分岐命令

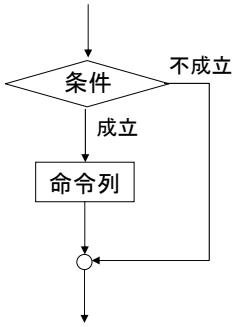
■ 条件判断（if 文）や繰返し（for 文，while 文）の制御構造はどのようにつくるか

■ 制御構造を実現するための命令

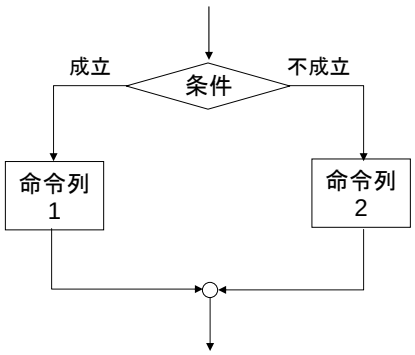
- 分岐命令（条件分岐命令，無条件分岐命令）
- 比較命令

プログラムの制御構造（C言語）

条件判定： if - then 型

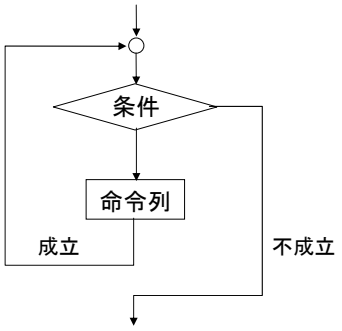


条件判定： if - then - else 型

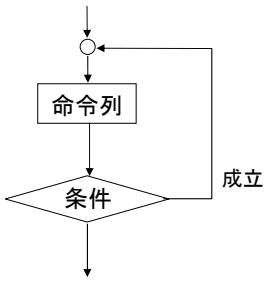


プログラムの制御構造（C言語）

繰返し： while 型



条件判定： do - while 型



（ for 文 はwhile型の特殊な場合 ）

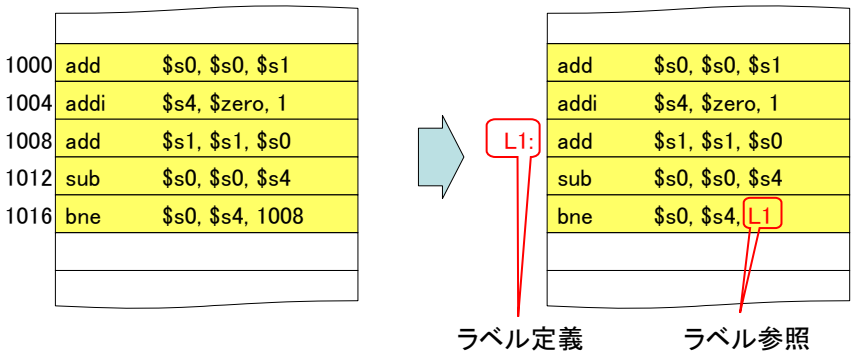
通常の実行では，実行した命令の次の番地（アドレス）の命令が実行される

繰返しや条件判定などの制御構造を作るには，任意の番地に分岐(ジャンプ)できる命令が必要になる

- ◆ 条件を判定し，条件の成立・不成立によって次に実行する命令の番地を変更する命令
  （条件分岐命令 / 条件ジャンプ命令）
  : ただし条件によらず，「常に分岐 / ジャンプ」する命令（無条件分岐/無条件ジャンプ命令）もある。
- ◆ 分岐 / ジャンプ先の番地を指定するにはラベル（label）を用いる

ラベル（label）

ラベル: 命令やデータが配置されている主記憶上の番地につけられた名前



※ ラベルはアセンブラによって主記憶の番地に変換される

条件分岐命令(1)

一致（＝ 条件）分岐　　beq $Ri, $Rj, *Label*

- “ branch on equal ”
- $Ri と $Rj が等しければ，ラベル *Label* に分岐

  さもなければ，次のアドレスの命令を実行

```
        …
        beq  $s1, $s0, L1
        …
        …
L1:     …
```

条件分岐命令(2)

不一致(≠ 条件)分岐　bne $Ri, $Rj, *Label*

- " branch on not equal "
- $Ri と $Rj が等しくなければ，ラベル *Label* に分岐

さもなければ，次のアドレスの命令を実行

```
        …
        bne  $s1,  $s0, L2
        …
        …
L2:     …
```

無条件分岐命令

無条件分岐　b *Label*

- " branch "
- 無条件で(常に)，ラベル *Label* に分岐

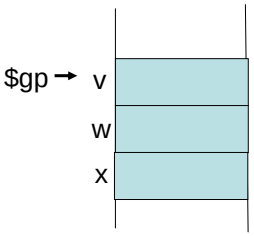
```
        …
        b  L3
        …
        …
L3:     …
```

条件：if - then型

C プログラム
1: if (v == w) {
2: x = x+1 ;
3: }

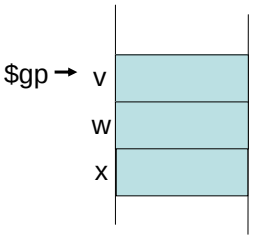
条件判定の扱い
プログラミング言語
： 成立したら次の文を実行
アセンブラ言語（機械語命令）
： 成立したら分岐する
→ アセンブラ言語では
逆の条件で調べたほうが
プログラム構造は考えやすい

```
1      lw  $s0, 0($gp)      # $s0 ← v
2      lw  $s1, 4($gp)      # $s1 ← w
3                           # $s0≠$s1 かどうか？
4      lw  $s3, 8($gp)      # $s0=$s1ならば,
5      addi $s3, $s3, 1     # $s3 ← $s3 + 1
6      sw $s3, 8($gp)       # x ← $s3
7 exit:
```



条件：if - then - else 型

C プログラム
1: if (v == w) {
2: x = x+1 ;
3: } else {
4: x = x – 1;
5: }




```
1      lw  $s0, 0($gp)      # $s0 ← v
2      lw  $s1, 4($gp)      # $s1 ← w
3                           # $s0≠$s1かどうか？
4      lw  $s2, 8($gp)      # $s2 ← x
5      addi $s2, $s2, 1     # $s2 ← $s2+1
6      sw $s2, 8($gp)       # x ← $s2
7                           # else 部をスキップ
8 else: lw $s2, 8($gp)      # $s2 ← x
9      addi $s2, $s2, -1    # $s2 ← $s2 −1
10     sw $s2, 8($gp)       # x ← $s2
11 exit:
```

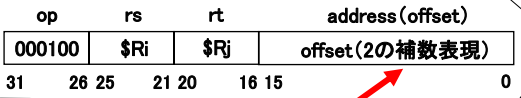
分岐命令の命令形式

「現在のPCの値」は現在実行している命令の次の命令の番地になっていることに注意

※ 分岐先の番地は現在のPCの値からのオフセット（変位；何番地離れているか）で指定する
（「何命令前か，何命令後か」という指定形式：PC相対指定）

命令の先頭番地は4番地ごとだから

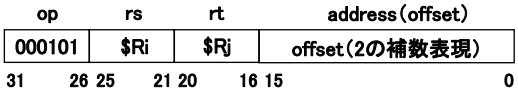
beq $Ri, $Rj, *Label*



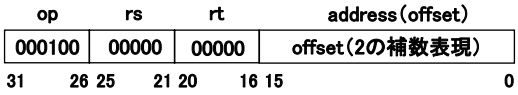
PCの値はbeq命令の次の命令の番地を指しているから

{（Label の番地）－ 4 －（この命令のある番地）}÷4

bne $Ri, $Rj, *Label*



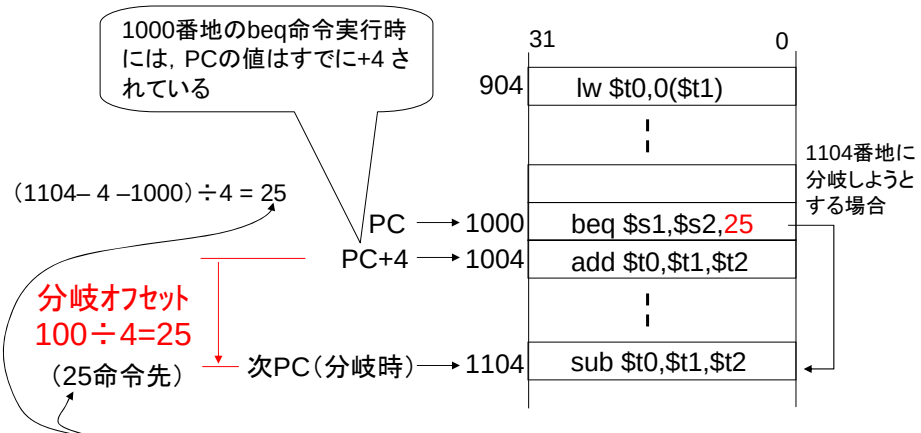
b *Label*



b 命令は beq 命令で，$Ri=$Rj=$zero としたもの
（ $zero=$zero だから常に成立）

分岐先アドレス（offset）の指定（後方の指定）

1000番地のbeq命令実行時には，PCの値はすでに+4 されている

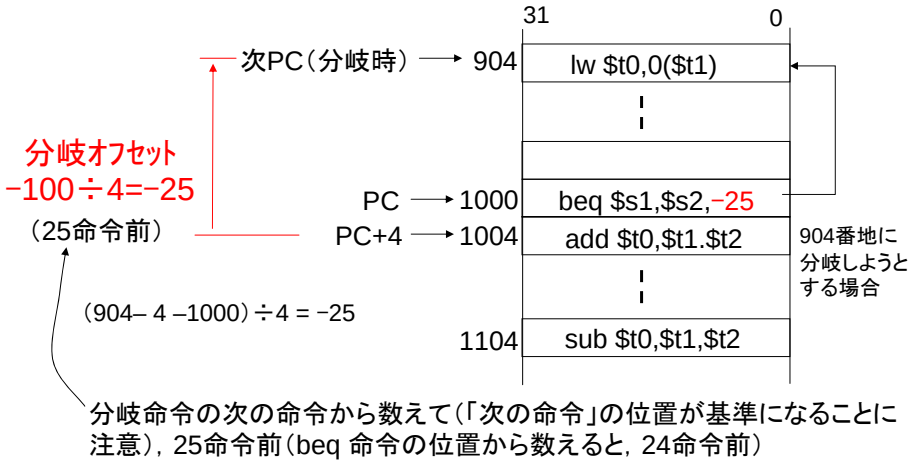


（1104－ 4 －1000）÷4 = 25

分岐オフセット
100÷4=25
（25命令先）

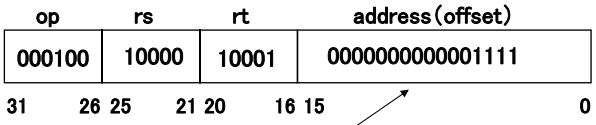
分岐命令の次の命令から数えて（「現在のPCの値」が基準になることに注意），25命令後（beq 命令の位置から数えると，26命令後方）

分岐先アドレス（offset）の指定（前方の指定）



例　1200番地　beq $s0, $s1, L1

（ラベルL1は1264番地に対応するものとする）

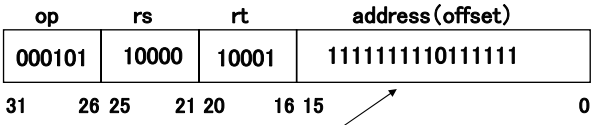


16進表記　：**1211000F**

分岐先は，このbeq命令の次の命令から15
命令先にあるので，オフセットは 15 となる
（1264−4−1200）/4=15 として計算できる

例　6400番地　bne $s0, $s1, L2

（ラベルL2は6144番地に対応するものとする）



16進表記　：**1611FFBF**

分岐先は，このbeq命令の次の命令から65
命令前にあるので，オフセットは −65となる
（6144−4−6400）/4=−65 として計算できる