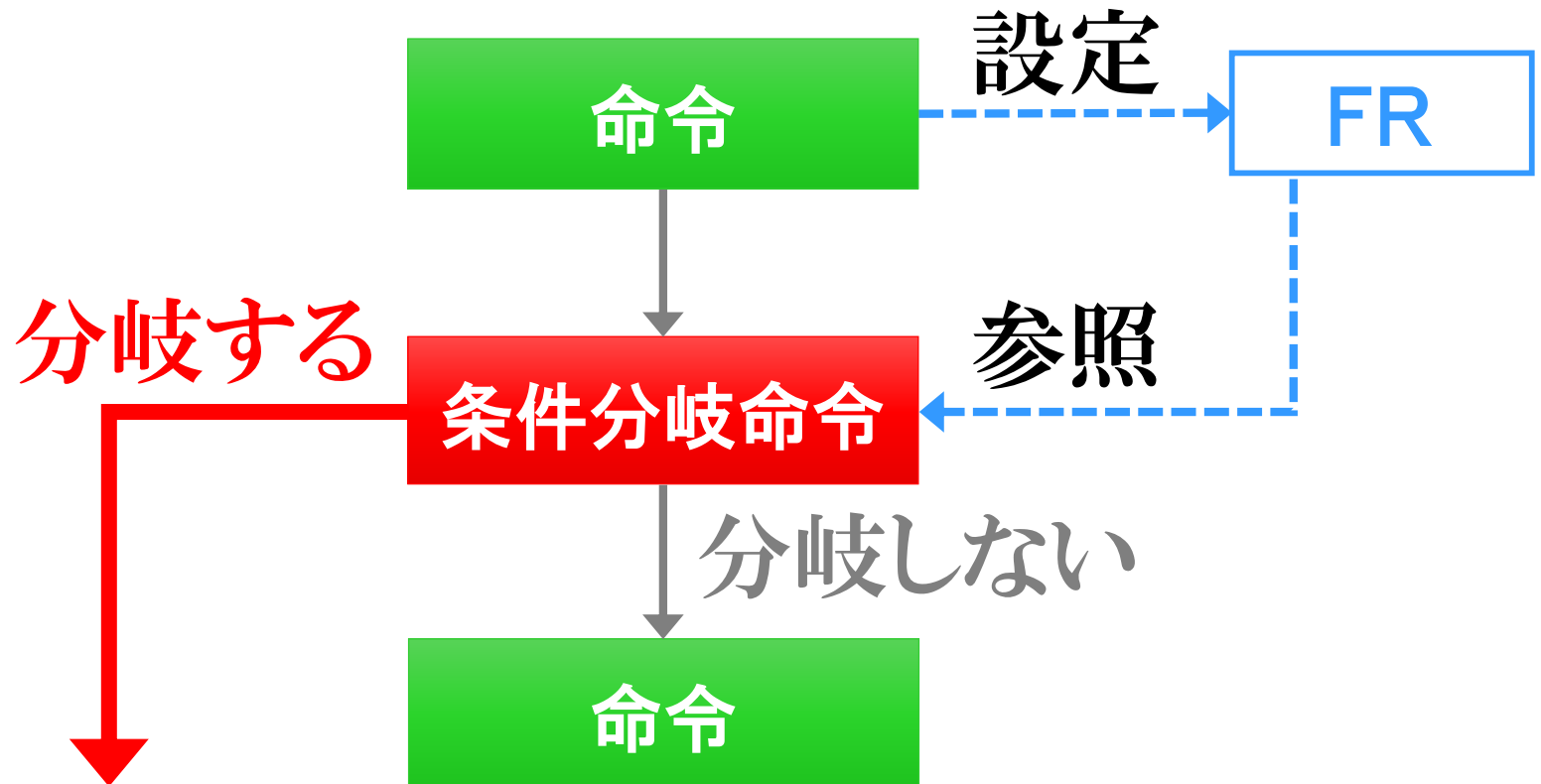
r1 - r2の実行結果からFRを設定する。

CPA r1, adr

r1 - adr番地の値の実行結果からFRを設定する。

条件分岐命令

FRの値によって、分岐するか、しないかを決定する。



条件分岐命令

重要!

JMI adr 負分岐

負 ($SF = 1$) のとき **adr番地** へ移動

JPL adr 正分岐

正 ($SF = 0, ZF = 0$) のとき **adr番地** へ移動

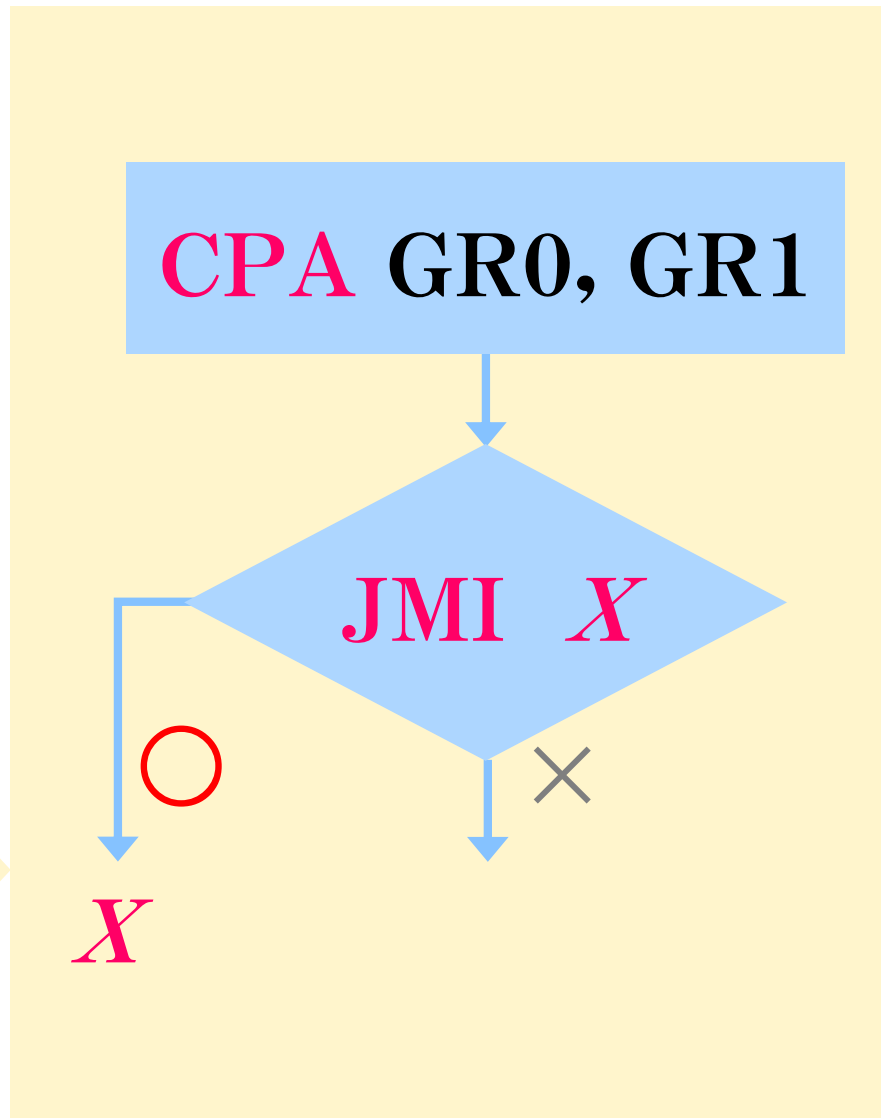
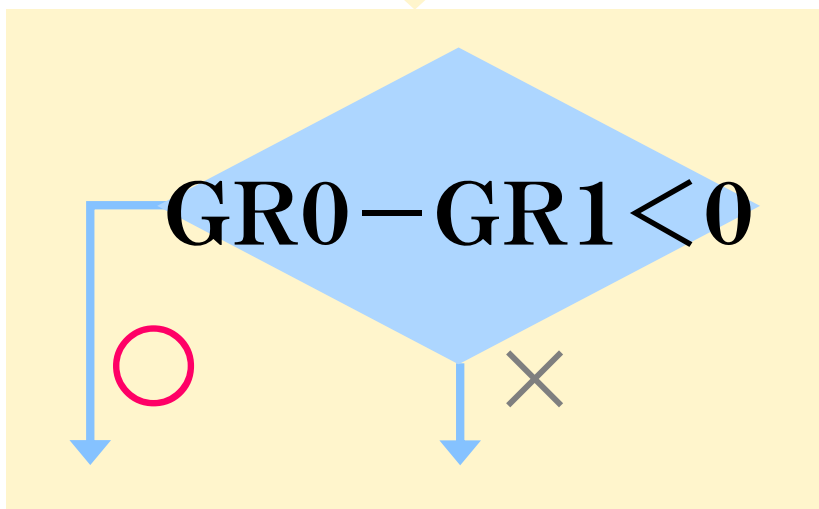
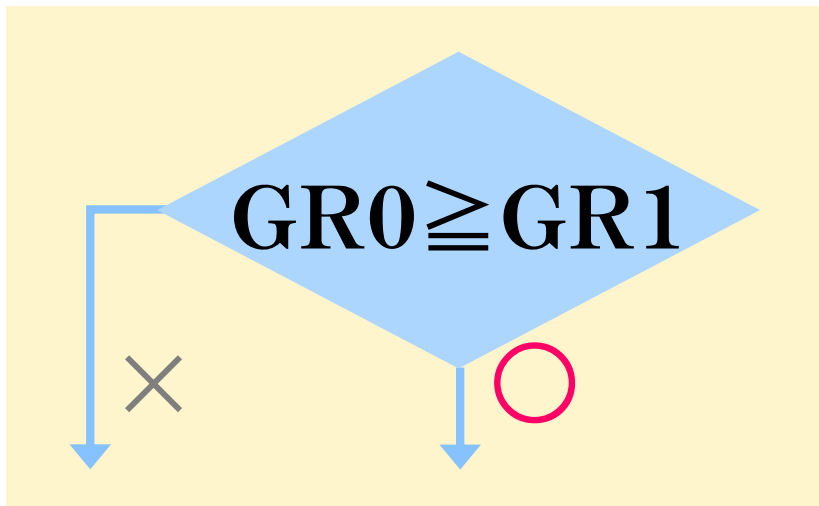
JZE adr 零分岐

0 ($ZF = 1$) のとき **adr番地** へ移動

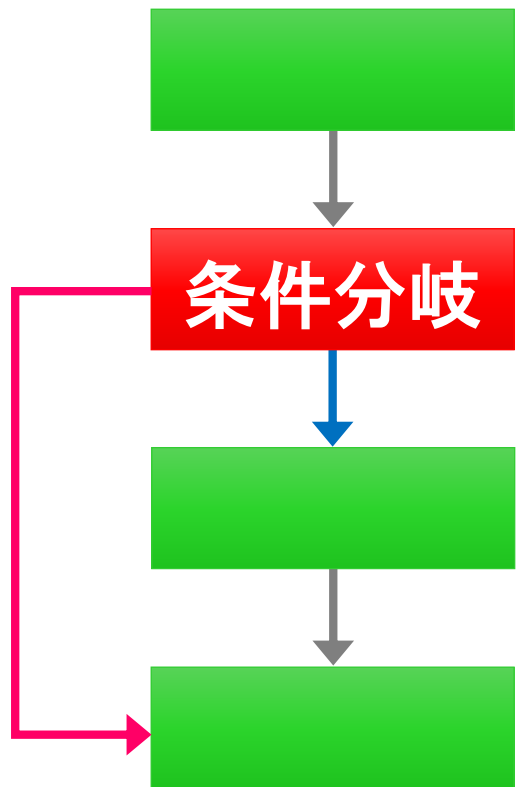
JNZ adr 非零分岐

0でない ($ZF = 0$) のとき **adr番地** へ移動

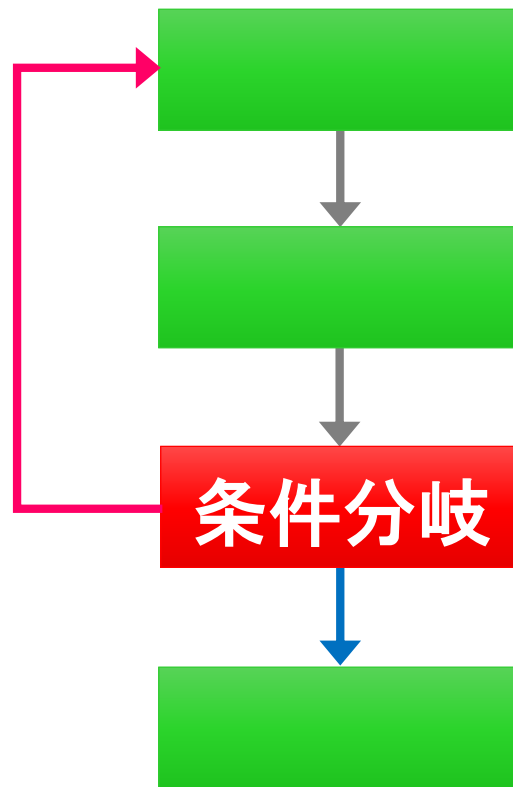
条件分岐の実現



条件分岐の使用

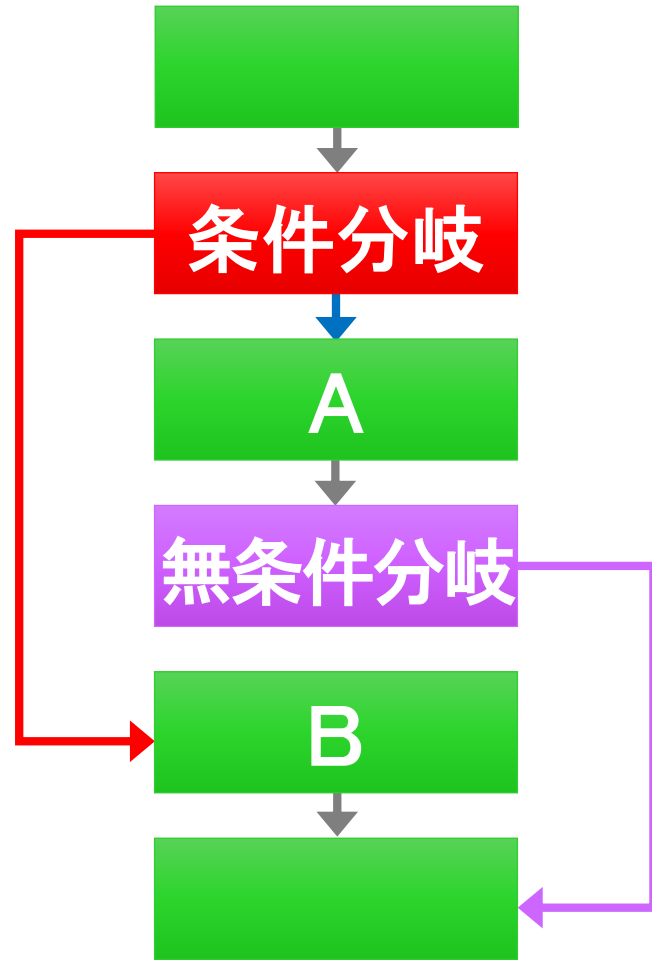


if文型

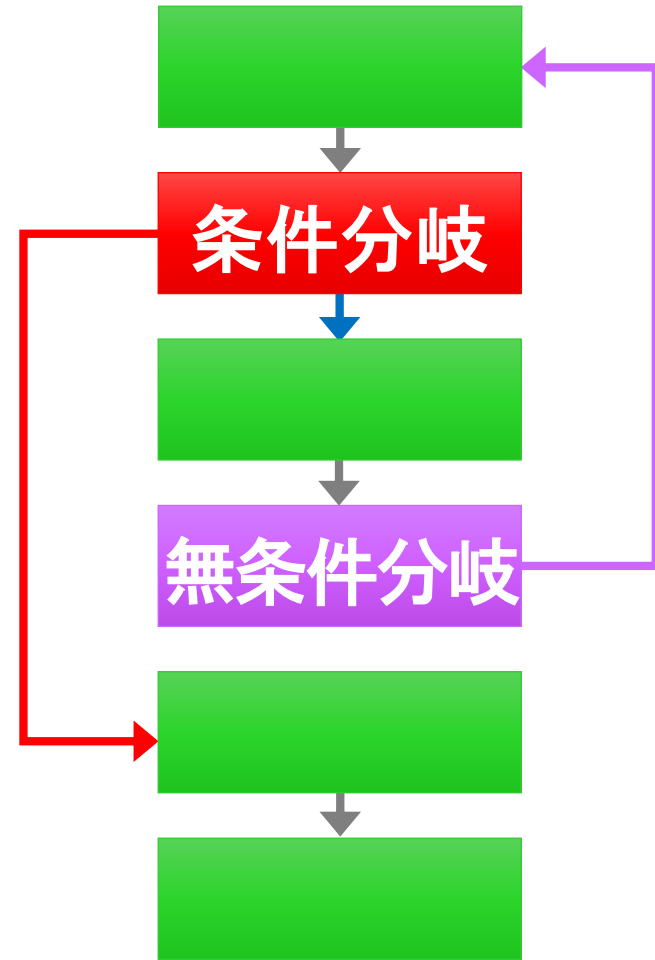


do-while文型

条件分岐と無条件分岐の使用



if-else文型



while文型

課題 3

❖ 配付プリント

❖ これまでの講義についての感想
どのような内容(感想・要望)でも良い。

提出日時: 1月7日(土) 講義開始前

❖ 理大専用のレポート用紙に書くこと。

❖ ホッチキスまたは糊で綴じること。

❖ 学生番号、氏名、講義名、提出日を書くこと。

❖ 途中の計算過程を書くこと。

定期試験 1月23日

CPUアーキテクチャ

重要!

♣ CISC 複雑命令セットコンピュータ

1つの命令で複雑な処理を実行できる。
組み込まれている命令数は多い。

♣ RISC 縮小命令セットコンピュータ

1つの命令は単純な処理だけを行う。
組み込まれている命令数は少ない。

※ CPUの進歩により、現在は明確な区別が無くなってきている。

CISCとRISCの特徴

重要!

| | CISC | RISC |
|----|---|---|
| 利点 | 小さいプログラムで、複雑な処理が行える。 | 命令の実行時間が短く、どれも同じ長さである。 CPUの回路構造が簡単になる。 |
| 欠点 | 命令の実行時間が長く、命令ごとに時間が違う。 CPUの回路構造が複雑になる。 | 複雑な処理をしたいとき、プログラムが大きくなる。 |
| 用途 | パソコン用CPUなど | 組み込み機器用小型マイコンなど |

CPUの高速化法

✦ パイプライン処理

✦ マルチプロセッサ, マルチコア

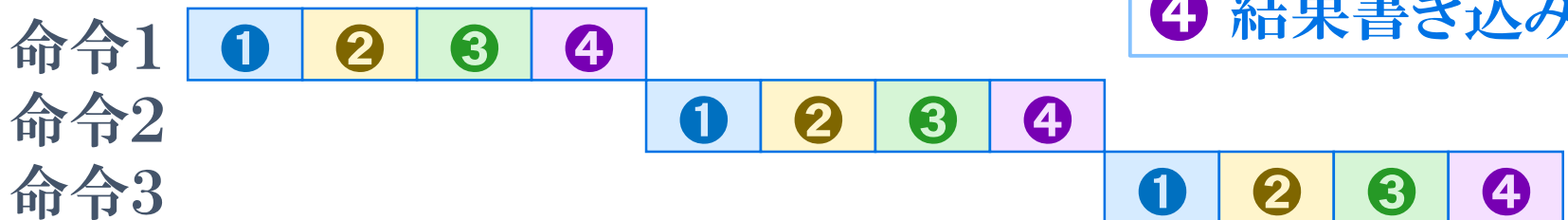
✦ SIMD演算

パイプライン処理

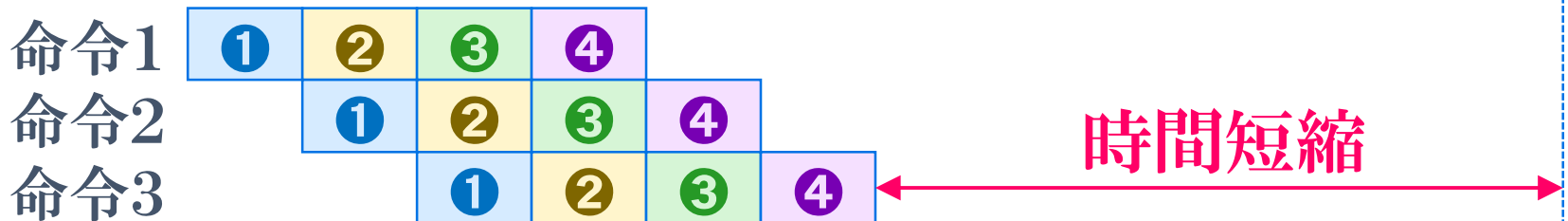
重要!

複数の命令を並行して実行することで、処理速度を上げる。

逐次処理



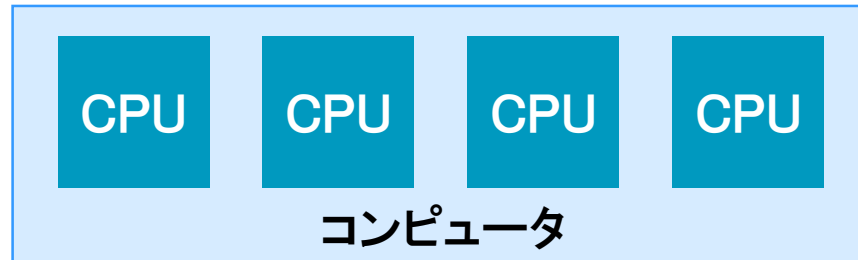
パイプライン処理



マルチプロセッサ／マルチコア

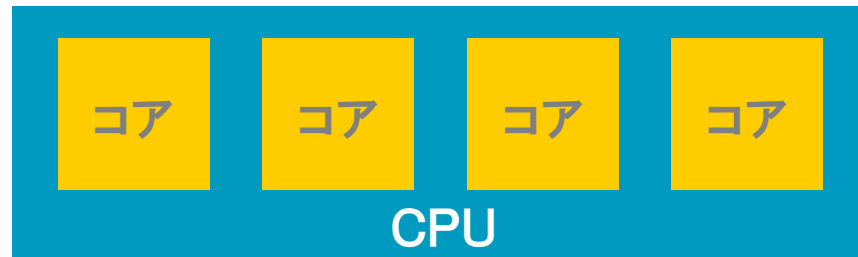
❖ マルチプロセッサ

1台のコンピュータの中に複数のCPUを搭載



❖ マルチコア

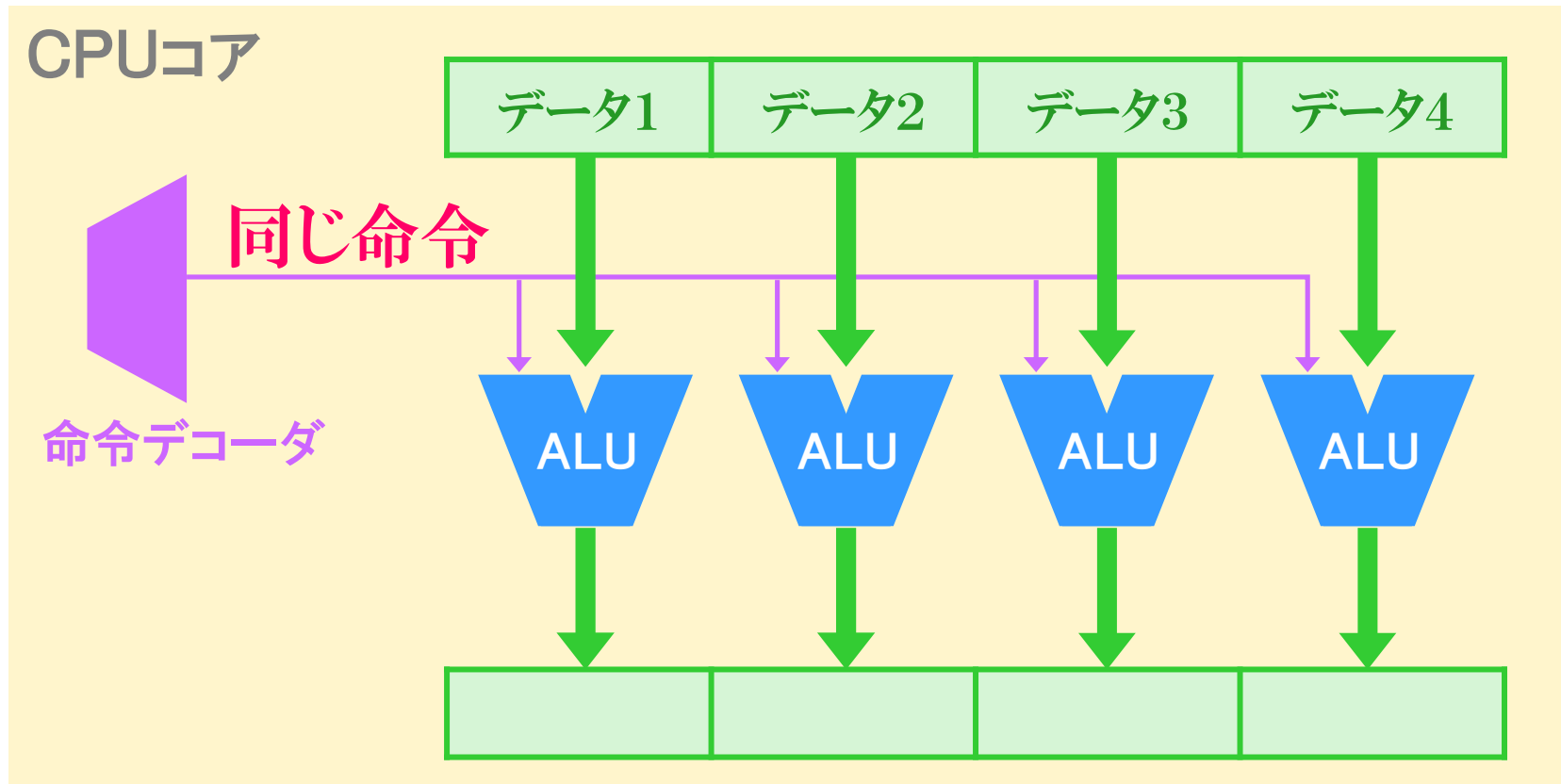
1個のCPUの中に複数のCPUコアを搭載



SIMD方式

重要!

1つの命令で複数のデータを処理することにより、処理速度を上げる。



CPUの性能評価

重要!

❖ 平均命令実行時間

1命令の実行に要する時間(秒)の平均値

❖ MIPS

1秒間に実行できる命令数(百万単位)

❖ FLOPS

1秒間に実行できる浮動小数点数演算の回数

定期試験(予定)

日時 2017年1月23日 13時15分

場所 10242講義室

出題範囲

- 順序回路(フリップフロップ)
- CPUの構造と働き
- アセンブリ言語
- CPUの高速化、CPUの性能評価
- 中間試験範囲から基礎的問題を数問