

as-5. サブルーチン呼び出しの メカニズム

(68000 アセンブラ)

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

金子邦彦



種々のオペランド

```
move.w #200, %d2
```

```
/* #200 は10進数. d2 の上位バイトは変化しない */
```

```
move.l #0x01, %d1
```

```
/* #0x01 は16進数. d0 に 0x00000000 がセットされる */
```

```
.equ BUFFER_SIZE 100
```

```
move.w #BUFFER_SIZE, %d2
```

割り込み禁止の
決まり文句

```
move.w #0x2700, %sr
```

```
/* 割り込み禁止 */
```

```
move.l %d0, %d1
```

```
move.l %d0, (%a0)
```

オペランド
「操作」すべきデータの
対象がかかっている部分

```
/* アドレスレジスタ a0 がポイントするメモリに転送 (a0は変化しない) */
```

```
move.b #0x00, (%a0)
```

```
/* アドレスレジスタ a0 がポイントするメモリに 0 をセット (.b なので1バイト) */
```

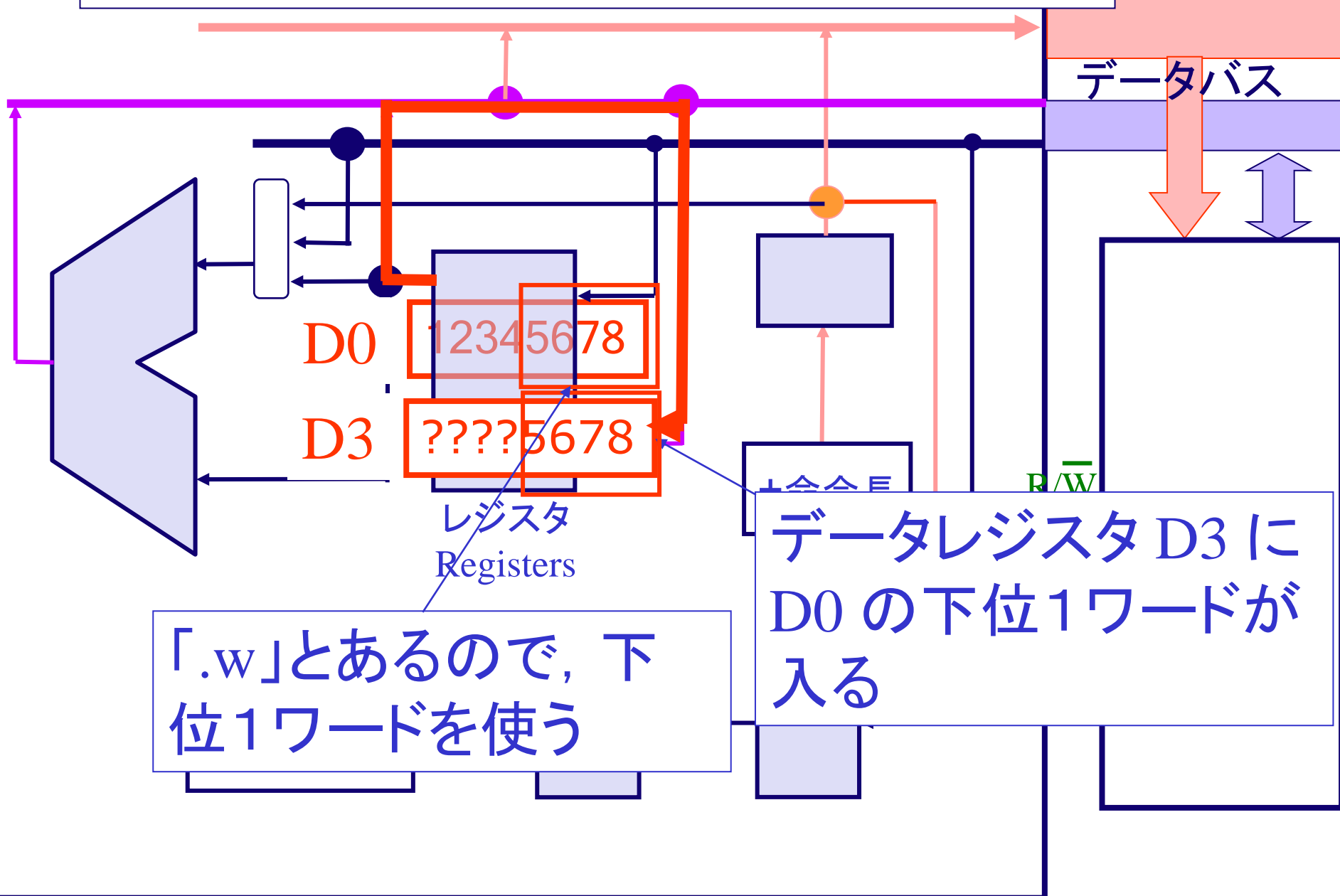
データレジスタ直接 (data register direct)

例:

```
move .w %D0, %D3
```

• 記法 %Dn

move.w %D0, %D3 の命令実行



アドレスレジスタ直接 (address register direct)

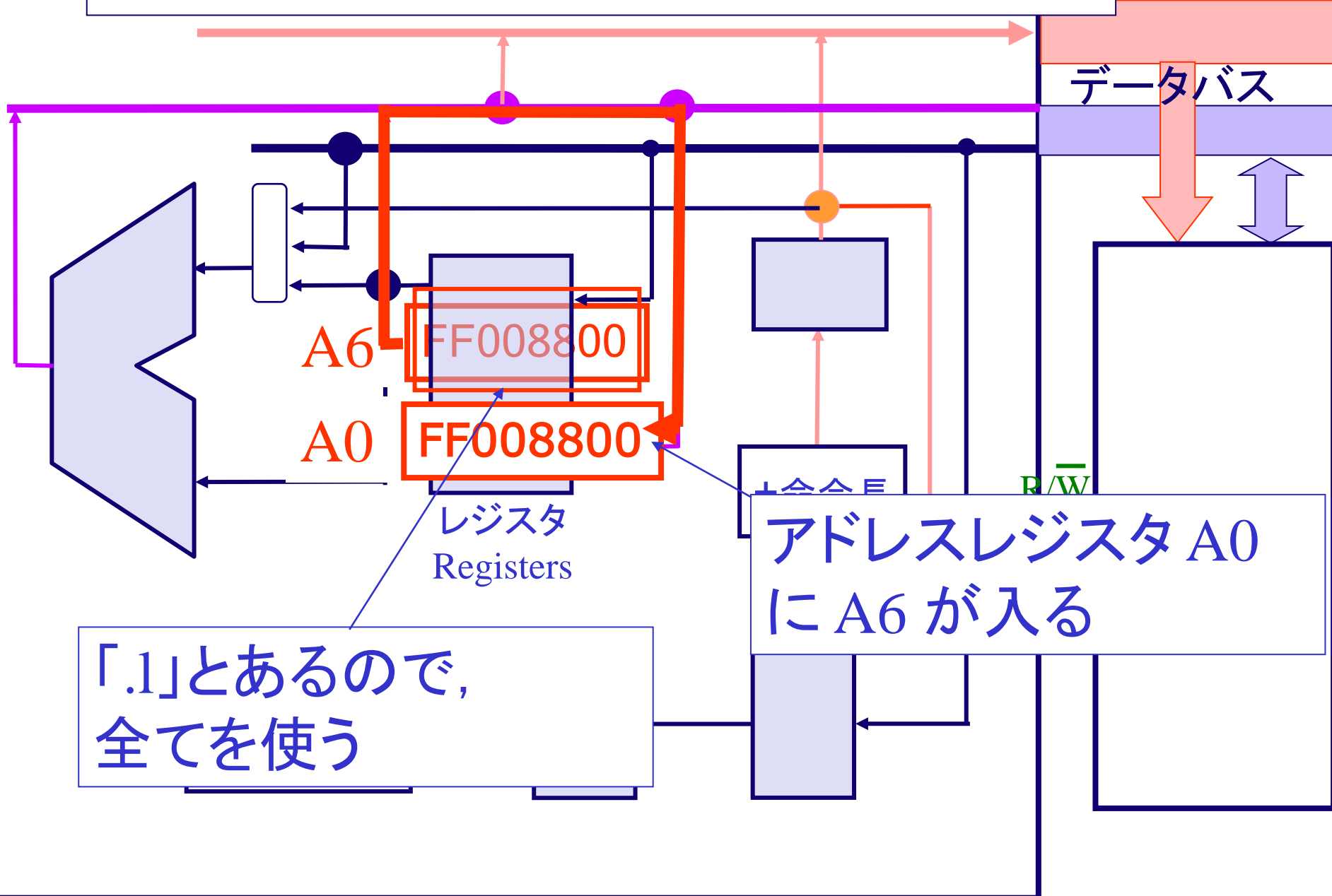
例:

```
move .1 %A6, %A0
```

• 記法

%An

move.l %A6, %A0 の命令実行



アドレスバス
データバス

A6 FF008800

A0 FF008800

レジスタ
Registers

命令実行
アドレスレジスタ A0
に A6 が入る

「.l」とあるので、
全てを使う

R/W

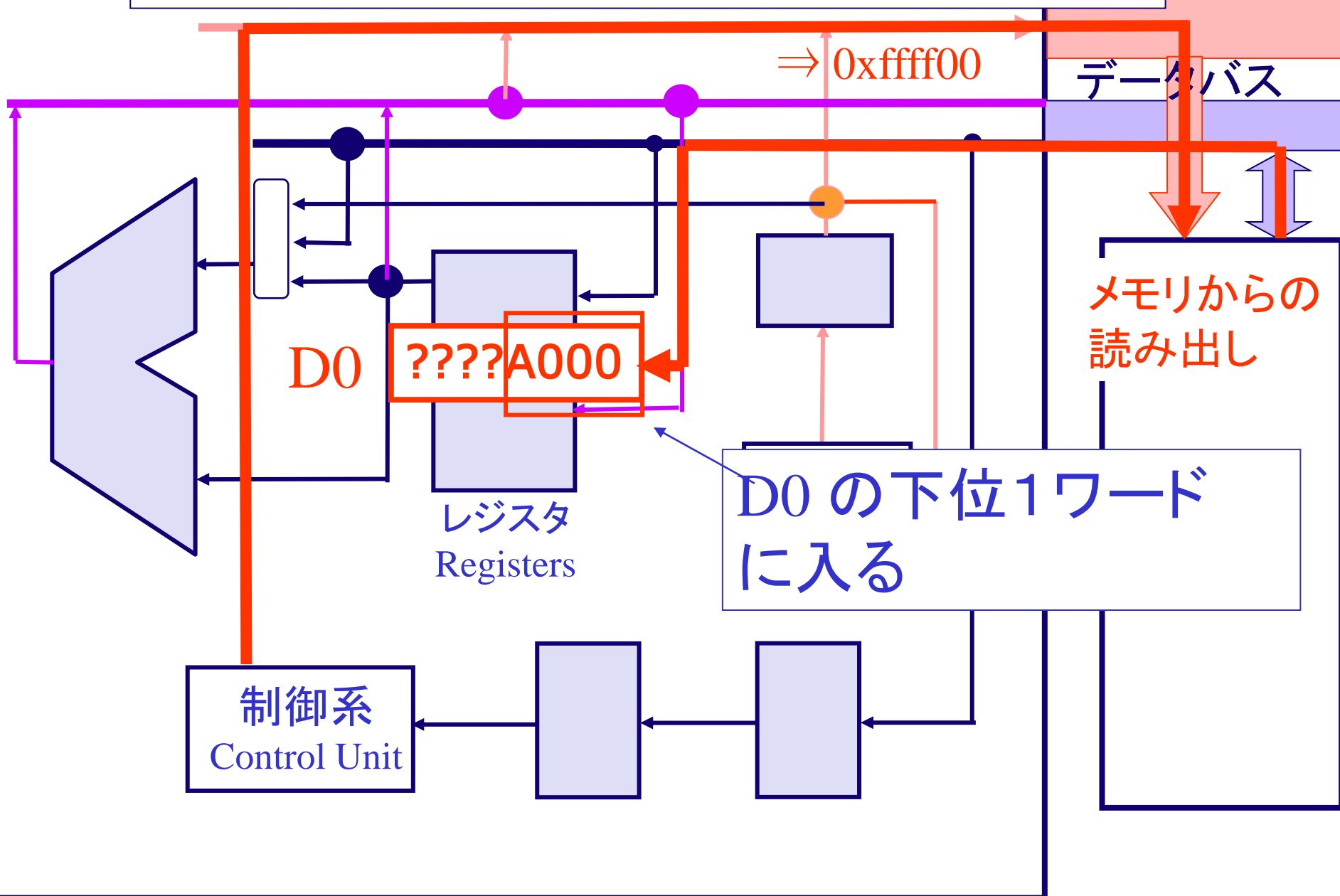
アブソリュート (absolute)

例:

```
.equ ADDR 0xffffffff00  
move.w ADDR, %D0  
move.w %D1, ADDR
```

※ メモリアドレスの値を指定

move.w ADDR, %D0 の命令実行



D0

????A000

D0 の下位1ワード
に入る

制御系
Control Unit

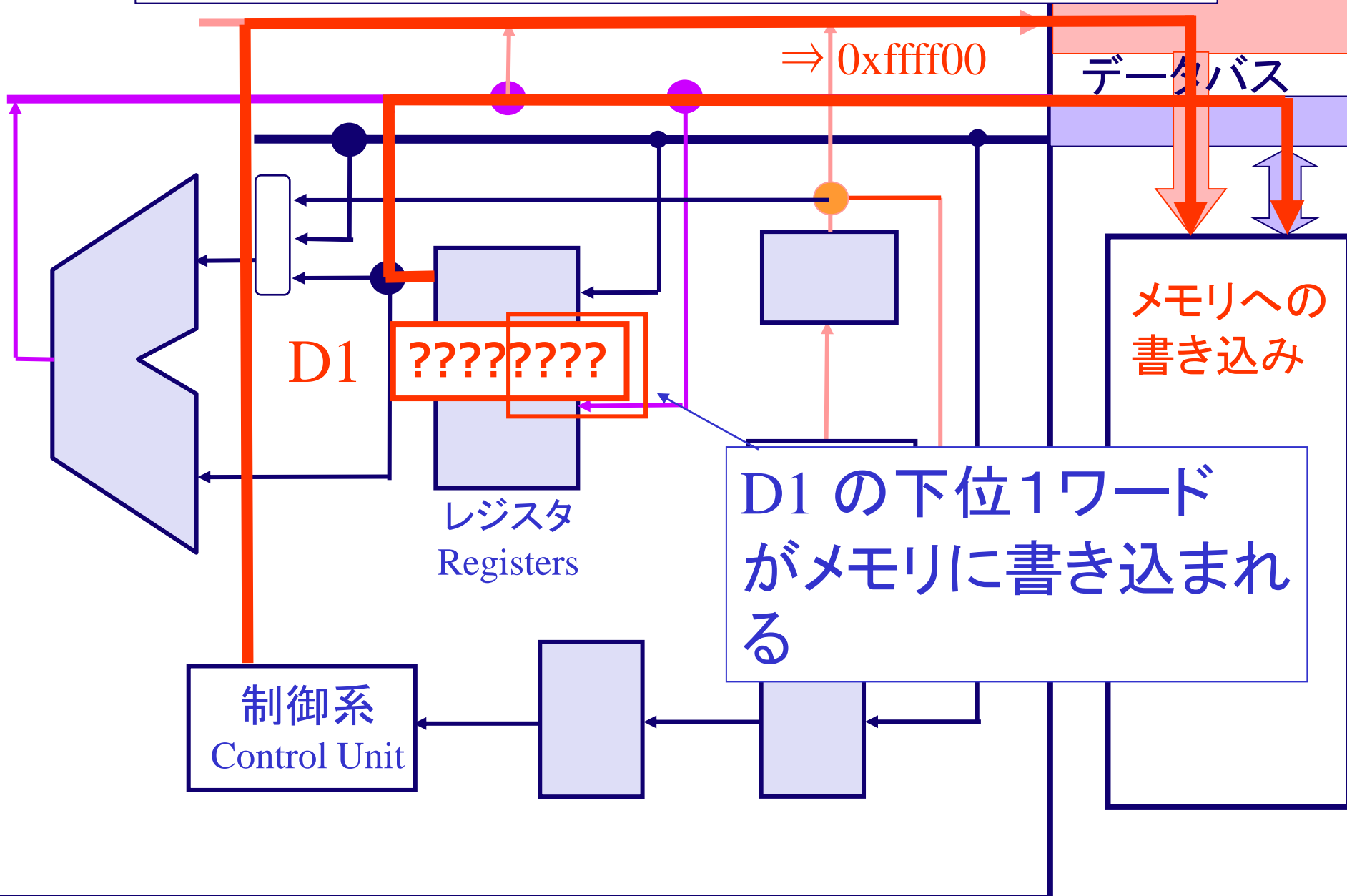
レジスタ
Registers

メモリからの
読み出し

データバス

⇒ 0xffff00

move.w %D1, ADDR の命令実行



D1 の下位1ワード
がメモリに書き込まれ
る

メモリへの
書き込み

⇒ 0xffff00

データバス

制御系
Control Unit

レジスタ
Registers

D1

????????

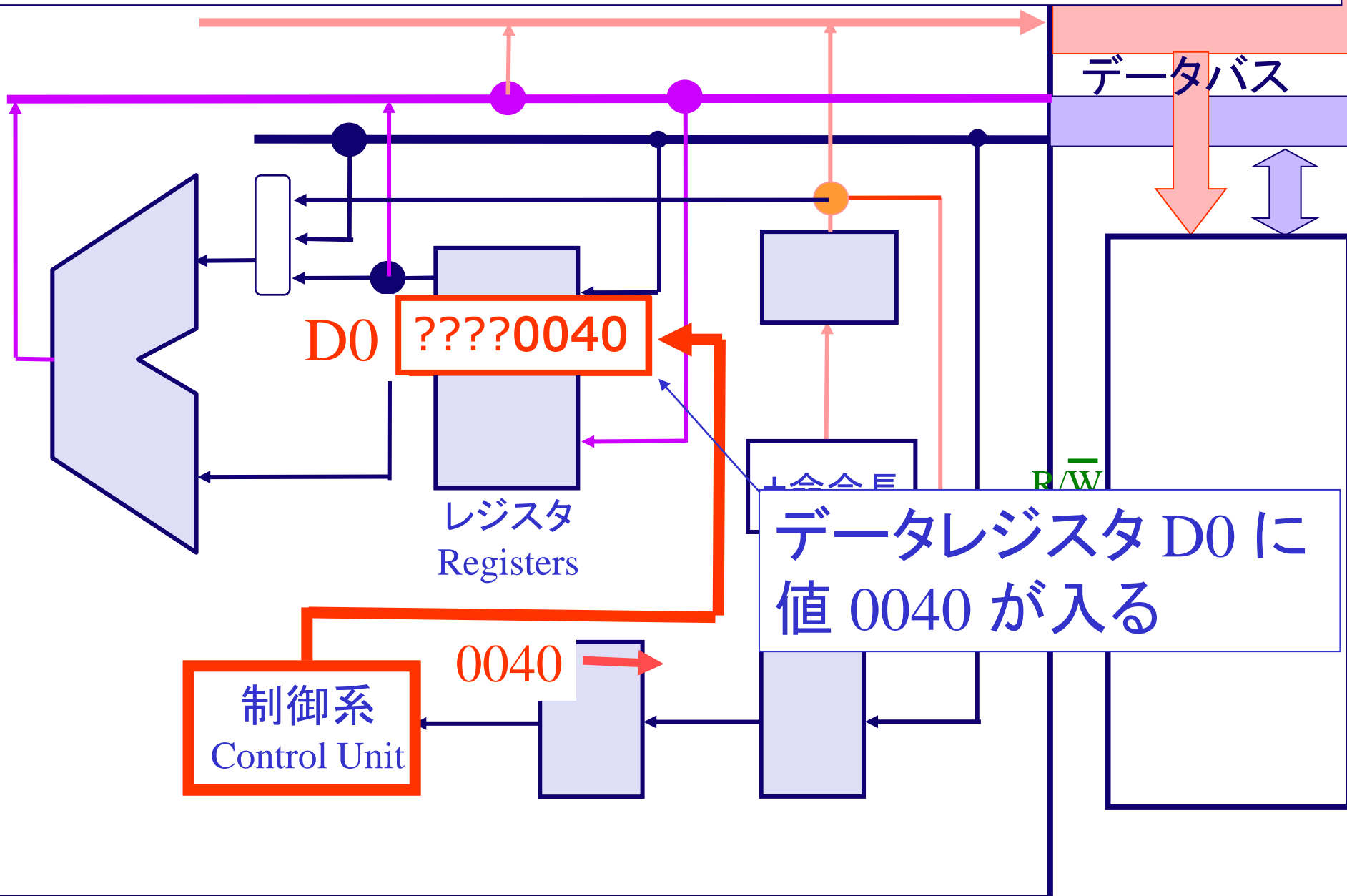
イミディエート (immediate)

例:

```
move .w #0x0040, %D0
```

- 記法 #data

move.w #0x0040, %D0 の命令実行では



- イミディエート

```
.equ ADDR      0x00ffff00
      move.w    #0x1000,%d0
      move.l    #ADDR,%a0
```

※ 値を扱う

- アブソリュート

```
.equ ADDR      0xffff00
      move.w    ADDR,%d0
```

※ メモリの読み出し, 書き込み

レジスタ間接 (register indirect)

例:

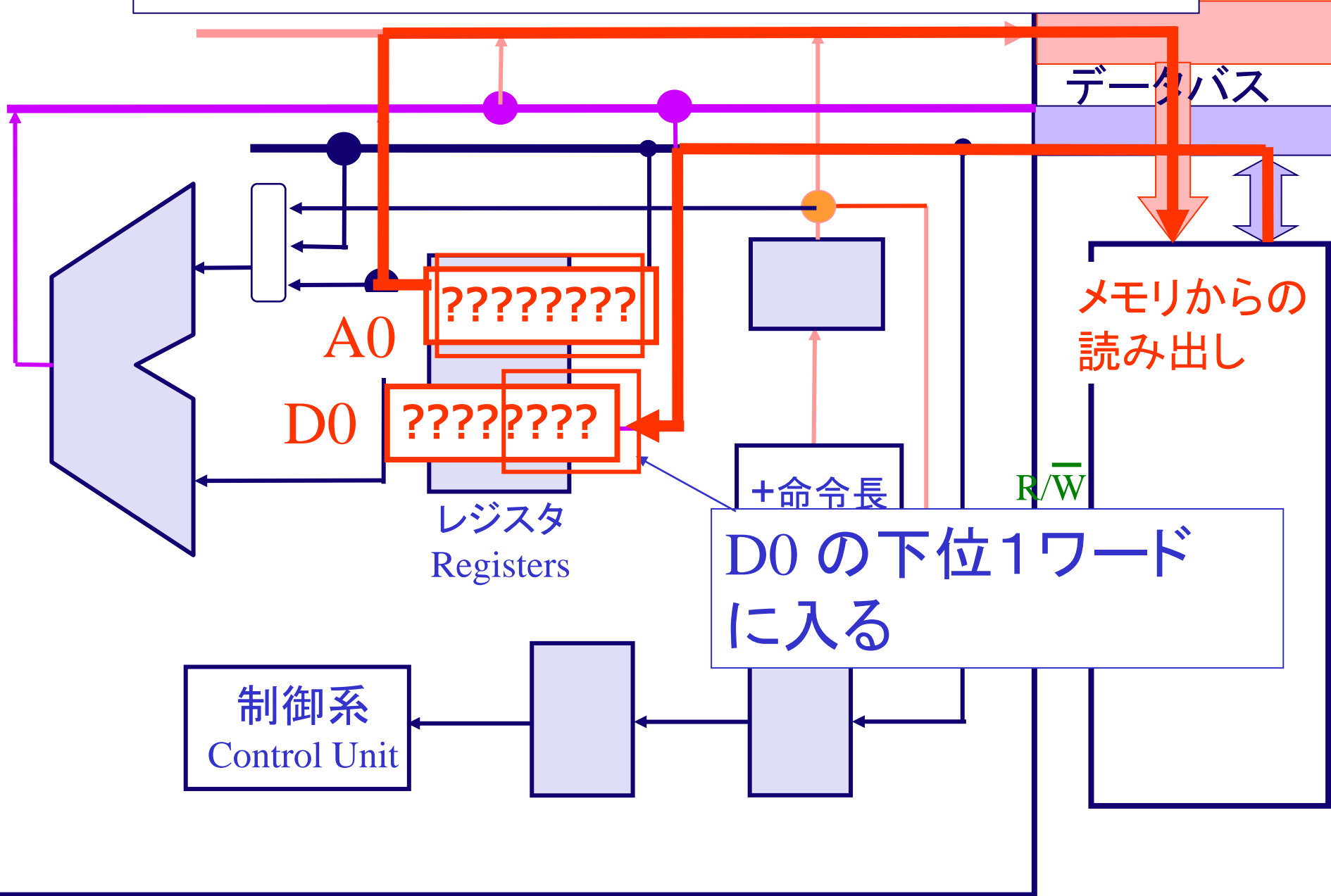
```
move.w (%A0), %D0
```

```
move.w %D3, (%A1)
```

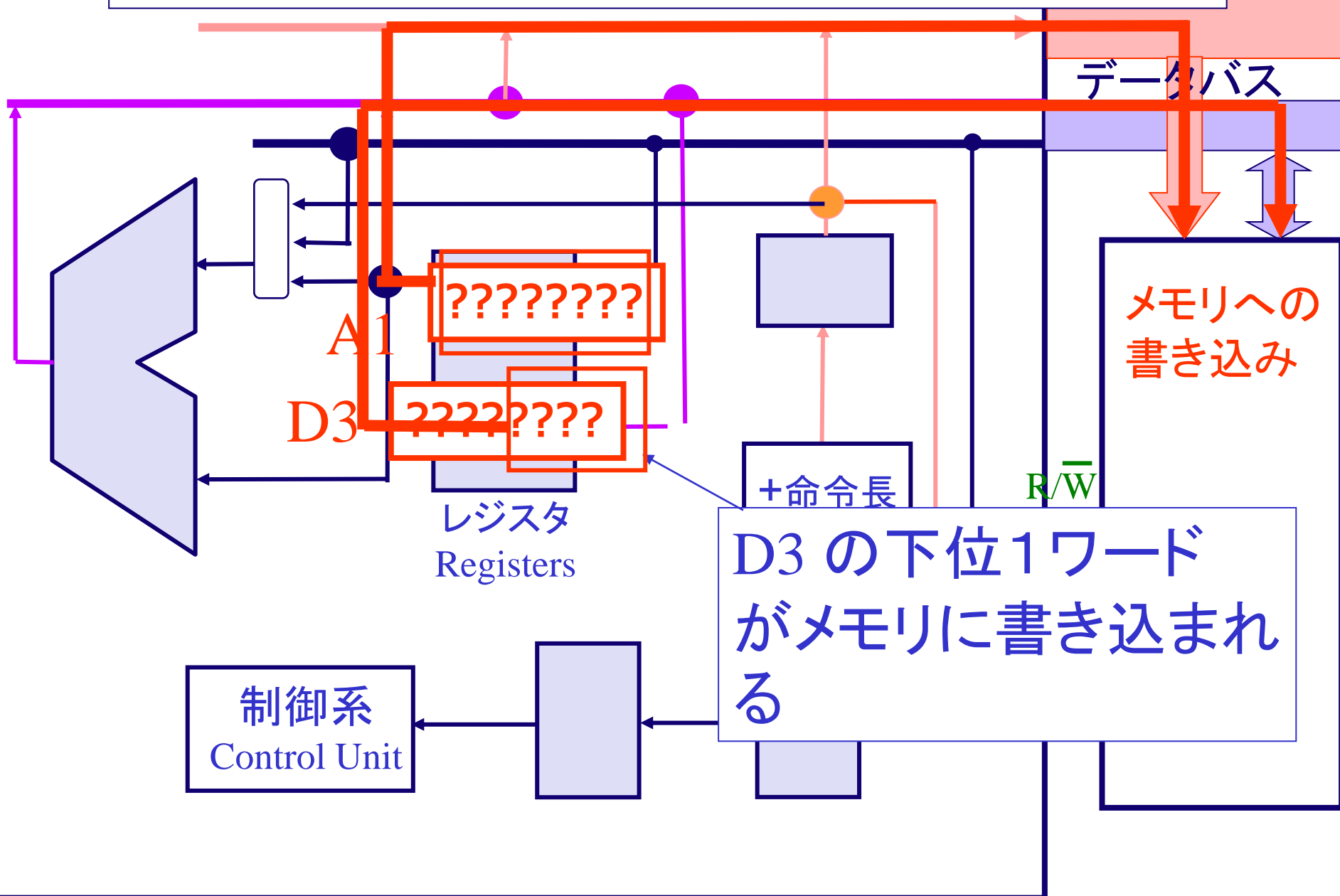
- 記法 (%An)

※ (%A0) や (%A7) は可. (%D0) などは許されない

move.w (%A0), %D0 の命令実行



move.w %D3, (%A1) の命令実行



D3 の下位1ワード
がメモリに書き込まれる

メモリへの
書き込み

R/W

レジスタ
Registers

制御系
Control Unit

+命令長

D3

A1

データバス

レジスタ直接とレジスタ間接の違い

- レジスタ直接

```
move.l    #0, %D0
```

```
move.l    %D2, %D0
```

→ d0 が書き換わる

- レジスタ間接

```
.equ     ADDR      0xffff00
```

```
move.l   #ADDR, %A0
```

```
move.w   %D0, (%A0)
```

→ A0 に入っているメモリアドレスを使って、データの書き込みが行われる

種々のオペランド(その2)

```
move.w  %sr, -(%sp)
```

```
/* システムスタックエリアに sr の中身をプッシュ */
```

```
move.w  (%sp)+, %sr
```

```
/* システムスタックエリアから sr の中身をポップ */
```

```
move.w  #0x0010, 20(%A0)
```

```
/* A0 に 20 足したメモリアドレスに, 0x0010 を書き込む */
```

```
move.w  ※※, -(%sp)
```

```
move.w  (%sp)+, ※※
```

システムスタックエリアへの
プッシュとポップの決まり文句

ポストインクリメント・レジスタ間接

例: `move.w (%a0)+, %d0`

a0 に 0x00002000 が入っているとしよう

「`move.w (%a0), %d0`」と同様の処理が行われた後に、自動的に a0 に2が足される。 a0 の値は、0x00002002 になる

「.b」なら1足される

「.w」なら2足される

「.l」なら4足される

プリデクリメント・レジスタ間接

例: `move.w -(%a0), %d0`

a0 に 0x00002000 が入っているとしよう

最初に、自動的に a0 から2が引かれる。 a0 の値は、0x00001ffe になる。その後で、「move.w (%a0), %d0」と同様の処理が行われる

「.b」なら1引かれる

「.w」なら2引かれる

「.l」なら4引かれる

ディスプレイースメント付きレジスタ間接

例: `move.l 0x20(%a0), %d0`

a0 に 0x00002000 が入っているとしよう

0x00002000 に 0x20 が足されて、メモリアドレス
0x00002020 から、4バイト読み込まれて、d0 に
入る

- 数値(%レジスタ名)

例: `lea 0x2000, %a0` /* a0 に 0x2000 をセット*/

`move.b 4(%a0), %d0` /* 0x2004 から1バイト読みこみ */

`move.w 4(%a0), %d0` /* 0x2004 から2バイト読みこみ */

`move.l 4(%a0), %d0` /* 0x2004 から4バイト読みこみ */

- 構造体

例: 住所録

名前 20バイト

身長 2バイト

年齢 1バイト

性別 1バイト

→ 全体で 24バイトのデータ



`move.w 20(%a0), %d0`

`move.b 22(%a0), %d0`

`move.b 23(%a0), %d0`

68000 のアドレッシングモード

モード		記法	例
データレジスタ直接	データレジスタ	%Dn	%D0, %D1, %D7
アドレスレジスタ直接	アドレスレジスタ	%An	%A0, %A1, %A7
アブソリュート	メモリアドレス	xx	0x00ffffff00, ADDR
レジスタ間接	括弧付き	(%An)	(%A0), (%A1), (%A7)
ポストインクリメント・レジスタ間接	後ろに+付き	(%An)+	(%A0)+, (%A1)+, (%A7)+
プリデクリメント・レジスタ間接	前に-付き	-(%An)	-(%A0), -(%A1), -(%A7)
ディスプレイースメント付きレジスタ間接	前に数値付き	d16(%An)	20(%A0), BOTTOM(%A0)
イミディエート	数値	#data	#3, #0x2000, #ADDR

アドレッシングモード

- 命令の種類によって、書くことができるアドレッシングモードに制限がある

例: `cmp` 命令では

`cmp.w (%a2)+, %d2` は動く

`cmp.w %d2, (%a2)+` は動かない

配布資料「37 CMP」のページを見よ

`CMP.{.B/.W/.L} <ea>, Dn`

とあるのは、2番目のオペランドにデータレジスタしか書けないという意味

C 言語での関数呼び出し

例題1

- 文字列の長さを数える関数

#inc 関数名 .h> 関数の入力(パラメータ)

```
int stringlength( char* const str )
{
    short int len;
    char *s;

    len = 0;
    s = str;
    while ( (*s) != '\0' ) {
        s++;
        len++;
    }
    return len;
}
```

```
int main()
{
    short int n;
    n = stringlength( "My Name is David\n" );
    fprintf( stderr, "n = %d\n", n );
    return 0;
}
```

関数の
本体

例題1. 関数呼び出し(1)

```
#include <stdio.h>
```

```
int stringlength( char* const str )  
{  
    short int len;  
    char *s;  
  
    len = 0;  
    s = str;  
    while ( (*s) != '\0' ) {  
        s++;  
        len++;  
    }  
    return len;  
}
```

1つの関数

関数呼び出し

```
int main()  
{  
    short int n;  
    n = stringlength( "My Name is David\n" );  
    fprintf( stderr, "n = %d\n", n );  
    return 0;  
}
```

1つの関数

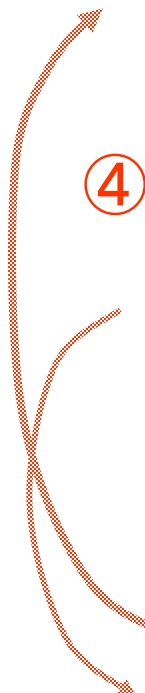
```
#include <stdio.h>

int stringlength( char* const str )
{
    short int len;
    char *s;

    ② len = 0;
    ③ s = str;
    ④ while ( (*s) != '\0' ) {
        s++;
        len++;
    }
    ⑤ return len; 戻り
}
```

プログラム実行は
メイン関数から始まる

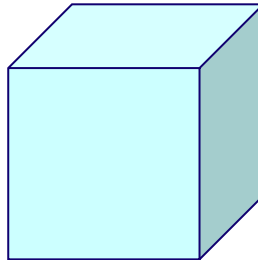
```
int main()
{
    short int n;
    ① n = stringlength( "My Name is David\n" );
    ⑥ fprintf( stderr, "n = %d\n", n );
    ⑦ return 0;
}
```



変数

- 変数 データ(数値や文字)を入れるもの
- 変数名 英数字かアンダーバー(_)で作られる
最初の文字には数字は使えない
大文字と小文字を区別する

変数 i



```
#include <stdio.h>
```

```
int stringlength( char* const str )
```

```
{  
    short int len; } 変数lenをメモリエリア中に確保
```

```
    char *s; } 変数sをメモリエリア中に確保
```

```
    len = 0;
```

```
    s = str;
```

```
    while ( (*s) != '\0' ) {
```

```
        s++;
```

```
        len++;
```

```
    }
```

```
    return len;
```

```
}
```

ここで使用

```
int main()
```

```
{
```

```
    short int n;
```

```
    n = stringlength( "My Name is David\n" );
```

```
    fprintf( stderr, "n = %d\n", n );
```

```
    return 0;
```

```
}
```

```
#include <stdio.h>

int stringlength( char* const str )
{
    short int len;
    char *s;

    len = 0;
    s = str;
    while ( (*s) != '\0' ) {
        s++;
        len++;
    }
    return len;
}

int main()
{
    short int n;
    n = stringlength( "My Name is I" );
    fprintf( stderr, "n = %d\n", n );
    return 0;
}
```

変数は2種類使っている

2バイトデータ(整数)
を扱う short int 型

メモリアドレスを扱う
char * 型

short int が2バイトになる
という決まりは無いが、
2バイトになっていることが多い

#include <stdio.h> 関数の入力(パラメータ)

```
int stringlength( char* const str )
{
    short int len;
    char *s;

    len = 0;
    s = str;
    while ( (*s) != '\0' ) {
        s++;
        len++;
    }
    return len;
}

int main()
{
    short int n;
    n = stringlength( "My Name is I" );
    fprintf( stderr, "n = %d\n", n );
    return 0;
}
```

変数は2種類使っている

2バイトデータ(整数)を扱う short int 型

メモリアドレスを扱う char * 型

short int が2バイトになるという決まりは無いが、2バイトになっていることが多い


```

#include <stdio.h>

int stringlength( char* const str )
{
    short int len;
    char *s;

    len = 0;
    s = str;
    while ( (*s) != '\0' ) {
        s++;
        len++;
    }
    return len;
}

int main()
{
    short int n;
    n = stringlength( "stringlength" );
    fprintf( stderr, "%d\n", n );
    return 0;
}

```

この時点では



str	文字列の先頭 アドレス	char *
s	文字列の先頭 アドレス + 17	char *
len	17	int

実際のメモリの中身

str	文字列の先頭 アドレス	char *
s	文字列の先頭 アドレス + 17	char *
len	17	int

len s

```
      : 00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f | 0123456789abcdef
-----|-----
ffbffad0 : 00 00 00 00 ff 3a 20 00 00 00 00 04 00 00 00 04 | .....t .....
ffbffae0 : ff bf fb 20 00 01 0a 74 ff 3e ed b4 ff bf fa ccl | ... ..t.>.....
ffbffa0 : 00 00 01 00 00 00 00 11 00 00 00 08 00 00 00 04 | .....
ffbffb00 : 00 00 00 11 ff ff ff ff ff ff 00 11 00 01 0c 71 | .....q
ffbffb10 : 00 00 03 00 00 00 00 00 00 00 00 00 00 00 00 00 | .....
ffbffb20 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | .....
ffbffb30 : ff 3a 20 00 ff 36 42 24 ff 36 9b 00 ff 3e c9 64 | : ..6B$.6...>.d
ffbffb40 : 00 00 00 01 ff bf fb fc ff bf fc 04 00 02 0f 70 | .....P
ffbffb50 : ff 3a 00 c0 ff 3a 01 00 ff bf fb 98 00 01 05 c4 | .....
ffbffb60 : 00 00 00 00 00 01 0c 60 00 00 00 00 00 00 00 00 | .....
ffbffb70 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | .....
ffbffb80 : 00 00 00 04 ff bf fc 04 00 00 00 00 00 00 00 00 | .....
ffbffb90 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | .....
ffbffb
```

str

len と s があるメモリアリアは、関数 stringlength 実行のために
ダイナミックに確保されたエリア

(参考) 実際のメモリの中身

```
#include <stdio.h>

int stringlength
{
    short int len;
    char *s;

    len = 0;
    s = str;
    while ( (*s) != '\0' )
        s++;
        len++;
    }
    return len;
}

int main()
{
    short int n;
    n = stringlength( "My Name is David\n" );
    fprintf( stderr, "n = %d\n", n );
    return 0;
}
```

10c50	:	6c 65 6e 20 3d 20 25 64 0a 00 00 00 00 00 00 00	len = %d.....
10c60	:	4d 79 20 4e 61 6d 65 20 69 73 20 44 61 76 69 64	My Name is David
10c70	:	0a 00 00 00 00 02 0d 30 00 02 0e 04 00 02 0e 140.....
10c80	:	00 02 0e 10 00 02 0e 0c 00 02 0e 08 00 02 0d f8
10c90	:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
10ca0	:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
10cb0	:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
10cc0	:	00 00 00 00 00 00 00 00 00 00 00 00 03 00 00 300
10cd0	:	30 bf ff f3 01 00 00 00 03 00 00 3c 30 bf ff f0	0.....<0...
10ce0	:	01 00 00 00 03 00 00 48 30 bf ff ed 01 00 00 00H0.....
10cf0	:	03 00 00 54 30 bf ff ea 01 00 00 00 03 00 00 60	...T0.....
10d00	:	30 bf ff e7 01 00 00 00 03 00 00 6c 30 bf ff e4	0.....10...
10d10	:	01 00 00 00 03 00 00 78 30 bf ff e1 01 00 00 00x0.....
10d20	:	03 00 00 84 30 bf ff de 01 00 00 00 01 00 00 00	...0.....
10d30	:	00 00 00 01 00 00 01 01 00 00 00 0c 00 01 0a f4
10d40	:	00 00 00 0d 00 01 0b 10 00 00 00 04 00 01 00 e8

文字列データは, len, s などとは別のメモリエリアに入っている

関数呼び出し

1. 制御の流れ
2. 関数のパラメータ
3. 関数内のローカル変数
4. 関数からの返り値

68000 アセンブラ言語での 関数呼び出し

C言語

```
#include <stdio.h>

int stringlength( char* const str )
{
    short int len;
    char *s;

    len = 0;
    s = str;
    while ( (*s) != '\0' ) {
        s++;
        len++;
    }
    return len;
}

int main()
{
    short int n;
    n = stringlength( "My Name is David\n" );
    fprintf( stderr, "n = %d\n", n );
    return 0;
}
```

等価

68000アセンブラ言語

```
.data
str1:
    .ascii "My Name is David!¥0"
SYS_STK:
    .ds.b 0x4000
SYS_STK_TOP:

.text
stringlength:
    link.w %a6, #-8
    clr.w -2(%a6)
    move.l 8(%a6), -6(%a6)
start1:
    move.l -6(%a6), %a0
    cmp.b #0, (%a0)
    beq break1
    addq.l #1, -6(%a6)
    addq.w #1, -2(%a6)
    bra start1
break1:
    move.w -2(%a6), %a0
    move.l %a0, %d0
    unlk %a6
    rts

main:
    lea.l SYS_STK_TOP, %a7

    lea.l str1, %a0
    move.l %a0, -(%a7)
    jsr stringlength
    addq.l #4, %a7

    .dc.w 0x4848
    stop #0

.end
```

C言語

```
#include <stdio.h>

int stringlength( char* const str )
{
    short int len;
    char *s;

    len = 0;
    s = str;
    while ( (*s) != '\0' ) {
        s++;
        len++;
    }
    return len;
}

int main()
{
    short int n;
    n = stringlength( "My Name is David\n" );
    fprintf( stderr, "n = %d\n", n );
    return 0;
}
```

関数呼び出し

(文字列の先頭アドレスを関数に渡す)

68000アセンブラ言語

```
.data
str1:
    .ascii "My Name is David!¥0"
SYS_STK:
    .ds.b 0x4000
SYS_STK_TOP:

.text
stringlength:
    link.w %a6, #-8
    clr.w -2(%a6)
    move.l 8(%a6), -6(%a6)
start1:
    move.l -6(%a6), %a0
    cmp.b #0, (%a0)
    beq break1

    unlk %a6
    rts

main:
    lea.l SYS_STK_TOP, %a7

    lea.l str1, %a0
    move.l %a0, -(%a7)
    jsr stringlength
    addq.l #4, %a7

    .dc.w 0x4848
    stop #0

.end
```

C言語

```
#include <stdio.h>

int stringlength( char* const str )
{
    short int len;
    char *s;

    len = 0;
    s = str;
    while ( (*s) != '\0' ) {
        s++;
        len++;
    }
    return len;
}

int main()
{
    short int n;
    n = stringlength( "My Name is David\n" );
    fprintf( stderr, "n = %d\n", n );
    return 0;
}
```

リターン

68000アセンブラ言語

```
.data
str1:
    .ascii "My Name is David!¥0"
SYS_STK:
    .ds.b 0x4000
SYS_STK_TOP:

.text
stringlength:
    link.w %a6, #-8
    clr.w -2(%a6)
    move.l 8(%a6), -6(%a6)
start1:
    move.l -6(%a6), %a0
    cmp.b #0, (%a0)
    beq break1
    addq.l #1, -6(%a6)
    addq.w #1, -2(%a6)
    bra start1

break1:
    move.w -2(%a6), %a0
    move.l %a0, %d0
    unlk %a6
    rts

main:
    lea.l SYS_STK_TOP, %a7

    lea.l str1, %a0
    move.l %a0, -(%a7)
    jsr stringlength
    addq.l #4, %a7

    .dc.w 0x4848
    stop #0

.end
```


C言語

```
#include <stdio.h>

int stringlength( char* const str )
{
    short int len;
    char *s;

    len = 0;
    s = str;
    while ( (*s) != '\0' )
        s++;
        len++;
}

return len;
```

```
int main()
{
    short int n;
    n = stringlength( "My Name is David\n" );
    fprintf( stderr, "n = %d\n", n );
    return 0;
}
```

関数での処理結果を、
呼び出し側に引き渡す

68000アセンブラ言語

```
.data
str1:
    .ascii "My Name is David!¥0"
SYS_STK:
    .ds.b 0x4000
SYS_STK_TOP:

.text
stringlength:
```

```
bra start1
break1:
    move.w -2(%a6),%a0
    move.l %a0,%d0
    unlk %a6
    rts

main:
    lea.l SYS_STK_TOP,%a7
    lea.l str1,%a0
    move.l %a0,-(%a7)
    jsr stringlength
    addq.l #4,%a7

    .dc.w 0x4848
    stop #0

.end
```

return len;

move.w -2(%a6),%a0
move.l %a0,%d0
unlk %a6
rts

lea.l str1,%a0
move.l %a0,-(%a7)
jsr stringlength
addq.l #4,%a7

n = stringlength("My Name is David\n");

C言語

```
#include <stdio.h>

int strlenlength( char* const str )
{
    short int len;
    char *s;

    len = 0;
    s = str;
    while ( (*s) != '\0' ) {
        s++;
        len++;
    }
    return len;
}

int main()
{
    short int n;
    n = strlenlength( "My Name is David" );
    fprintf( stdout, "%d\n", n );
    return 0;
}
```

68000アセンブラ言語

```
.data
str1:
    .ascii "My Name is David!¥0"
SYS_STK:
    .ds.b 0x4000
SYS_STK_TOP:

.text
strlenlength:
    link.w %a6, #-8
    clr.w -2(%a6)
    move.l 8(%a6), -6(%a6)
start1:
    move.l -6(%a6), %a0
    cmp.b #0, (%a0)
    beq break1
    addq.l #1, -6(%a6)
    addq.w #1, -2(%a6)
    bra start1
break1:
    move.w -2(%a6), %a0
    move.l %a0, %d0
    unlk %a6
    rts
```

関数実行のために、メモリ
エリアをダイナミックに確保
(関数実行の終わりで解放)

short int len;
char *s;

link.w %a6, #-8

return len;

move.w -2(%a6), %a0
move.l %a0, %d0
unlk %a6
rts

C言語

```
#include <stdio.h>

int stringlength( char* const str )
{
    short int len;
    char *s;

    len = 0;
    s = str;
    while ( (*s) != '\0' ) {
        s++;
        len++;
    }
    return len;
}

int main()
{
    short int n;
    n = stringlength( "My Name is David\n" );
    fprintf( stderr, "n = %d\n", n );
    return 0;
}
```

68000アセンブラ言語

```
.data
str1:
    .ascii "My Name is David!¥0"
SYS_STK:
    .ds.b 0x4000
SYS_STK_TOP:

.text
stringlength:
    link.w %a6, #-8
    clr.w -2(%a6)
    move.l 8(%a6), -6(%a6)
start1:
    move.l -6(%a6), %a0
    cmp.b #0, (%a0)
    beq break1
    addq.l #1, -6(%a6)
    addq.w #1, -2(%a6)
    bra start1
break1:
    move.w -2(%a6), %a0
    move.l %a0, %d0
    unlk %a6
    rts

main:
    lea.l SYS_STK_TOP, %a7

.end
```

「スタックエリア」の確保と、
そのメモリアドレスを A7 にセット

C言語

```
#include <stdio.h>

int stringlength( char* const str )
{
    short int len;
    char *s;

    len = 0;
    s = str;
    while ( (*s) != '\0' ) {
        s++;
        len++;
    }
    return len;
}

int main()
{
    short int n;
    n = stringlength( "My Name is David\n" );
    fprintf( stderr, "n = %d\n", n );
    return 0;
}
```

68000アセンブラ言語

```
.data
str1:
    .ascii "My Name is David!¥0"
SYS_STK:
    .ds.b 0x4000
SYS_STK_TOP:

.text
stringlength:
    link.w %a6, #-8
    clr.w -2(%a6)
    move.l 8(%a6), -6(%a6)
start1:
    move.l -6(%a6), %a0
    cmp.b #0, (%a0)
    beq break1
    addq.l #1, -6(%a6)
    addq.w #1, -2(%a6)
    bra start1
break1:
    move.w -2(%a6), %a0
    move.l %a0, %d0
    unlk %a6
    rts

main:
    lea.l SYS_STK_TOP, %a7

    lea.l str1, %a0
    move.l %a0, -(%a7)
    jsr stringlength
    addq.l #4, %a7

    .dc.w 0x4848
    stop #0

.end
```

short int len;
char *s;

len = 0;
s = str;
while ((*s) != '\0') {
 s++;
 len++;
}

return len;

int main()
{
 short int n;
 n = stringlength("My Name is David\n");
 fprintf(stderr, "n = %d\n", n);
 return 0;
}

link.w %a6, #-8

clr.w -2(%a6)
move.l 8(%a6), -6(%a6)
start1:
move.l -6(%a6), %a0
cmp.b #0, (%a0)
beq break1
addq.l #1, -6(%a6)
addq.w #1, -2(%a6)
bra start1

break1:
move.w -2(%a6), %a0
move.l %a0, %d0
unlk %a6
rts

main:
lea.l SYS_STK_TOP, %a7

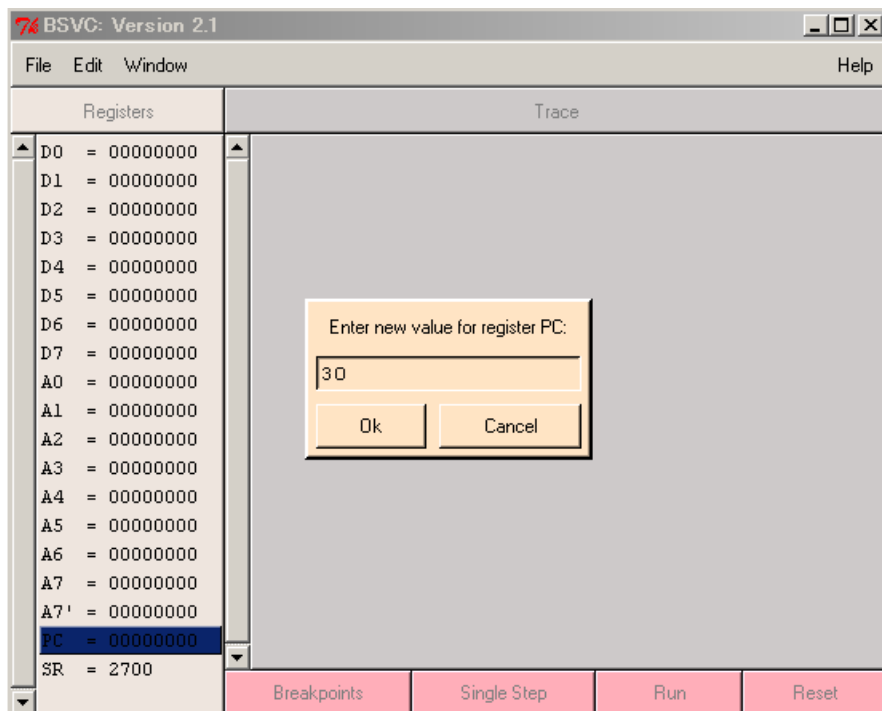
lea.l str1, %a0
move.l %a0, -(%a7)
jsr stringlength
addq.l #4, %a7

.dc.w 0x4848
stop #0

.end

(参考) BSVC での実行

最初に PC の値を手動でセット
(メイン関数から開始するように)



```
9
10 .text
11 stringlength:
12     link.w %a6,#-8
13     /* -2(%a6) = 'len' (lo
14     /* -6(%a6) = 's' (loca
15     /* 8(%a6) = 'str' (pa
000000 4E56 FFF8
16     clr.w -2(%a6)
17     move.l 8(%a6),-6(%a6)
17
18 start1:
00000e 206E FFFA     19     move.l -6(%a6),%a0
000012 0C10 0000     20     cmp.b #0,(%a0)
000016 6700 000E     21     beq break1
00001a 52AE FFFA     22     addq.l #1,-6(%a6)
00001e 526E FFFE     23     addq.w #1,-2(%a6)
000022 6000 FFEA     24     bra start1
25 break1:
000026 306E FFFE     26     move.w -2(%a6),%a0
00002a 2008         27     move.l %a0,%d0
00002c 4E5E         28     unlk %a6
00002e 4E75         29     rts
30
31 main:
000030 4FF9 0000     32     lea.l SYS STK TOP,%a7
        40A8         32
33
000036 41F9 0000     34     lea.l str1,%a0
        0096         34
00003c 2F08         35     move.l %a0,-(%a7)
00003e 4EBA FFC0     36     jsr stringlength
000042 588F         37     addq.l #4,%a7
38
000044 4848         39     .dc.w 0x4848
000046 4E72 0000     40     stop #0
41 .end
```

(1) システムスタックエリアの
確保と, A7 へのセット

```
.data
str1:
    .ascii "My Name is David!¥0"
SYS_STK:
    .ds.b 0x4000
SYS_STK_TOP:

.text
stringlength:
    link.w %a6, #-8
    clr.w -2(%a6)
    move.l 8(%a6), -6(%a6)

start1:
    move.l -6(%a6), %a0
    cmp.b #0, (%a0)
    beq break1
    addq.l #1, -6(%a6)
    addq.w #1, -2(%a6)
    bra start1

break1:
    move.w -2(%a6), %a0
    move.l %a0, %d0
    unlk %a6
    rts

main:
    lea.l SYS_STK_TOP, %a7

    lea.l str1, %a0
    move.l %a0, -(%a7)
    jsr stringlength
    addq.l #4, %a7

    .dc.w 0x4848
    stop #0

.end
```

0x4000 バイト (16進数)
のメモリエリア確保

例えば、0x0000005e ~ 0x00000405e

この場合ラベル
SYS_STK_TOP
は 0x0000405e を指す

A7 に 0x0000405e
をセット

★ A7 は「スタックポインタ」のこと
(CPU内のレジスタ)である。

A7に 0x0000405e
をセット

メモリ

システム
スタックエリア

0x4000 バイトの
メモリエリア
0x0000005e
~
0x00000405e

A7のポイント先 →

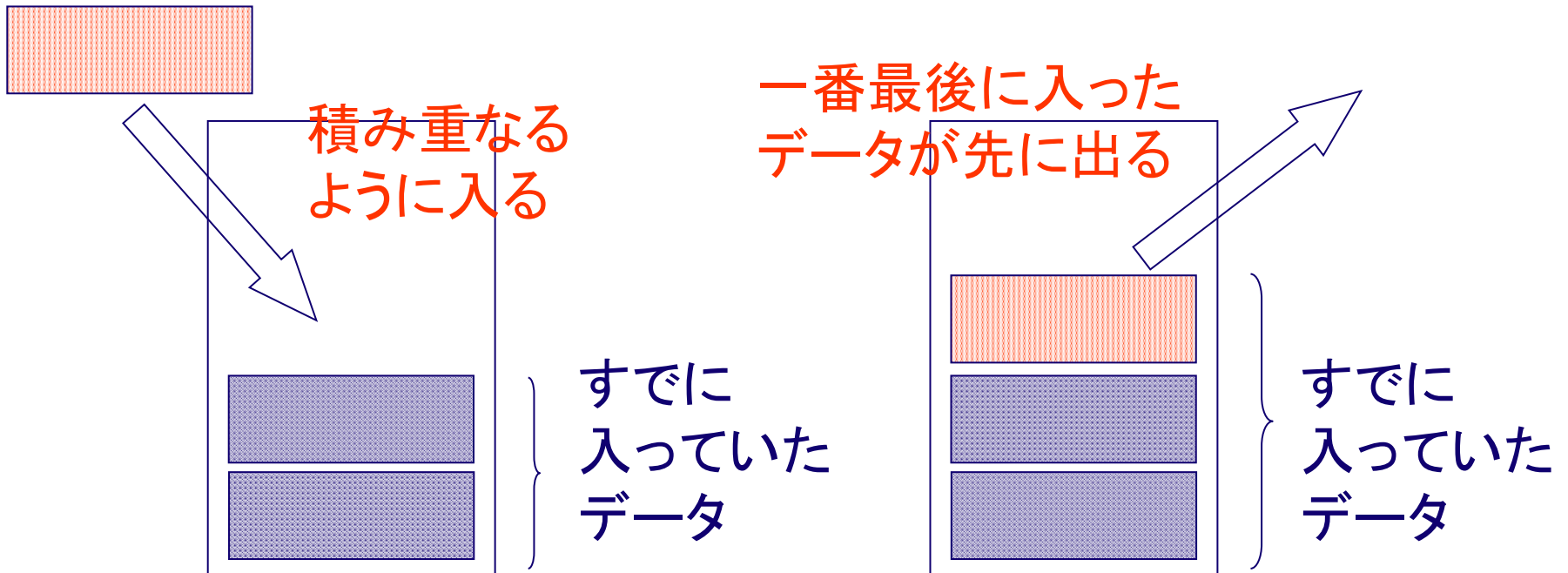
プログラム実行の最初の時点で、
A7に「システムスタックエリア」の末尾+1
のメモリアドレスをセットしておく

(2) 関数のパラメータを, システムスタックエリアにプッシュ
(push)

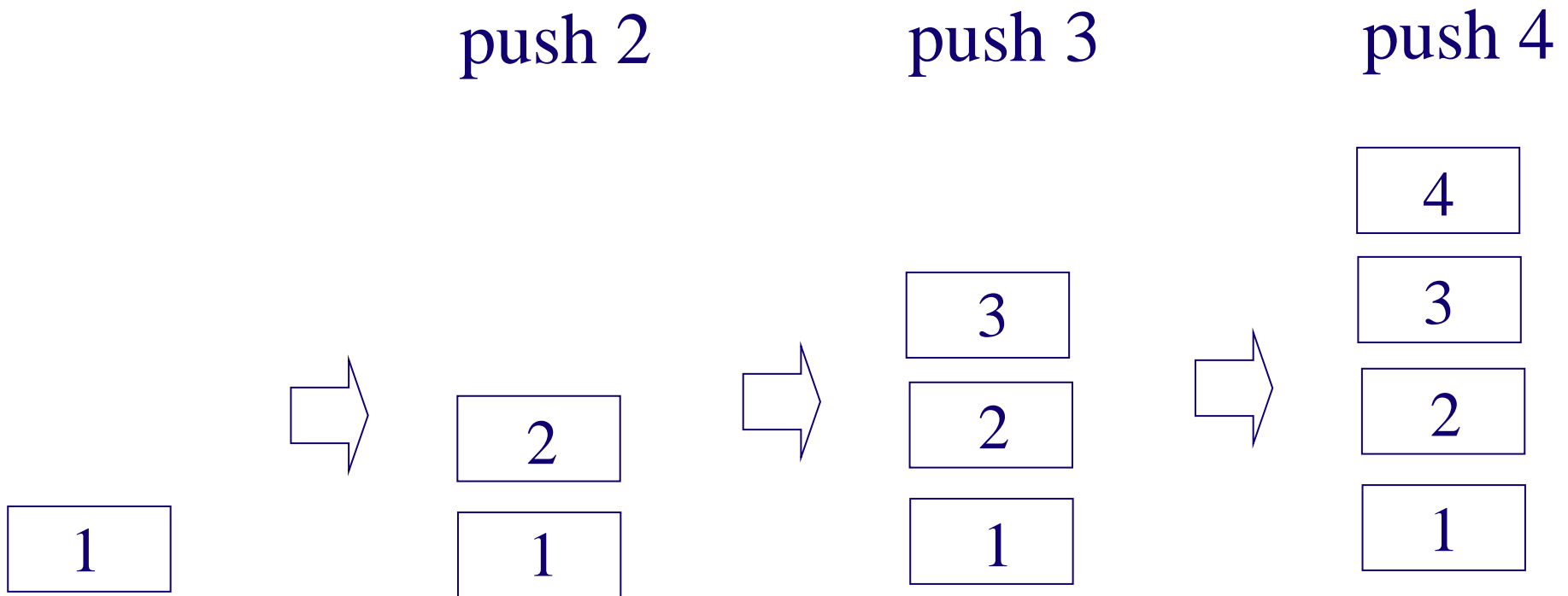
スタック

- プッシュ (push)

- ポップ (pop)



スタック



スタックとキュー

- スタックは、データの後入れ・先出し(last in first out)を行う
 - スタックの push 1, push 2, push 3, pop, pop では、1番目のpopで3が、2番目のpopで2が出て、1は残っている.
- キューは、データの先入れ・先出しを行う.

システムスタックエリアに
「4バイト」のデータを push

例) `move.l %a0, -(%a7)`

メモリ

push 前

push 後

システム
スタックエリア

A0 の中身
(4バイトデータ)

A7の
ポイント先

A7の
ポイント先
(空なので「全体の末尾」
+1をポイントしている)

説明上、
最初は空
とする

4減る

(3) 関数呼び出しとリターン

```

.data
str1:
    .ascii "My Name is David!¥0"
SYS_STK:
    .ds.b 0x4000
SYS_STK_TOP:

.text
stringlength:
    link.w %a6, #-8
    clr.w -2(%a6)
    move.l 8(%a6), -6(%a6)

start1:
    move.l -6(%a6), %a0
    cmp.b #0, (%a0)
    beq break1
    addq.l #1, -6(%a6)
    addq.w #1, -2(%a6)
    bra start1

break1:
    move.w -2(%a6), %a0
    move.l %a0, %d0
    unlk %a6
    rts

main:
    lea.l SYS_STK_TOP, %a7

    lea.l str1, %a0
    move.l %a0, -(%a7)
    jsr stringlength
    addq.l #4, %a7

    .dc.w 0x4848
    stop #0

.end

```

jsr stringlength

サブルーチン呼び出し

(jsr = jump subroutine)

分岐が行われるとともに、
「戻り番地」が**保存**される

保存先

⇒システムスタックエリア

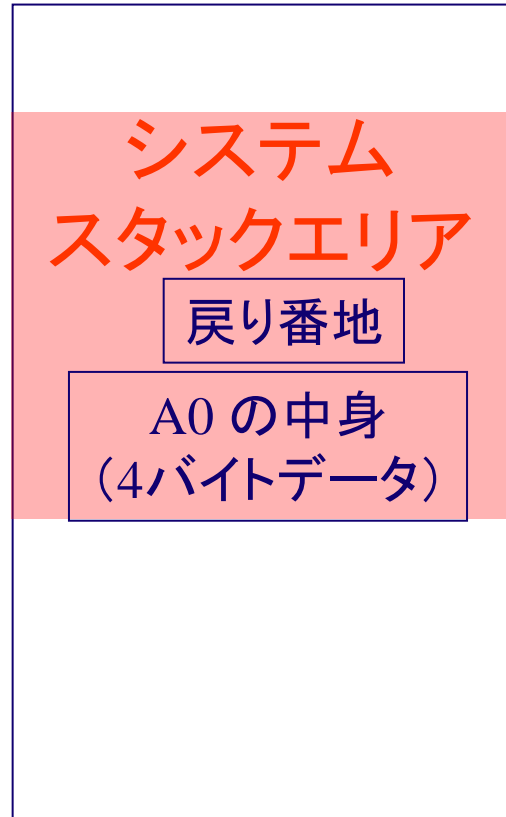
システムスタックエリアに 戻り番地を push

例) `jsr stringlength`

push 前

メモリ

push 後



A7の
ポイント先 →
(空なので「全体の末尾」
+1をポイントしている)

← A7の
ポイント先

4減る

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

命令が届く

プログラムカウンタ
Program Counter

+命令長

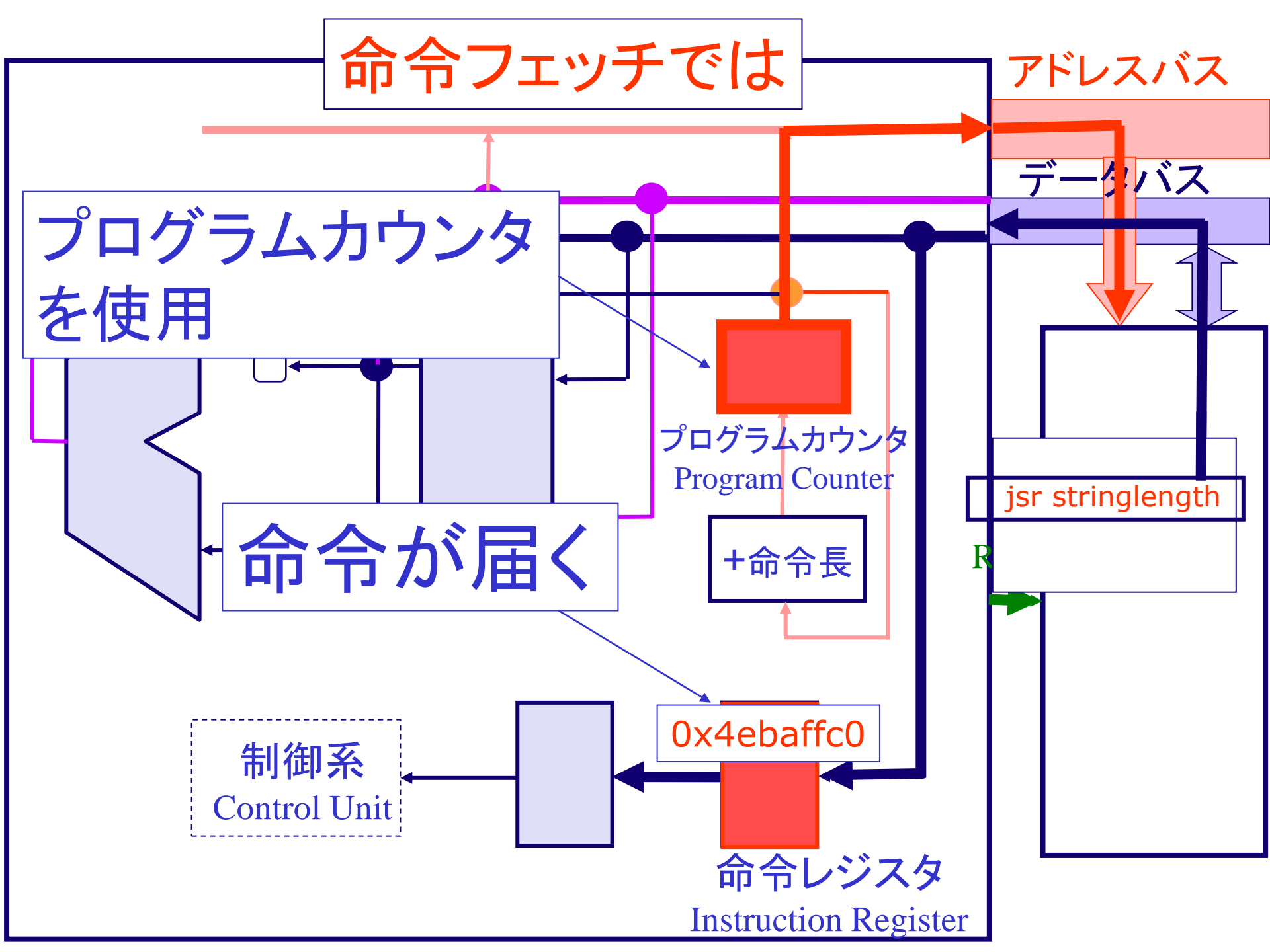
jsr stringlength

R

制御系
Control Unit

0x4ebaffc0

命令レジスタ
Instruction Register



命令フェッチでは

アドレスバス

データバス

ffc0 は、
分岐先のメモリアドレス
を表している

+命令長

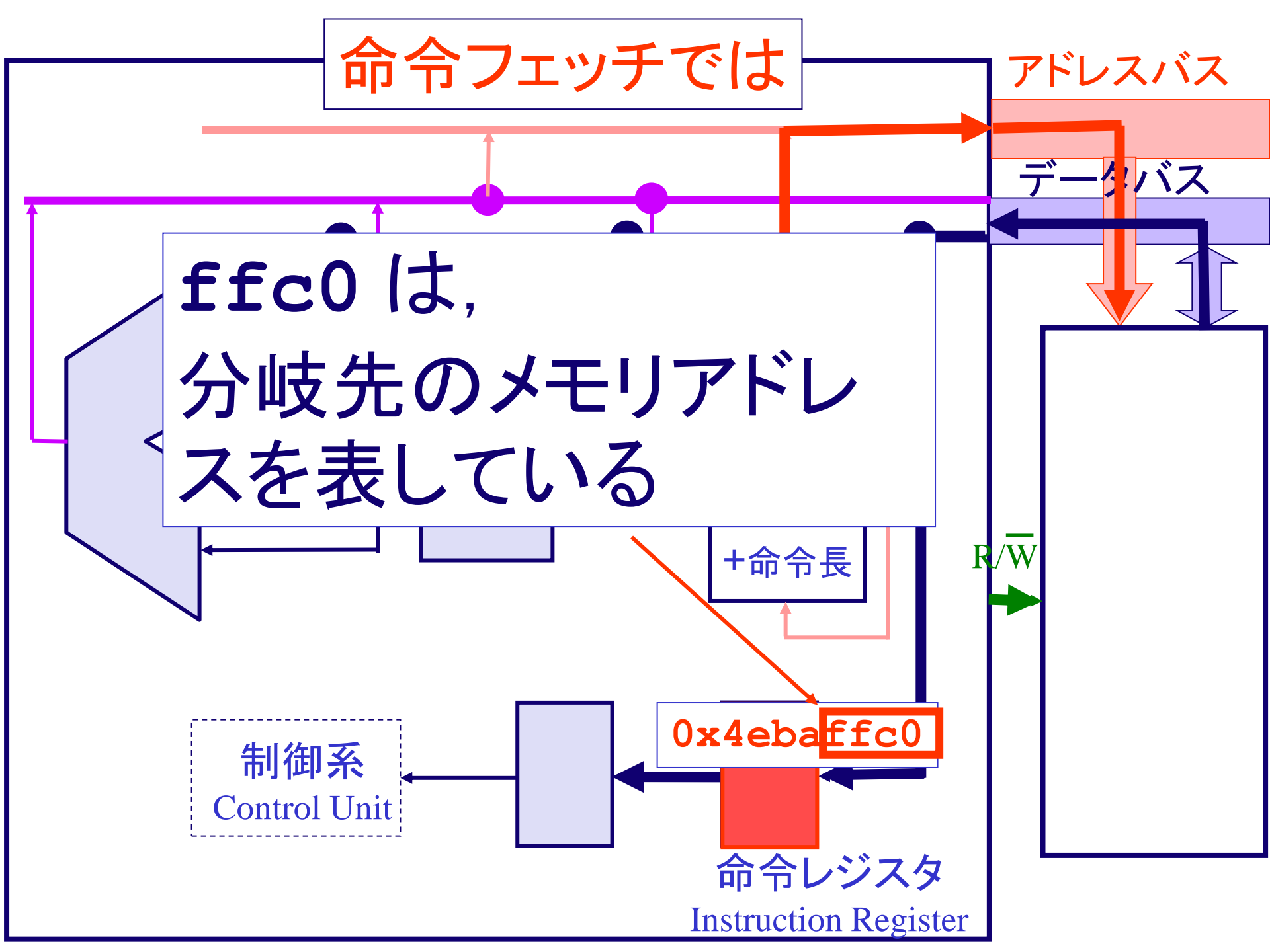
0x4ebaffc0

制御系
Control Unit

命令レジスタ

Instruction Register

R/ \bar{W}



命令デコードでは

アドレスバス

データバス

まず、プログラムカウンタが「jsr stringlength」の次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

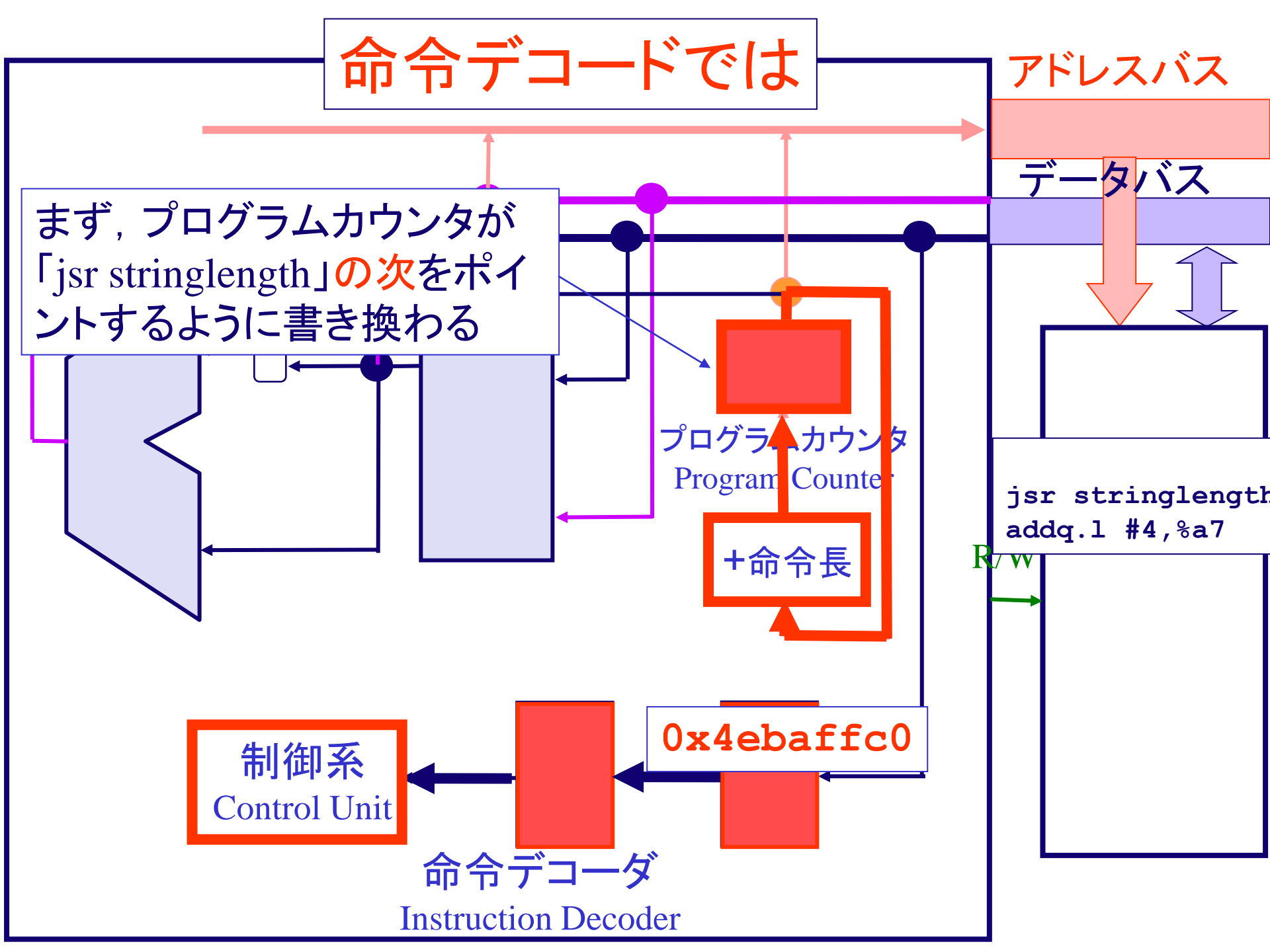
```
jsr stringlength  
addq.1 #4,%a7
```

R/w

制御系
Control Unit

命令デコーダ
Instruction Decoder

0x4ebaffc0



命令実行では (1/2)

現在のプログラム
カウンタの値
(= 戻り番地) をシ
ステムスタックエリ
アに push
(A7 が4減る)

プログラムカウンタ
Program Counter

+命令長

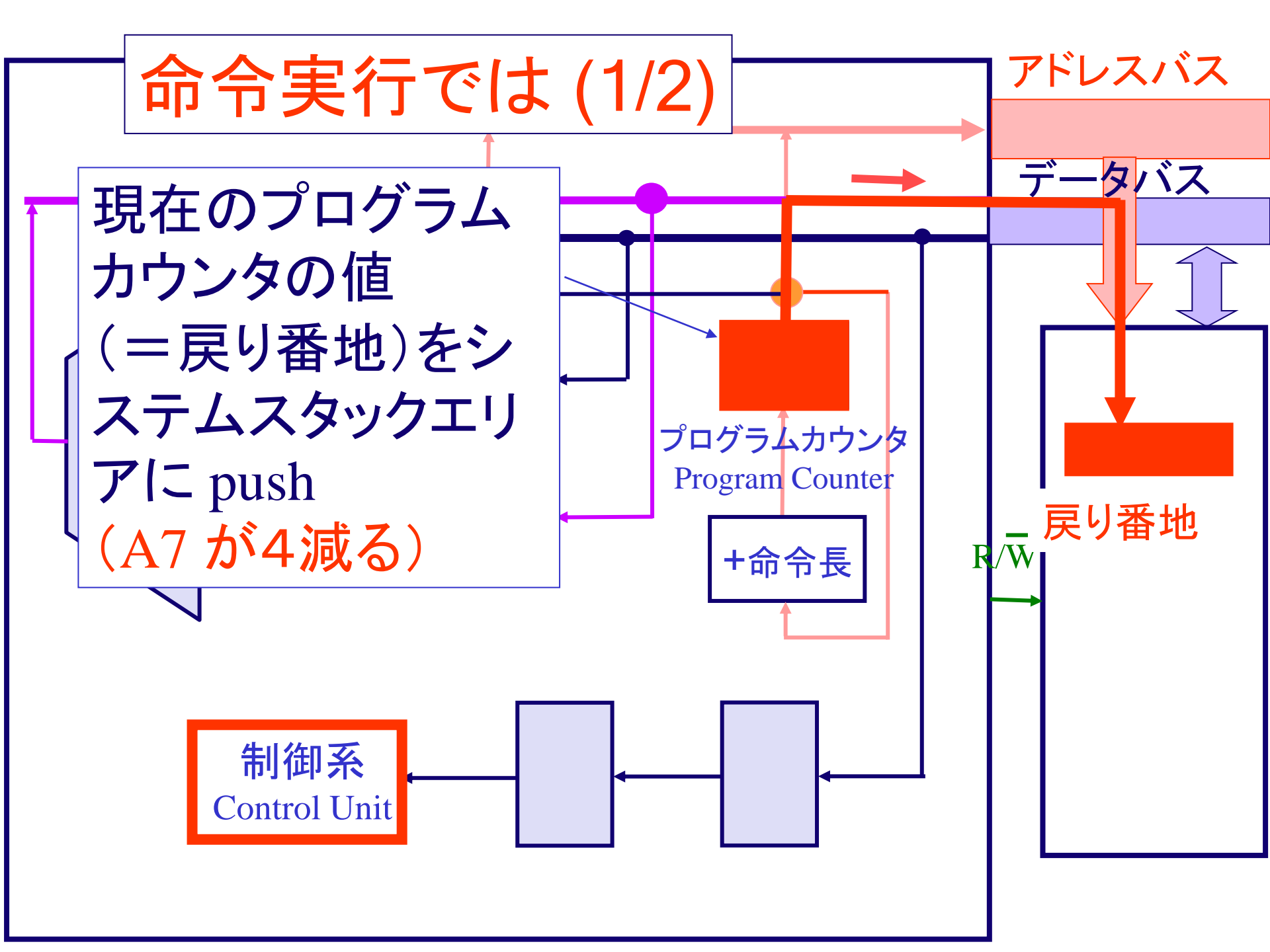
制御系
Control Unit

アドレスバス

データバス

戻り番地

R/W



命令実行では (2/2)

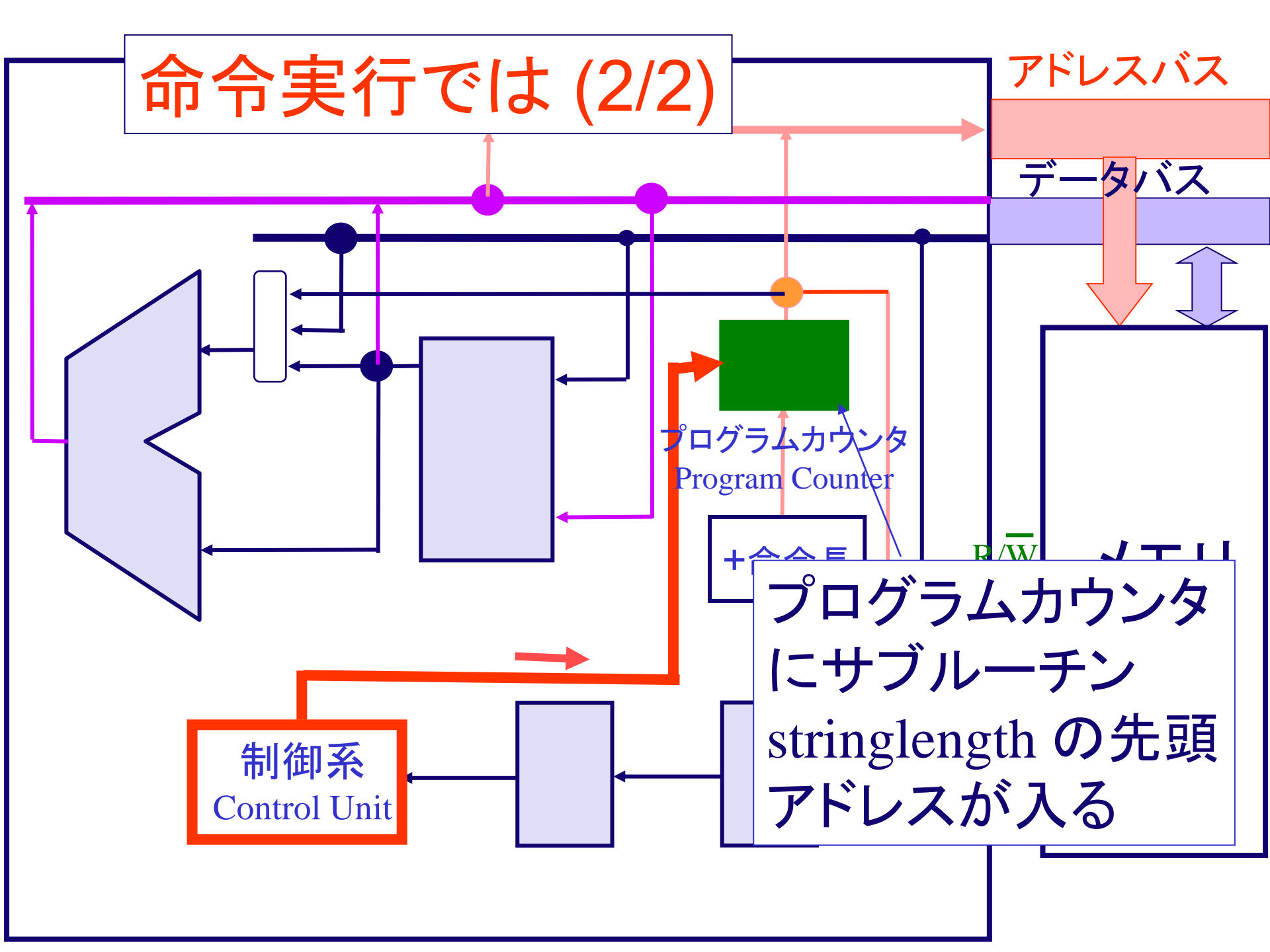
アドレスバス

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
にサブルーチン
stringlength の先頭
アドレスが入る



```
.data
str1:
    .ascii "My Name is David!¥0"
SYS_STK:
    .ds.b 0x4000
SYS_STK_TOP:

.text
stringlength:
    link.w %a6, #-8
    clr.w -2(%a6)
    move.l 8(%a6), -6(%a6)

start1:
    move.l -6(%a6), %a0
    cmp.b #0, (%a0)
    beq break1
    addq.l #1, -6(%a6)
    addq.w #1, -2(%a6)
    bra start1

break1:
    move.w -2(%a6), %a0
    move.l %a0, %d0
    unlk %a6
    rts

main:
    lea.l SYS_STK_TOP, %a0
    lea.l str1, %a0
    move.l %a0, -(%a7)
    jsr stringlength
    addq.l #4, %a7

    .dc.w 0x4848
    stop #0

.end
```

rts

(rts = return subroutine)

システムスタックエリア
から4バイト pop し、
プログラムカウンタに上書き

rts では、戻り先がプログラム中のどこにも書かれていない。戻り先は、ダイナミックに保存されるから。

rts の命令実行では

システムスタックエリアから4バイト pop され、プログラムカウンタに上書き.
(このとき A7 が4増える)

アドレスバス

データバス

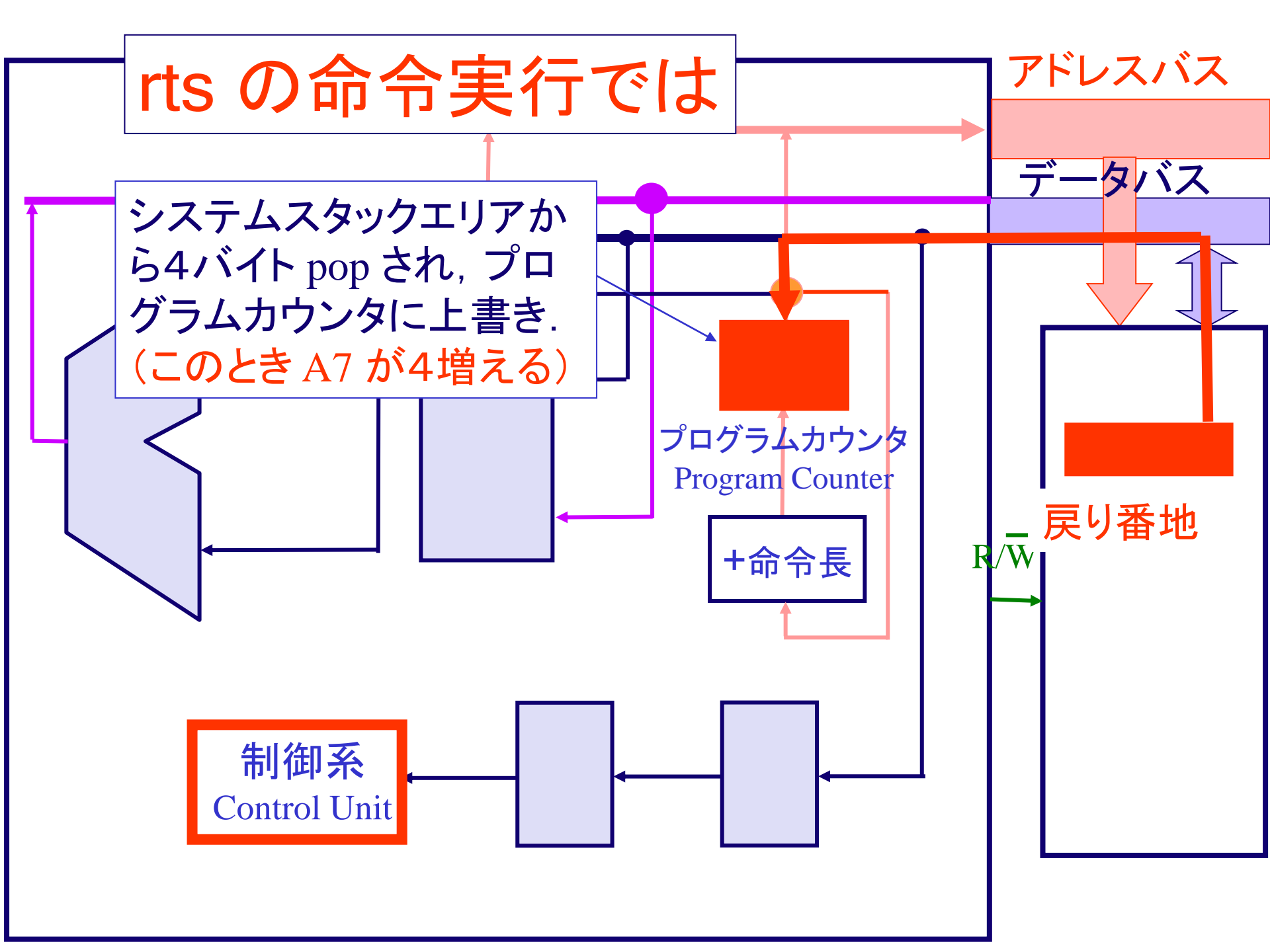
プログラムカウンタ
Program Counter

+命令長

戻り番地

制御系
Control Unit

R/W



(4) 関数実行の始めに, メモリ
エリアをダイナミックに確保
(終わりで解放)

```

.data
str1:
    .ascii "My Name is David!¥0"
SYS_STK:
    .ds.b 0x4000
SYS_STK_TOP:

.text
stringlength:
    link.w %a6, #-8
    clr.w -2(%a6)
    move.l 8(%a6), -6(%a6)
start1:
    move.l -6(%a6), %a0
    cmp.b #0, (%a0)
    beq break1
    addq.l #1, -6(%a6)
    addq.w #1, -2(%a6)
    bra start1
break1:
    move.w -2(%a6), %a0
    move.l %a0, %d0
    unlk %a6
    rts

main:
    lea.l SYS_STK_TOP, %a7

    lea.l str1, %a0
    move.l %a0, -(%a7)
    jsr stringlength
    addq.l #4, %a7

    .dc.w 0x4848
    stop #0

.end

```

確保

システムスタックエリア内に
8バイトを確保せよ

解放

```

.data
str1:
    .ascii "My Name is David!¥0"
SYS_STK:
    .ds.b 0x4000
SYS_STK_TOP:

.text
stringlength:
    link.w %a6, #-8
    clr.w -2(%a6)
    move.l 8(%a6), -6(%a6)
start1:
    move.l -6(%a6), %a0
    cmp.b #0, (%a0)
    beq break1
    addq.l #1, -6(%a6)
    addq.w #1, -2(%a6)
    bra start1
break1:
    move.w -2(%a6), %a0
    move.l %a0, %d0
    unlk %a6
    rts

main:
    lea.l SYS_STK_TOP, %a7

    lea.l str1, %a0
    move.l %a0, -(%a7)
    jsr stringlength
    addq.l #4, %a7

    .dc.w 0x4848
    stop #0

.end

```

確保

- ① A6 をシステムスタックエリアに push (A6 の保存)
- ② A6 ← A7
(A7 の現時点の値の記録)
- ③ A7 ← A7 - 8
(メモリエリアの確保)

解放

- ① A7 ← A6
- ② システムスタックエリアから pop して, A6 に入れる
(A6, A7 が元に戻る)

```

.text
stringlength:
    link.w %a6, #-8
    clr.w -2(%a6)
    move.l 8(%a6), -6(%a6)
start1:
    move.l -6(%a6), %a0
    cmp.b #0, (%a0)
    beq break1
    addq.l #1, -6(%a6)
    addq.w #1, -2(%a6)
    bra start1
break1:
    move.w -2(%a6), %a0
    move.l %a0, %d0
    unlk %a6
    rts

```

確保

- ① A6 をシステムスタックエリア
に push (A6 の保存)
- ② A6 ← A7
(A7 の現時点の値の記録)
- ③ A7 ← A7 - 8
(メモリエリアの確保)

link 前

A6: 0x00000000
A7: 0x00004056

システムスタックエリアの中身(一部)

link 後

A6: 0x00004052
A7: 0x0000404a

```

004000: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004010: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004040: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004050: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

```

```

004000: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004010: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004040: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004050: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

```

戻り番地 パラメータ

旧A6値

メモリエリア

確保

```
.text
stringlength:
    link.w %a6, #-8
    clr.w -2(%a6)
    move.l 8(%a6), -6(%a6)
start1:
    move.l -6(%a6), %a0
    cmp.b #0, (%a0)
    beq break1
    addq.l #1, -6(%a6)
    addq.w #1, -2(%a6)
    bra start1
break1:
    move.w -2(%a6), %a0
    move.l %a0, %d0
    unlk %a6
    rts
```

メモリエリア内の2つのデータ

1. 2バイトデータ
(長さを数える)
-2(%a6)

2. メモリアドレス
(文字列データの各文字を
ポイントする)
-6(%a6)

link 後

A6: 0x00004052

004000:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004010:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004020:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004030:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004040:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004050:	00 00 00 00 00 00 00 00 00 42 00 00 00 00 00 00

メモリエリア

(5) 関数内での パラメータの使用

関数のパラメータの渡し方

1. レジスタを使用

2. システムスタックエリアを使用

こちらを説明


```

.data
str1:
    .ascii "My Name is David!¥0"
SYS_STK:
    .ds.b 0x4000
SYS_STK_TOP:

.text
stringlength:
    link.w %a6, #-8
    clr.w -2(%a6)
    move.l 8(%a6), -6(%a6)
start1:
    move.l -6(%a6), %a0
    cmp.b #0, (%a0)
    beq break1
    addq.l #1, -6(%a6)
    addq.w #1, -2(%a6)
    bra start1
break1:
    move.w -2(%a6), %a0
    move.l %a0, %d0
    unlk %a6
    rts

main:
    lea.l SYS_STK_TOP, %a7
    lea.l str1, %a0
    move.l %a0, -(%a7)
    jsr stringlength
    addq.l #4, %a7

    .dc.w 0x4848
    stop #0

.end

```

プッシュした値を、
ここで読み出し

A0 の値をプッシュ
(文字列の先頭アドレス
0x0000004c を push)

link 後
A6: 0x00004052

004000:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004010:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004020:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004030:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004040:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004050:	00 00 00 00 00 00 00 00 00 42 00 00 00 4c

```

.data
str1:
    .ascii "My Name is David!¥0"
SYS_STK:
    .ds.b 0x4000
SYS_STK_TOP:

.text
stringlength:
    link.w %a6, #-8
    clr.w -2(%a6)
    move.l 8(%a6), -6(%a6)
start1:
    move.l -6(%a6), %a0
    cmp.b #0, (%a0)
    beq break1
    addq.l #1, -6(%a6)
    addq.w #1, -2(%a6)
    bra start1
break1:
    move.w -2(%a6), %a0
    move.l %a0, %d0
    unlk %a6
    rts

main:
    lea.l SYS_STK_TOP, %a7
    lea.l str1, %a0
    move.l %a0, -(%a7)
    jsr stringlength
    addq.l #4, %a7

    .dc.w 0x4848
    stop #0

.end

```

プッシュした値を、
ここで読み出し

A0 の値をプッシュ
(文字列の先頭アドレス
0x0000004c をプッシュ)

pop しても仕方が無いので、
A7 に 4 足すだけ

(6) 関数での処理結果の,
呼び出し側への引渡し

C言語

68000アセンブラ言語

```
#include <stdio.h>
```

```
int stringlength
```

```
{  
    short int len;  
    char *s;
```

```
    len = 0;  
    s = str;  
    while ( (*s)  
            s++;  
            len++;  
    }
```

```
    return len;  
}
```

```
int main()
```

```
{  
    short int n;  
    n = stringlength( "My Name is David\n" );  
    fprintf( stderr, "n = %d\n", n );  
    return 0;  
}
```

```
.data  
str1:
```

```
.ascii "My Name is David!¥0"
```

```
SYS_STK:
```

```
.dc.b 0x4000
```

関数での処理結果を、
呼び出し側に引き渡す
(データレジスタ D0 を使用)

```
    addq.w #1, -2(%a0)  
    bra start1
```

```
break1:
```

```
    move.w -2(%a6), %a0  
    move.l %a0, %d0  
    unlk %a6  
    rts
```

```
main:
```

```
    lea.l SYS_STK_TOP, %a7
```

```
    lea.l str1, %a0  
    move.l %a0, -(%a7)  
    jsr stringlength  
    addq.l #4, %a7
```

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

```
.end
```

関数での処理結果の, 呼び出し側への引渡し

1. レジスタ
2. 所定のメモリアドレスに書き込み
3. 関数へのパラメータ(システムスタックエリア内)を書き換える