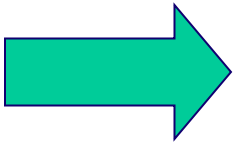
as-3. プログラムカウンタと命令実行サイクル

（68000 アセンブラ）

URL: https://www.kkaneko.jp/cc/as/index.html

金子邦彦

アセンブラ
プログラム
ファイル



アセンブラ
m68k-as など

HEX
ファイル

これは
ファイル

```
.data
x:
        .dc.w 10
y:
        .dc.w 20
z:
        .dc.w 0
.text
        move.w x,%d0
        add.w y,%d0
        move.w %d0,z

        .dc.w 0x4048
        stop #0
.end
```

テキストエディタなど
で記述．add.s など

```
S00600004844521B
S2140000003039000000018D0790000001A33C0000014
S20C000010001C40484E7200007F
S20A000018000A00140000BF
S5030003F9
S804000000FB
```

メモリのどこに何を
置くかを書いたファイル

自動生成．add.abs など

メモリにロード

```
000000: 30 39 00 00 00 18 d0 79 00 00 00 1a 33 c0 00 00
000010: 00 1c 40 48 4e 72 00 00 00 0a 00 14 00 00 00 00
000020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

メモリの中身

```
S0 06 0000484452 1B
S2 14 000000 303900000018D0790000001A33C00000 14
S2 0C 000010 001C40484E720000 7F
S2 0A 000018 000A00140000 BF
S5 03 0003 F9
S8 04 000000 FB
```

ステータスレコード（ファイル名など）

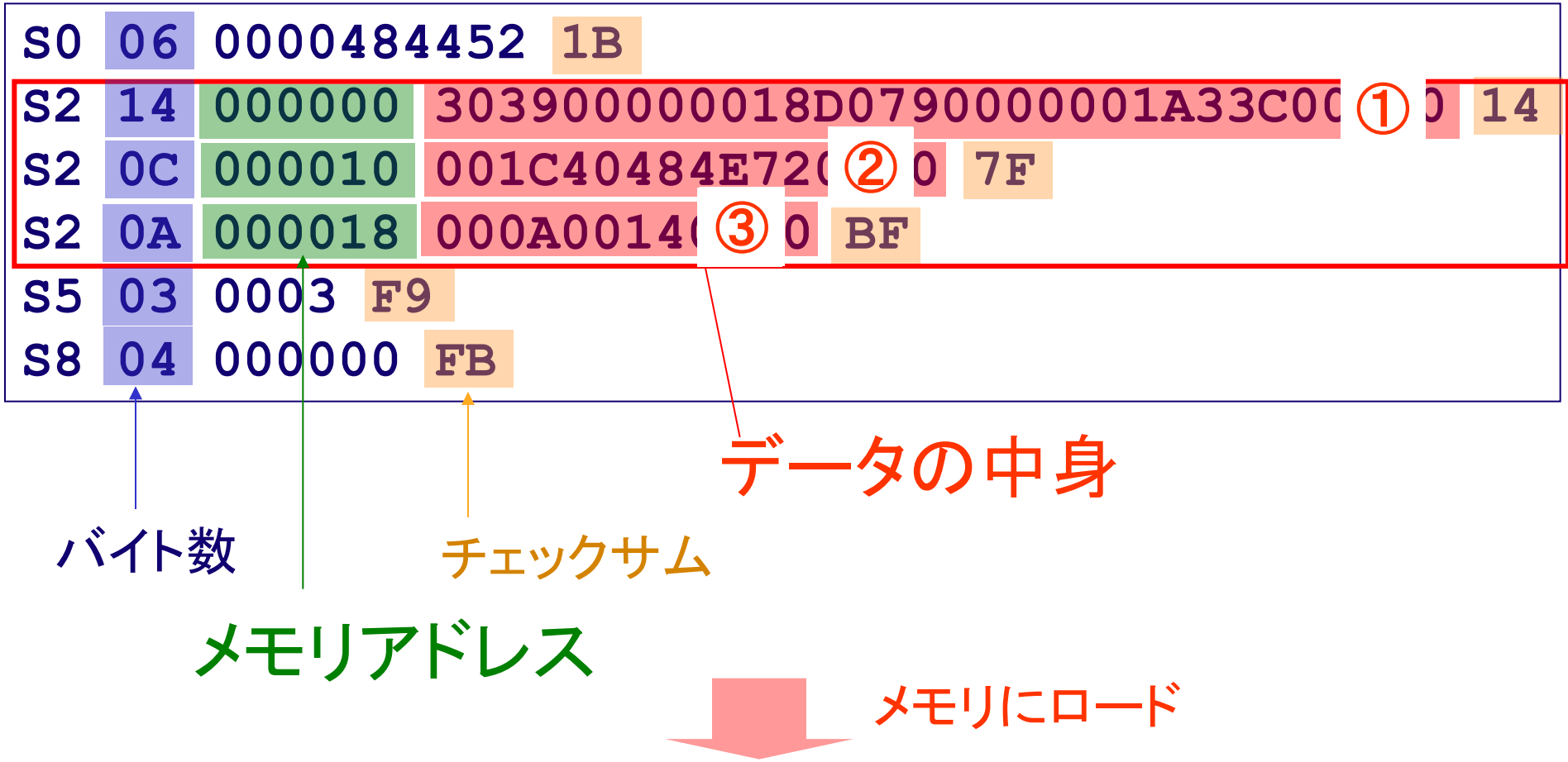
データレコード

データレコード数

終了を示す

データレコード：　メモリにロードされるべき中身

その他のレコード：　管理情報



メモリにロード

```
000000: 30 39 00 00 00 18 d0 79 00 00 00 1a 33 c0 00 00
000010: 00 1c 40 48 4e 72 00 00 00 0a 00 14 00 00 00 00
000020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

メモリの中身

```
.data
x:
        .dc.w 10
y:
        .dc.w 20
z:
        .dc.w 0
.text
        move.w x,%d0
        add.w y,%d0
        move.w %d0,z
```

x → 0x000018
y → 0x00001a
z → 0x00001c

メモリ読み出し (x から)
→ データレジスタ d0 に格納

アセンブラプログラムファイル中ではラベル名

```
000000: 30 39 00 00 00 18 d0 79 00 00 00 1a 33 c0 00 00
000010: 00 1c 40 48 4e 72 00 00 00 0a 00 14 00 00 00 00
000020: 00 00 00 00 0                    00 00 00 00 00
000030: 00 00 00 00 0                    00 00 00 00 00
```

メモリアドレス

```
.data
x:
        .dc.w 10
y:
        .dc.w 20
z:
        .dc.w 0
.text
        move.w x,%d0
        add.w y,%d0
        move.w %d0,z
```

x → 0x000018
y → 0x00001a
z → 0x00001c

メモリ読み出し (x から)
→ データレジスタ d0 に格納

アセンブラプログラムファイル中ではラベル名

```
000000: 30 39 00 00 00 18 d0 79 00 00 00 1a 33 c0 00 00
000010: 00 1c 40 48 4e 72 00 00 00 0a 00 14 00 00 00 00
0                 0 00                 00 00 00 00
0                 0 00                 00 00 00 00
```

1. データ転送を行え
2. 転送先は D0
3. 処理はワード単位

メモリアドレス 0x00000018 を扱う

```
.data
x:
        .dc.w 10
y:
        .dc.w 20
z:
        .dc.w 0
.text
        move.w x,%d0
        add.w y,%d0
        move.w %d0,z
```

x → 0x000018
y → 0x00001a
z → 0x00001c

メモリ読み出し（yから）と加算
→ データレジスタ d0 との加算
→ 加算の結果はd0に格納

```
S0 06 0000484452 1B
S2 14 000000 303900000018D0790000001A33C00000 14
S2 0C 000010 001C40484E720000 7F
S2 0A 000
S5 03 000
S8 04 000
```

1. 加算を行え
2. 使用するレジスタは D0
3. 処理はワード単位

メモリアドレス 0x0000001A を扱う

```
.data
x:
        .dc.w 10
y:
        .dc.w 20
z:
        .dc.w 0
.text
        move.w x,%d0
        add.w y,%d0
        move.w %d0,z
```

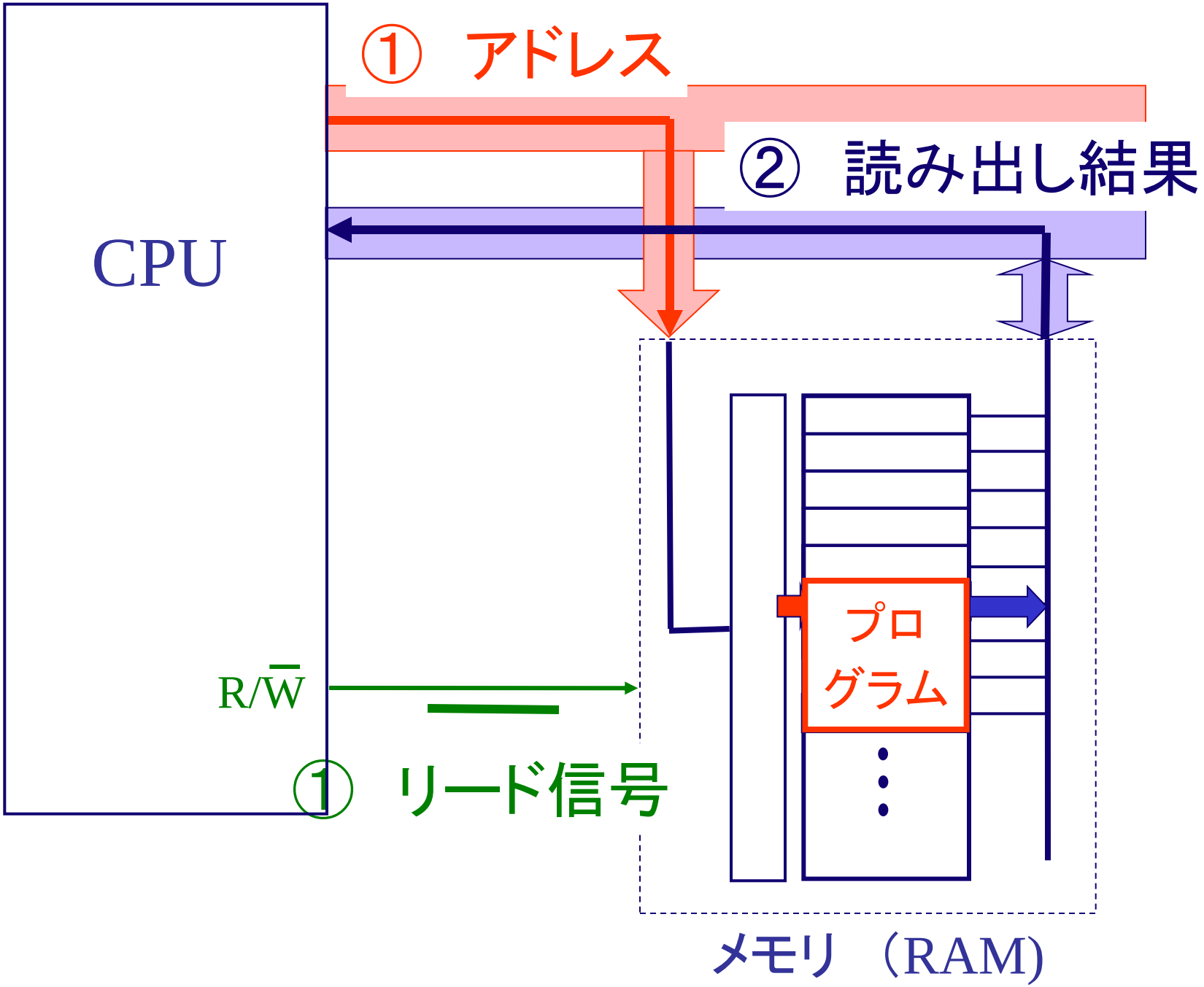
x → 0x000018
y → 0x00001a
z → 0x00001c

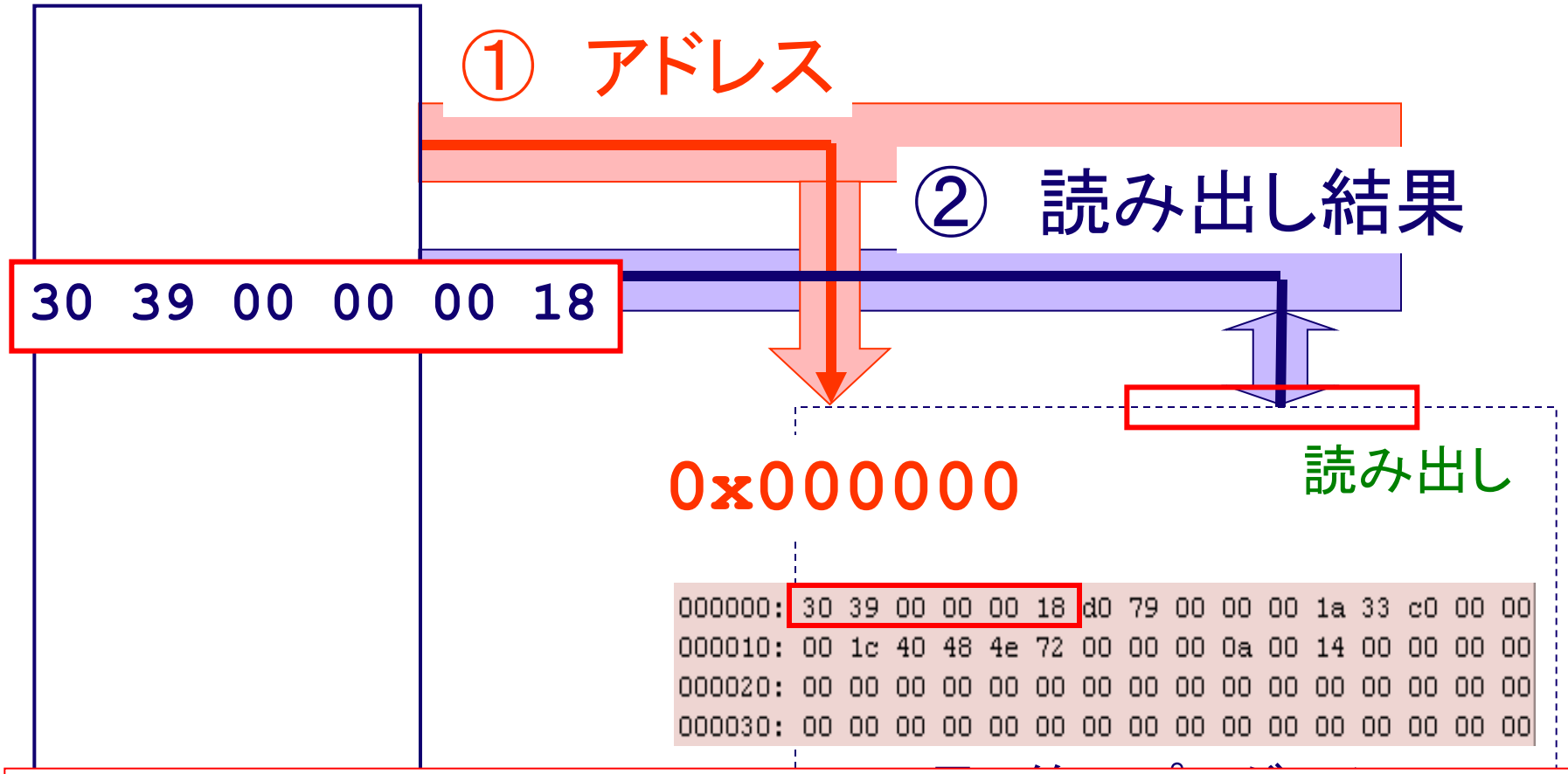
メモリ書き込み (z へ)
→ データレジスタ d0 の中身を書き込む

```
S0 06 0000484452 1B
S2 14 000000 303900000018D0790000001A33C00000 14
S2 0C 000010 001C40484E720000 7F
S2 0A 000018 000A00140000 BF
S5 03
S8 04
```

メモリアドレス 0x0000001C を扱う

1. データ転送を行え
2. 転送元は D0
3. 処理はワード単位

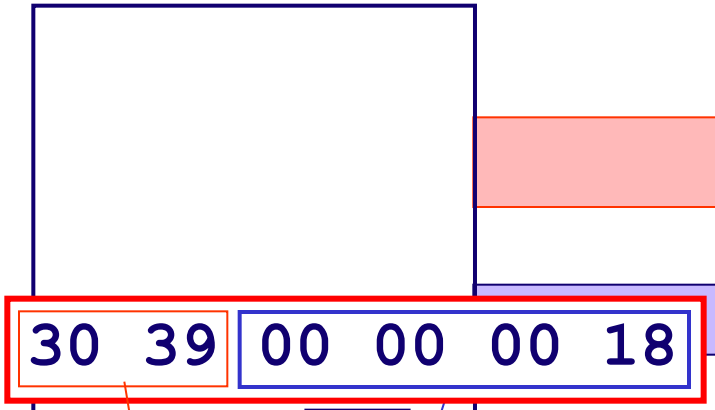

CPU
① アドレス
② 読み出し結果
R/$\overline{W}$
① リード信号
プロ
グラム
メモリ （RAM)



命令フェッチ

メモリ中の「プログラム命令」を，CPUへ読み込むこと

この授業では，説明を簡単にするために，1命令分が1度に読み出される説明としている



1. データ転送を行え
2. 転送先は D0
3. 処理はワード単位

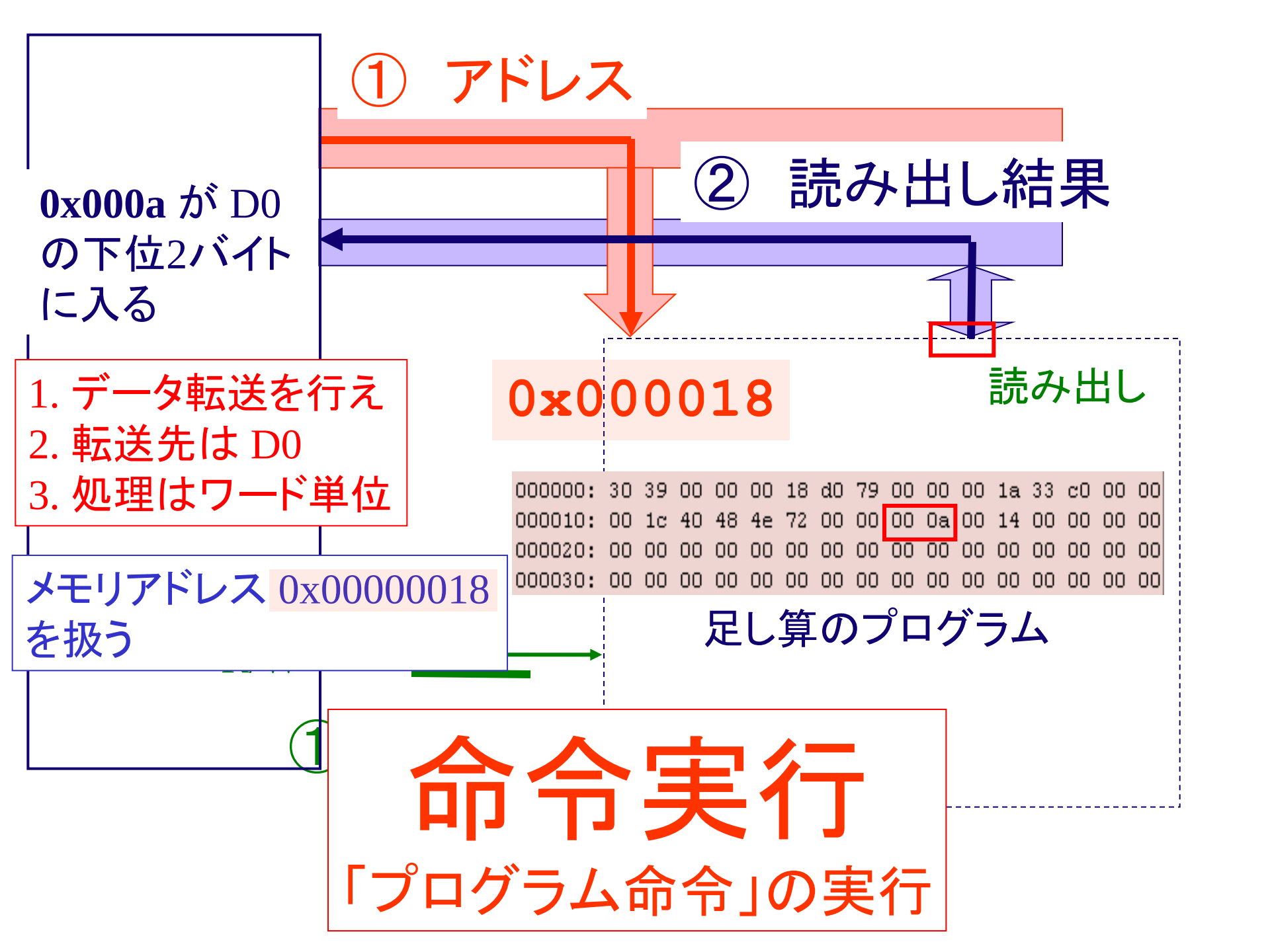
メモリアドレス 0x00000018 を扱う

```
000000: 30 39 00 00 00 18 d0 79 00 00 00 1a 33 c0 00 00
000010: 00 1c 40 48 4e 72 00 00 00 0a 00 14 00 00 00 00
000020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

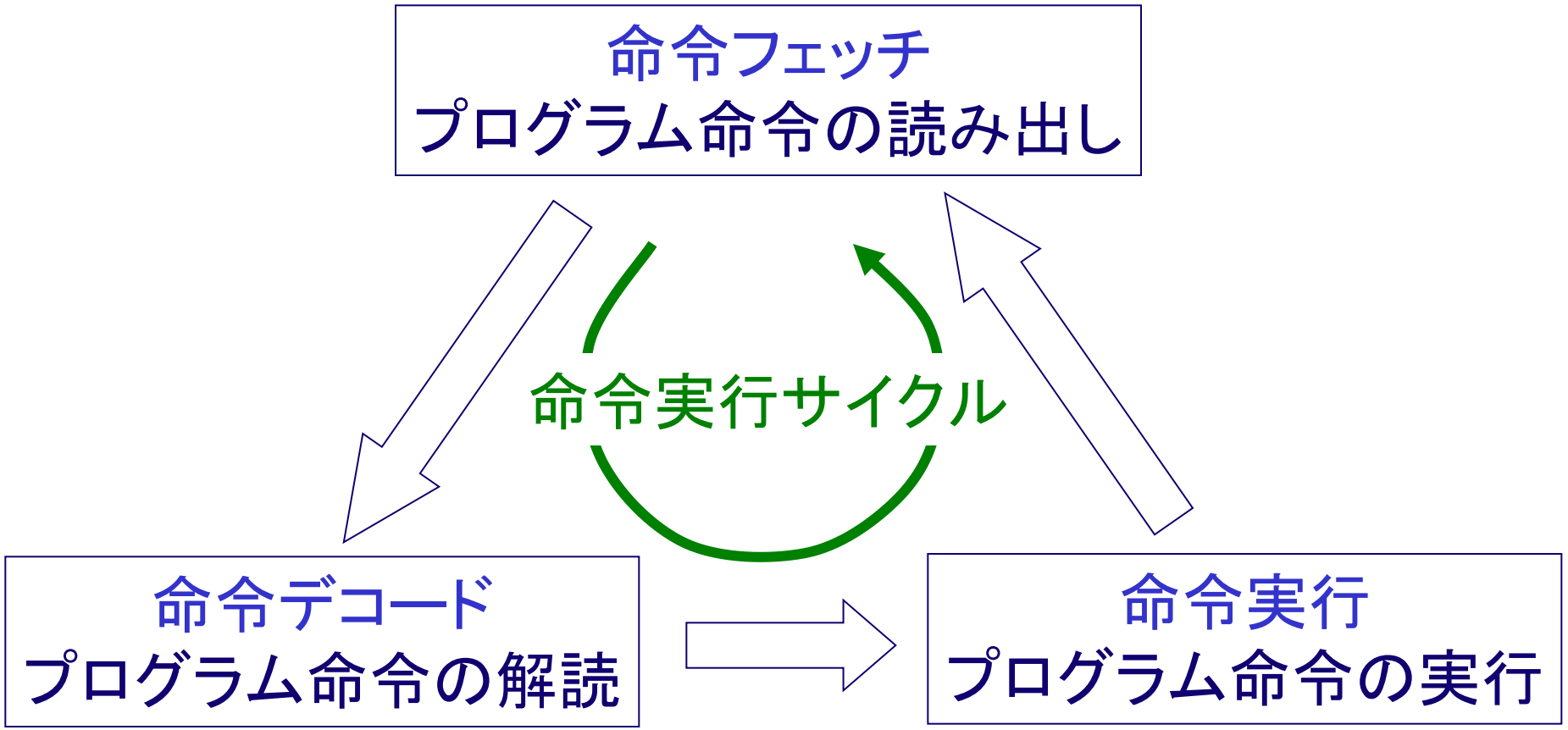
足し算のプログラム

命令デコード

CPU内で，プログラム命令の解読が行われること


① アドレス
② 読み出し結果
0x000a が D0 の下位2バイトに入る
1. データ転送を行え
2. 転送先は D0
3. 処理はワード単位
0x000018
読み出し
000000: 30 39 00 00 00 18 d0 79 00 00 00 1a 33 c0 00 00
000010: 00 1c 40 48 4e 72 00 00 00 0a 00 14 00 00 00 00
000020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
メモリアドレス 0x00000018 を扱う
足し算のプログラム
命令実行
「プログラム命令」の実行

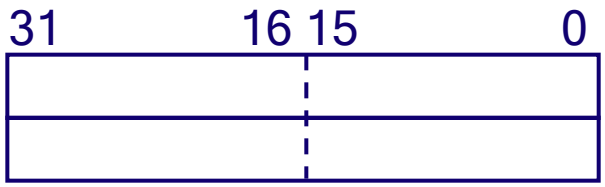
命令実行サイクル



各命令ごとに「命令実行サイクル」を繰り返す

CPU 68000 では

データレジスタ (data



ユーザスタックポインタ

スーパバイザスタックポインタ

(user stack pointer, supervisor stack pointer)

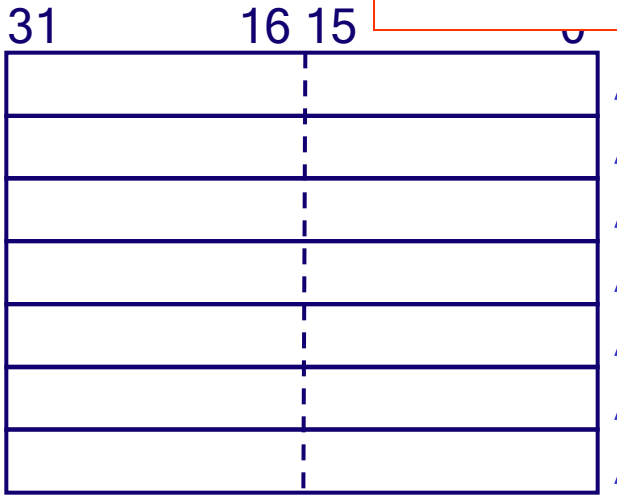
SPとも書く

32ビット長

プログラムカウンタの機能について，今から説明する

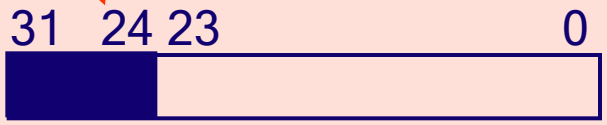
ステータスレジスタ (status register)

16ビット長



アドレスレジスタ (address registers)

32ビット長



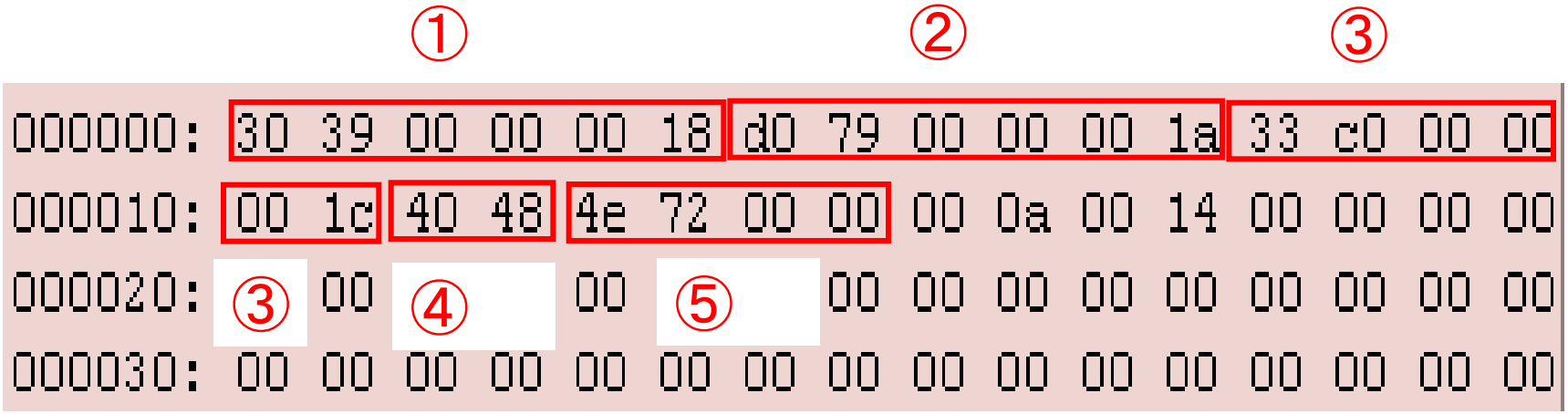
PCプログラムカウンタ (program counter)

32ビット長

68000アセンブラ
プログラムファイル

```
.data
x:
        .dc.w 10
y:
        .dc.w 20
z:
        .dc.w 0
.text
        ① move.w x,%d0
        ② add.w y,%d0
        ③ move.w %d0,z

        ④ .dc.w 0x4048
        ⑤ stop #0
.end
```

プログラムは順次実行される ①→②→③→・・・



メモリの中身

命令フェッチは，順次行われる ①→②→③→・・・

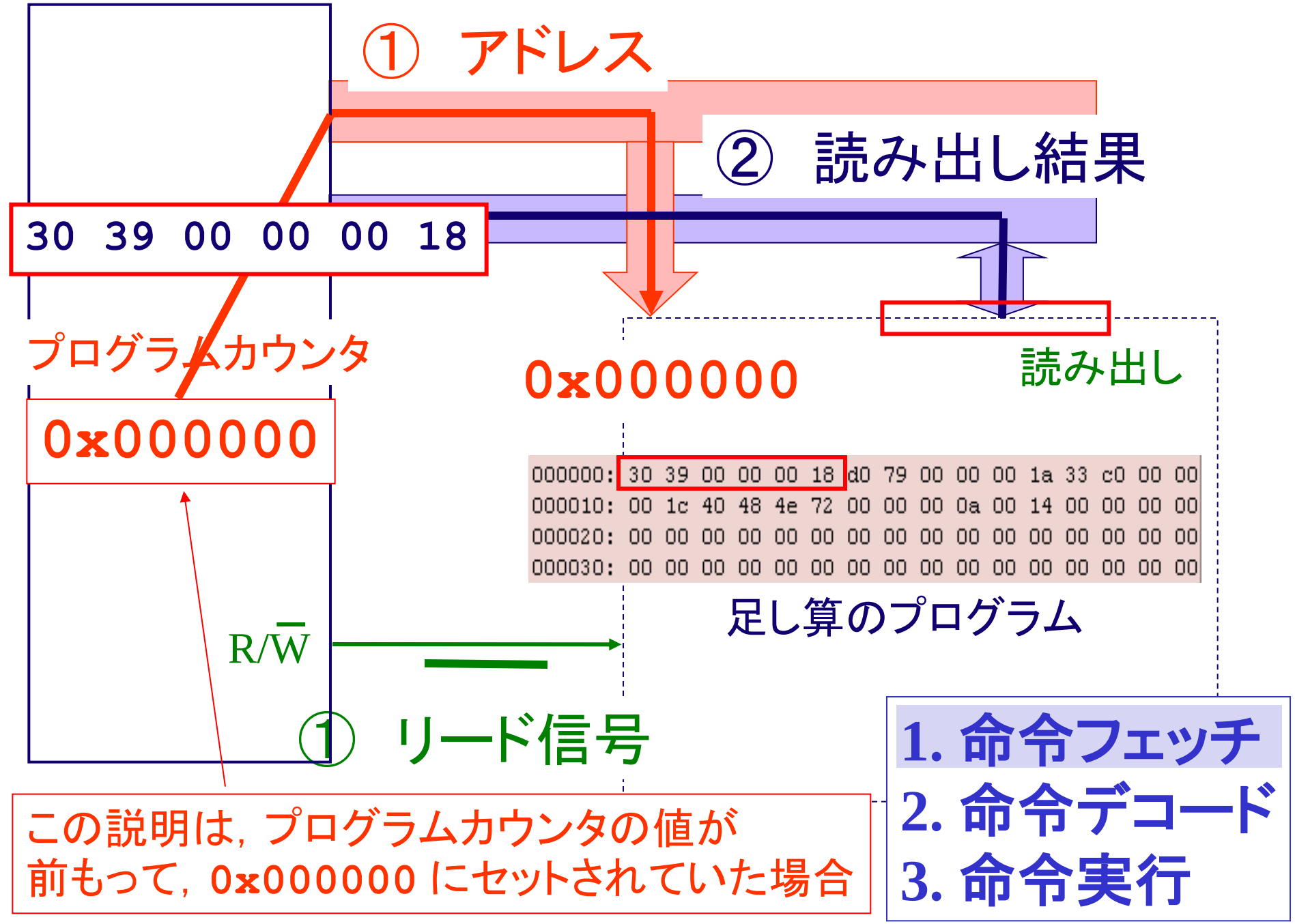
開始は 0x000000

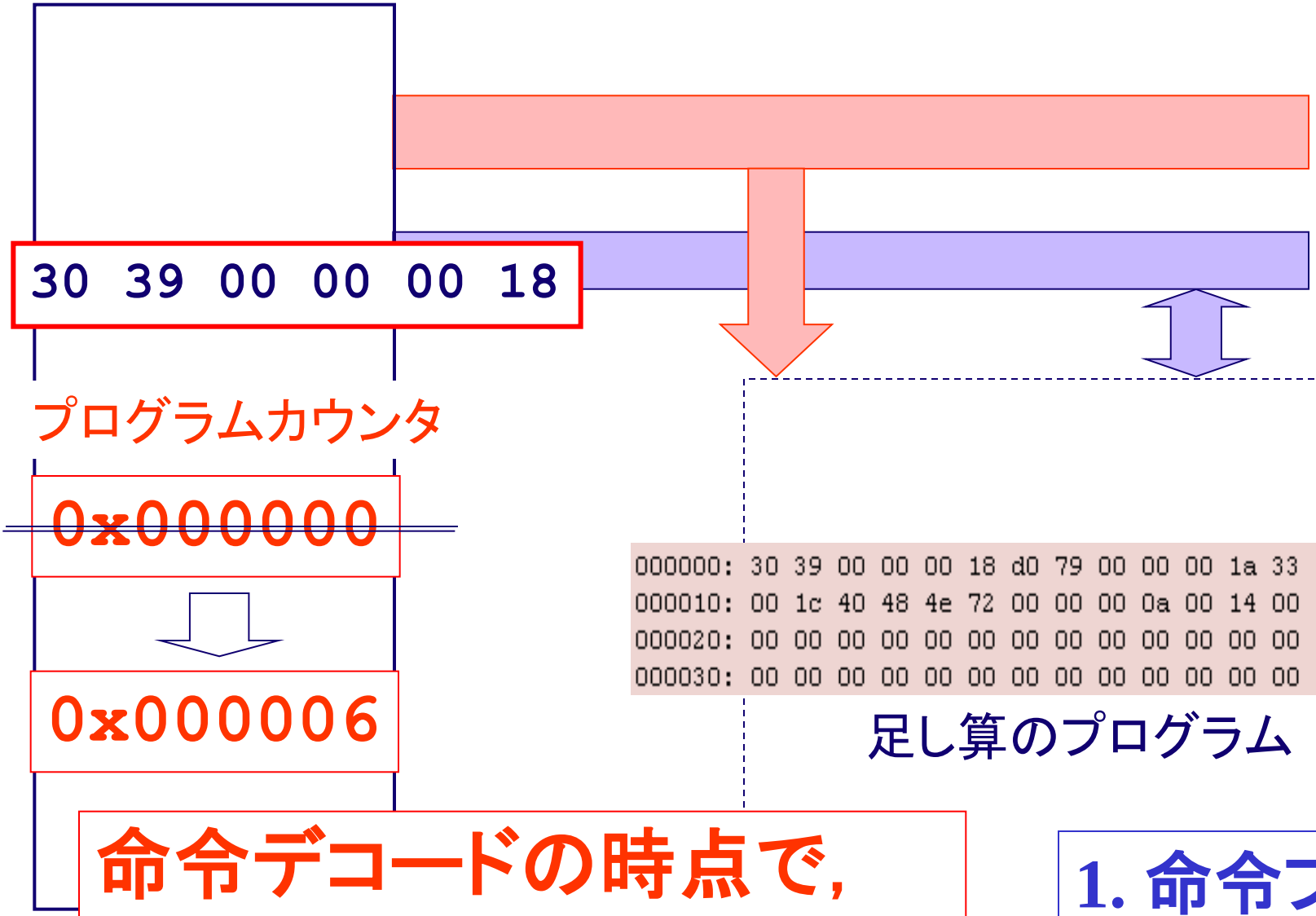
① 0x000000 から命令フェッチ

② 0x000006 から命令フェッチ

③ 0x00000c から命令フェッチ

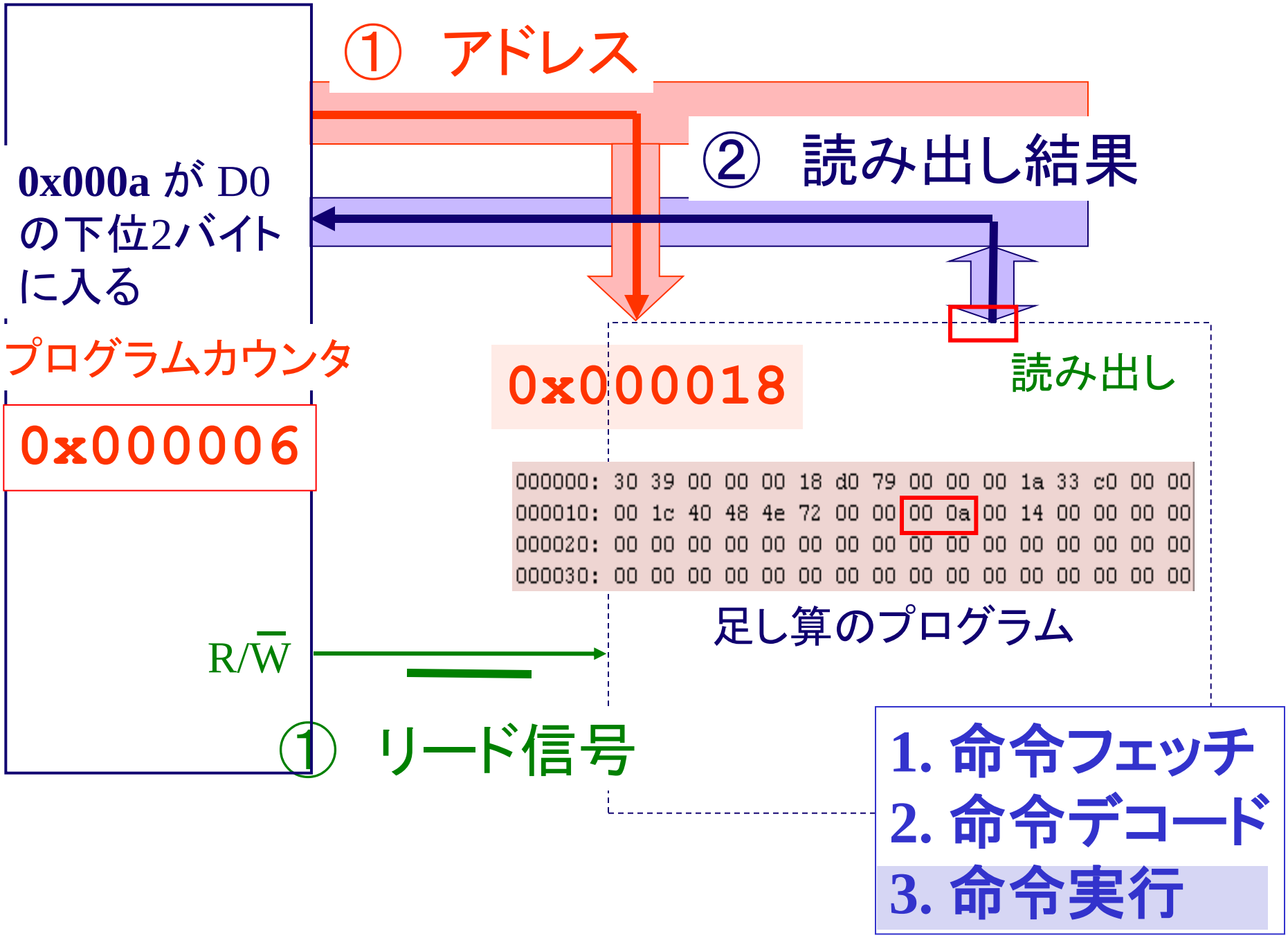
…

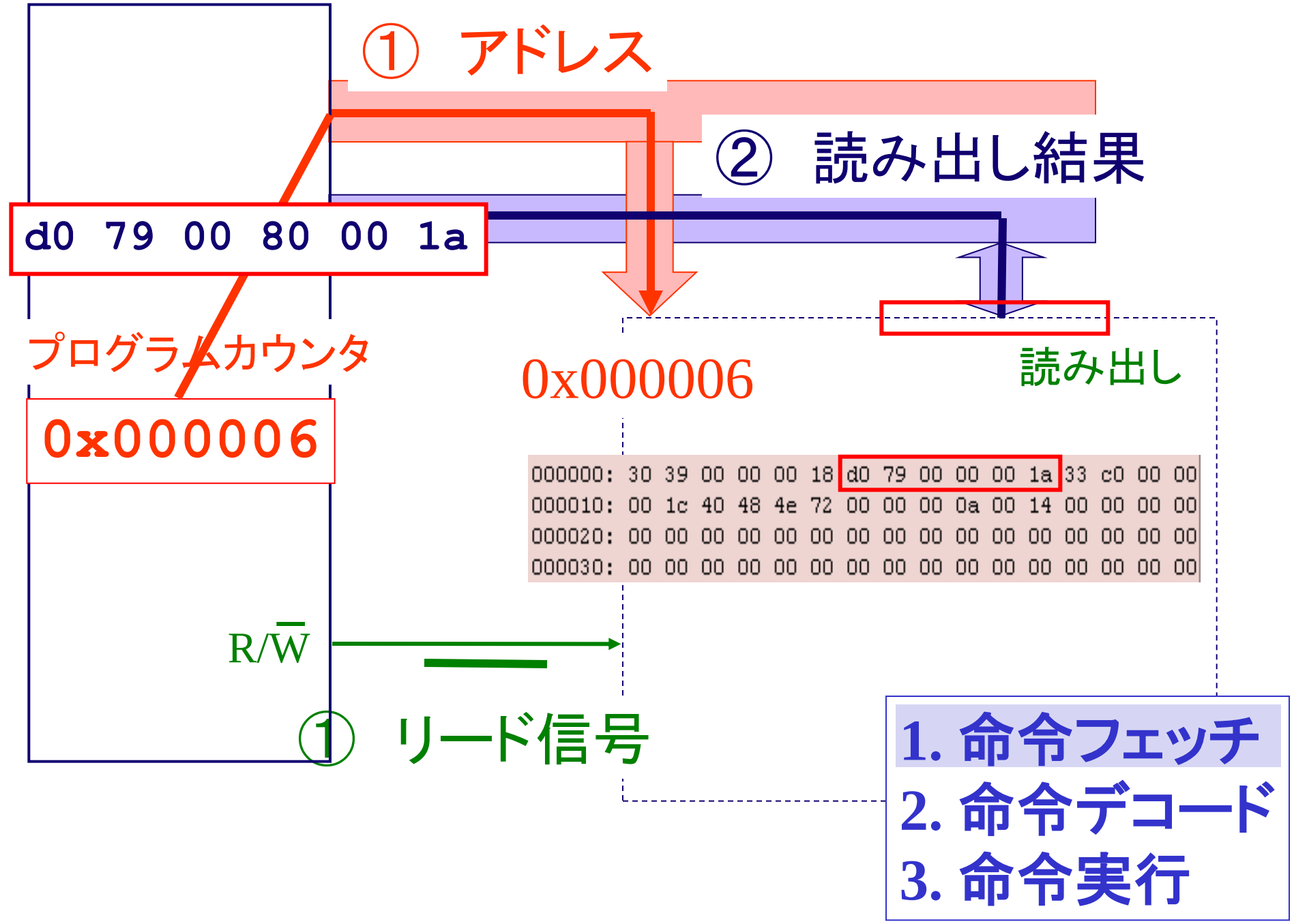

① アドレス
② 読み出し結果
30 39 00 00 00 18
プログラムカウンタ
0x000000
0x000000
読み出し
000000: 30 39 00 00 00 18 d0 79 00 00 00 1a 33 c0 00 00
000010: 00 1c 40 48 4e 72 00 00 00 0a 00 14 00 00 00 00
000020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
足し算のプログラム
R/W
① リード信号
この説明は，プログラムカウンタの値が
前もって，0x000000 にセットされていた場合
1. 命令フェッチ
2. 命令デコード
3. 命令実行

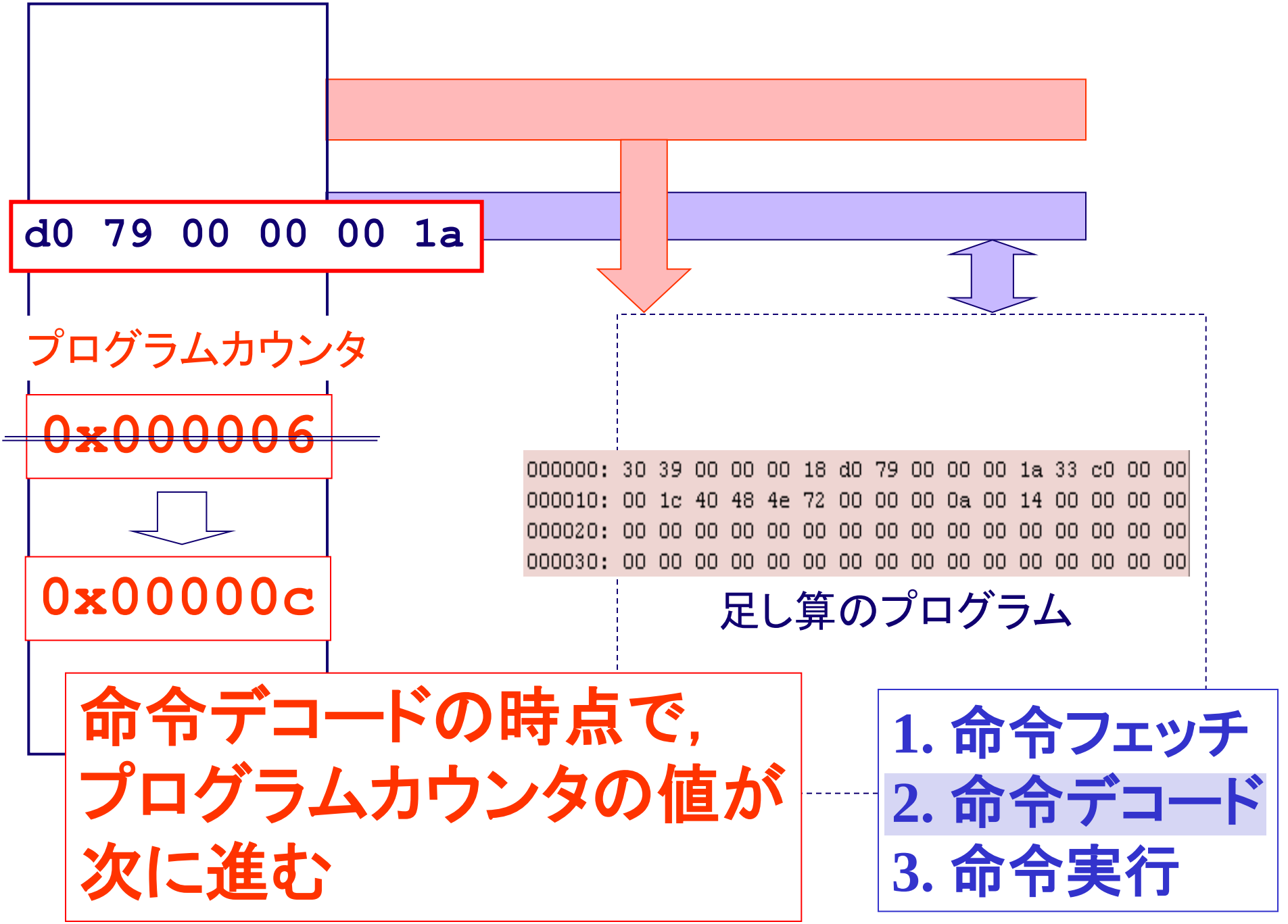


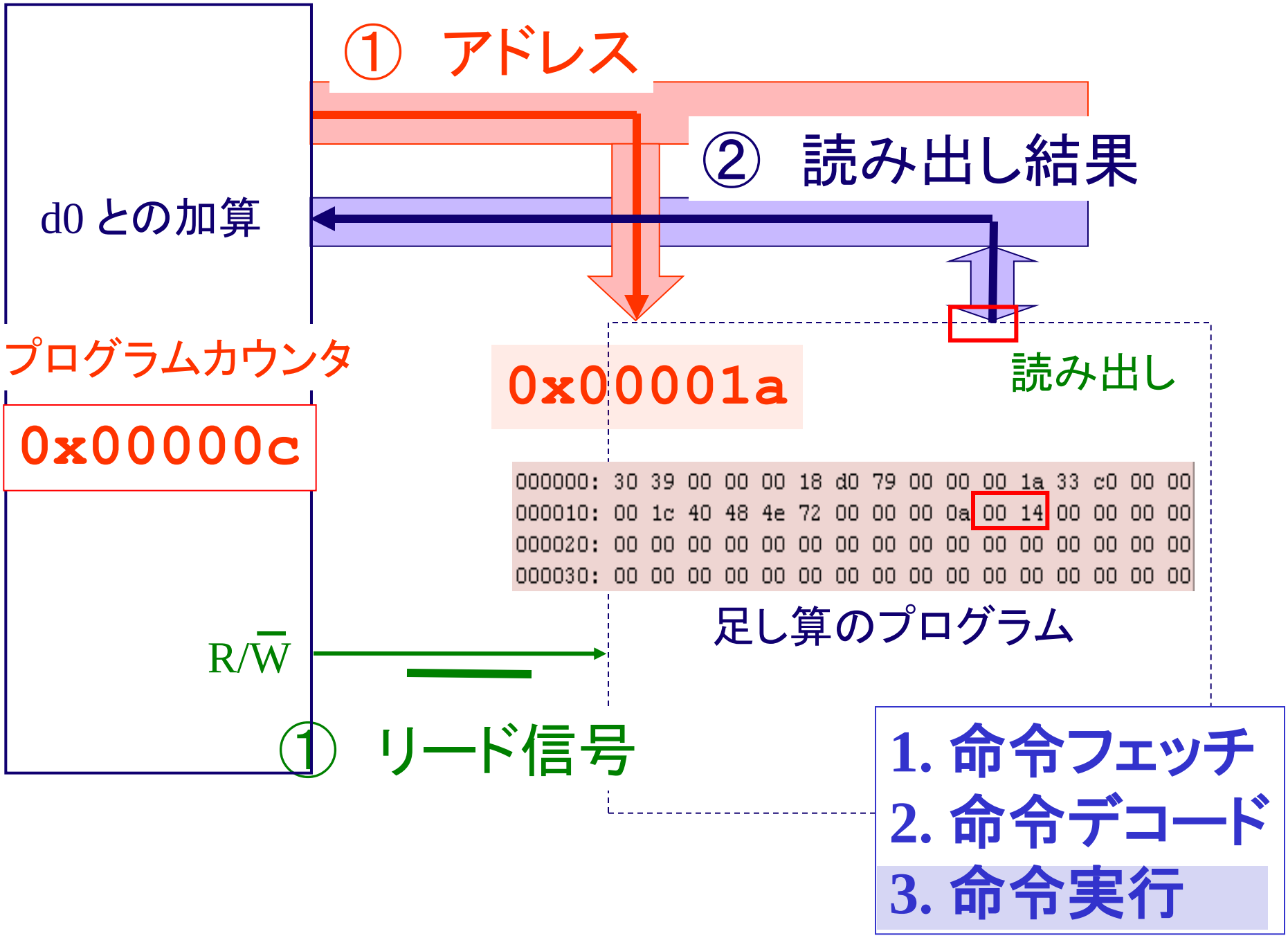
命令デコードの時点で，
プログラムカウンタの値が
次に進む

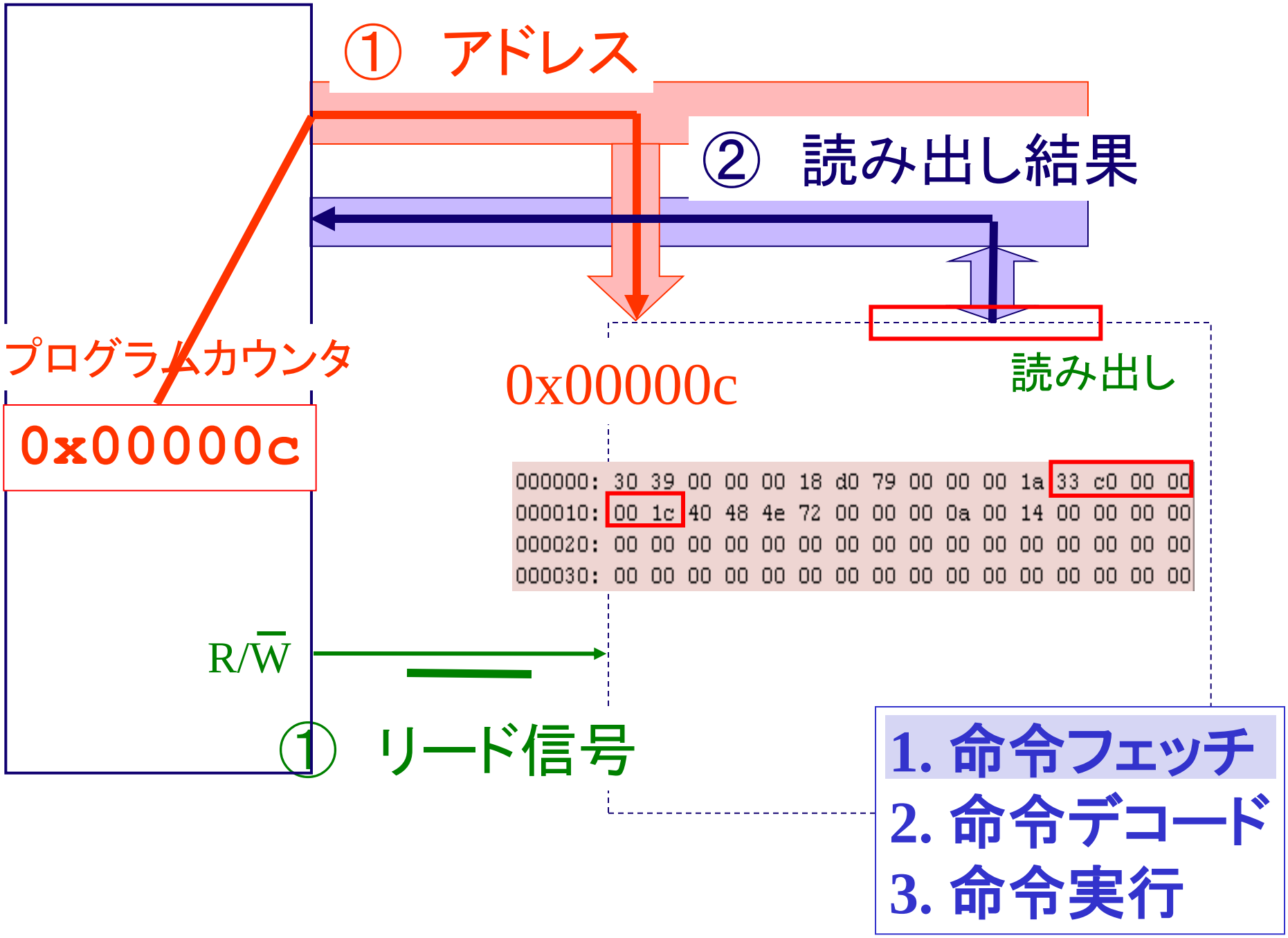
1. 命令フェッチ
2. 命令デコード
3. 命令実行


① アドレス
② 読み出し結果
0x000a が D0 の下位2バイトに入る
プログラムカウンタ
0x000006
0x000018
読み出し
000000: 30 39 00 00 00 18 d0 79 00 00 00 1a 33 c0 00 00
000010: 00 1c 40 48 4e 72 00 00 00 0a 00 14 00 00 00 00
000020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
足し算のプログラム
R/W
① リード信号
1. 命令フェッチ
2. 命令デコード
3. 命令実行


① アドレス
② 読み出し結果
d0 79 00 80 00 1a
プログラムカウンタ
0x000006
0x000006
読み出し
000000: 30 39 00 00 00 18 d0 79 00 00 00 1a 33 c0 00 00
000010: 00 1c 40 48 4e 72 00 00 00 0a 00 14 00 00 00 00
000020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
R/W
① リード信号
1. 命令フェッチ
2. 命令デコード
3. 命令実行

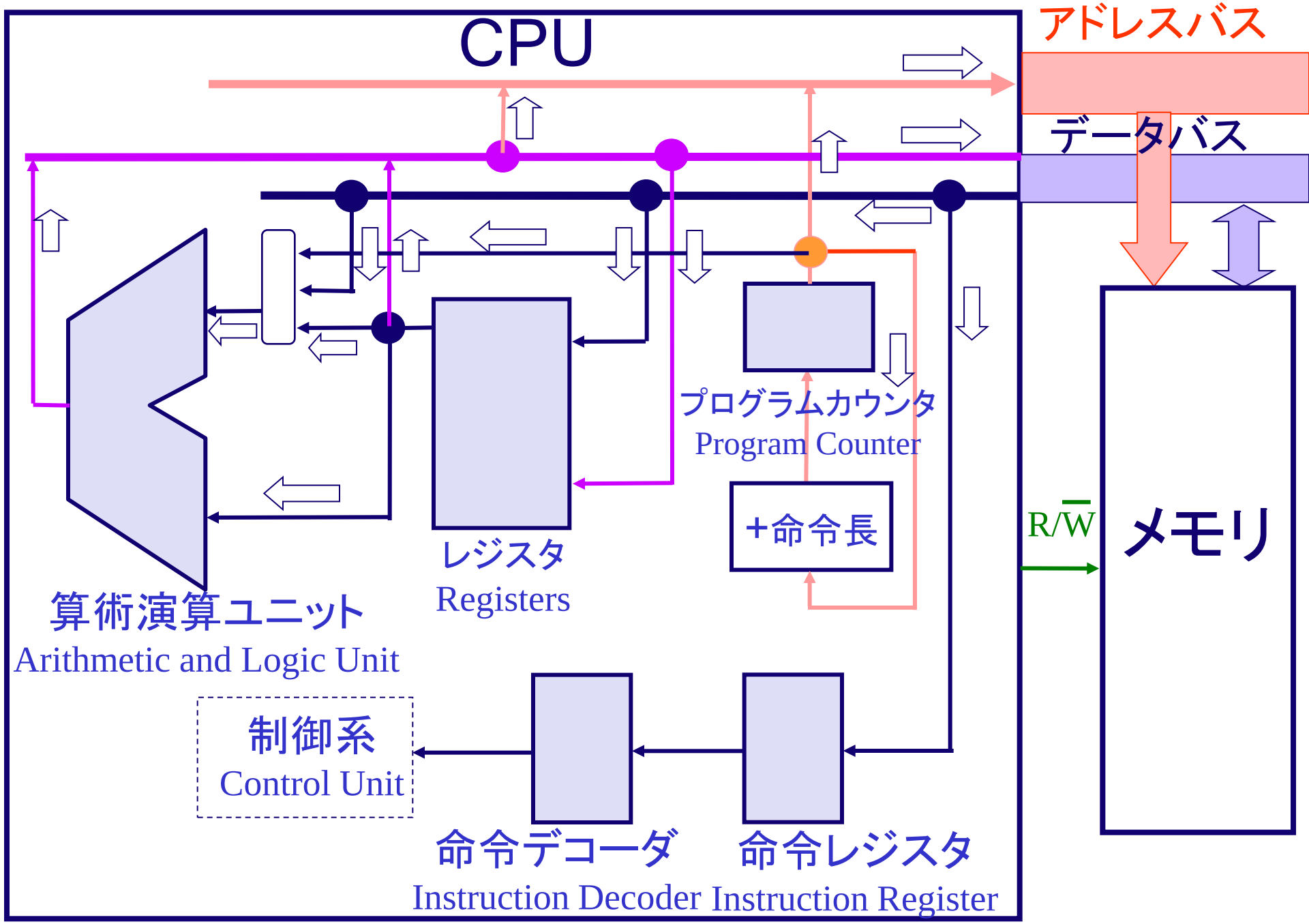

d0 79 00 00 00 1a
プログラムカウンタ
0x000006
0x00000c
000000: 30 39 00 00 00 18 d0 79 00 00 00 1a 33 c0 00 00
000010: 00 1c 40 48 4e 72 00 00 00 0a 00 14 00 00 00 00
000020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
足し算のプログラム
命令デコードの時点で，
プログラムカウンタの値が
次に進む
1. 命令フェッチ
2. 命令デコード
3. 命令実行

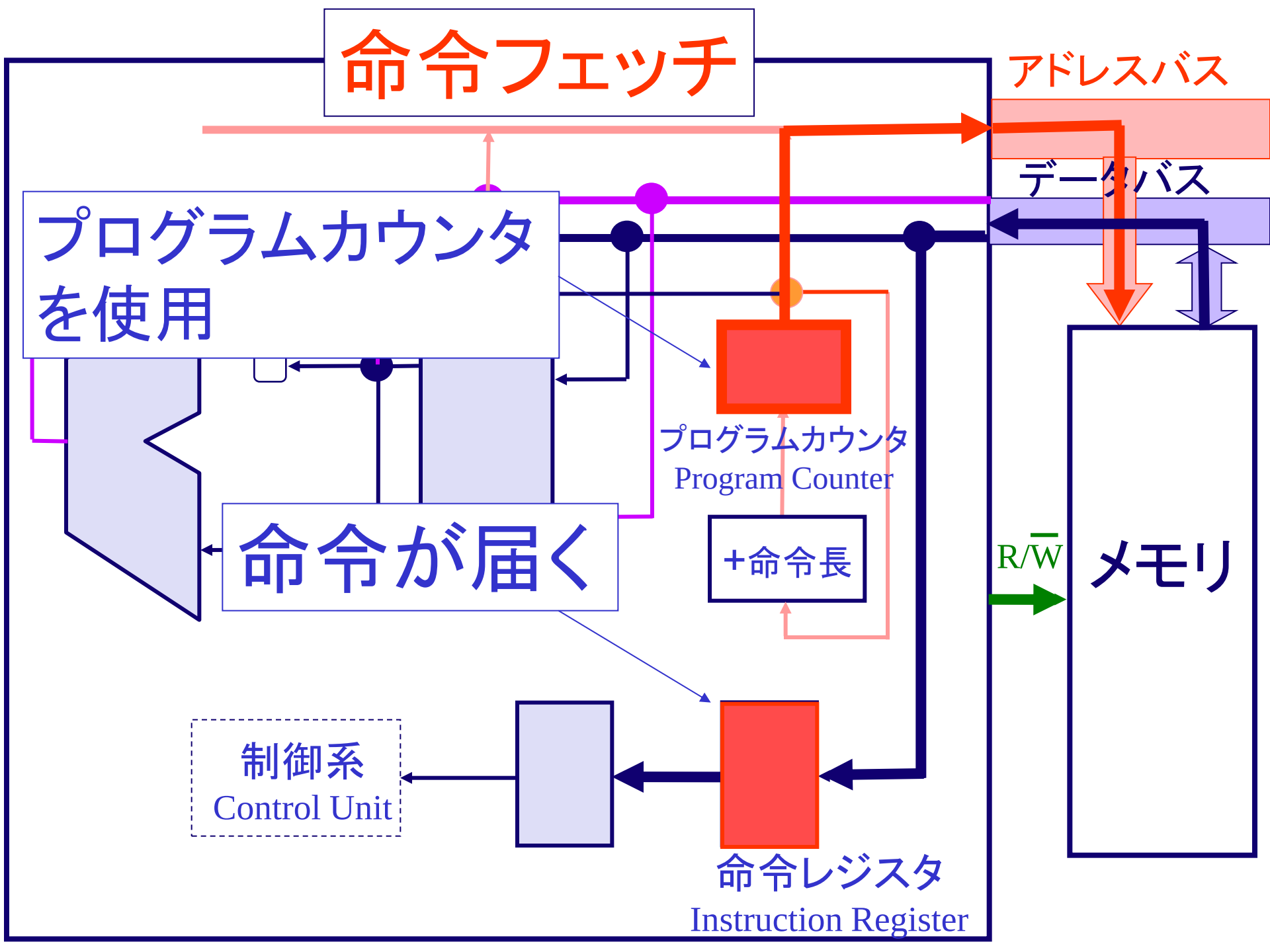

① アドレス
② 読み出し結果
d0 との加算
プログラムカウンタ
0x00000c
0x00001a
読み出し
000000: 30 39 00 00 00 18 d0 79 00 00 00 1a 33 c0 00 00
000010: 00 1c 40 48 4e 72 00 00 00 0a 00 14 00 00 00 00
000020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
足し算のプログラム
R/W
① リード信号
1. 命令フェッチ
2. 命令デコード
3. 命令実行

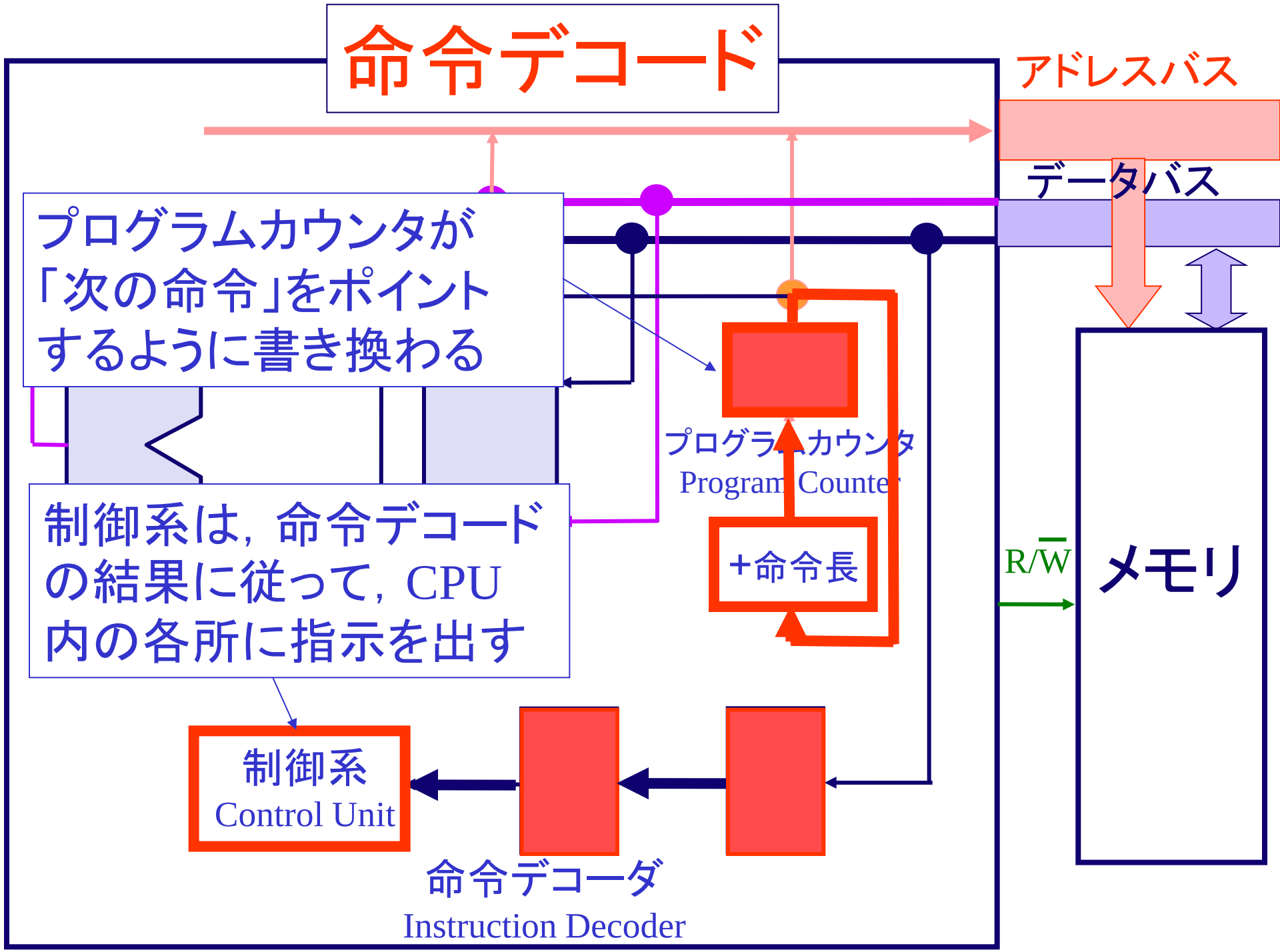

① アドレス
② 読み出し結果
プログラムカウンタ
0x00000c
0x00000c
読み出し
000000: 30 39 00 00 00 18 d0 79 00 00 00 1a 33 c0 00 00
000010: 00 1c 40 48 4e 72 00 00 00 0a 00 14 00 00 00 00
000020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
R/W
① リード信号
1. 命令フェッチ
2. 命令デコード
3. 命令実行

命令実行サイクル

1. 命令フェッチ
   - 次に実行すべきプログラム命令を，プログラムカウンタを使って，メモリから読み出す
2. 命令デコード
   - 読み出した命令を解読し，命令の種類，オペランドの種類，演算結果の格納場所の情報を得る
   - 命令デコードの時点で，プログラムカウンタの値が次に進む
3. 命令実行
   - 命令デコードの結果に従って，必要となるデータにアクセス
   - 演算装置に供給し，結果を適当な場所に格納する


CPU
アドレスバス
データバス
プログラムカウンタ
Program Counter
+命令長
R/W
メモリ
レジスタ
Registers
算術演算ユニット
Arithmetic and Logic Unit
制御系
Control Unit
命令デコーダ
Instruction Decoder
命令レジスタ
Instruction Register


命令フェッチ
アドレスバス
データバス
プログラムカウンタ
を使用
プログラムカウンタ
Program Counter
命令が届く
+命令長
R/W
メモリ
制御系
Control Unit
命令レジスタ
Instruction Register


命令デコード
アドレスバス
データバス
プログラムカウンタが
「次の命令」をポイント
するように書き換わる
プログラムカウンタ
Program Counter
制御系は，命令デコード
の結果に従って，CPU
内の各所に指示を出す
+命令長
R/W
メモリ
制御系
Control Unit
命令デコーダ
Instruction Decoder

例題. 繰り返し

- 次の例を使って，今までの内容を説明する

$$s = \sum_{\mathrm{i}=1}^{3} i$$

繰り返し

```
.data
s:
        .dc.l 0
.text
        /* %d0 = 1, 2, 3 */
        moveq.l #1,%d0
start1:
        cmp.l #3,%d0
        bhi break1
        add.l %d0,s
        addq.l #1,%d0
        bra start1
break1:

        .dc.w 0x4848
        stop #0
```

繰り返し

```
.data
s:
        .dc.l 0
.text
        /* %d0 = 1, 2, 3 */
        moveq.l #1,%d0
start1:
        cmp.l #3,%d0
        bhi break1
        add.l %d0,s
        addq.l #1,%d0
        bra start1
break1:

        .dc.w 0x4848
        stop #0
```

データレジスタ D0
と「3」との比較

ジャンプ

「3以下」のとき
のみ実行される
部分

繰り返し

```
.data
s:
        .dc.l 0
.text
        /* %d0 = 1, 2, 3 */
        moveq.l #1,%d0
start1:
        cmp.l #3,%d0
        bhi break1
        add.l %d0,s
        addq.l #1,%d0
        bra start1
break1:

        .dc.w 0x4848
        stop #0
```

データレジスタ D0
と「3」との比較

3を超えたときは

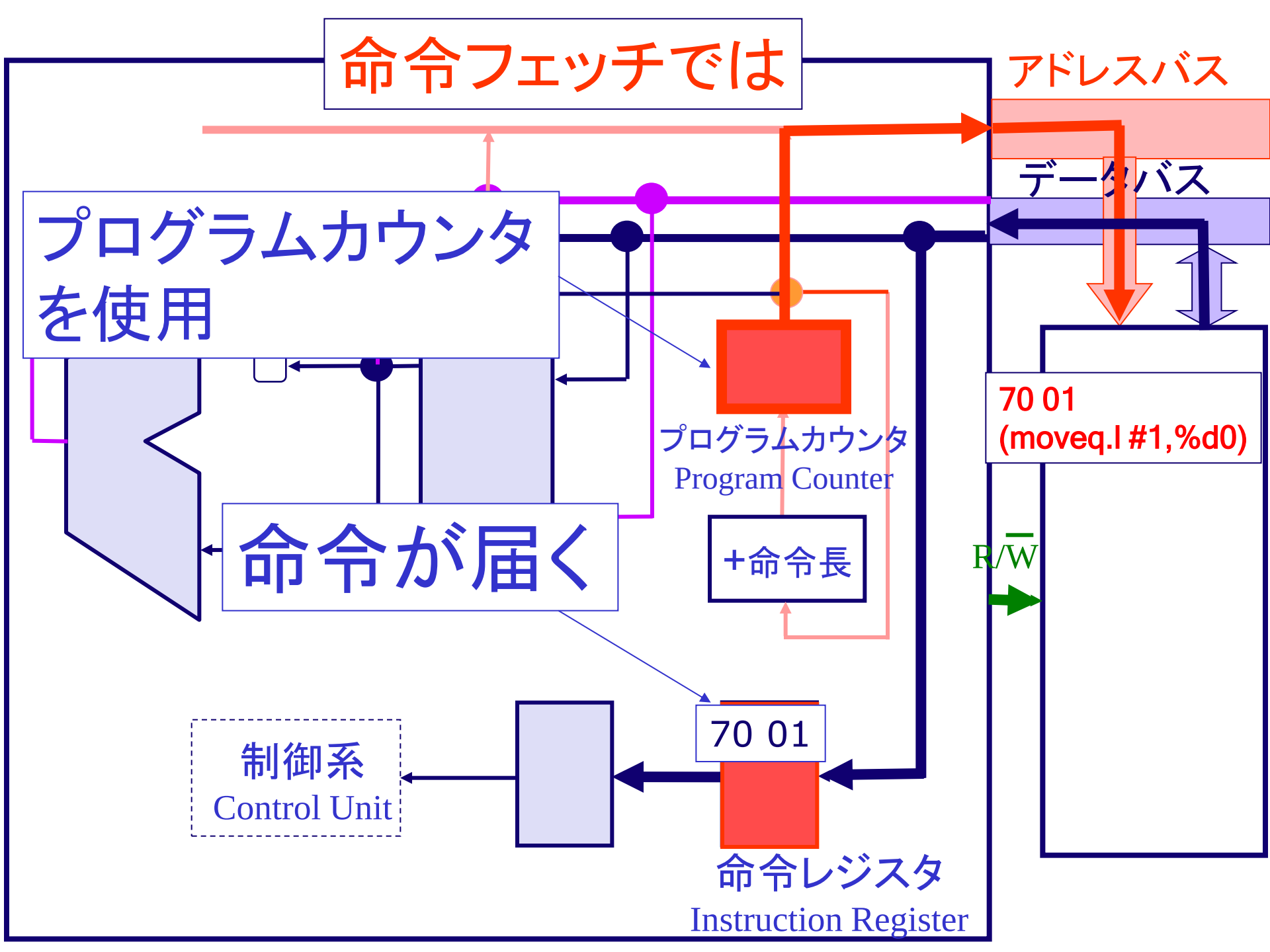
「3以下」のとき
のみ実行される
部分

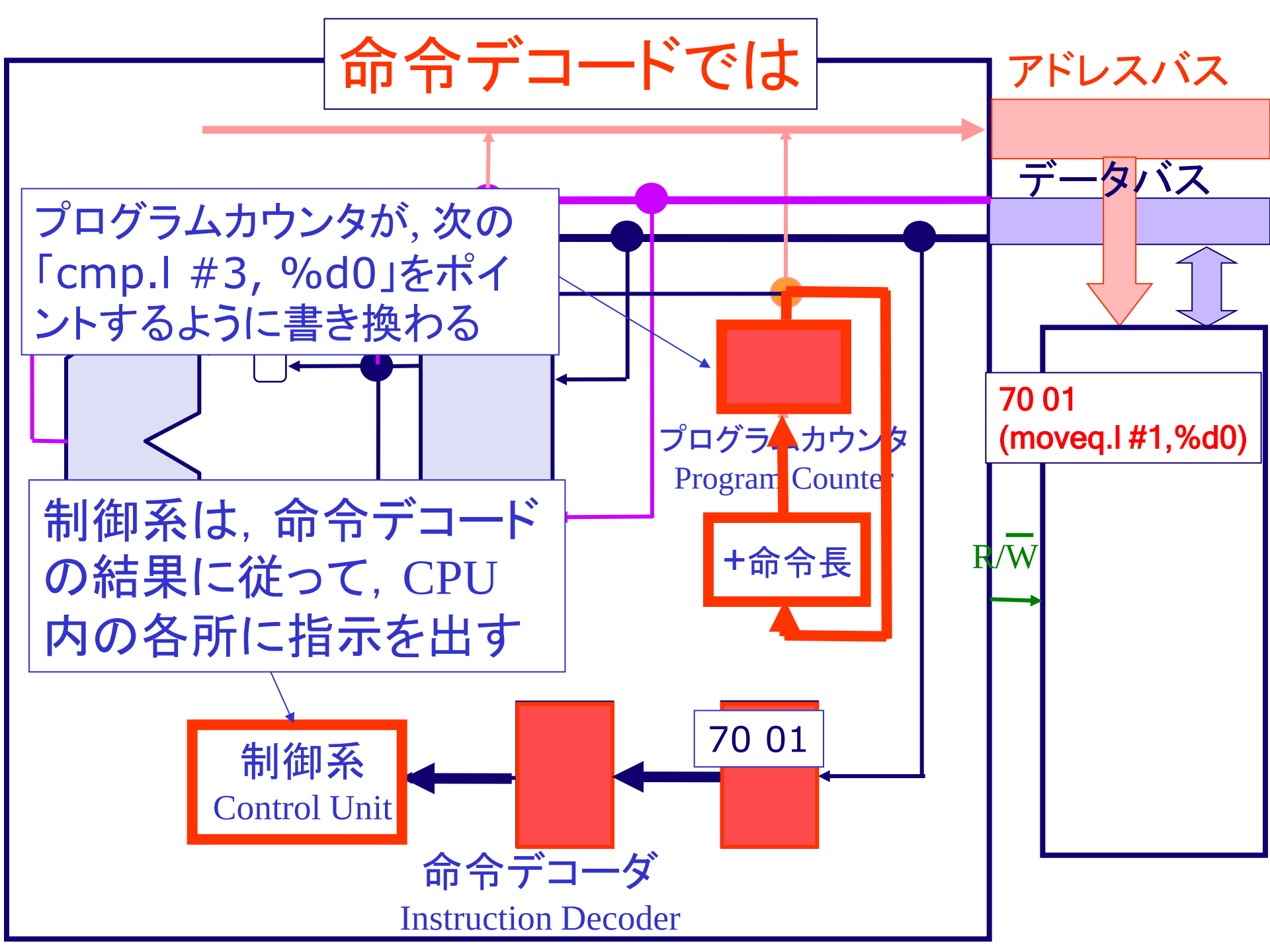
ジャンプ

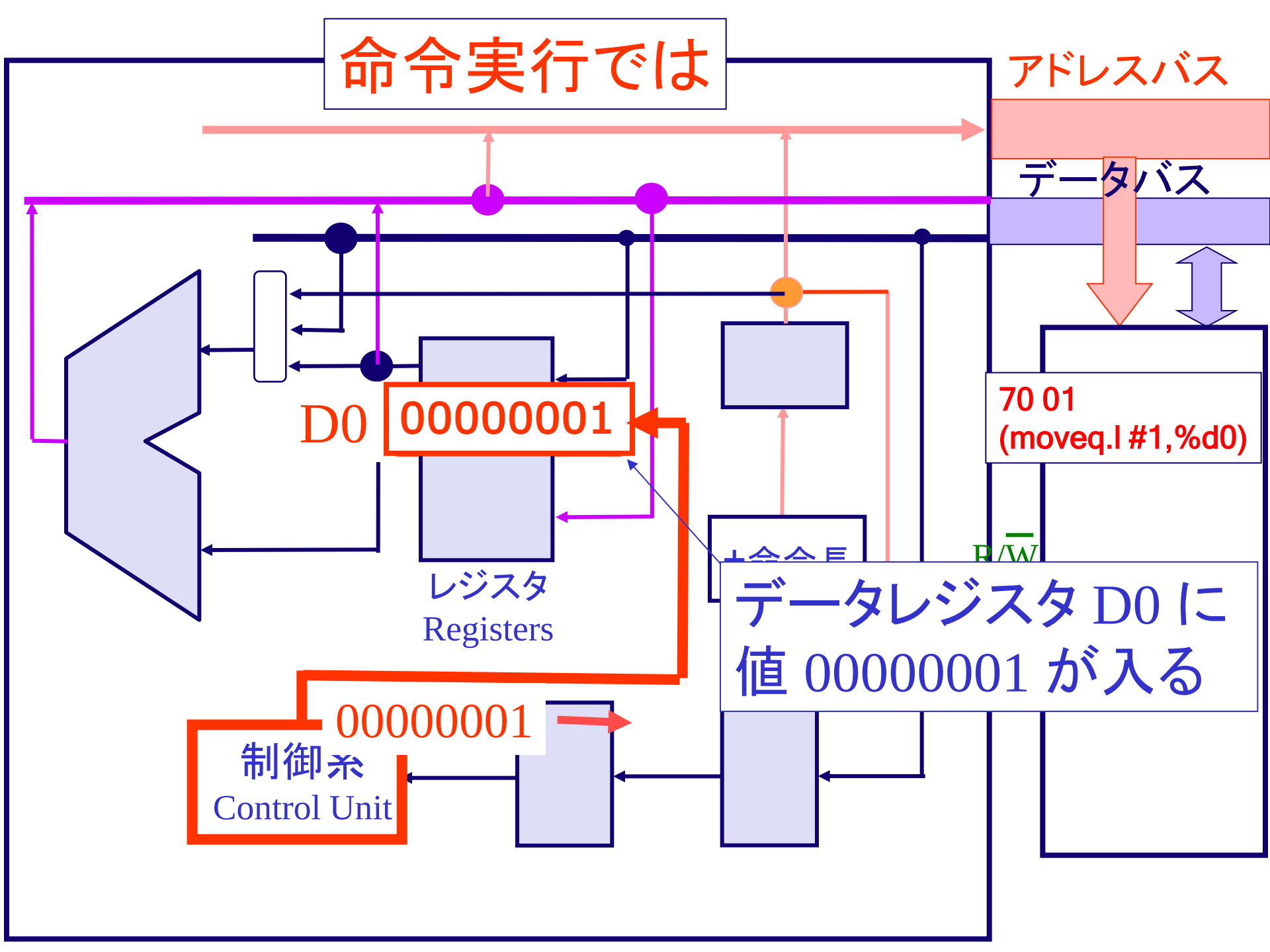
繰り返し

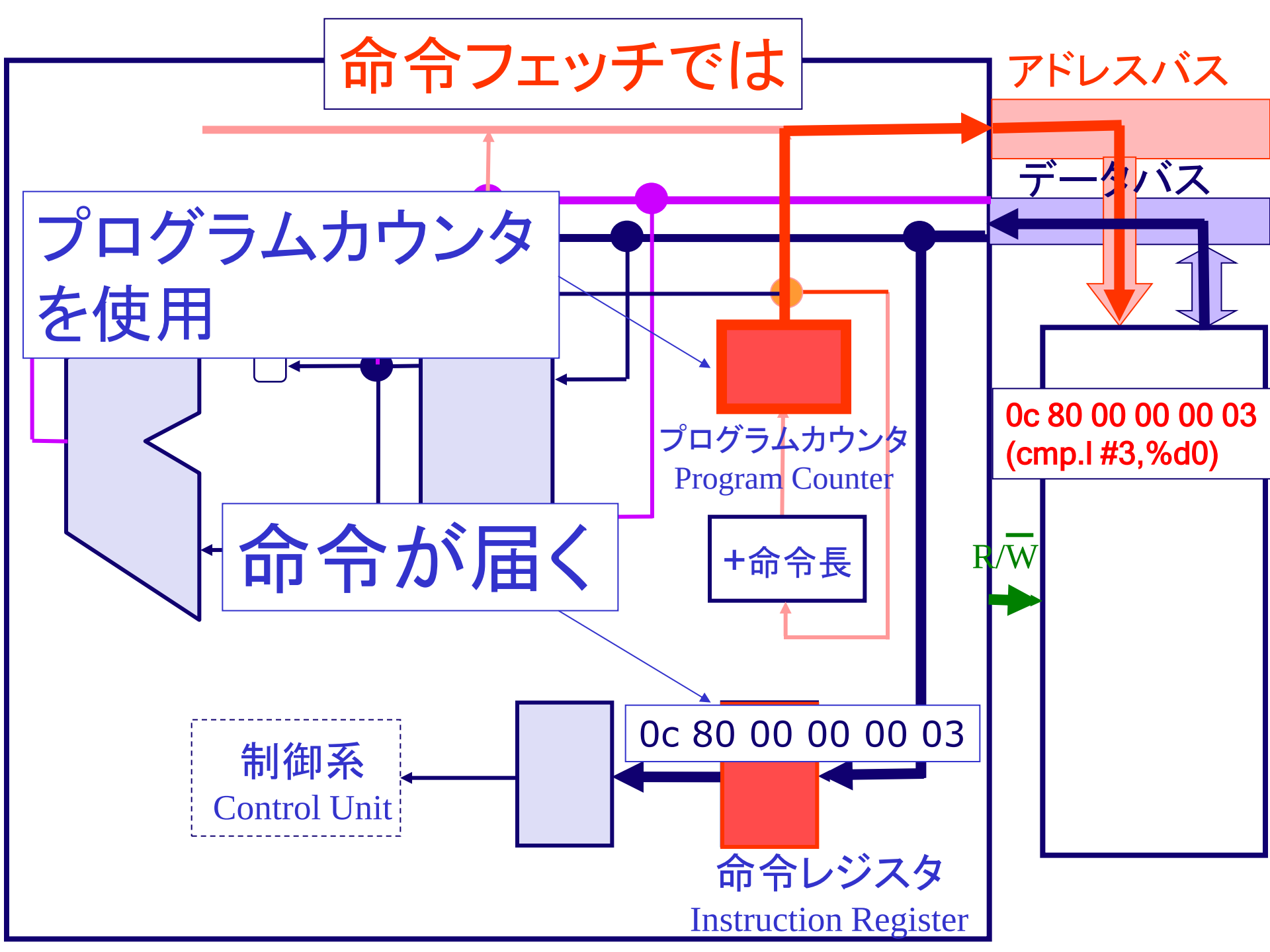
```
.data
s:
        .dc.l 0
.text
        /* %d0 = 1, 2, 3 */
    ①   moveq.l #1,%d0
start1:
②⑦⑫⑰   cmp.l #3,%d0
③⑧⑬⑱   bhi break1
④⑨⑭     add.l %d0,s
⑤⑩⑮     addq.l #1,%d0
⑥⑪⑯     bra start1
break1:

    ⑲   .dc.w 0x4848
        stop #0
```

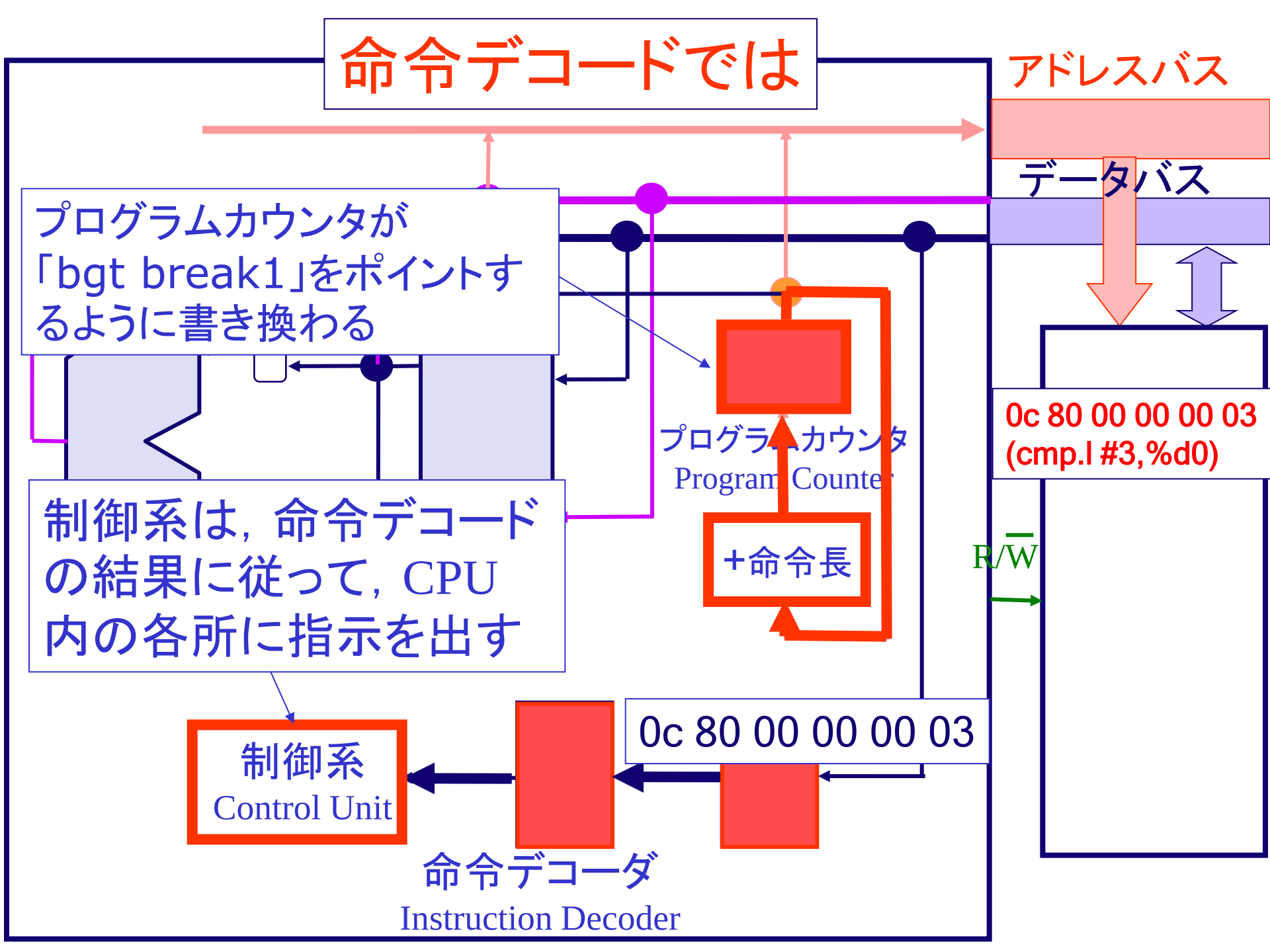
最初はここ

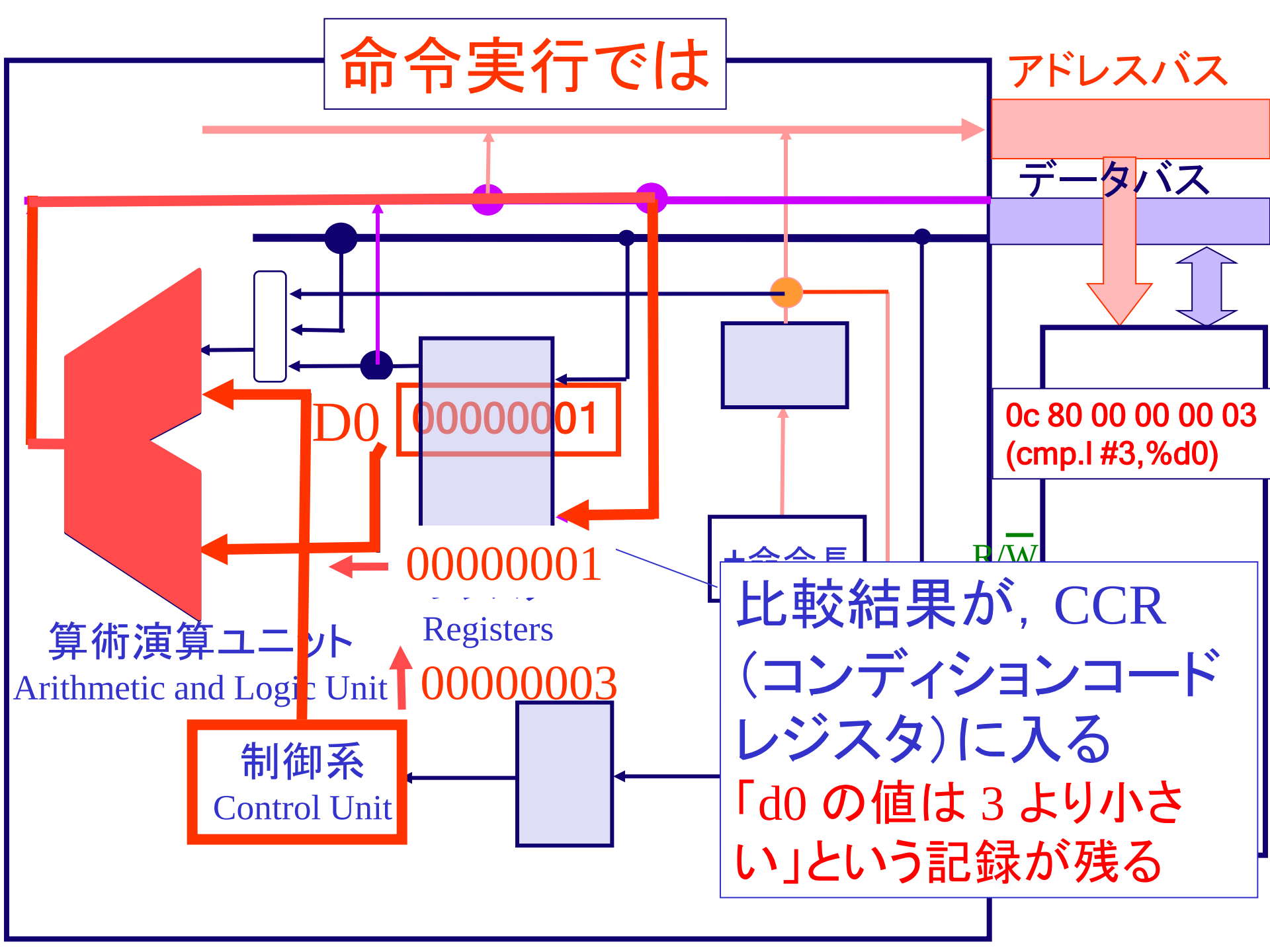

命令フェッチでは
アドレスバス
データバス
プログラムカウンタを使用
70 01
(moveq.l #1,%d0)
プログラムカウンタ
Program Counter
+命令長
R/W
命令が届く
70 01
制御系
Control Unit
命令レジスタ
Instruction Register

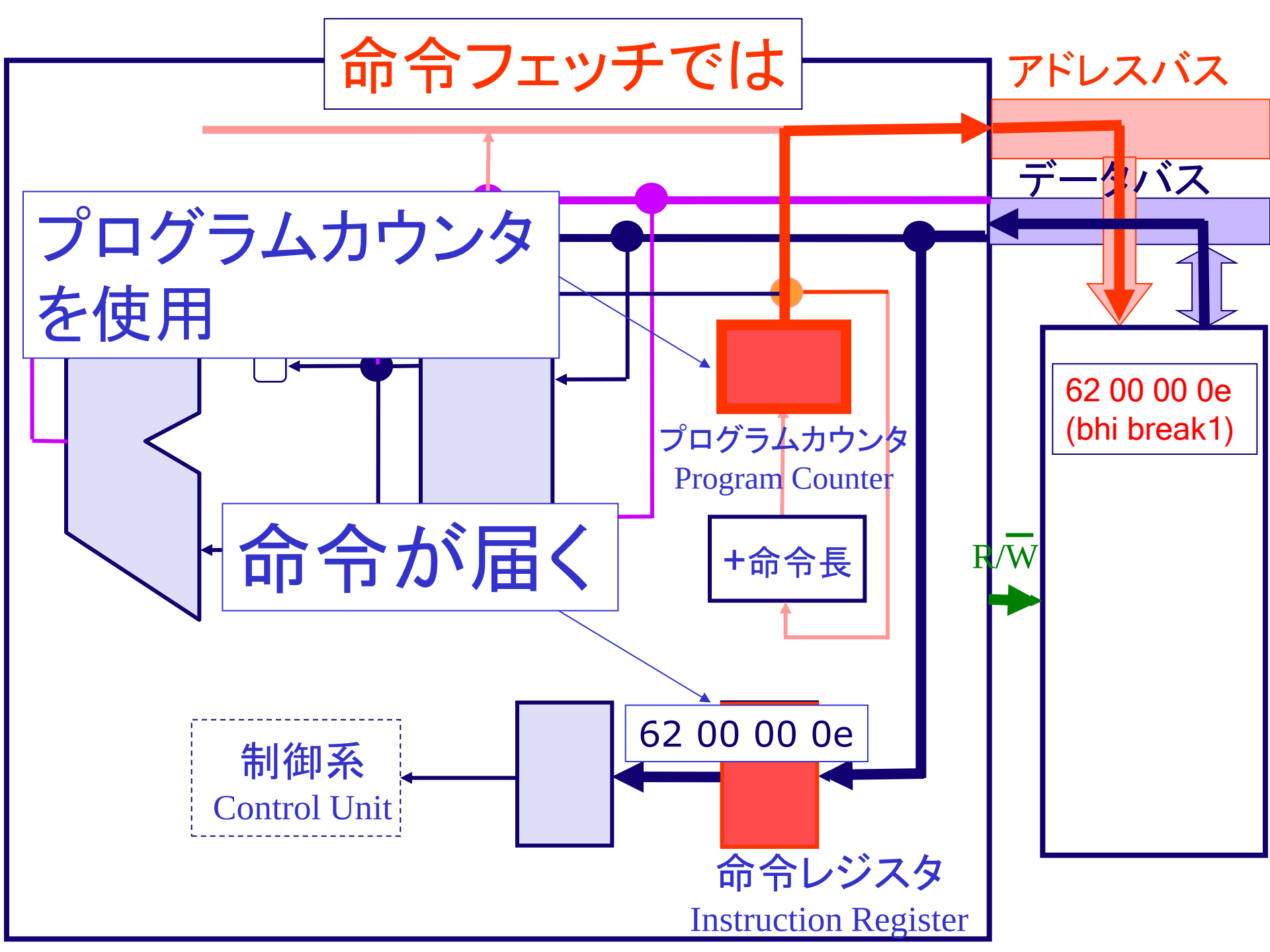

命令デコードでは
アドレスバス
データバス
プログラムカウンタが，次の「cmp.l #3, %d0」をポイントするように書き換わる
70 01
(moveq.l #1,%d0)
プログラムカウンタ
Program Counter
制御系は，命令デコードの結果に従って，CPU内の各所に指示を出す
+命令長
R/W
70 01
制御系
Control Unit
命令デコーダ
Instruction Decoder

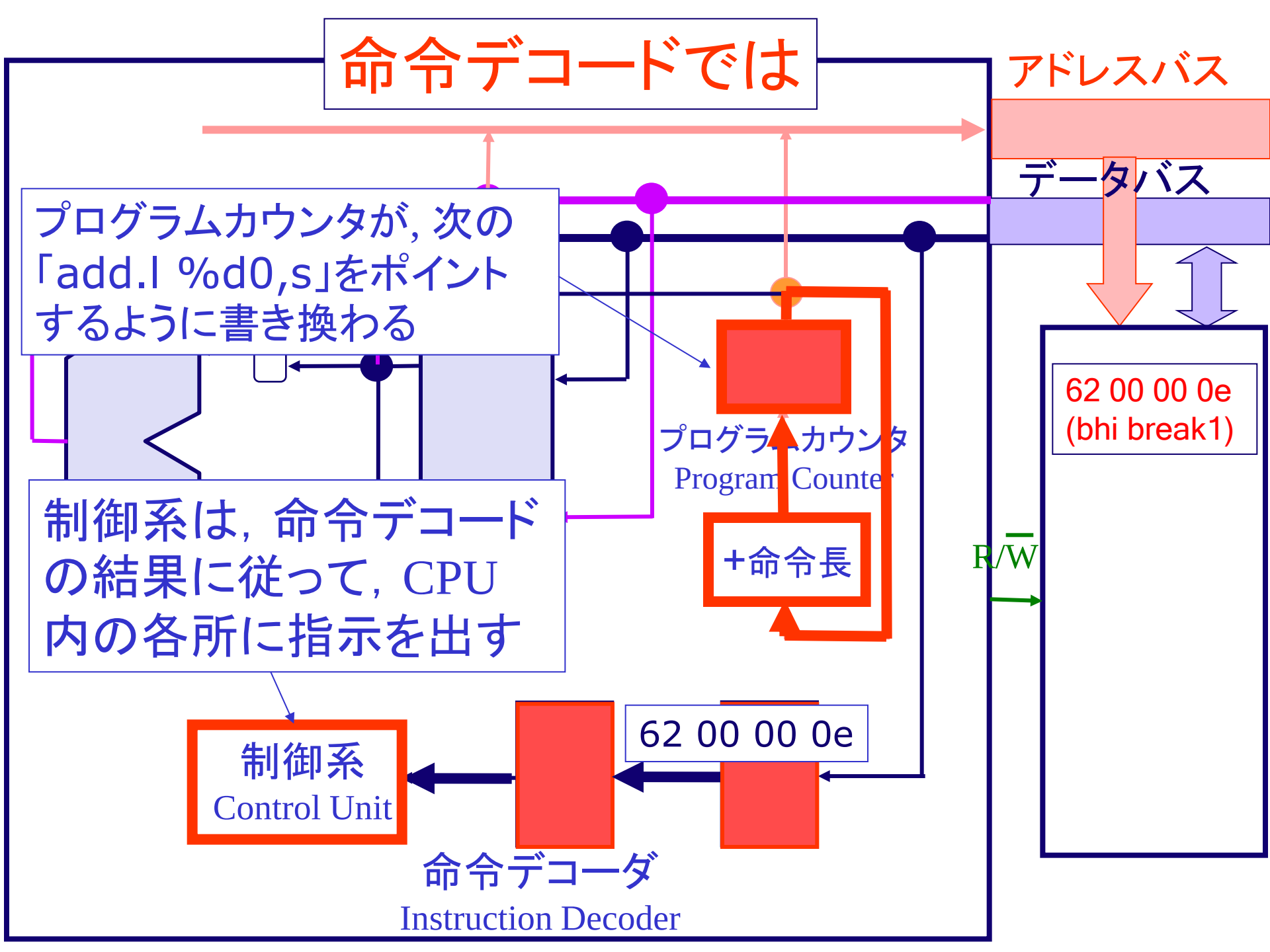

命令実行では
アドレスバス
データバス
D0
00000001
70 01
(moveq.l #1,%d0)
レジスタ
Registers
データレジスタ D0 に
値 00000001 が入る
00000001
制御ﾈ
Control Unit

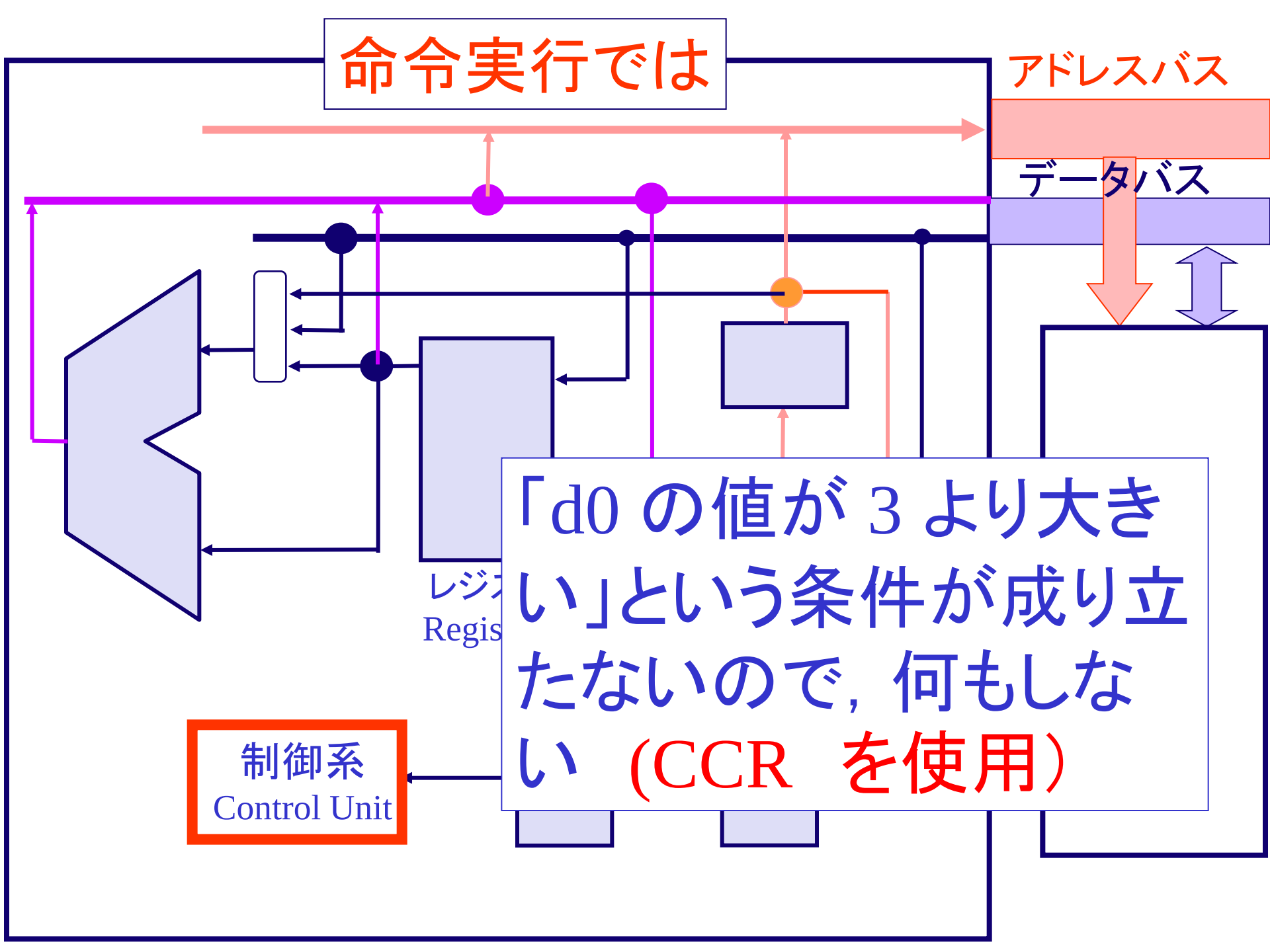

命令フェッチでは
アドレスバス
データバス
プログラムカウンタを使用
0c 80 00 00 00 03
(cmp.l #3,%d0)
プログラムカウンタ
Program Counter
命令が届く
+命令長
R/W
0c 80 00 00 00 03
制御系
Control Unit
命令レジスタ
Instruction Register


命令デコードでは
アドレスバス
データバス
プログラムカウンタが「bgt break1」をポイントするように書き換わる
0c 80 00 00 00 03
(cmp.l #3,%d0)
プログラムカウンタ
Program Counter
制御系は，命令デコードの結果に従って，CPU内の各所に指示を出す
+命令長
R/W
制御系
Control Unit
0c 80 00 00 00 03
命令デコーダ
Instruction Decoder

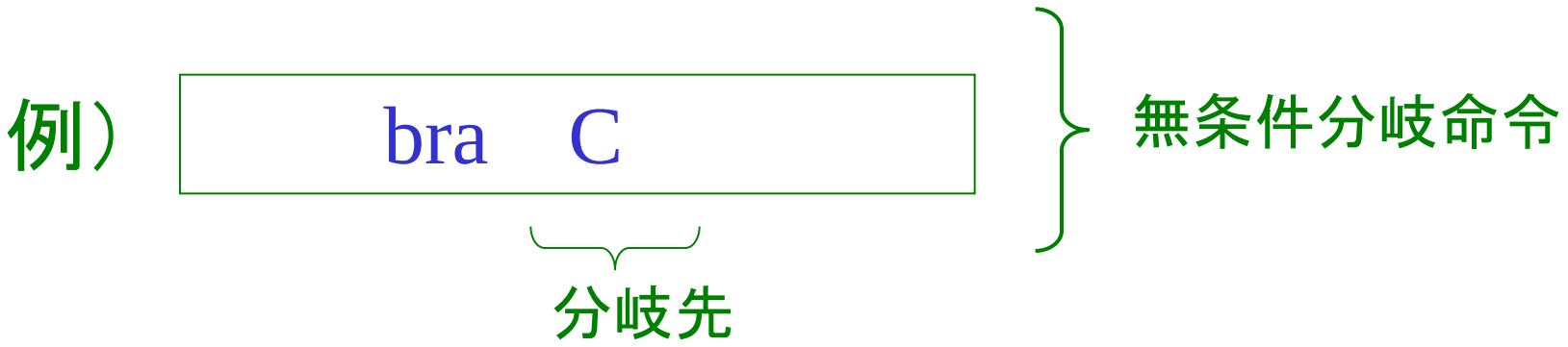

命令実行では
アドレスバス
データバス
D0
00000001
0c 80 00 00 00 03
(cmp.l #3,%d0)
00000001
算術演算ユニット
Arithmetic and Logic Unit
Registers
00000003
制御系
Control Unit
比較結果が，CCR（コンディションコードレジスタ）に入る
「d0 の値は 3 より小さい」という記録が残る


命令フェッチでは
アドレスバス
データバス
プログラムカウンタを使用
62 00 00 0e
(bhi break1)
プログラムカウンタ
Program Counter
+命令長
R/W
命令が届く
62 00 00 0e
制御系
Control Unit
命令レジスタ
Instruction Register

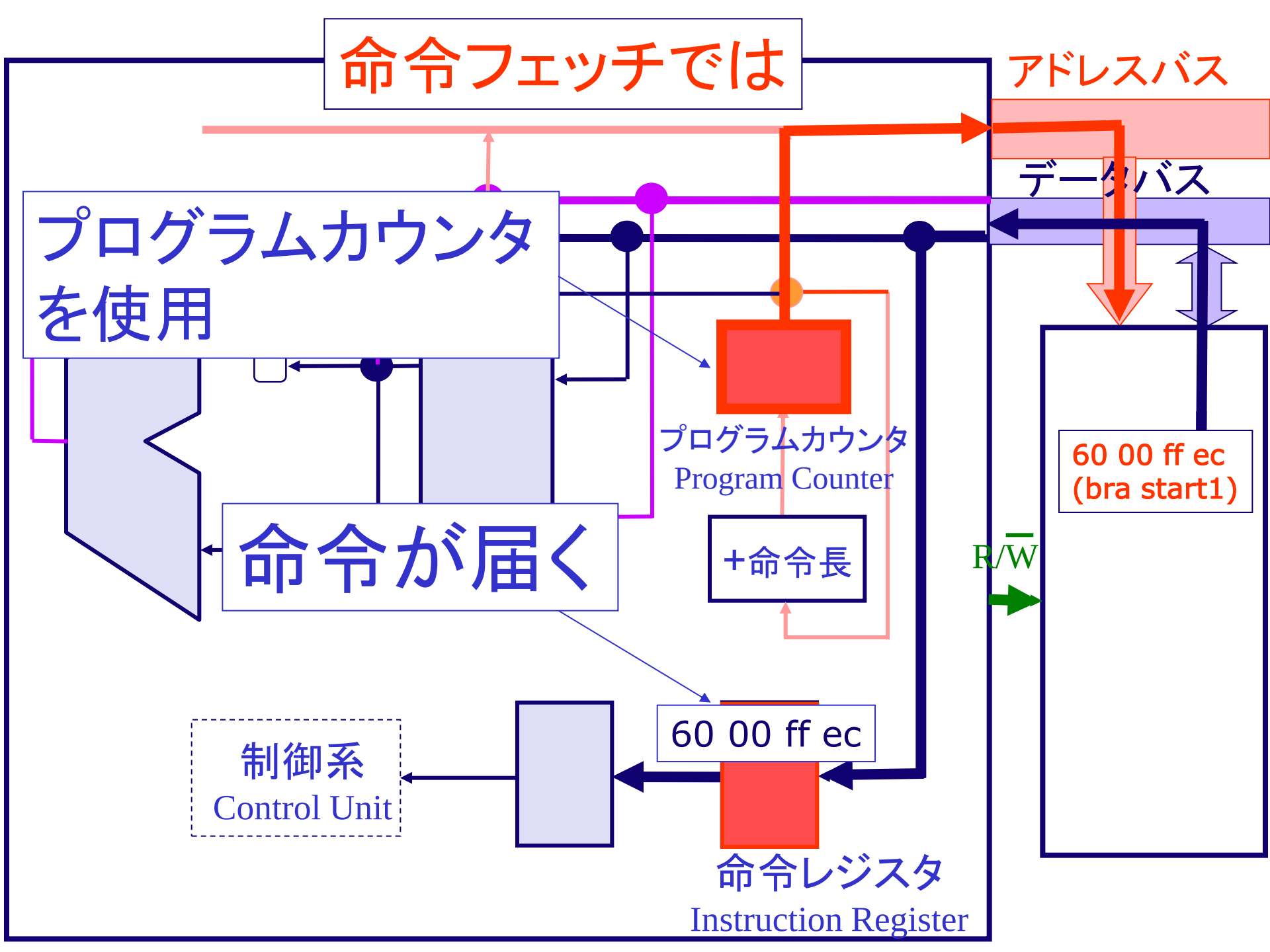

命令デコードでは
アドレスバス
データバス
プログラムカウンタが，次の「add.l %d0,s」をポイントするように書き換わる
62 00 00 0e
(bhi break1)
プログラムカウンタ
Program Counter
制御系は，命令デコードの結果に従って，CPU内の各所に指示を出す
+命令長
R/W
62 00 00 0e
制御系
Control Unit
命令デコーダ
Instruction Decoder

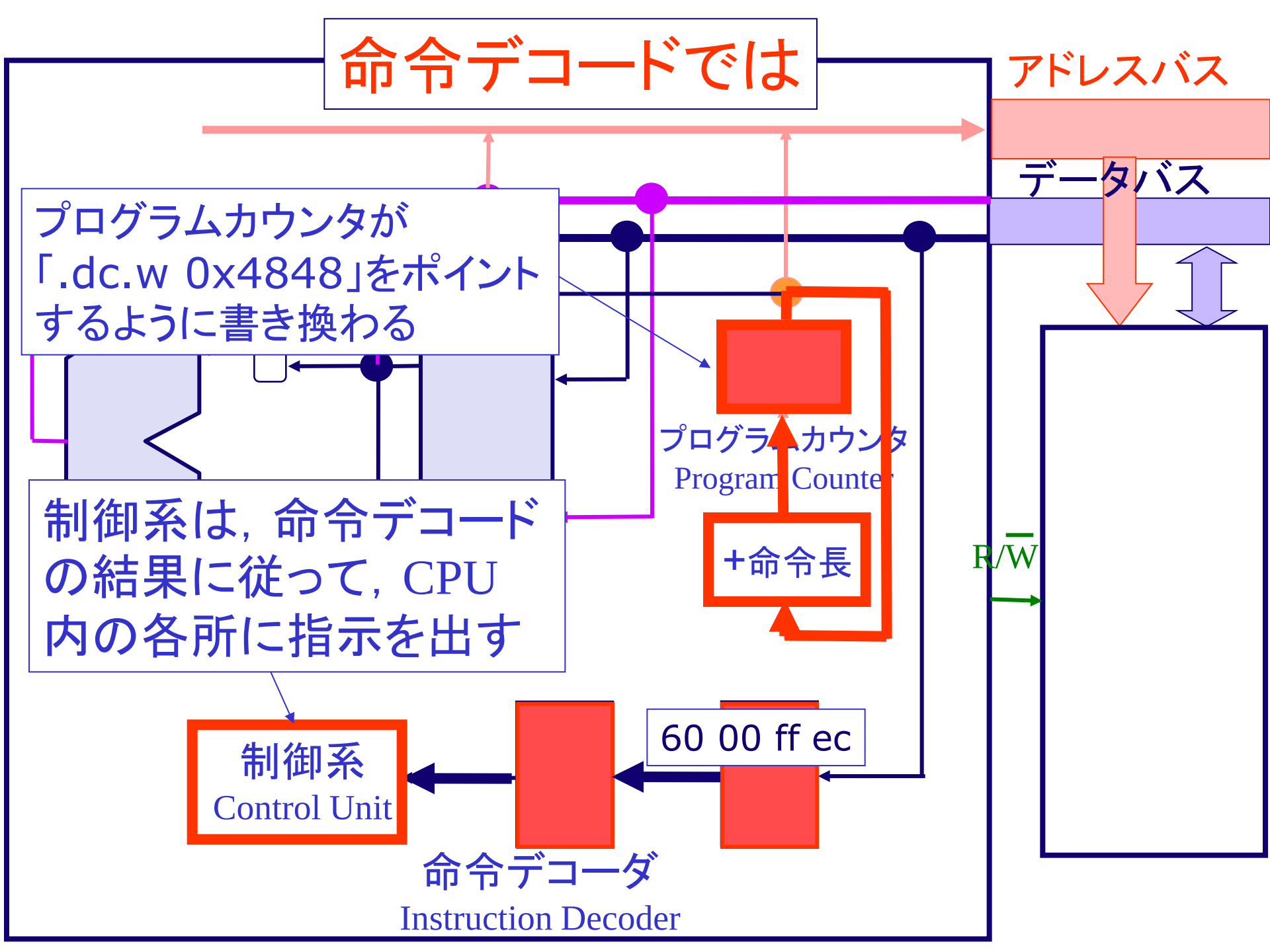

命令実行では
アドレスバス
データバス
「d0 の値が 3 より大きい」という条件が成り立たないので，何もしない（CCR を使用）
制御系
Control Unit

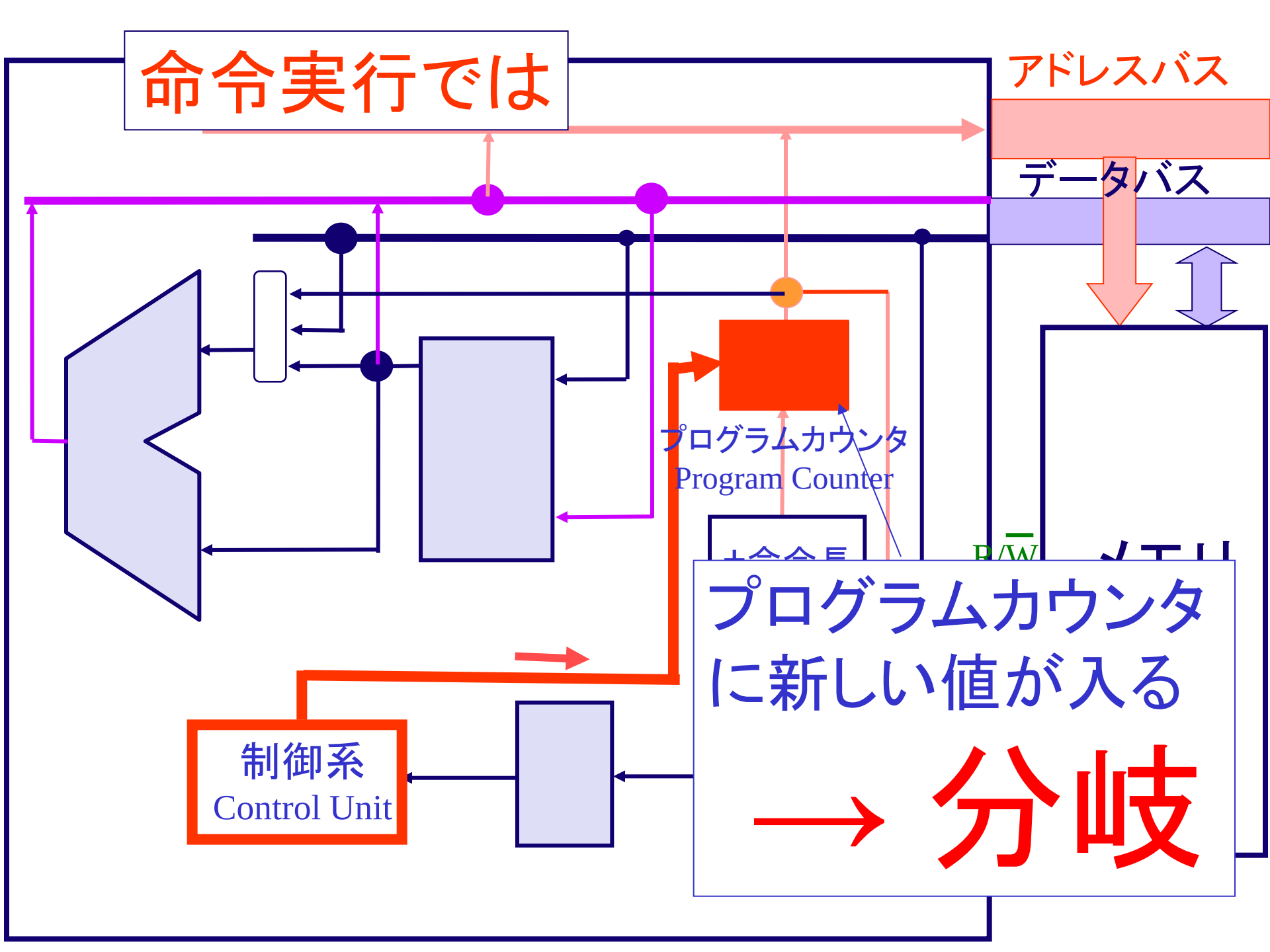
無条件分岐命令 bra



- 分岐先：
  - 分岐先のラベルを書く
  - ラベルが，分岐先のメモリアドレスであると解釈される


命令フェッチでは
アドレスバス
データバス
プログラムカウンタを使用
プログラムカウンタ
Program Counter
60 00 ff ec
(bra start1)
命令が届く
+命令長
R/W
60 00 ff ec
制御系
Control Unit
命令レジスタ
Instruction Register


命令デコードでは
アドレスバス
データバス
プログラムカウンタが
「.dc.w 0x4848」をポイント
するように書き換わる
プログラムカウンタ
Program Counter
+命令長
R/W
制御系は，命令デコード
の結果に従って，CPU
内の各所に指示を出す
制御系
Control Unit
60 00 ff ec
命令デコーダ
Instruction Decoder


命令実行では
アドレスバス
データバス
プログラムカウンタ
Program Counter
制御系
Control Unit
プログラムカウンタ
に新しい値が入る
→ 分岐

まとめ

- 分岐命令では、プログラムカウンタの書き換えが起こる
- 命令は、メモリに格納され、プログラムカウンタを使い読みだされる