

as-4. 条件分岐と繰り返し

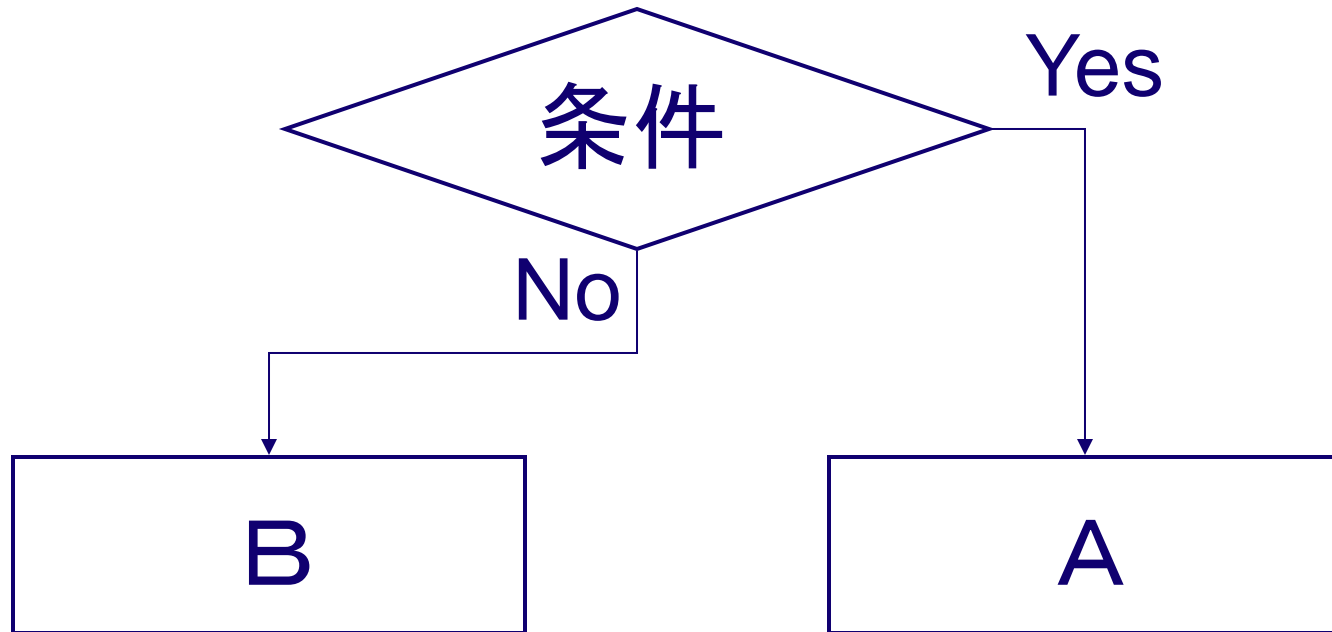
(68000アセンブラプログラミング)

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

金子邦彦



条件分岐とは



- 「ある条件」が成り立てばAを、成り立たなければBを実行

例題1. 2数の最大値

- データレジスタ D0, データレジスタ D1 のうち大きい方をデータレジスタ D2 にセットする

例) D0 の中身: 0x 0000 0030

D1 の中身: 0x 0000 0040

のとき

D2 の中身: 0x 0000 0040

比較命令 (D0 と D1 の比較)

条件分岐命令

.text

```
cmp.l %d0, %d1
```

```
bhi ELSE1 /* d1 > d0 */
```

```
move.l %d0, %d2
```

```
bra ENDIF1
```

条件が成り立つ場合に
実行される部分

ELSE1:

```
move.l %d1, %d2
```

条件が成り立たない場合
に実行される部分

ENDIF1:

```
.dc.w 0x4848
```

```
stop #0
```

.end

実行順

「D1 の中身 > D0 の中身」が成り立つ場合

.text

① **cmp.l %d0, %d1**

② **bhi ELSE1 /* d1 > d0 */**

move.l %d0, %d2

bra ENDIF1

ELSE1:

③ **move.l %d1, %d2**

ENDIF1:

④ **.dc.w 0x4848**

⑤ **stop #0**

.end

実行順

「D1 の中身 > D0 の中身」が成り立たない場合

.text

① **cmp.l %d0, %d1**

② **bhi ELSE1 /* d1 > d0 */**

③ **move.l %d0, %d2**

④ **bra ENDIF1**

ELSE1:

move.l %d1, %d2

ENDIF1:

⑤ **.dc.w 0x4848**

⑥ **stop #0**

.end

D0, D1 の中身を比較. 比較結果による条件分岐

```
cmp.l %d0, %d1
```

```
bhi ELSE1 /* d1 > d0 */
```

条件分岐の一般形

<コンディションコードレジスタを変化させる命令>

<条件分岐命令>

bhi, bcc, beq

bne, bcs, bls など

cmp など

```
cmp.l %d0, %d1
```

```
bhi ELSE1 /* d1 > d0 */
```

条件分岐命令	分岐する条件
bhi	D0の中身 < D1の中身
bcc	D0の中身 ≤ D1の中身
beq	D0の中身 = D1の中身
bne	D0の中身 ≠ D1の中身
bcs	D0の中身 > D1の中身
bls	D0の中身 ≥ D1の中身


```
cmp.l %d0, %d1
bhi ELSE1 /* d1 > d0 */
```

分岐条件

bhi, bcc, beq
bne, bcs, bls など

分岐先の
メモリアドレス
※ プログラムカウンタ
にセットされる

例題2. 条件分岐

- 条件分岐の例として, 次の例を考える

$$y = 8 \times x \quad (x > 5 \text{のとき})$$

$$y = 0 \quad (x \leq 5 \text{のとき})$$

(一部の内容は, 復習を兼ねる)

```
.data
```

```
x:
```

```
.dc.l 7
```

```
y:
```

```
.dc.l 0
```

それぞれ、1ロングワードの
データエリアをメモリ中に確保

```
.text
```

```
moveq.l #5, %d0
```

```
cmp.l x, %d0
```

```
bcc else1
```

```
move.l x, %d0
```

```
lsl.l #3, %d0
```

```
move.l %d0, y
```

```
bra endif1
```

```
else1:
```

```
clr.l y
```

```
endif1:
```

```
.dc.w 0x4848
```

```
stop #0
```

```
.end
```

ロングワード

1ロングワードは4バイト

プログラム中使用

- .l ロングワード(4バイト)
- .w ワード(2バイト)
- .b バイト(1バイト)

```

.data
x:
    .dc.l 7
y:
    .dc.l 0

.text
    moveq.l #5, %d0
    cmp.l x, %d0
    bcc else1
    move.l x, %d0
    lsl.l #3, %d0
    move.l %d0, y
    bra endif1
else1:
    clr.l y
endif1:

    .dc.w 0x4848
    stop #0

.end

```

x と 5 の比較
 (D0 を使用)

← 比較結果による分岐

$x > 5$ のとき実行
 される部分

$x > 5$ が成り立たない
 とき実行される部分

```
.data
```

もし $x > 5$ ならば

実行順

① `moveq.l #5, %d0`

② `cmp.l x, %d0`

③ `bcc else1` 分岐しない

④ `move.l x, %d0`

⑤ `lsl.l #3, %d0` $x > 5$ のとき実行される部分

⑥ `move.l %d0, y`

⑦ `bra endif1`

```
    clr.l y
```

`endif1:`

スキップ

⑧ `.dc.w 0x4848` 以後省略

```
    stop #0
```

```
.end
```

ジャンプ

分岐命令

ラベル `endif1` へ

分岐せよという指示

※ `bra` は必ず分岐する

```
.data
```

もし $x \leq 5$ ならば

実行順 : 1 0

- ① `moveq.l #5, %d0`
- ② `cmp.l x, %d0`
- ③ `bcc else1`

分岐する

ラベル `else1` へ分岐せよという指示

ジャンプ

```
move.l x, %d0  
lsl.l #3, %d0  
move.l %d0, y  
bra endif1
```

スキップ

`else1:`

- ④ `clr.l y`

$x > 5$ が成り立たないとき
実行される部分

`endif1:`

- ⑤ `.dc.w 0x4848` 以後省略
`stop #0`

```
.end
```

5(数値)と X の中身を比較. 比較結果による条件分岐

D0 に 0x0000 0005 を入れる

```
moveq.l #5, %d0  
cmp.l x, %d0  
bcc else1
```

x の中身と D0 の比較

※ メモリからの読み出し

※ D0 を比較のために使用

条件分岐条件

D0 に 0x0000 0005 を入れる

```
moveq.l #5, %d0  
cmp.l x, %d0  
bcc else1
```

x の中身と D0 の比較
(メモリからの読み出し)

分岐条件

bhi

bcc

beq

bne

bcs

bls

この場合の意味

bhi xの中身 < D0の中身

bcc xの中身 \leq **D0の中身**

beq xの中身 = D0の中身

bne xの中身 \neq D0の中身

bcs xの中身 > D0の中身

bls xの中身 \geq D0の中身

5(数値)と X の中身を比較. 比較結果による条件分岐

```
cmp.l x, %d0
```

```
bcc else1
```

条件分岐の一般形

<コンディションコードレジスタを変化させる命令>

<条件分岐命令>

bhi, bcc, beq

bne, bcs, bls など

cmp など

「cmp.l x,%d0」の実行では

アドレスバス

データバス

xの中身

x

R/W

「引き算」が
実行される

D0 00000005

00000005 スタ
registers

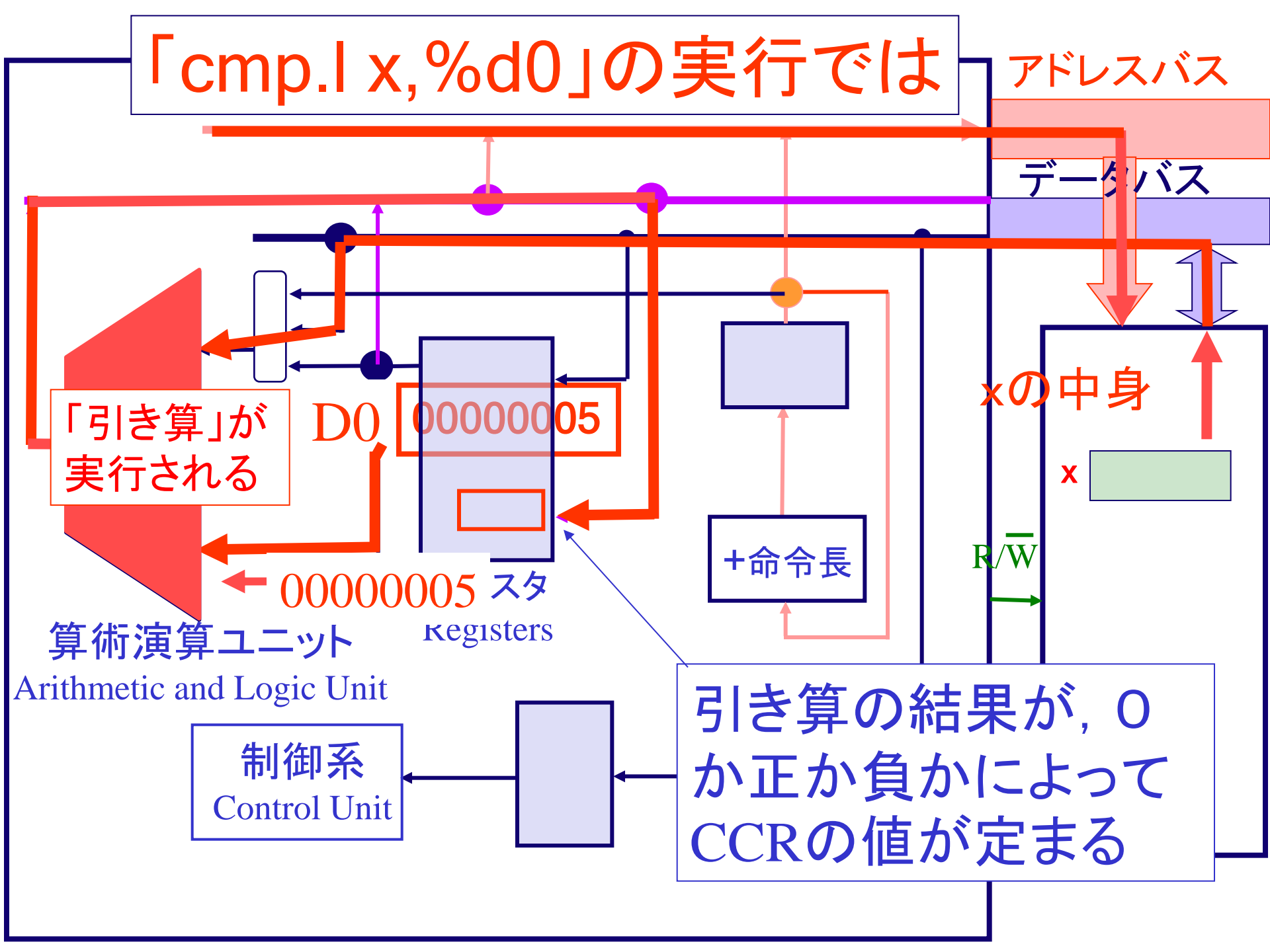
+命令長

算術演算ユニット

Arithmetic and Logic Unit

制御系
Control Unit

引き算の結果が、0
か正か負かによって
CCRの値が定まる



「bcc else1」の実行では

アドレスバス

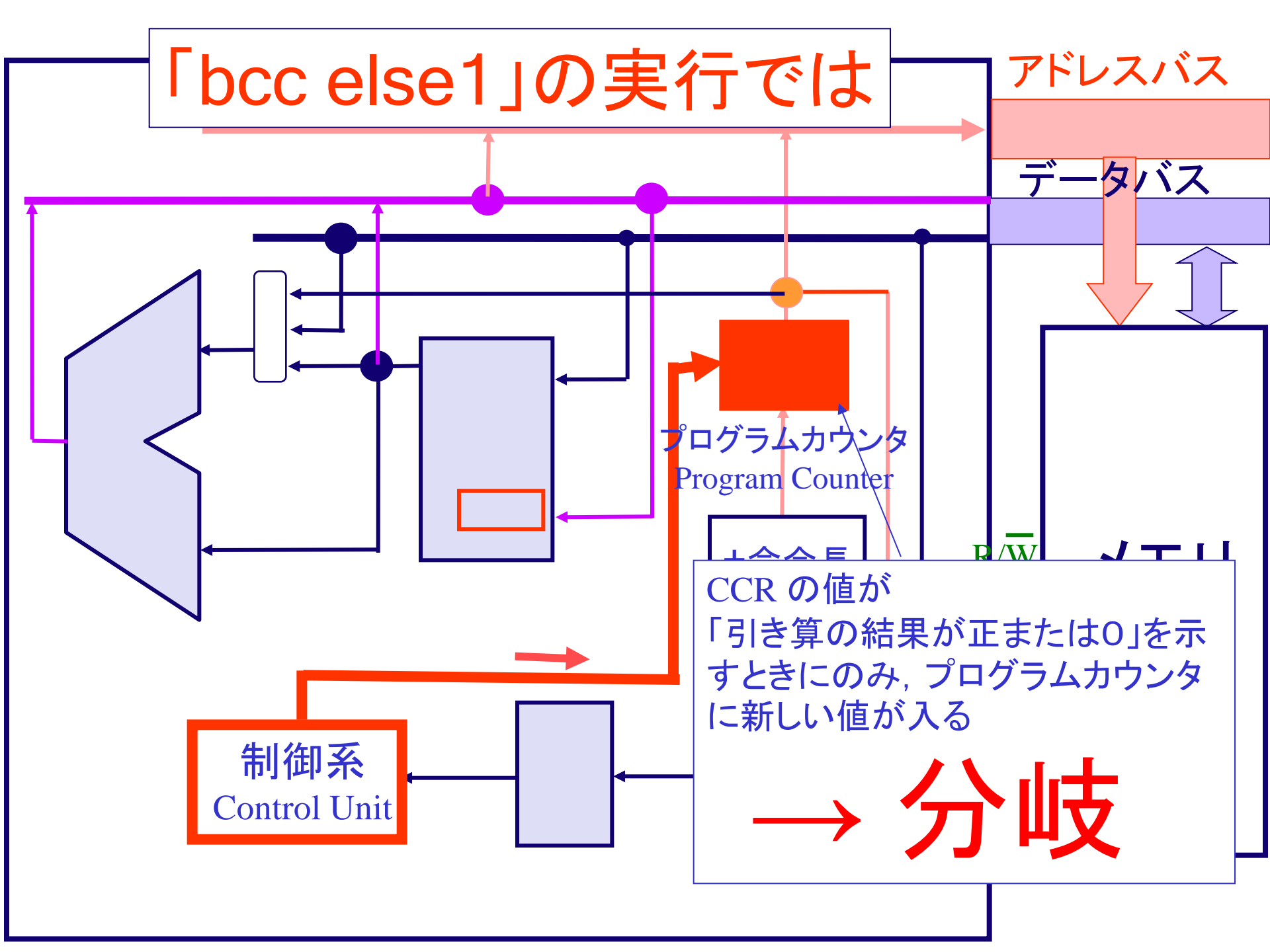
データバス

プログラムカウンタ
Program Counter

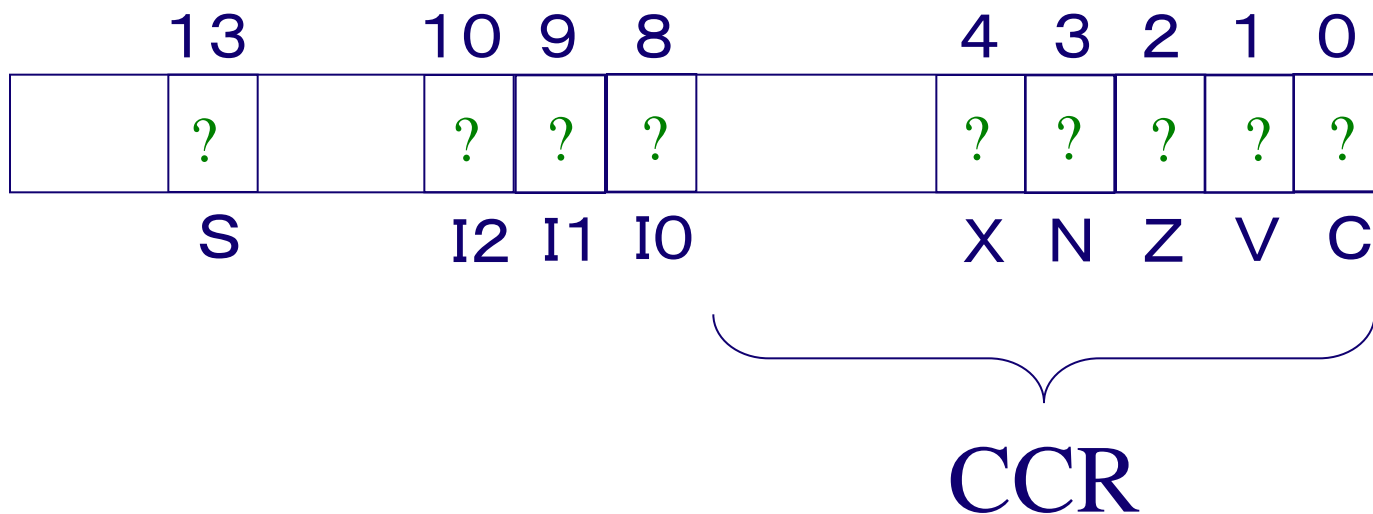
制御系
Control Unit

CCR の値が
「引き算の結果が正または0」を示
すときにのみ、プログラムカウンタ
に新しい値が入る

→ 分岐



ステータスレジスタ



ステータスレジスタ(16ビット)
の下位1バイトが、コンディション
コードレジスタ(CCR)

コンディションコードレジスタの フラグの振る舞い

- フラグは X, N, Z, V, C の5つ
- フラグはすべて1ビット(0か1の値をとる)

cmp 命令での振る舞い

cmp.l <ソース>, <ディスティネーション> 2つの比較

Nフラグ: $\text{<ディスティネーション>} - \text{<ソース>} < 0$ なら1
さもなければ0

Zフラグ: $\text{<ディスティネーション>} - \text{<ソース>} = 0$ なら1
さもなければ0

2数の引き算の結果

※ cmp 命令以外でも CCR の値は変化する

(参考) MOVE 命令でのコンディ ションコードレジスタの変化

X: 変化せず

N: 転送結果の「最上位ビット」が1ならセット(1)、
それ以外はリセット(0)

Z: 転送結果が「全てのビット」が0ならセット(1)、
それ以外はリセット(0)

V: 常にリセット(0)

C: 常にリセット(0)

※ 2つの比較ではなく、
「転送したデータ」の値に
よる変化

例題3. 繰り返し

- 次の例で、条件分岐命令を見る

$$S = \sum_{i=1}^3 i$$

繰り返しの一般形

START:

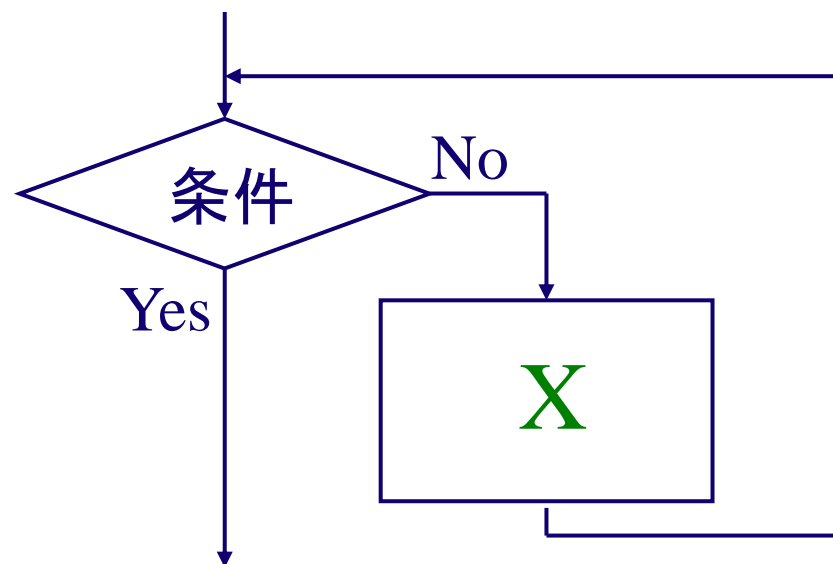
命令 (比較命令, 演算など)

b?? QUIT

命令
命令
...

bra START

QUIT:



- 何かの処理の繰り返し
- 繰り返しのたびに **条件分岐命令**が実行され, 指定された条件が成り立たない限り, 実行が繰り返される

繰り返し

繰り返しの終了条件

「D0 の中身 ≤ 3 」が成り立たない

D0 の中身
と 3 の比較

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

.end
```

分岐条件

bhi	D0	>	3
bcc	D0	\geq	3
beq	D0	=	3
bne	D0	\neq	3
bcs	D0	<	3
bls	D0	\leq	3

D0 の中身 > 3
のときはジャンプ

繰り返し

D0 の中身
と 3 の比較

ジャンプ

繰り返しを続ける

D0 の中身 ≤ 3
のとき

```
.data
s:
    .dc.l 0
.text
/* %d0 = 1, 2, 3 */
moveq.l #1, %d0
rt1:
    cmp.l #3, %d0
    bhi break1
    add.l %d0, s
    addq.l #1, %d0
    bra start1
break1:
    .dc.w 0x4848
    stop #0
.end
```

例題4. ワードデータの配列

- 10個のワードデータに, 順に, 0000, 0001, 0002, ..., 0009 をセットするプログラム

実行前

プログラム本体そのものが入っているエリア

000000:	42	40	41	f9	00	00	00	20	0c	40	00	09	62	00	00	0c
000010:	30	80	52	40	54	88	60	00	ff	f0	48	48	4e	72	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
000040:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

未使用

10ワード(=20バイト)
のデータエリア

実行後

プログラム本体そのものが入っているエリア

000000:	42	40	41	f9	00	00	00	20	0c	40	00	09	62	00	00	0c
000010:	30	80	52	40	54	88	60	00	ff	f0	48	48	4e	72	00	00
000020:	00	00	00	01	00	02	00	03	00	04	00	05	00	06	00	07
000030:	00	08	00	09	00	00	00	00	00	00	00	00	00	00	00	00
000040:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

未使用

10ワード(=20バイト)のデータエリア

```
.data
```

```
a:  
    .ds.w 10
```

「10」ワード分のデータエリアを確保

```
.text
```

```
    clr.w %d0  
    lea a, %a0
```

```
start1:
```

```
    cmp.w #9, %d0  
    bhi break1  
    move.w %d0, (%a0)  
    addq.w #1, %d0  
    addq.l #2, %a0  
    bra start1
```

```
break1:
```

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

```
.end
```

※ `.dc.w 10`
1ワードの確保.
初期値を10にセット

※ `.ds.w 10`
10ワードの確保
(初期値は不定)

`.l` ロングワード(4バイト)
`.w` ワード(2バイト)
`.b` バイト(1バイト)

```
.data
```

```
a:  
    .ds.w 10
```

「10」ワード分のデータエリアを確保

```
.text
```

```
clr.w %d0  
lea a, %a0
```

D0 をクリア (0x0000 → D0 の下2バイト)
ラベル a のメモリアドレスを A0 にセット
(0x00000020 → A0)

```
start1:
```

```
cmp.w #9, %d0
```

D0 と 9 の比較

```
bhi break1
```

比較結果による分岐

```
move.w %d0, (%a0)  
addq.w #1, %d0  
addq.l #2, %a0  
bra start1
```

繰り返し実行

```
break1:
```

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

```
.end
```

```
.data
```

```
a:
```

```
.ds.w 10
```

```
.text
```

```
clr.w %d0
```

```
lea a, %a0
```

```
start1:
```

```
cmp.w #9, %d0
```

```
bhi break1
```

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

```
addq.w #1, %d0
```

```
addq.l #2, %a0
```

```
bra start1
```

```
break1:
```

```
.dc.w 0x4848
```

```
stop #0
```

```
.end
```

ラベル a のメモリアドレスを A0 にセット
(0x00000020 → A0)

A0がポイントしている
メモリアドレスに, D0 の中身を書き込む

D0 + 1 → D0 (下2バイト)
0x0000, 0x0001, 0x0002,
..., 0x0009

A0 + 1 → A0
0x00000020, 0x00000022,
..., 0x00000032

オペランドの#の意味

- プログラム命令では

- #付き : 値を表す

`move.w` #0x1000, %d0

→ メモリの読み書きを行わない

- #無し : メモリアドレスなどを表す

`move.w` ADDR, %d0

→ メモリの読み書きを行う

- 擬似命令では

- ふう # はない. 例えば

`.equ` ADDR 0xffff00

`.org` 0x0400

`.dc.w` 0x4848

`a:` `.ds.w` 1

記法 (%a0)

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

D0 の中身を, A0がポイントしている
メモリアドレスに書き込む

メモリへの書き込み

A0 の値が 0x00000020 ならば
0x20, 0x21 番地に, D0 の下2バイトを書き込む

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

D0 の中身を A0 の中にコピー
メモリの読み書き無し

lea 命令と move 命令

```
lea s, %a0
```

ラベル s が示すメモリアドレスの値を A0 に格納
0x00000020 → A0

メモリアクセス無し

```
move.l s, %a0
```

ラベル s が示すメモリの中身を A0 に格納
<0x00000020 の中身 4バイト分> → A0

メモリからの読み出し

例題5. 文字列の長さ

- 文字列の長さを数えるプログラム

今回の文字列データ: `My Name is David!`

- 文字列の先頭から1文字ずつ読み
 - 0 で無ければ, 「データレジスタD0 に1を足す」ことを繰り返す
 - 0 ならば処理を終える

実行後

文字列の末端を示す 0

プログラム本体そのものが入っているエリア

000000:	41	f9	00	00	00	20	42	80	0c	10	00	00	67	00	00	0a
000010:	52	88	52	80	60	00	ff	f2	48	48	4e	72	00	00	00	00
000020:	4d	79	20	4e	61	6d	65	20	69	73	20	44	61	76	69	64
000030:	21	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000040:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

未使用

文字列のデータ
(1文字で1バイト)

ASCIIコード

- パソコン, ワークステーションで英数文字データを扱うときの標準

ASCIIコード表

	0	1	2	3	4	5	6	7
0	NULL	DEL	SP	0	@	P		p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	“	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	\$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	(BS)	CAN	(8	H	X	h	x
9	(HT)	EM)	9	I	Y	i	y
A	(LF)	SUB	*	:	J	Z	j	z
B	(VT)	ESC	+	;	K	[k	{
C	(FF)	(FS)	,	<	L	¥	l	
D	(CR)	(GS)	-	=	M]	m	}
E	SO	(RS)	.	>	N	^	n	~
F	SI	(US)	/	?	O	_	o	DEL

青は特別用途
の1バイトデータ

緑は英数文字
データ

文字列

- My Name is David! のように、英数文字が並んだもの
- 各1文字は1バイト
- 末尾の「0x00」（これで1バイト）
文字列の終わりを意味する（文字列の長さは変化するので、終わりを示すための記号が必要）


```
.data
```

```
str1:
```

```
.ascii "My Name is David!\0"
```

```
.text
```

```
lea.l str1, %a0
```

```
clr.l %d0
```

```
start1:
```

```
cmp.b #0, (%a0)
```

```
beq break1
```

```
addq.l #1, %a0
```

```
addq.l #1, %d0
```

```
bra start1
```

```
break1:
```

```
.dc.w 0x4848
```

```
stop #0
```

```
.end
```

文字列データのデータ
エリアを確保

アセンブラプログラム中での
文字列の書き方

```
.ascii "<文字列> \0"
```

```
.data
str1:
    .ascii "My

.text

start1:

break1:

    .dc.w 0x4848
    stop #0

.end
```

ラベル str1 のメモリアドレスを A0 にセット
(0x00000020 → A0)
D0 をクリア (0x00000000 → D0)

```
lea.l str1, %a0
clr.l %d0
```

0 との比較
(文字列の末端か?)

```
cmp.b #0, (%a0)
```

```
beq break1
```

```
addq.l #1, %a0
```

```
addq.l #1, %d0
```

```
bra start1
```

繰り返し実行

```
.data
str1:
    .ascii "My Name is David!\0"

.text
lea.l str1, %a0
clr.l %d0

start1:
    cmp.b #0, (%a0)
    beq break1
    addq.l #1, %a0
    addq.l #1, %d0
    bra start1

break1:
    .dc.w 0x4848
    stop #0

.end
```

```
lea.l str1, %a0
clr.l %d0
```

```
cmp.b #0, (%a0)
beq break1
```

```
addq.l #1, %a0
addq.l #1, %d0
bra start1
```

A0 + 1 → A0
0x00000020, 0x00000021,
..., 0x00000031

D0 + 1 → D0
0x00000000, 0x00000001,
..., 0x00000011

おわりに

- 条件分岐、繰り返しには、分岐命令が登場する
- 分岐命令では、プログラムカウンタの強制的な書き換えが起こる
- 比較命令などで、コンディションコードレジスタ(ステータスレジスタの下位1バイト)が変化。分岐命令で利用される