

## 2. アセンブラ入門(2)

- 分岐, 繰り返し
- サブルーチンコール

# 無条件分岐命令 bra

bra.w    ラベル  
bra.b    ラベル                      (サイズを省略したときは w)

- ラベルで指定された番地に分岐.
- コードは, サイズが b のときは, 60?? で, ?? 部分は, 分岐先の命令との相対距離を表す1バイトの2の補数(−128~127).  
サイズが w のときは, 6000???? で, ???? 部分は分岐先の命令との相対距離を表す2バイトの2の補数.

※ アセンブラが自動的に相対距離を計算してくれるので, 心配なし.

- **この命令を実行すると,**

この命令の位置 + 2 + 相対距離( ?? または, ???? )  
にある命令に分岐する.

**注意: 本命令と分岐先の命令の相対距離が2バイトの2の補数で表せるときしか利用できない.**

# 無条件分岐命令 jmp

---

jmp 番地

jmp (%a0)

など（詳しくはマニュアル参照）

- オペランドで指定された値の番地に無条件分岐.
- この命令実行後, PC の値はオペランドで指定された値になる.
- たとえば, オペランドが (%a0) で, アドレスレジスタ a0 の値が 0x00001000 ならば,

jmp 0x00001000

と書いたのと同じで, 0x00001000 番地に分岐することになる.

※ (%a0) と書くと, 上記の例では, 1000番地にある値が示す番地に分岐するように思うかも知れないが, それは誤り.

オペランドで指定されるのは, イミディエイトデータ以外は場所である.

jmp %a0 だと, アドレスレジスタ a0 に分岐するという変なことになる!!

# 条件付き分岐命令 **bcc**

**bcc**. w    ラベル                      (サイズを省略したときは w)  
**bcc**. b    ラベル                      (**cc** は hi, cc, eq, ne などの条件)

- ラベルで指定された番地への相対距離による分岐 (bra 参照).
- 条件が成立するとき分岐し (PC を書き換える), 条件が成立しないときは本命令の次に位置する命令に制御が移る (PC を書き換えない).
- たとえば,

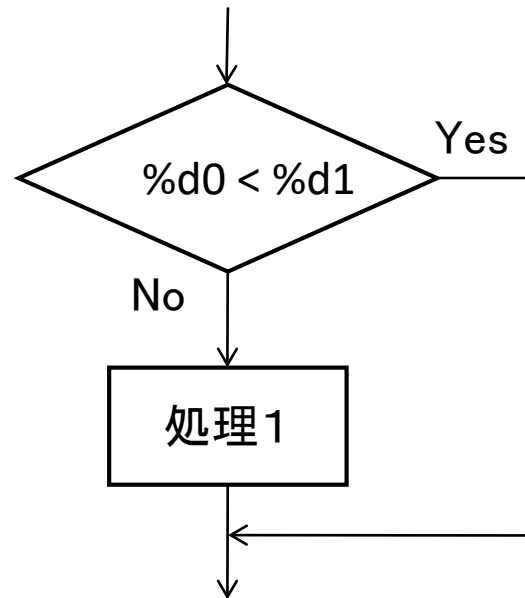
cmp. l    %d0, %d1    ← d1 - d0 をした結果が CCR に反映される

の後, 以下のそれぞれの条件分岐命令を実行すると...

```
bhi    label1    /* d0 < d1 のとき分岐 */
bcc    label1    /* d0 ≤ d1 のとき分岐 */
beq    label1    /* d0 = d1 のとき分岐 */
bne    label1    /* d0 ≠ d1 のとき分岐 */
```

(その他の条件についてはマニュアル参照)

# if 文 (C言語) に相当するプログラム

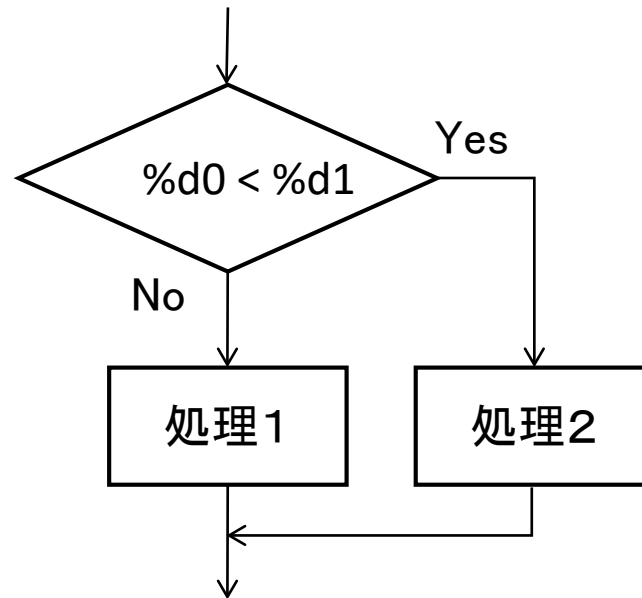


```
cmp. l    %d0, %d1
bhi       endif    /* d0 < d1 のとき endif に分岐 */


処理 1


endif:
```

# if ~ else文 (C言語) に相当するプログラム <sup>6</sup>

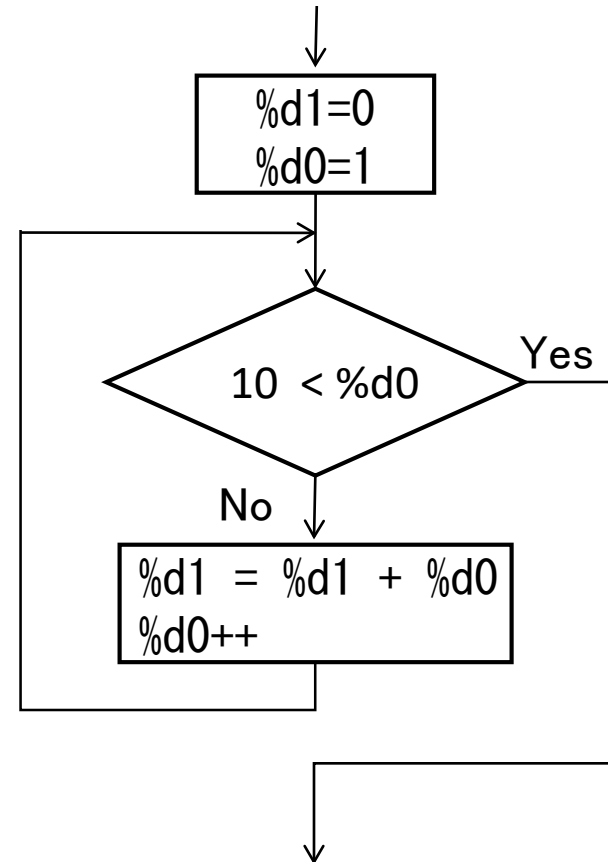


```
cmp. l    %d0, %d1
bhi       else    /* d0 < d1 のとき else に分岐 */
    処理 1
bra       endif
else:     処理 2
endif:
```

# 繰り返し

$$\%d1 \leftarrow \sum_{n=1}^{10} n$$

```
loop:  move.w  #0, %d1
       move.w  #1, %d0
       cmpi.w  #10, %d0
       bhi     endL
       add.w   %d0, %d1
       addi.w  #1, %d0
       bra     loop
endL:
```



# サブルーチン(1/)

---

bsr.w    ラベル

bsr.b    ラベル

ラベルで指定されたサブルーチンの呼び出し.

相対距離による呼び出し(bra 参照).

サブルーチンの呼び出しを行う命令に, 上記の他 jsr がある( bsr と jsr の相違は, bra と jmp の相違と同様).

本命令を実行すると,

- 現在の PC の値をシステムスタックへ PUSH  
( A7 の値から 4 減じ, その値のアドレスへ PC の値を格納)
- ラベルで指定されたサブルーチンの番地を PC に格納(bra参照)

rts

サブルーチンからの復帰命令.

本命令を実行すると,

- システムスタックからPCへPOP  
(A7 の値のアドレスから4バイトを PC へ転送し, A7 の値を 4 増やす)



# サブルーチン(2/)

```
000500 6100 00FE      bsr.w  abcd
000504      ????
```

```
000600      ??? ????  abcd:
```

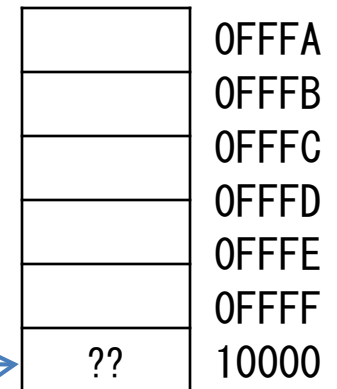
  

```
000620 4E75          rts
000622      ????
```

実行中

PC 00000504

a7 00010000



bsr

- 現在の PC の値をシステムスタック「a7」へ PUSH
- ラベルで指定されたサブルーチンの番地を PC に格納(bra参照)

# サブルーチン(2/)

```
000500 6100 00FE      bsr.w  abcd
000504      ????
```

```
000600      ??? ????  abcd:
```

```
000620 4E75          rts
000622      ????
```

実行直後

PC 00000504

a7 0000FFFB

	0FFFA
00	0FFFB
00	0FFFC
05	0FFFD
04	0FFFE
??	0FFFF
	10000

bsr

- 現在の PC の値をシステムスタック「a7」へ PUSH
- ラベルで指定されたサブルーチンの番地を PC に格納(bra参照)

## サブルーチン(2/)

```
000500 6100 00FE          bsr.w    abcd
000504 ?????

000600 ????? ?????    abcd:

000620 4E75          rts
000622 ?????
```

## 実行直後

PC	00000600
----	----------

a7	0000FFFB
----	----------

次は 0x600 番地  
から始まる命令  
を実行(サブ  
ルーチンコール)

	0FFFA
00	0FFFB
00	0FFFC
05	0FFFD
04	0FFFF
??	10000

bsr

- 現在の PC の値をシステムスタック「a7」へ PUSH
- ラベルで指定されたサブルーチンの番地を PC に格納 (bra 参照)

# サブルーチン(2/)

```
000500 6100 00FE          bsr.w    abcd
000504      ????
```


```
000600      ??? ????  abcd:
```

```
000620 4E75          rts
000622      ????
```

実行中

PC 00000622

a7 0000FFFB



	0FFFA
00	0FFFB
00	0FFFC
05	0FFFD
04	0FFFE
04	0FFFF
??	10000

rts

- システムスタックへから PC に POP

# サブルーチン(2/)

```
000500 6100 00FE          bsr.w    abcd
000504      ????
```

```
000600      ??? ????  abcd:
```

```
000620 4E75          rts
000622      ????
```

実行直後

PC 00000504

a7 00010000

次は 0x504 番地  
から始まる命令  
を実行(呼び出し  
側へ復帰)

	0FFFA
	0FFFB
	0FFFC
	0FFFD
	0FFFE
	0FFFF
??	10000

rts

- システムスタックへから PC に POP

# サブルーチン(2/)

---

## サブルーチンへのパラメータ(引数)の渡し方

- レジスタを利用      ←———— 基礎ソフト実験, ソフト実験1  
    メリット: 仕組みが簡単
- システムスタックを利用      ←———— ソフト実験2, 3  
    メリット: 再帰呼び出しが可能, C言語との相性が良い

## レジスタの PUSH, POP

サブルーチンで使用する(値を呼び出し側に返すためのレジスタ以外の)レジスタの値が, サブルーチンの実行により壊される.

➡ サブルーチンの初めでシステムスタックに PUS,  
サブルーチンの終わりでシステムスタックから POP

※ PUSH, POP には movem 命令を利用

PUSH と POP の対応がとれないと, rts により呼び出し側に復帰できず暴走