

as-2. メモリと CPU

(68000 アセンブラプログラミング)

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

金子邦彦



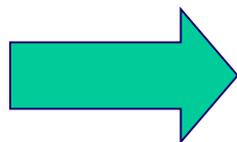
x と y の加算

- 簡単な例として, 次の例を考える

$$z \leftarrow x + y$$

但し, この例題では,
x, y, z はワードサイズ(2バイト)の整数データ

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
```

```
S00600004844521B
S214000000303900000018D07900000001A33C0000014
S20C000010001C40484E7200007F
S20A000018000A00140000BF
S5030003F9
S804000000FB
```

HEX ファイルは、メモリのどこ
に何を置くかを書いたファイル

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

テキストエディタなど
で記述. add.s など

S0	06	0000484452	1B	ステータスレコード(ファイル名など)
S2	14	000000	303900000018D0790000001A33C00000	14
S2	0C	000010	001C40484E720000	7F
S2	0A	000018	000A00140000	BF データレコード
S5	03	0003	F9	データレコード数
S8	04	000000	FB	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

S0	06	0000484452	1B
S2	14	000000	303900000018D0790000001A33C000 ① 0 14
S2	0C	000010	001C40484E7200 ② 0 7F
S2	0A	000018	000A001400 ③ 0 BF
S5	03	0003	F9
S8	04	000000	FB

データの中身

バイト数

チェックサム

メモリアドレス

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

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
    move.w x, %d0
    add.w y, %d0
    move.w %d0, z

    .dc.w 0x4048
    stop #0

.end
```

データ部 (.data) について、
HEX ファイル生成時に行われる
こと

1. メモリエリアの割り当て

x → 0x000018

y → 0x00001a

z → 0x00001c

2. HEX ファイル中に初期値
を入れる

```
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
```

HEX
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
```

プログラム本体 (.text)について、
HEX ファイル生成時に行われる
こと

各命令が数値化されて
HEX ファイルに入る

```
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
```

ここまでのまとめ

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
```

000000:	30 39 00 00 00 18	d0 79 00 00 00 1a	33 c0 00 00	①	②	③
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			
000030:	③	④	00	⑤	00 00 00 00 00 00 00 00	

メモリの中身

プログラム本体も
メモリ中にある

68000アセンブラ言語

C言語

```
static short int x = 10;  
static short int y = 20;  
static short int z = 0;
```

```
int main()  
{
```

```
    z = x + y;
```

```
    return 0;  
}
```

等価

```
.data  
x:  
    .dc.w 10  
y:  
    .dc.w 20  
z:  
    .dc.w 0
```

等価

```
.text  
    move.w x, %d0  
    add.w y, %d0  
    move.w %d0, z  
  
    .dc.w 0x4048  
    stop #0
```

関数の定義は、
今後の授業で触れる(第5回の講義)

実行結果の例

ここでは, x , y , z はともに2バイト
のデータ

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

$$z \leftarrow x + y$$

プログラム本体そのものが入っているエリア

ここでは, x, y, z はともに2バイトのデータ

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

x

y

z

未使用

データが入っているエリア

68000アセンブラ言語

```
.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
```

データエリアの確保

x, y, z (ともに2バイトデータ)
のためのデータエリアを確保せよ

プログラム本体

68000アセンブラ言語

```
.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
```

最初の時点

(プログラム全体をメモリ上にロードした時点であり、プログラムを実際に行う前)

000000:	30 39 00 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

68000アセンブラ言語

```
.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
```

「2バイトをデータエリア内に確保.
最初から「10(10進数)」にしておく.
x というラベルを付ける」という指示

「2バイトをデータエリア内に確保.
最初は「20(10進数)」にしておく.
y というラベルを付ける」という指示

「2バイトをデータエリア内に確保.
最初は「0(10進数)」にしておく.
z というラベルを付ける」という指示

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

プログラム全体をメモリ上に
ロードした時点で, x, y, z の値がセットされる

68000アセンブラ言語

```
.data
x:
    .dc.w 10
y:
    .dc.w 20
z:
    .dc.w 0
```

「2バイトをデータエリア内に確保。最初から「10(10進数)」にしておく。xというラベルを付ける」という指示

「2バイトをデータエリア内に確保。最初は「20(10進数)」にしておく。yというラベルを付ける」という指示

「2バイトをデータエリア内に確保。最初は「0(10進数)」にしておく。zというラベルを付ける」という指示

.text

```
move.w x, %d0
add.w y, %d0
move.w %d0, z
```

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

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

プログラム全体をメモリ上にロードした時点で、x, y, z の値がセットされる

```
.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

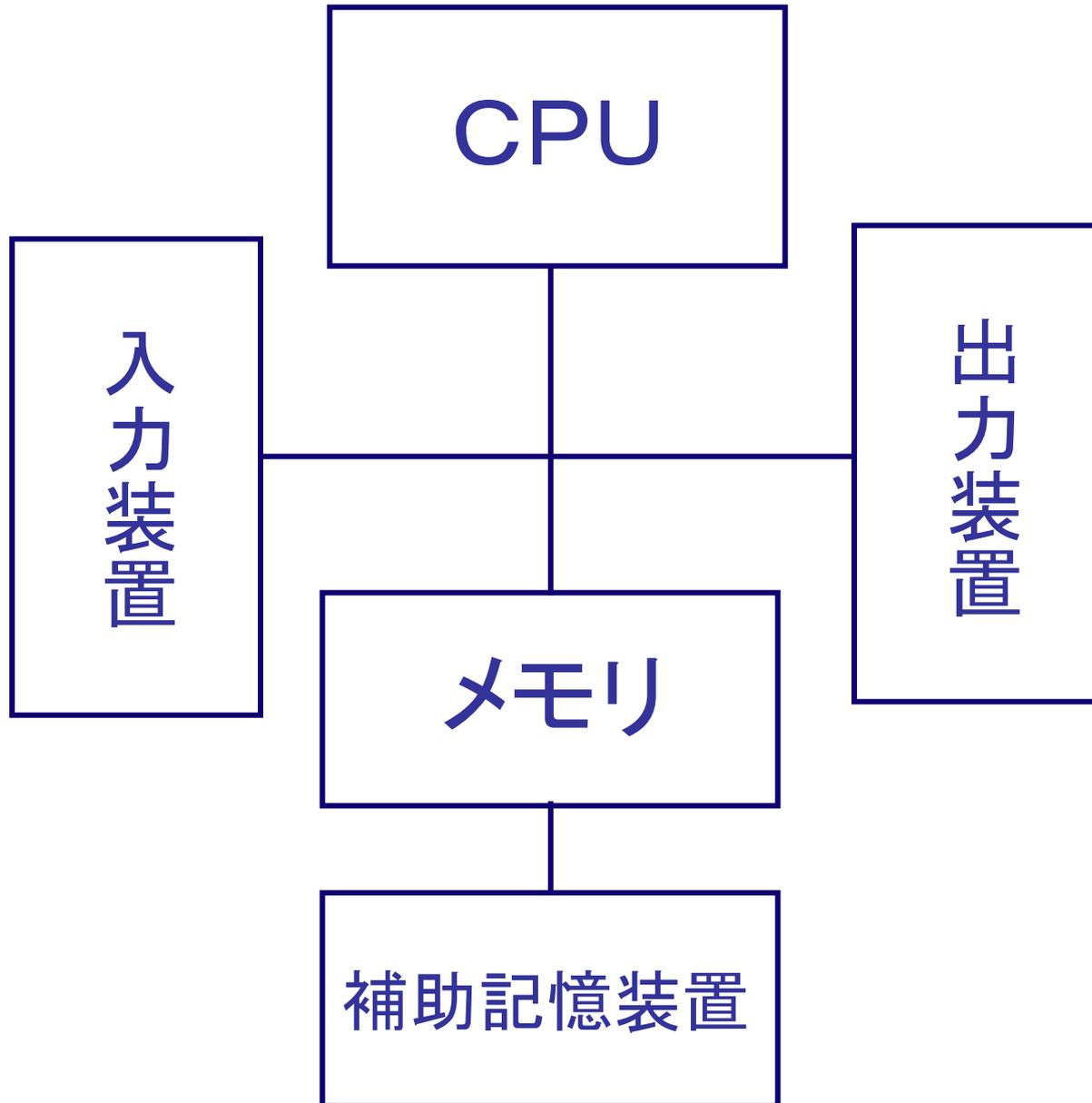
    .dc.w 0x4048
    stop #0
```

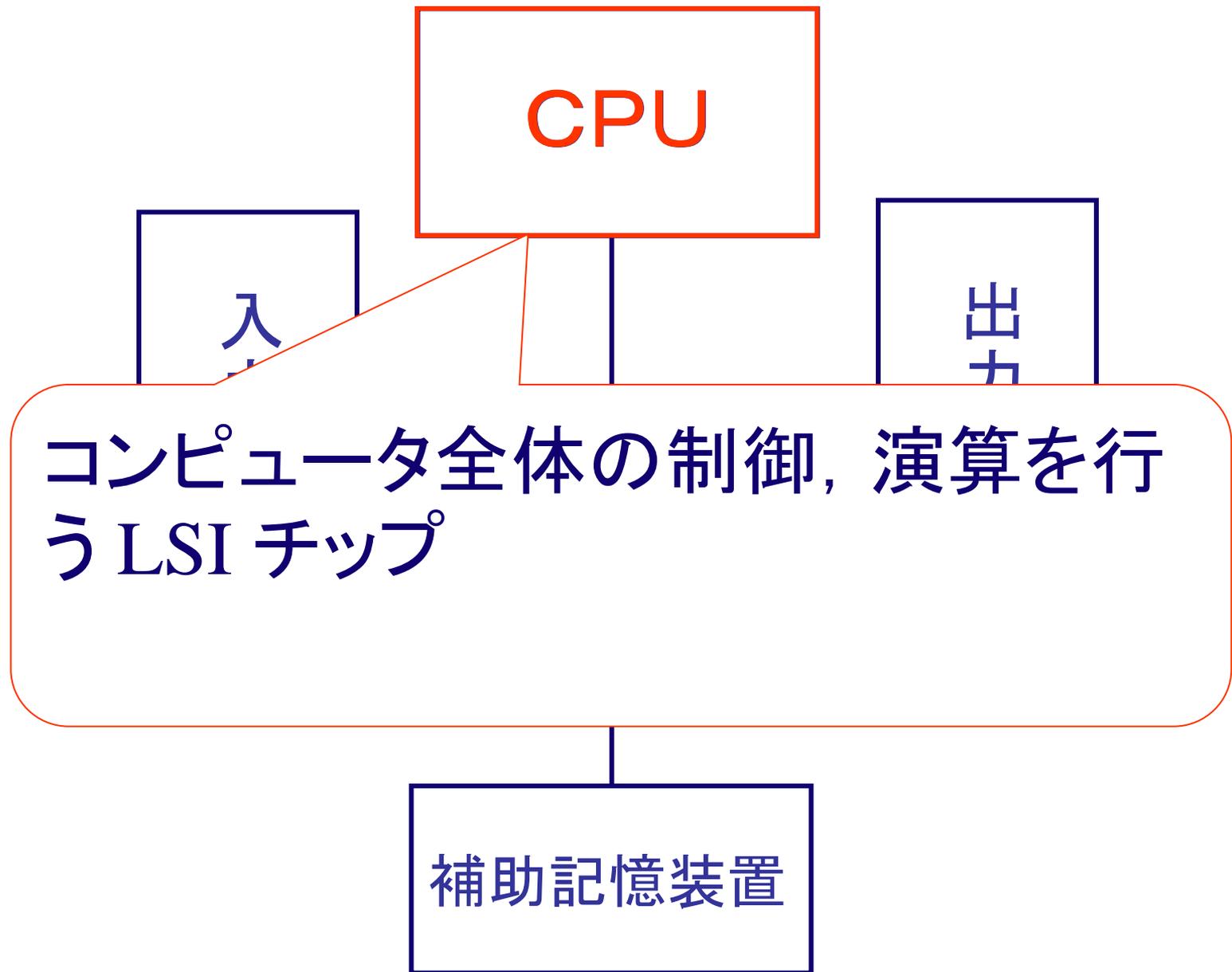
この授業では、
10進数は
10, 20 のように
16進数は
0x0a, 0x14 のように書く

プログラム本体

この理解には、CPU とメモリの
振る舞いを「頭の中にイメージ
できる」練習を必要とする

コンピュータのハードウェア構成





CPU

入

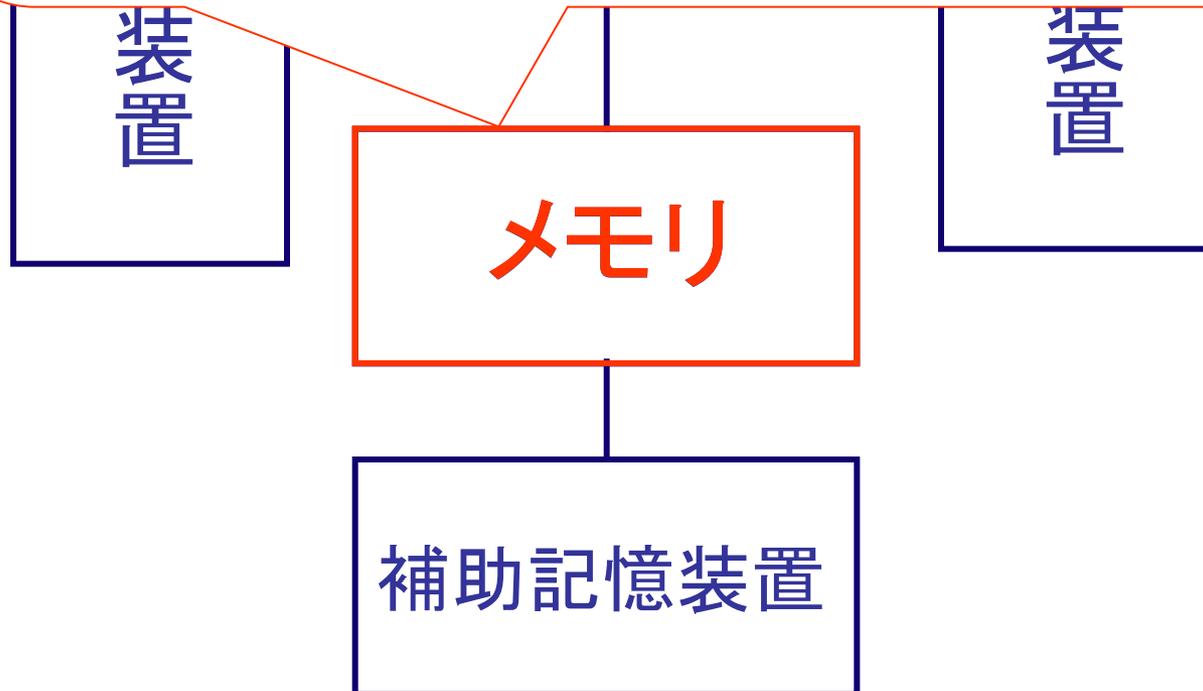
出

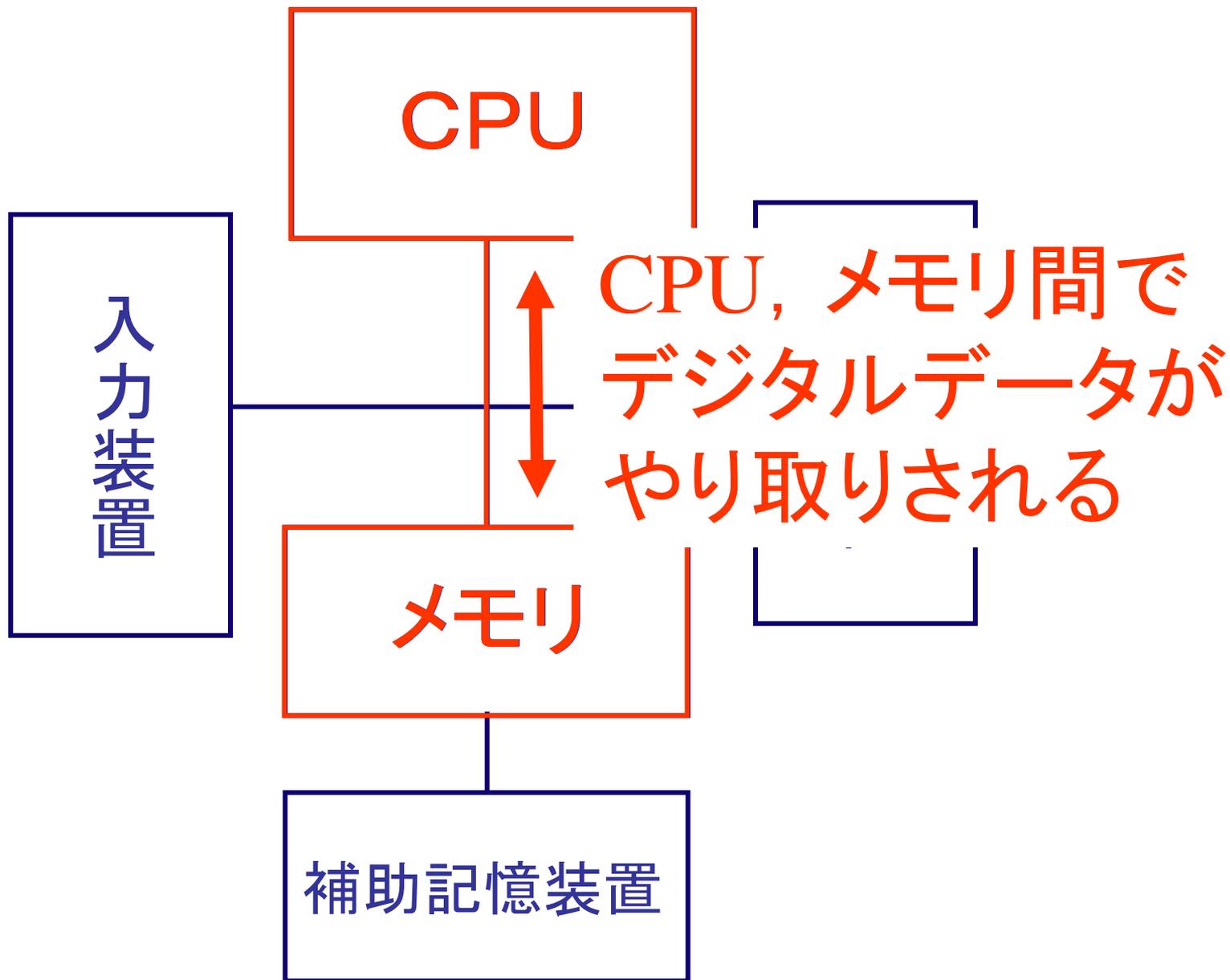
コンピュータ全体の制御, 演算を行
う LSI チップ

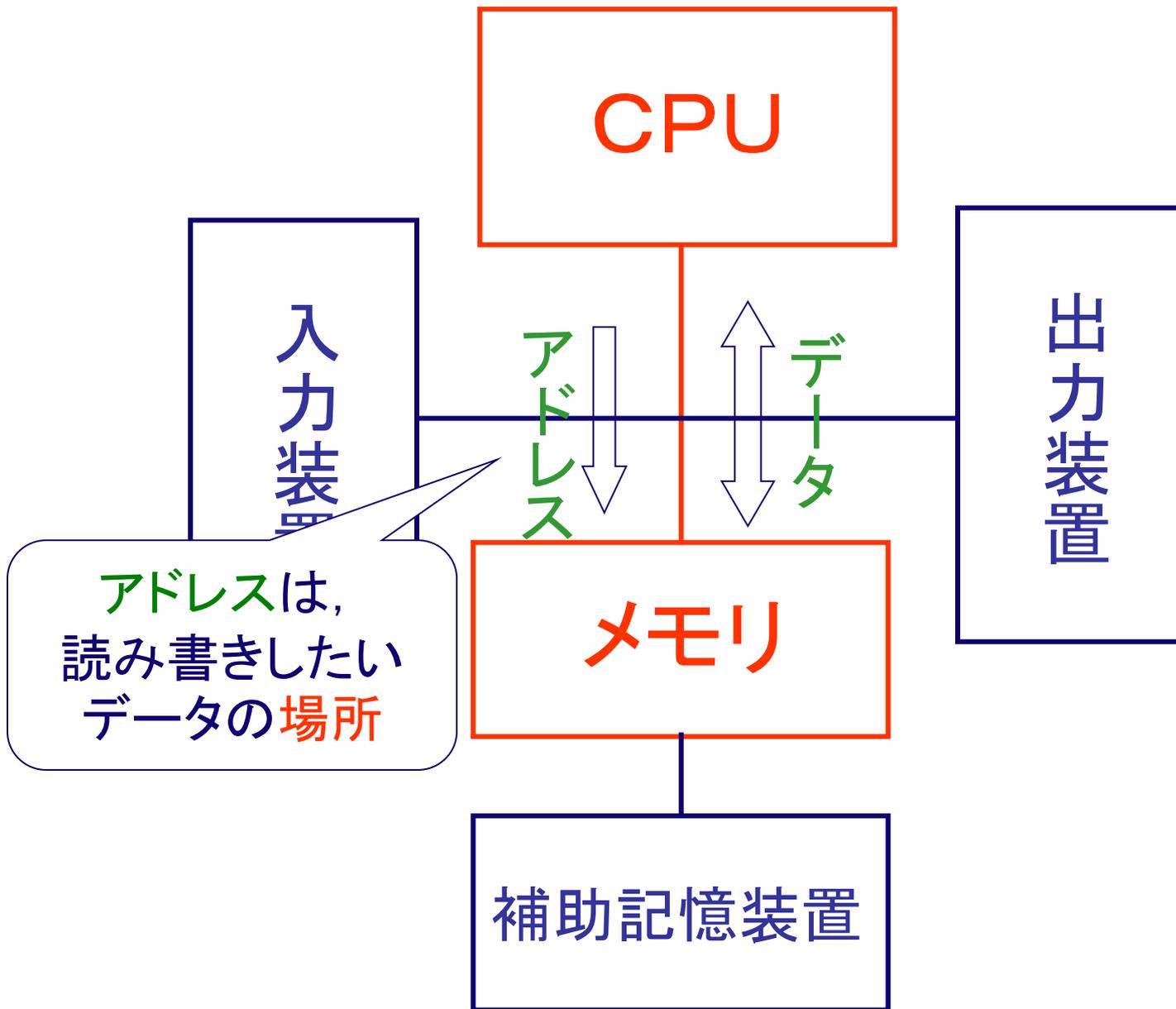
補助記憶装置

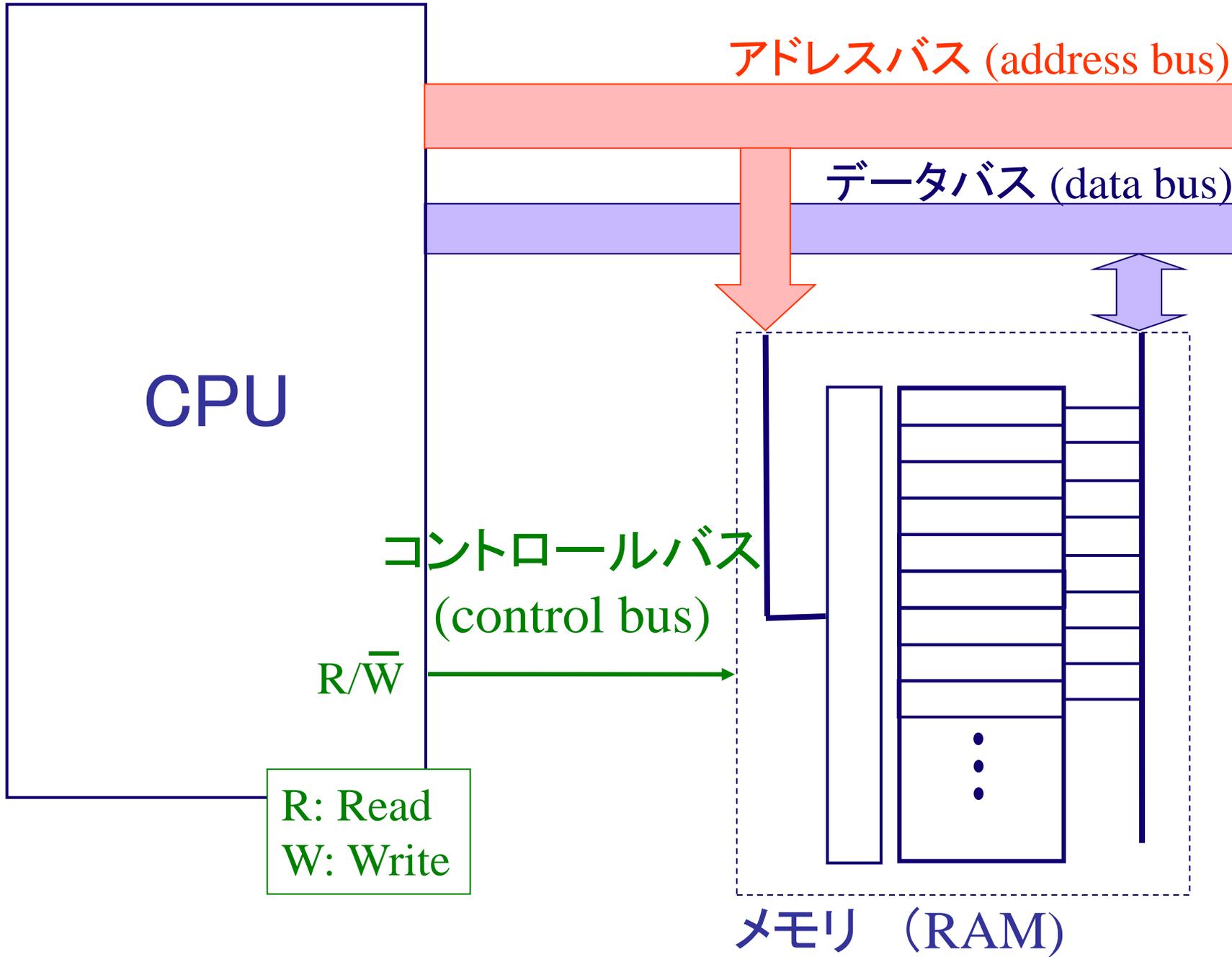
デジタルデータの記憶を行うLSI チップ

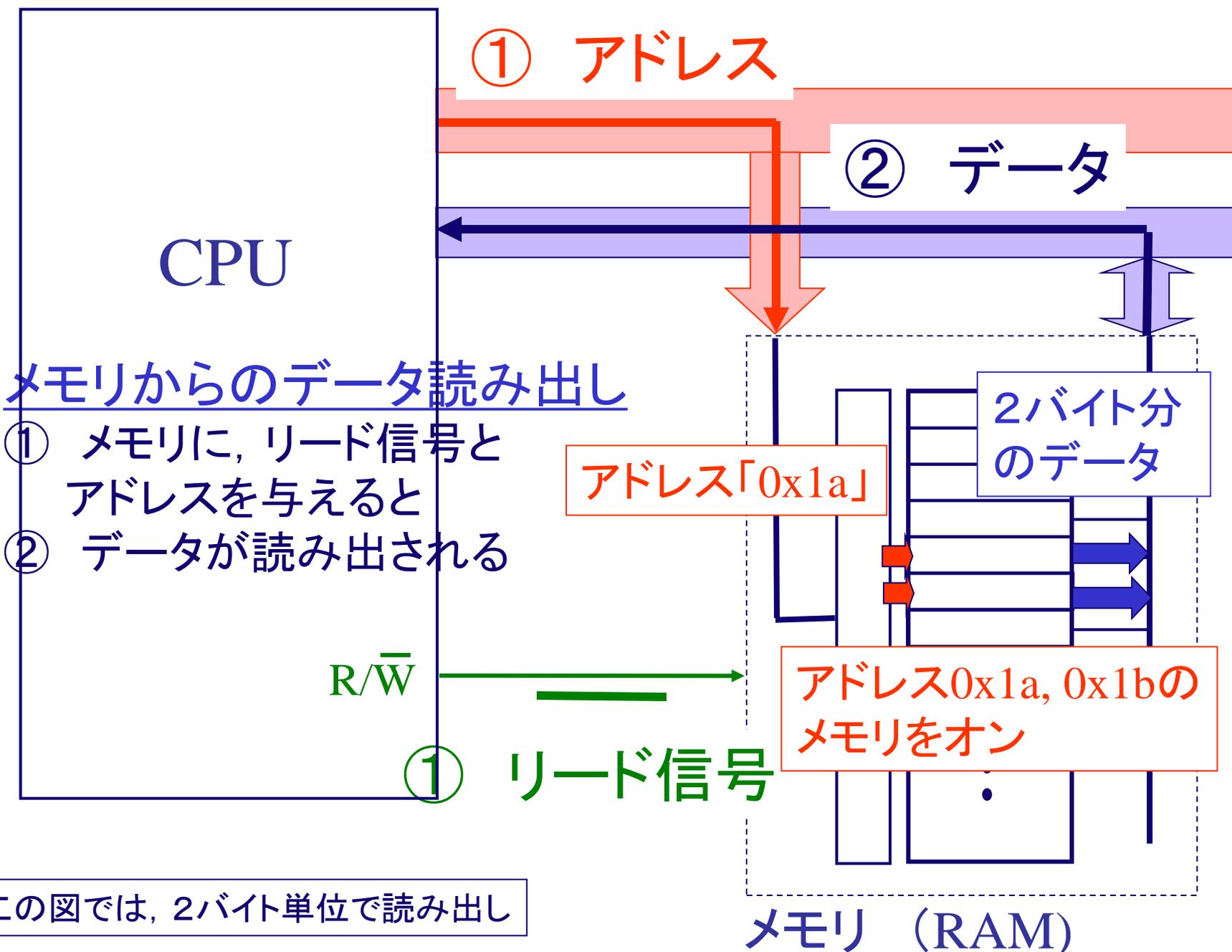
デジタルデータを覚えさせたり、取
り出したりの機能がある







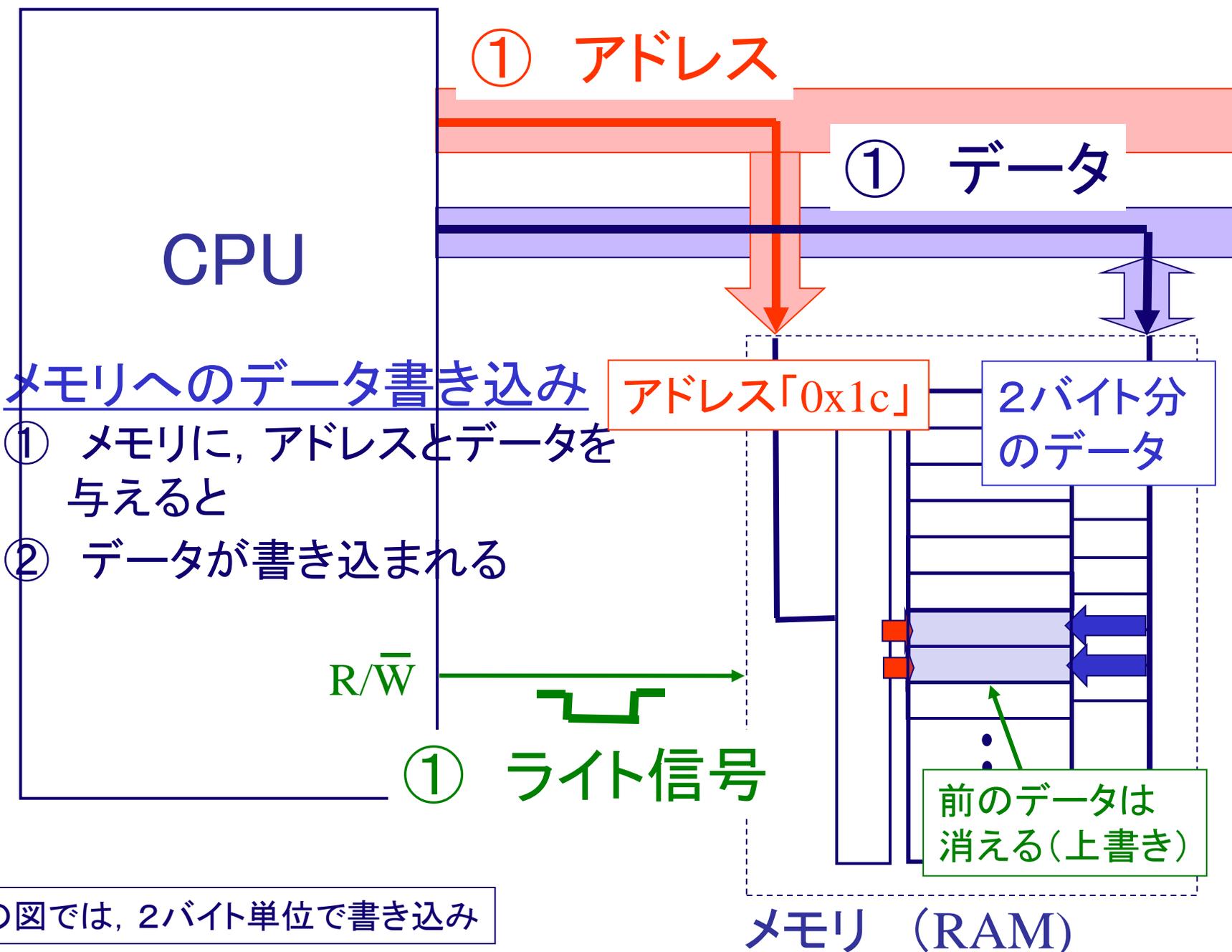




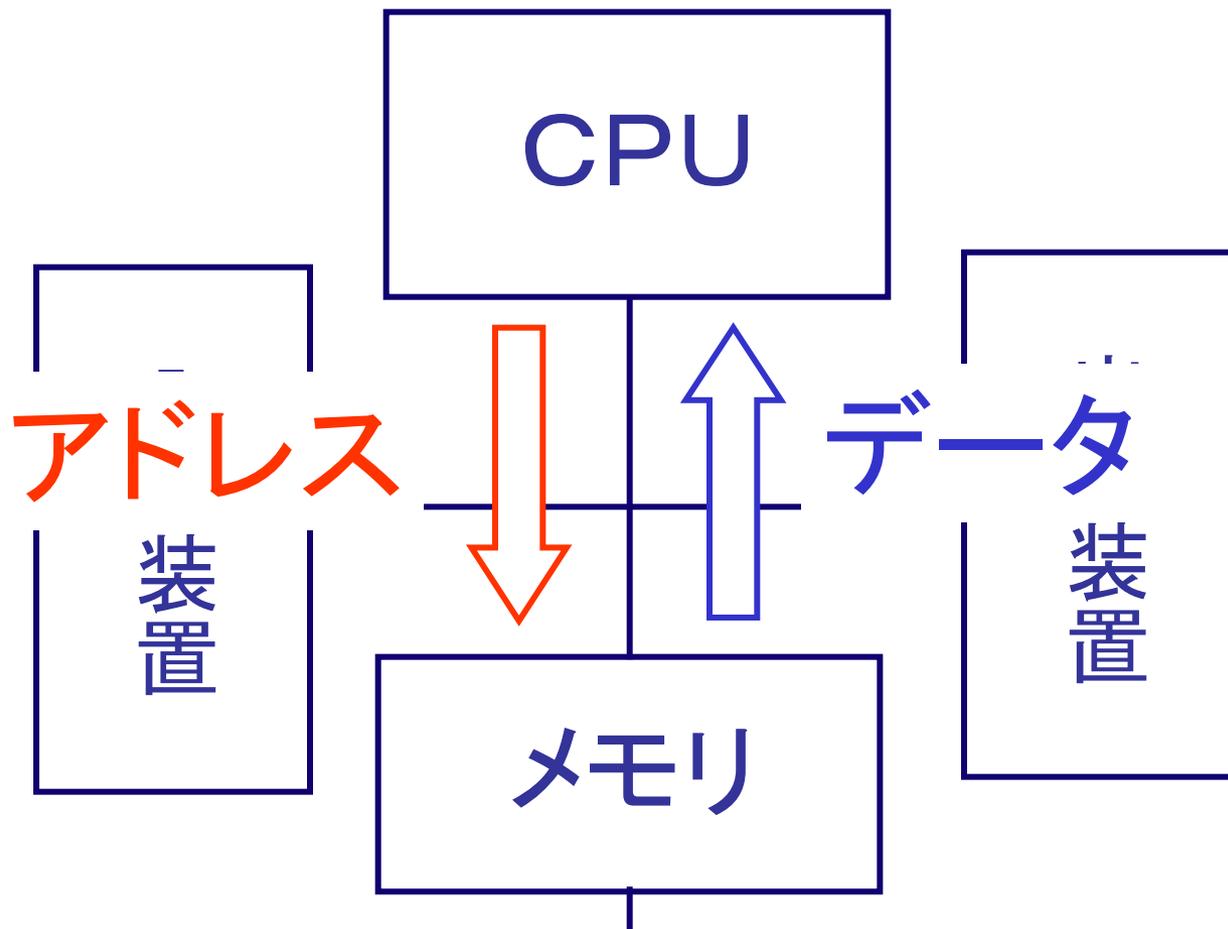
メモリからのデータ読み出し

- ① メモリに、リード信号とアドレスを与えると
- ② データが読み出される

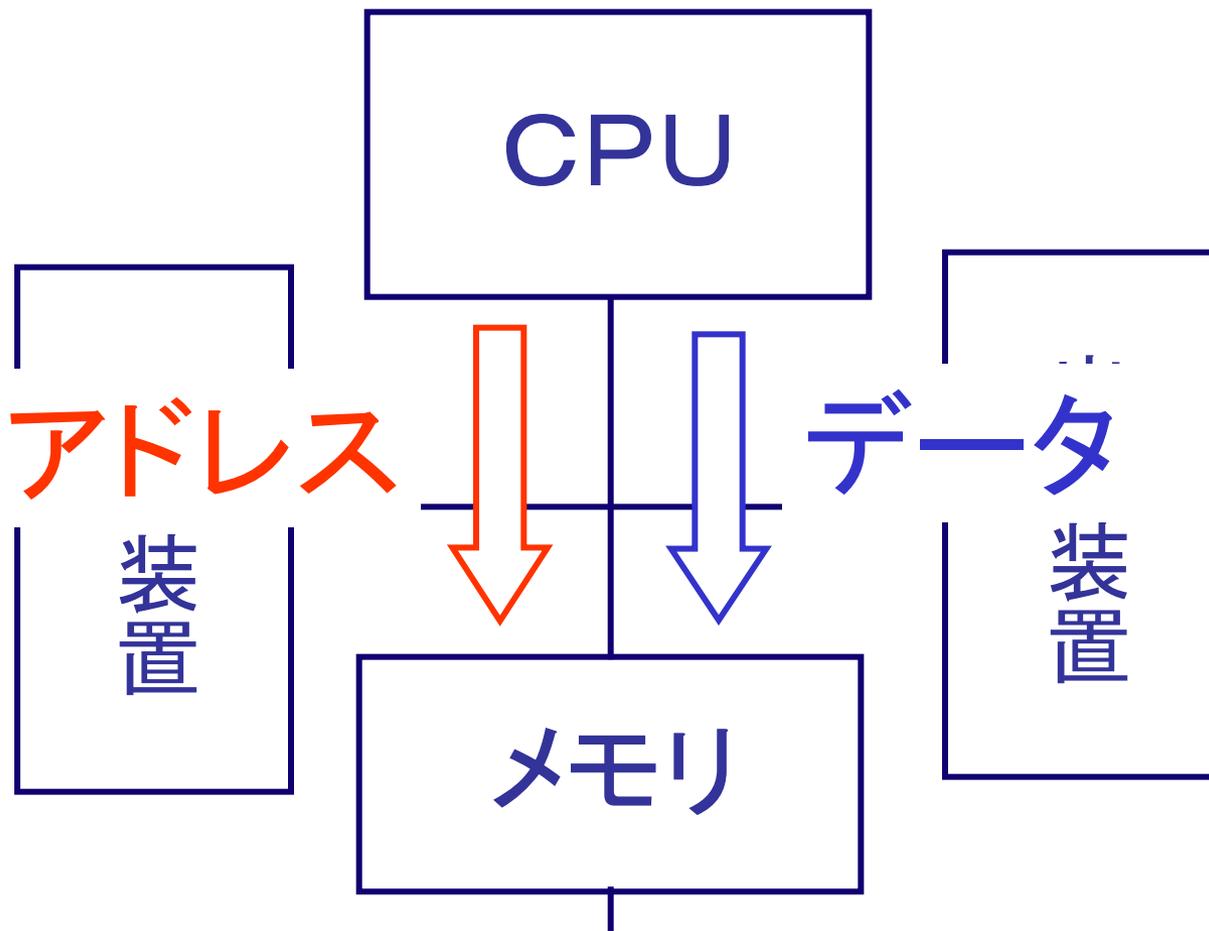
この図では、2バイト単位で読み出し



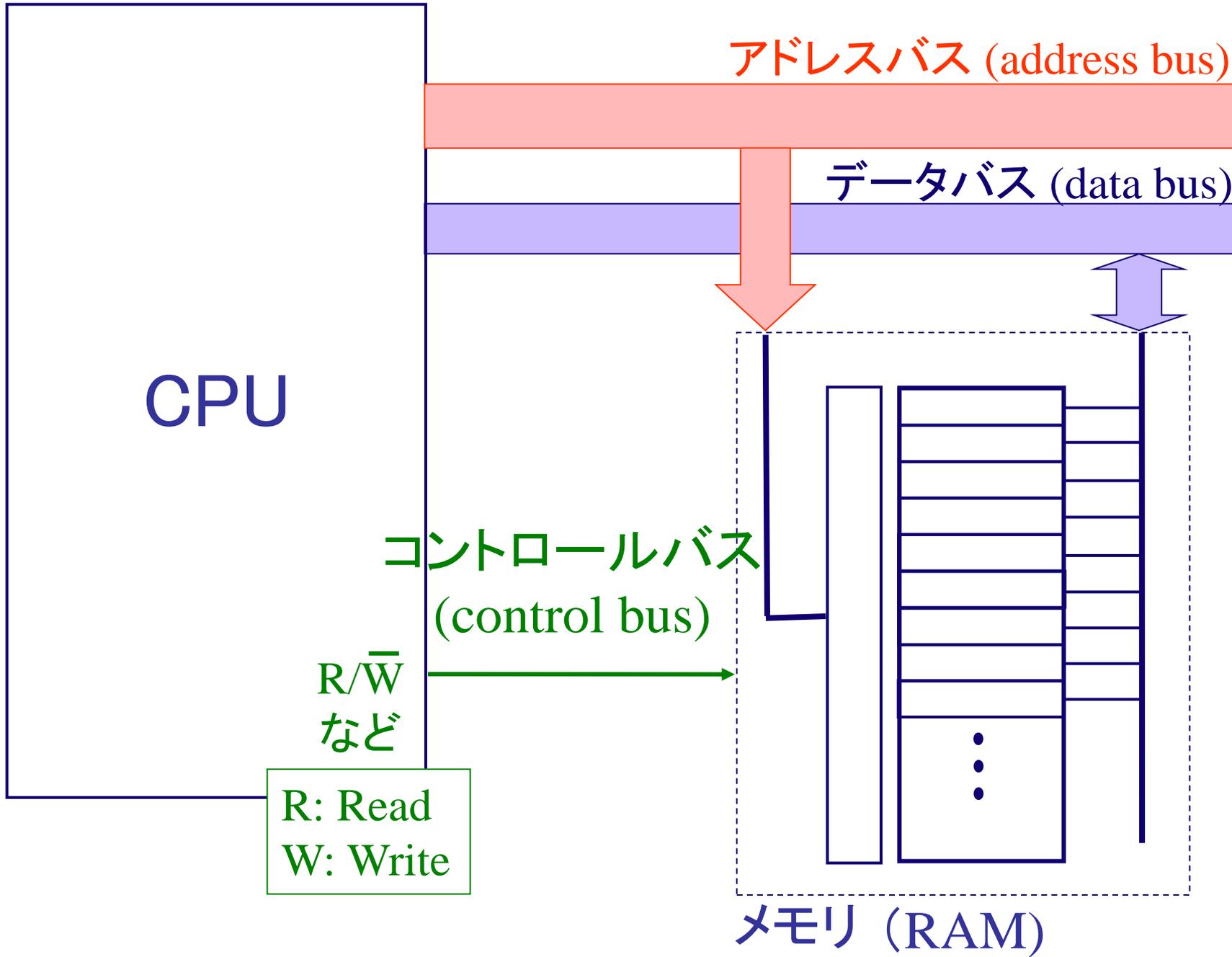
この図では、2バイト単位で書き込み



メモリからCPUへの読み出し



CPUからメモリへの書き込み



アドレスバス (address bus)

データバス (data bus)

CPU

コントロールバス
(control bus)

R/ \bar{W}
など

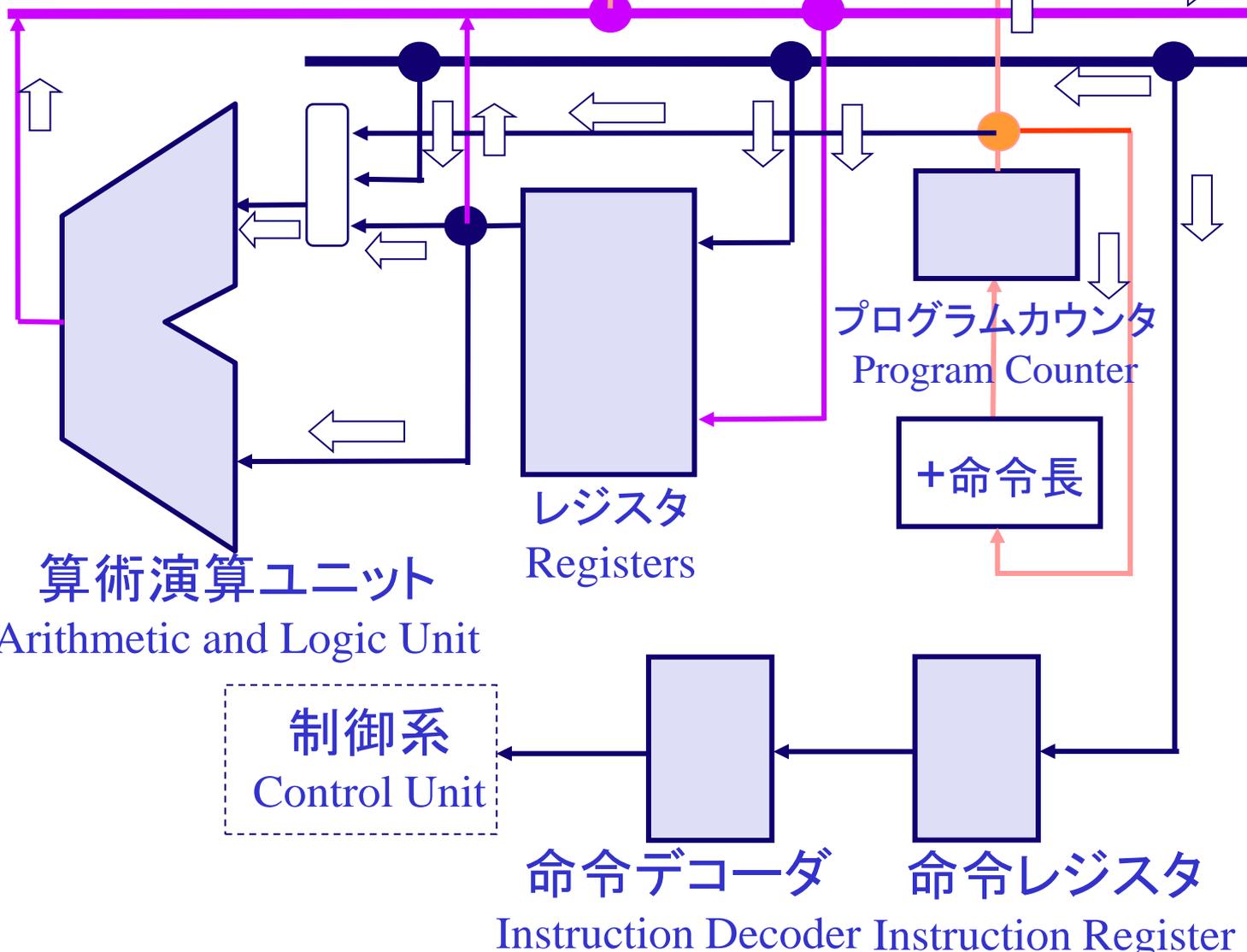
R: Read
W: Write

メモリ (RAM)

CPU

アドレスバス

データバス



プログラムカウンタ
Program Counter

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

R/W

メモリ



CPU

アドレスバス

データバス

次に実行すべき
プログラム命令の
メモリアドレスを記憶

算術演算, 論理
演算などの実行

データ等の記憶,
システムスタック
の管理,
比較の結果の保存

プログラムカウンタ
Program Counter

算術演算ユニット

Registers

制御系
Control Unit

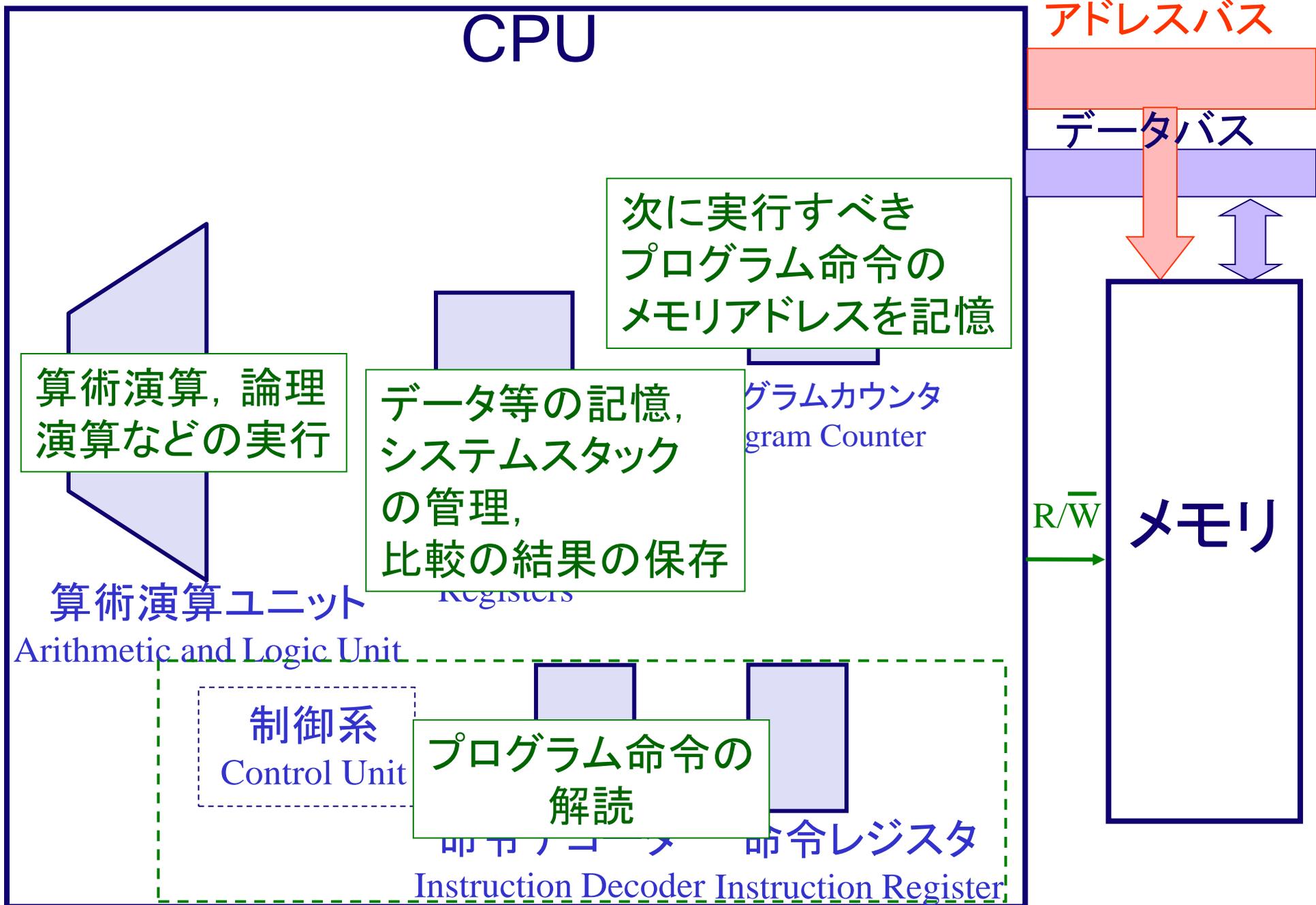
プログラム命令の
解読

命令レジスタ
Instruction Register

Instruction Decoder

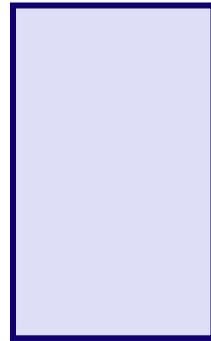
R/W

メモリ



CPU

この中身はCPUの種類によって異なる



レジスタ
Registers



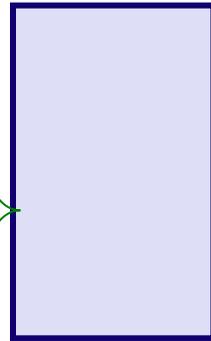
プログラムカウンタ
Program Counter

アセンブラプログラムでは、レジスタ、
プログラムカウンタの「名称」が現れる

CPU 68000 では

レジスタは4種類

1. データレジスタ
2. アドレスレジスタ
3. ユーザスタックポインタ, スーパーバイザスタックポインタ
4. ステータスレジスタ



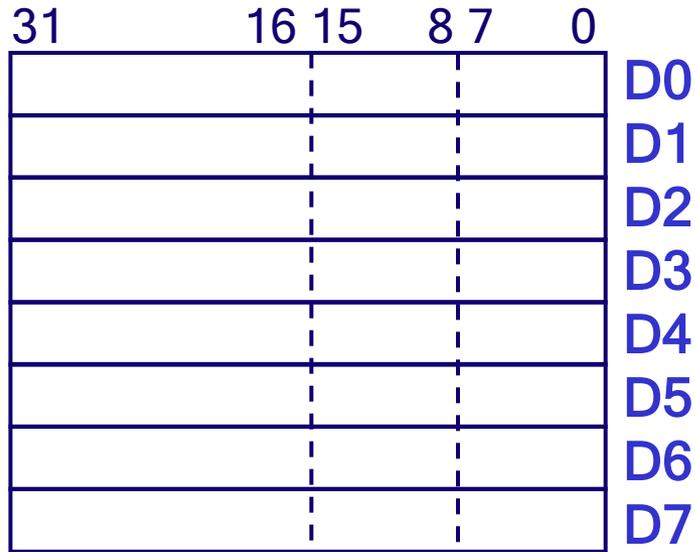
レジスタ
Registers



プログラムカウンタ
Program Counter

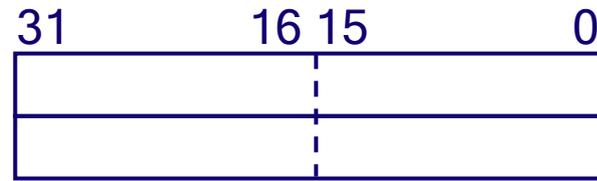
レジスタは, CPUの中にあって, データや制御情報等の一時格納を行う(一種のメモリ)

CPU 68000 では



データ
レジスタ
(data
registers)

32ビット長



ユーザ
スタックポインタ
A7
スーパーバイザ
スタックポインタ
(user stack
pointer,
supervisor stack
pointer)

32ビット長

同じ名前(間違いで
はない)SPとも書く

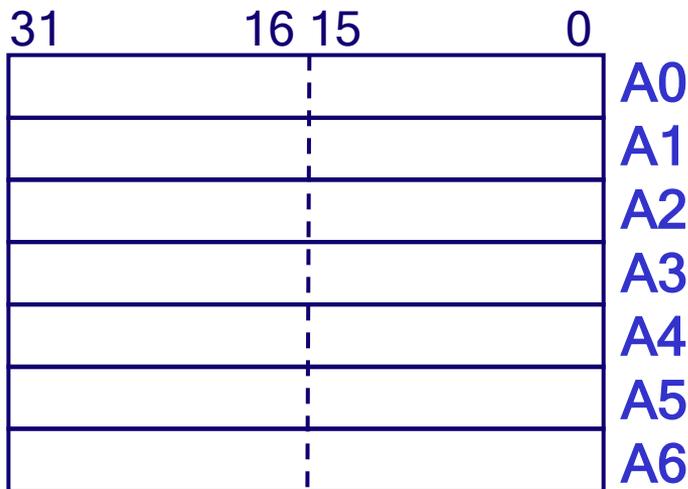
15 8 7 0



CCR

SR 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

CPU

アドレスバス

データバス

このデータと

このデータを
足して

ここに入れたい

算術演算ユニット

Arithmetic and Logic Unit

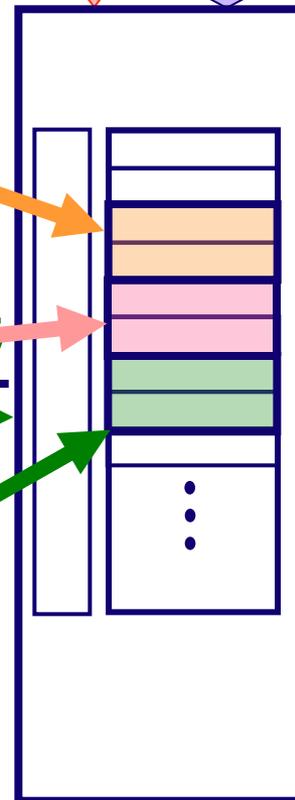
制御系

Control Unit

命令デコーダ

命令レジスタ

Instruction Decoder Instruction Register



メモリ

CPU

アドレスバス

データバス

算術演算ユニット

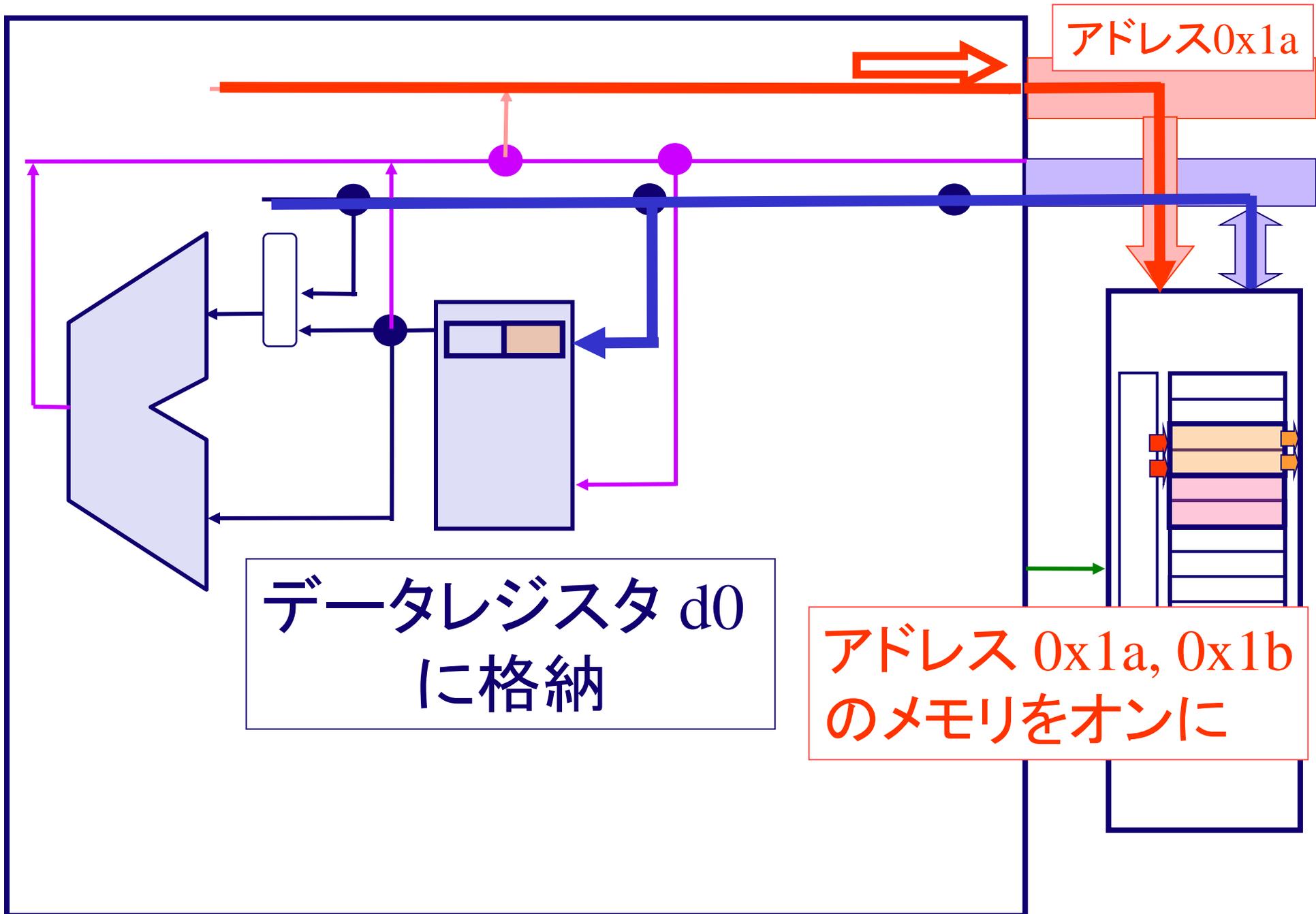
レジスタ
Registers

今回は、メモリから読み込んで来たデータの一時格納に使用

今回は、加算に使用

メモリ

Arithmetic and Logic Unit

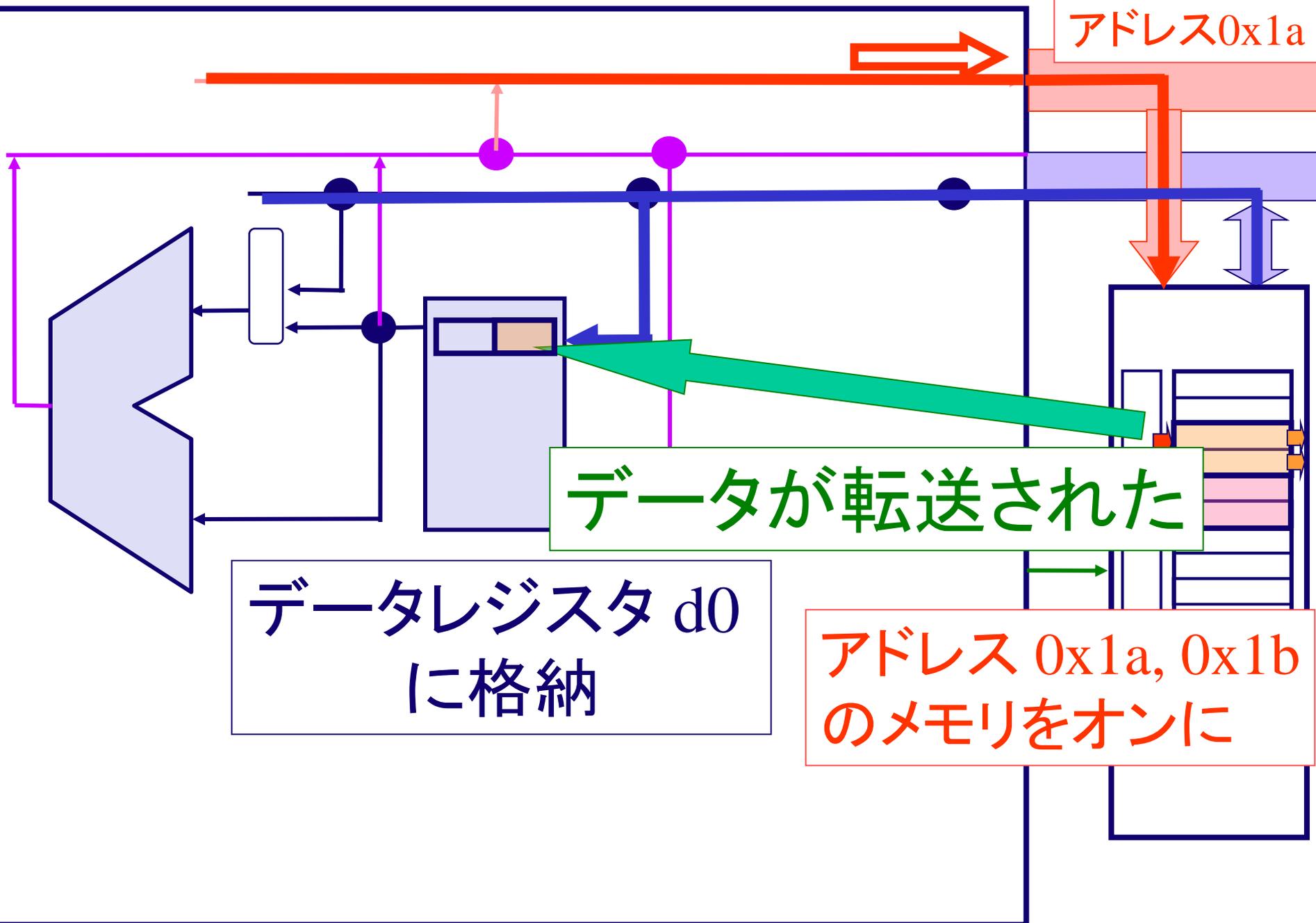


アドレス 0x1a

データレジスタ d0
に格納

アドレス 0x1a, 0x1b
のメモリをオンに

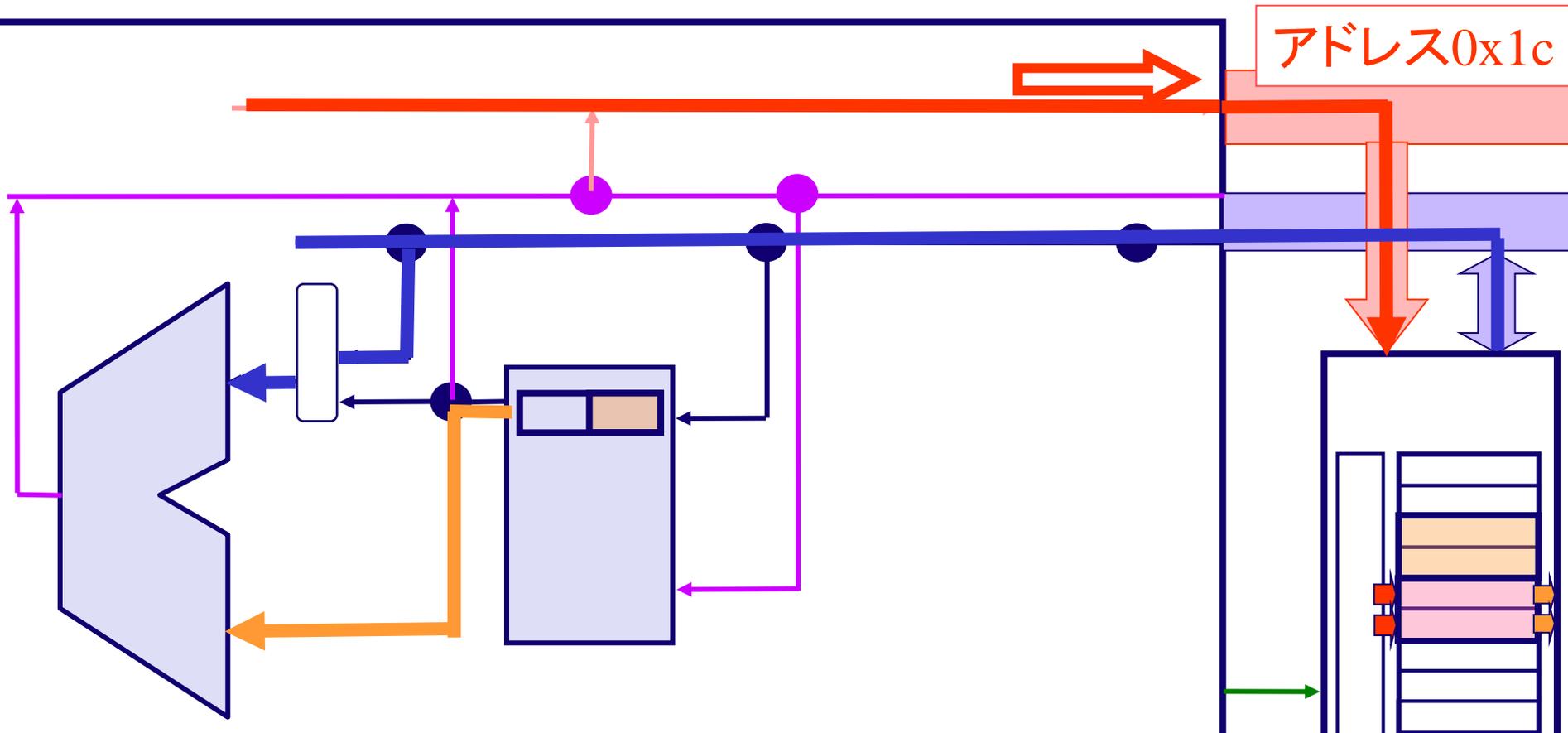
アドレス0x1a



データが転送された

データレジスタ d0
に格納

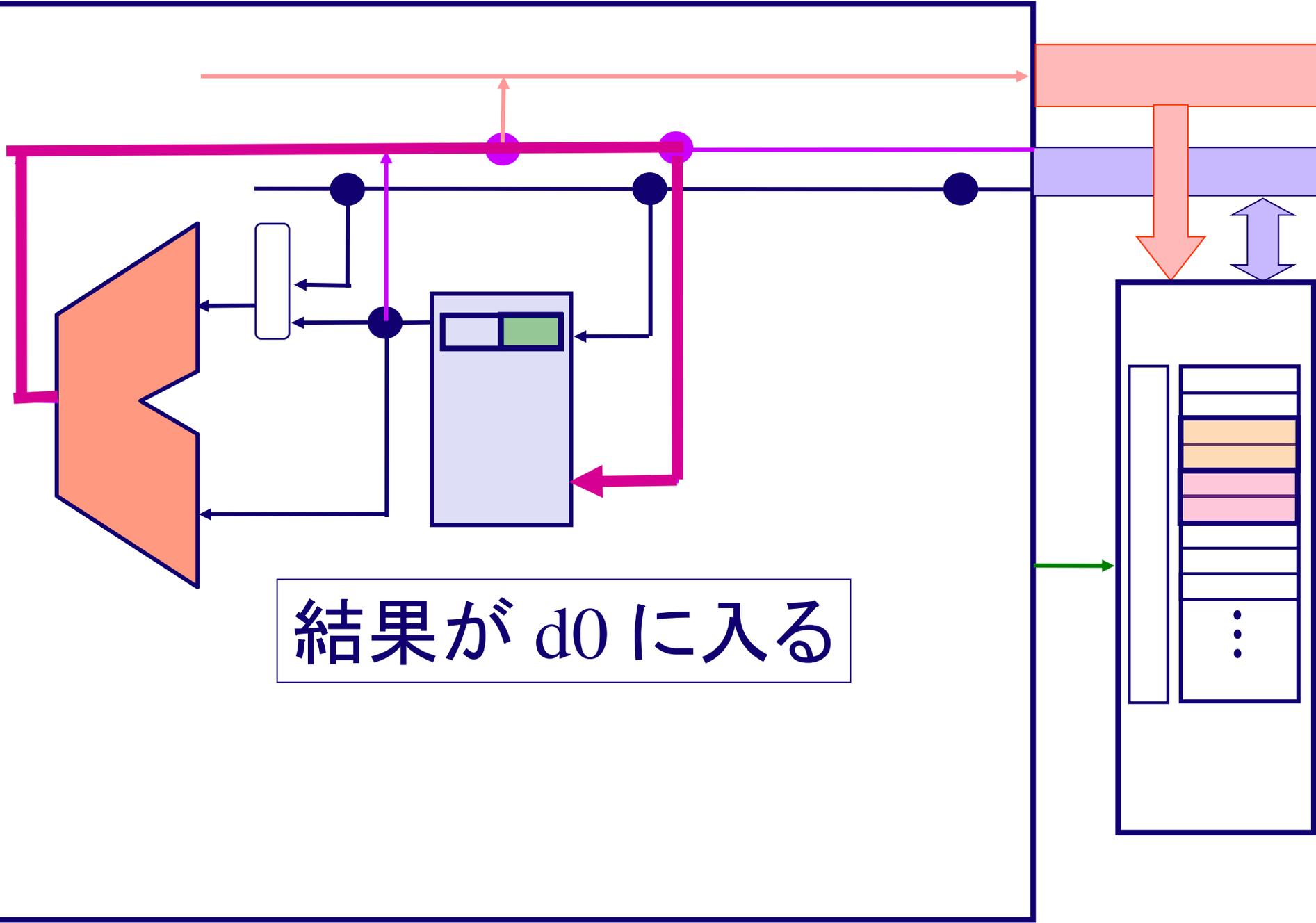
アドレス 0x1a, 0x1b
のメモリをオンに



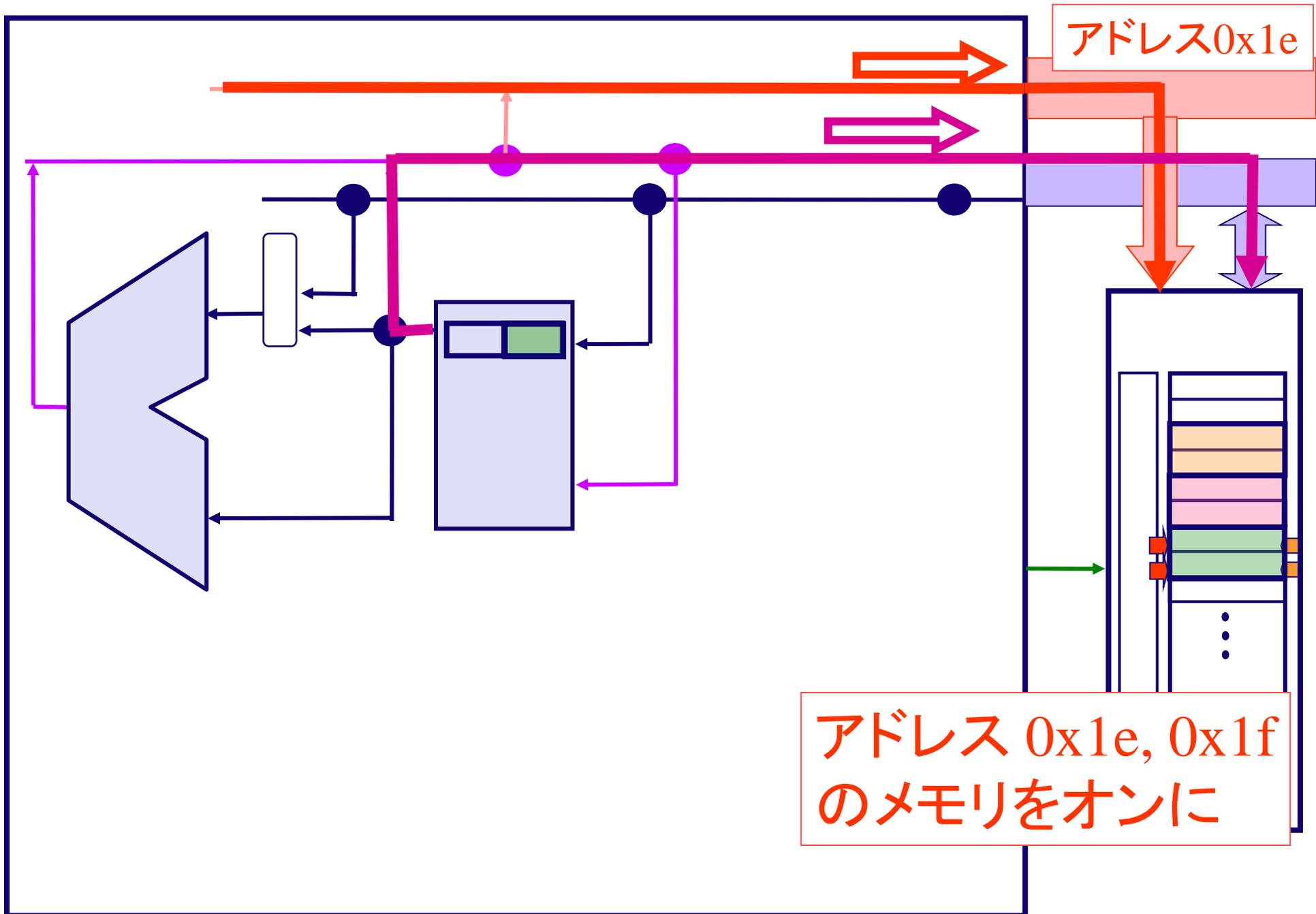
アドレス0x1c

2つのデータが
算術演算ユニットに
与えられる

アドレス 0x1c, 0x1d
のメモリをオンに



結果が d0 に入る



```
.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
```

メモリ読み出し

→ データレジスタ d0 に格納

1ワードの読み出し

データレジスタ長は4バイト
なので、d0 の下位2バイトに
入る

```
.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
```

メモリ読み出し

- データレジスタ d0 との加算
- 加算の結果はd0に格納

```
.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
```

メモリ書き込み

→ データレジスタ d0 の中身
を書き込む

1ワードの書き込み

データレジスタ長は4バイト
なので、d0 の下位2バイトが
書き込まれる

68000 アセンブラ言語

- CPU (Central Processing Unit; コンピュータの中央にあるチップのこと) の挙動を1ステップずつ指定する言語