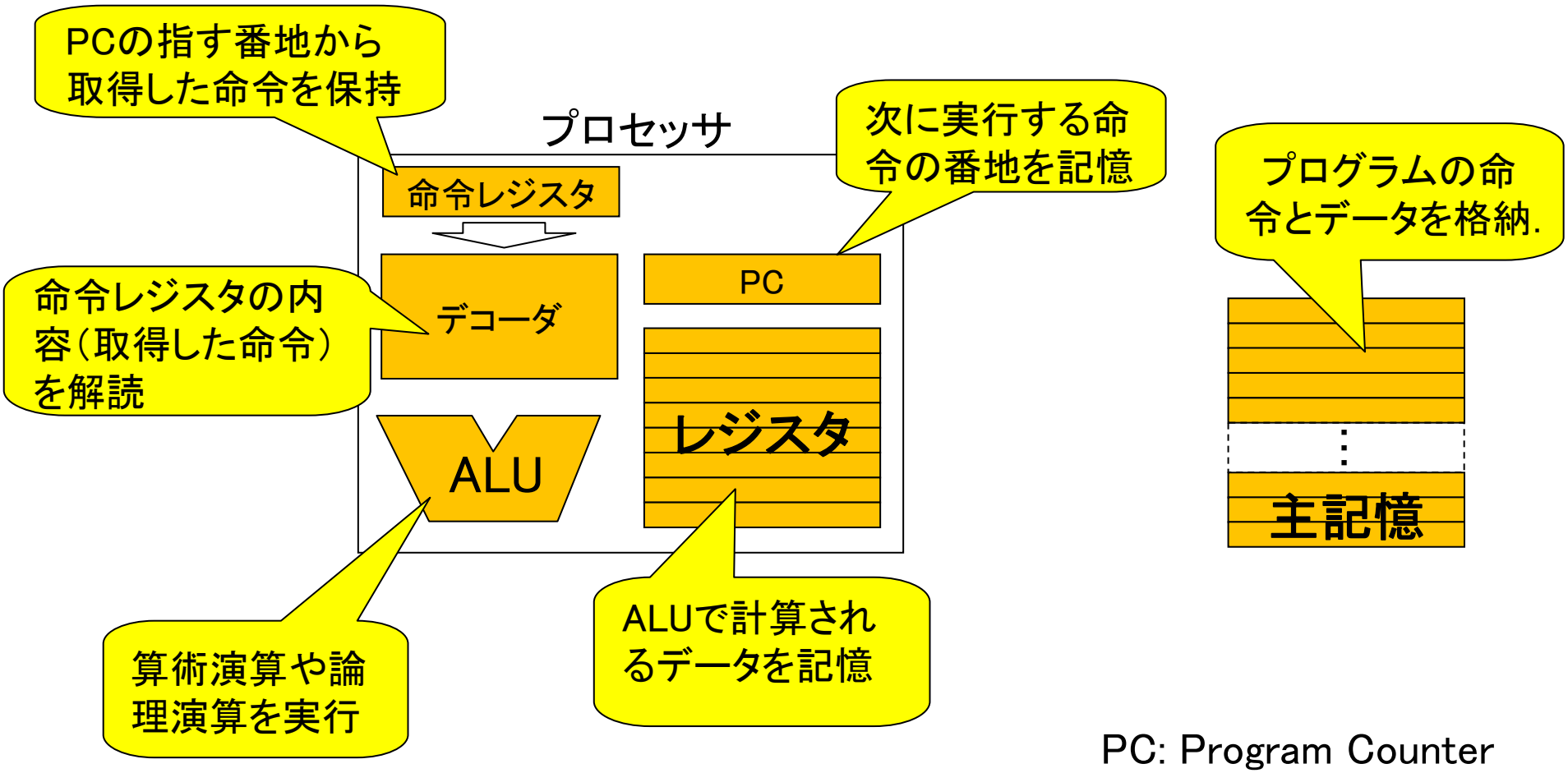
制御の流れ
（分岐命令）

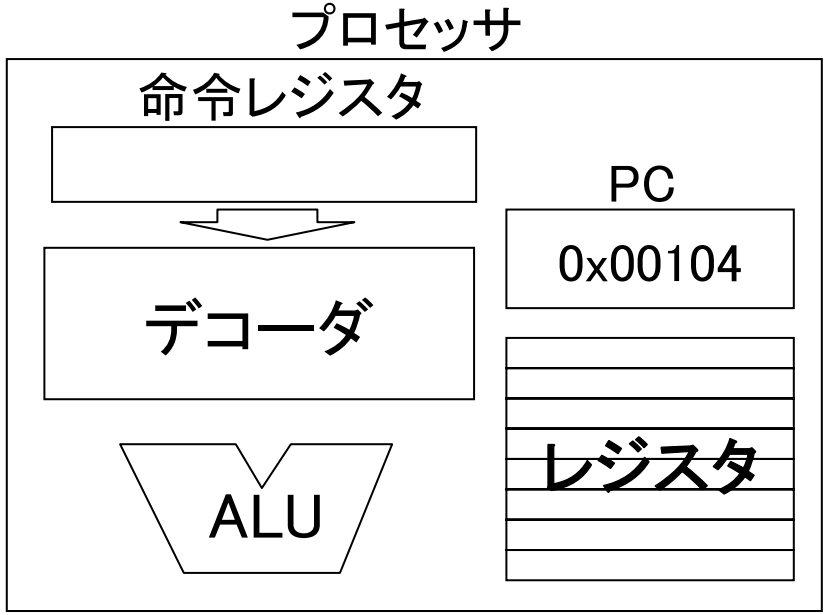
（教科書3.5節）

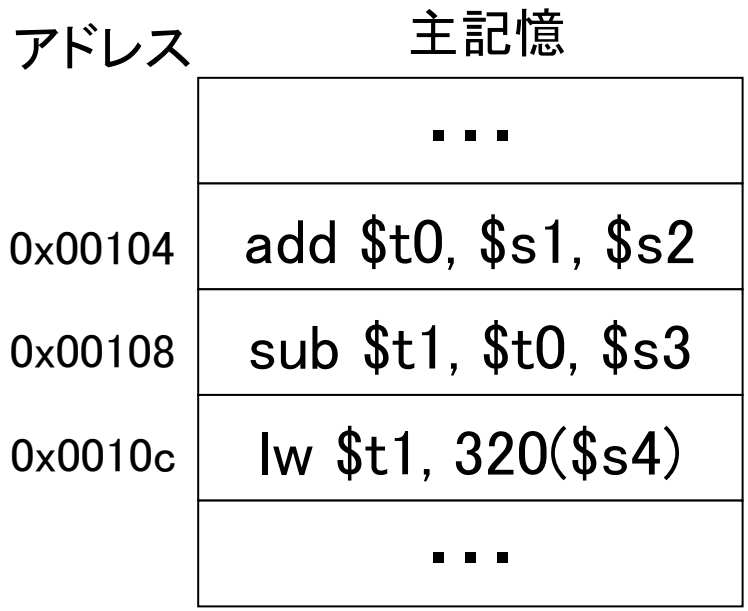
プロセッサの基本構成要素



PC: Program Counter

命令の実行手順





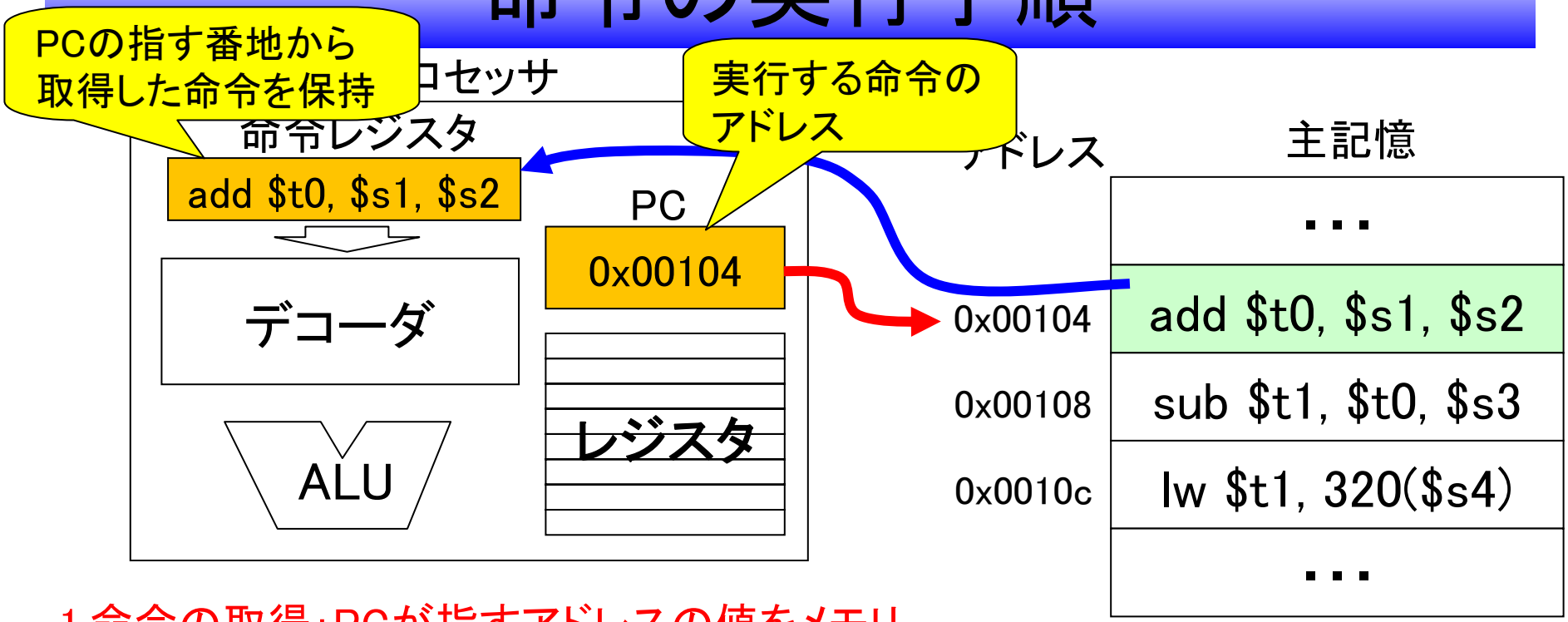
1.命令の取得：PCが指すアドレスの値をメモリから命令レジスタへ取得．同時に，次命令の取得に備えてPCの値を更新（例えば＋4）

2.命令の解読：命令レジスタの値をデコード

3.命令の実行：解読結果に従って実行

実際には，各命令は2進表現でメモリに格納されている

命令の実行手順



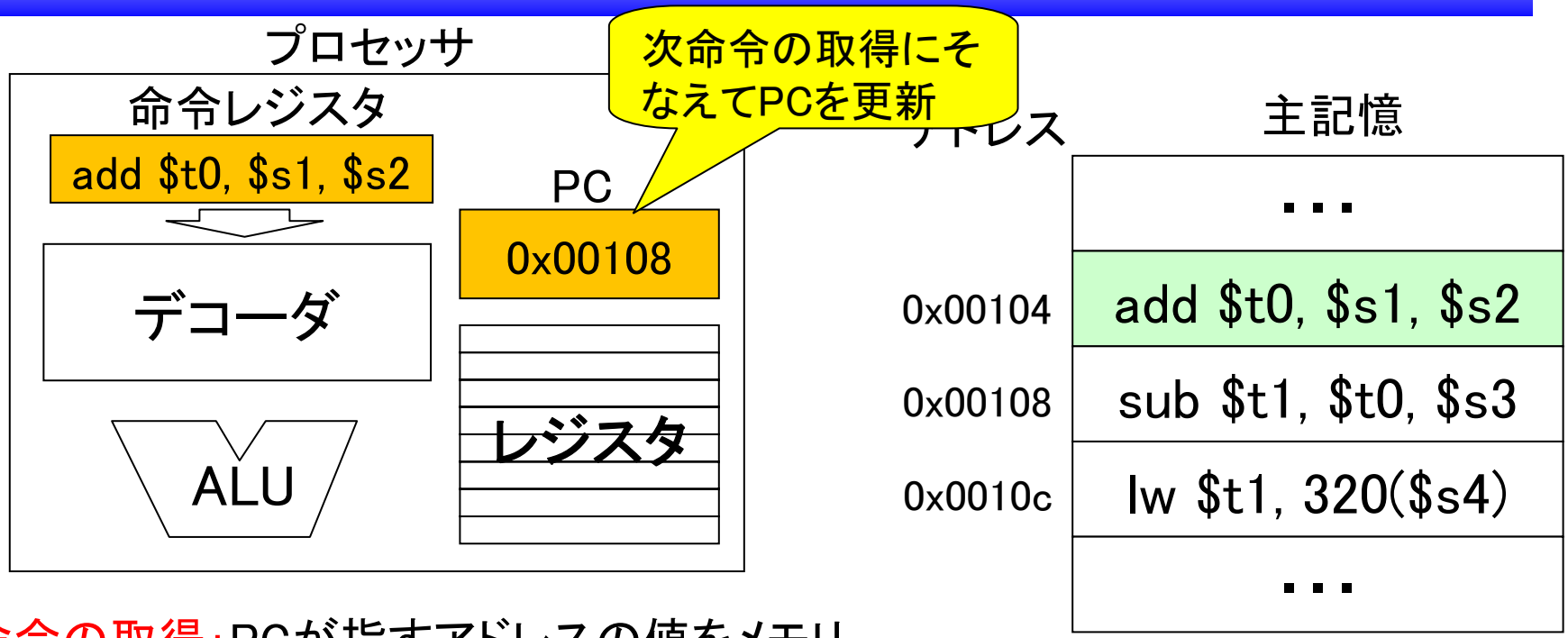
1.命令の取得：PCが指すアドレスの値をメモリから命令レジスタへ取得．同時に，次命令の取得に備えてPCの値を更新（例えば＋4）

2.命令の解読：命令レジスタの値をデコード

3.命令の実行：解読結果に従って実行

実際には，各命令は2進表現でメモリに格納されている

命令の実行手順



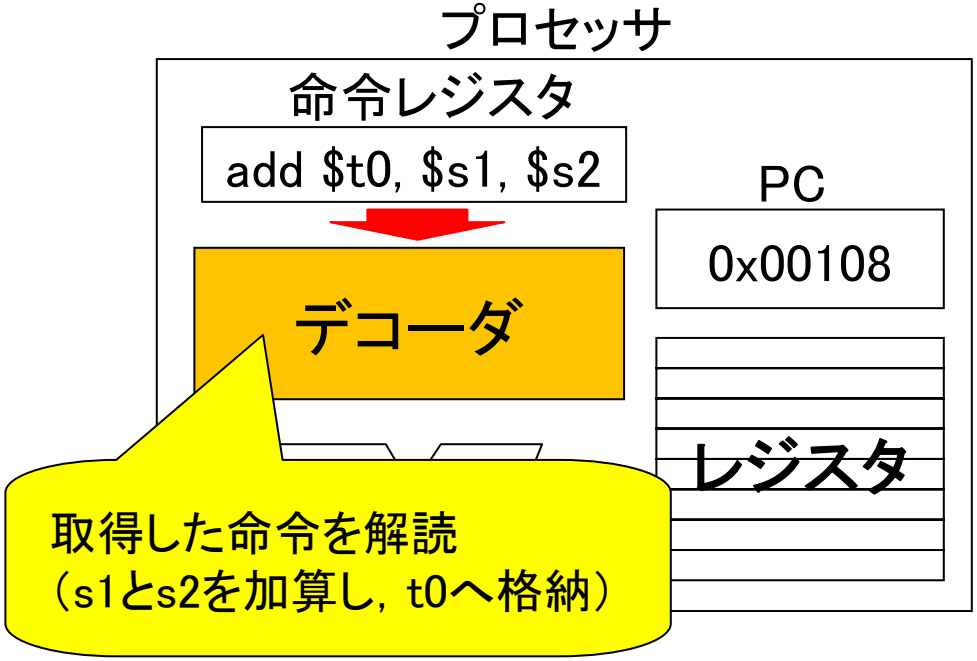
1.命令の取得：PCが指すアドレスの値をメモリから命令レジスタへ取得．同時に，次命令の取得に備えてPCの値を更新（例えば＋4）

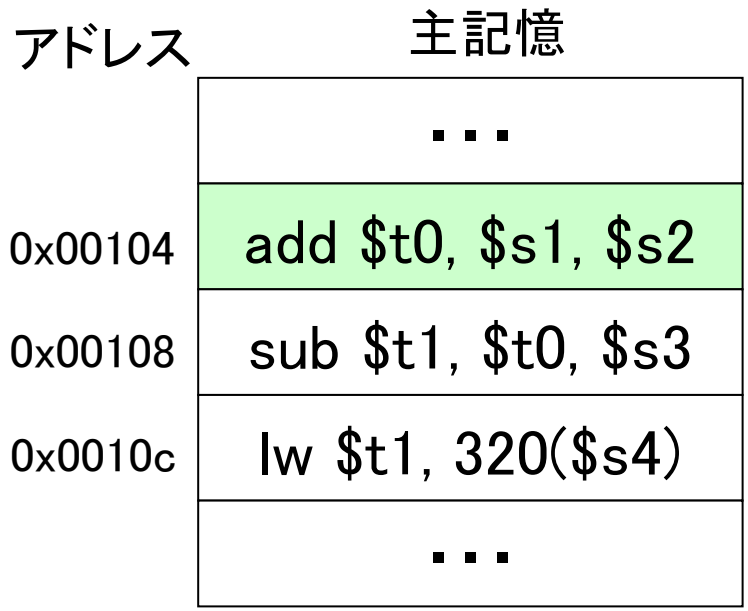
2.命令の解読：命令レジスタの値をデコード

3.命令の実行：解読結果に従って実行

実際には，各命令は2進表現でメモリに格納されている

命令の実行手順





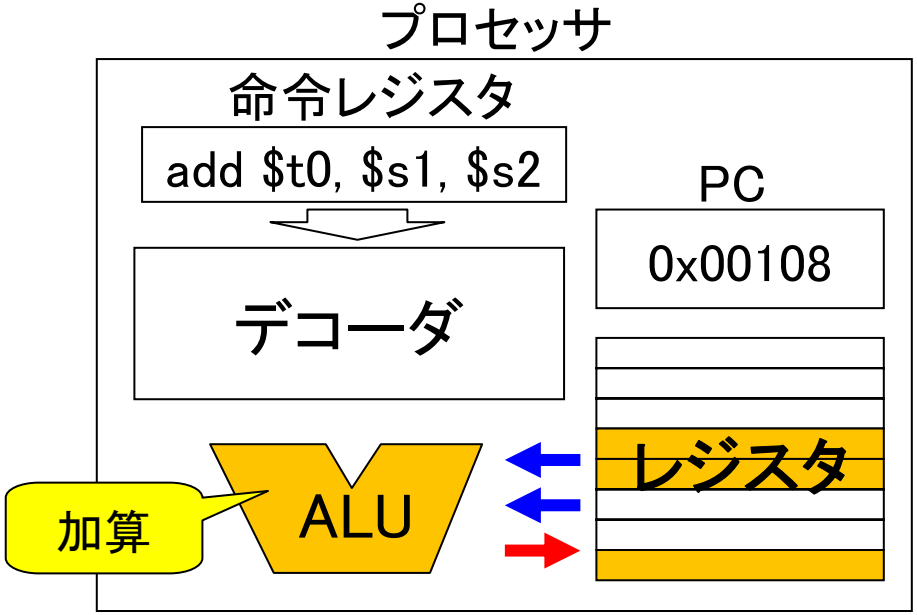
1.命令の取得：PCが指すアドレスの値をメモリから命令レジスタへ取得．同時に，次命令の取得に備えてPCの値を更新（例えば＋4）

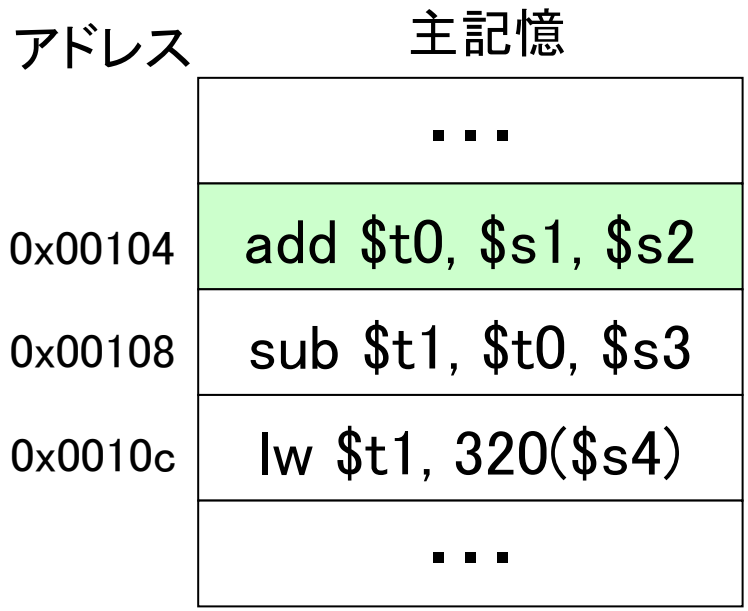
実際には，各命令は2進表現でメモリに格納されている

2.命令の解読：命令レジスタの値をデコード

3.命令の実行：解読結果に従って実行

命令の実行手順





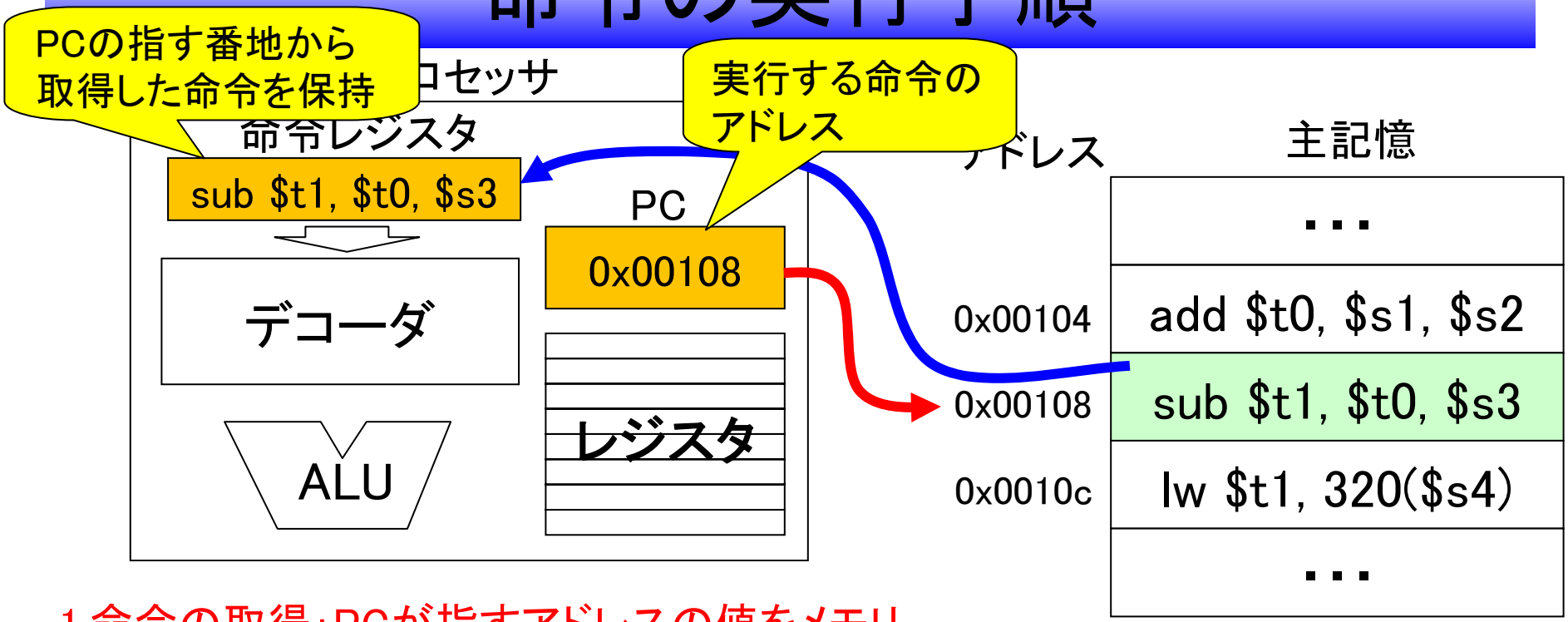
1.命令の取得：PCが指すアドレスの値をメモリから命令レジスタへ取得．同時に，次命令の取得に備えてPCの値を更新（例えば＋4）

2.命令の解読：命令レジスタの値をデコード

3.命令の実行：解読結果に従って実行

実際には，各命令は2進表現でメモリに格納されている

命令の実行手順



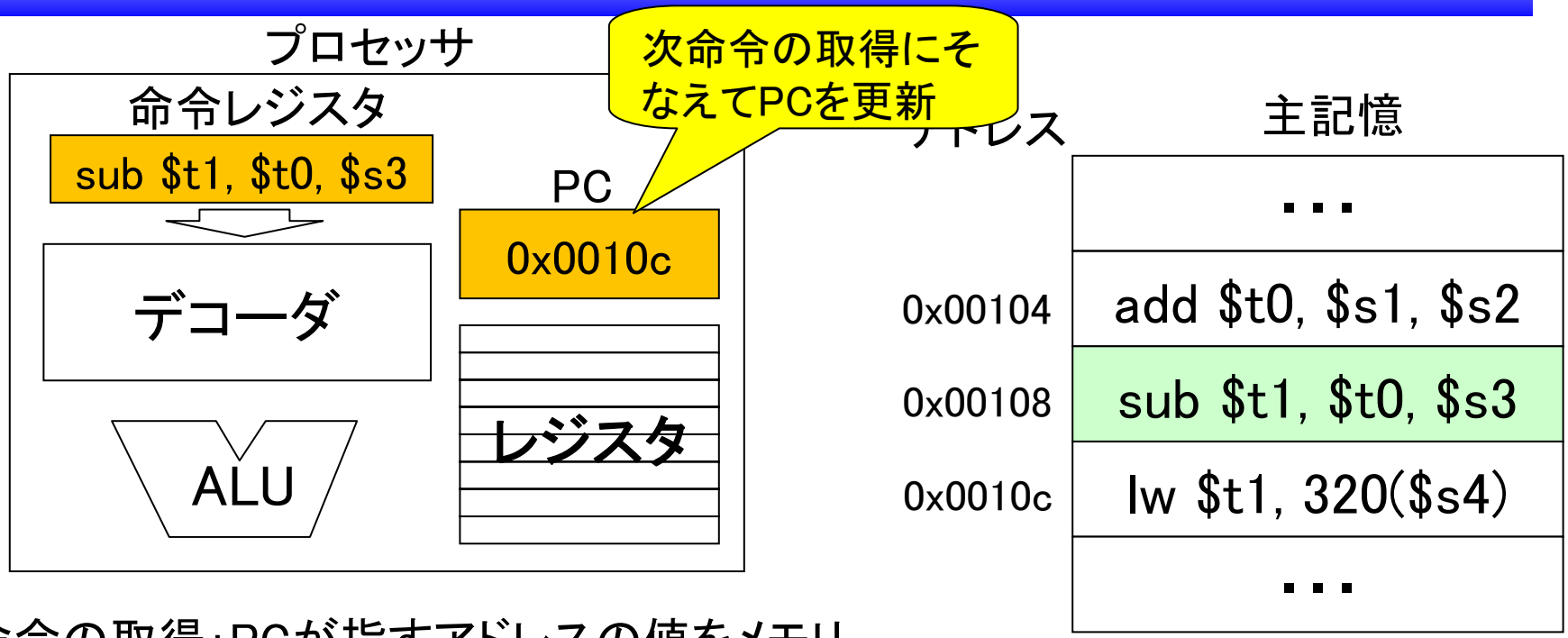
1.命令の取得：PCが指すアドレスの値をメモリから命令レジスタへ取得．同時に，次命令の取得に備えてPCの値を更新（例えば＋4）

2.命令の解読：命令レジスタの値をデコード

3.命令の実行：解読結果に従って実行

実際には，各命令は2進表現でメモリに格納されている

命令の実行手順



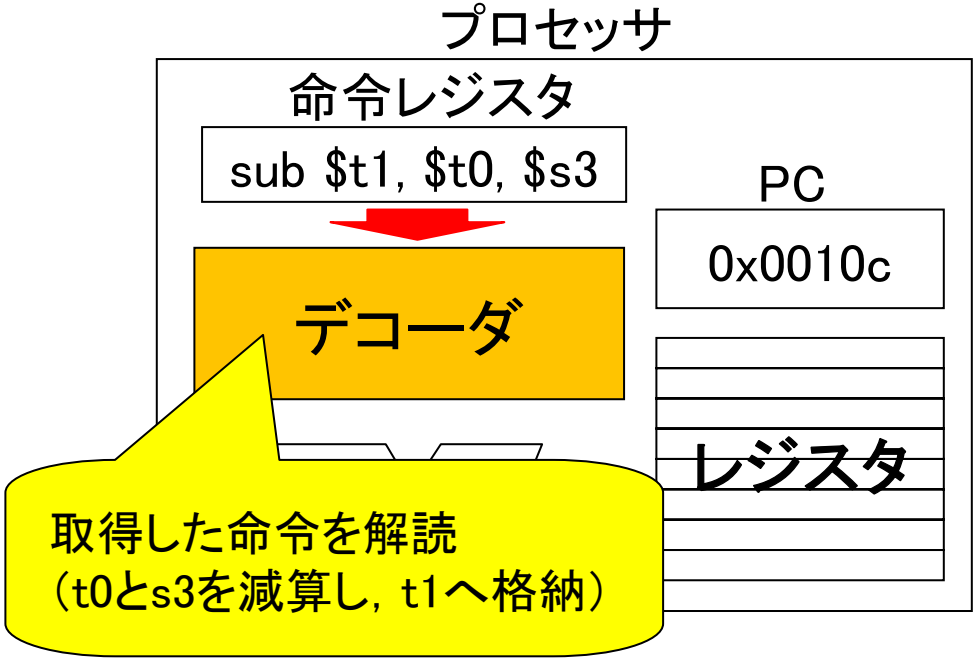
1.命令の取得：PCが指すアドレスの値をメモリから命令レジスタへ取得．同時に，次命令の取得に備えてPCの値を更新（例えば＋4）

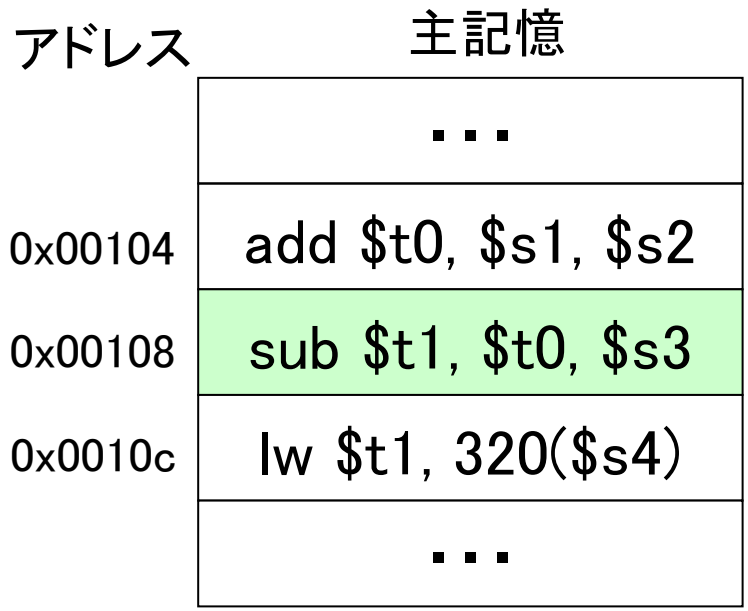
2.命令の解読：命令レジスタの値をデコード

3.命令の実行：解読結果に従って実行

実際には，各命令は2進表現でメモリに格納されている

命令の実行手順





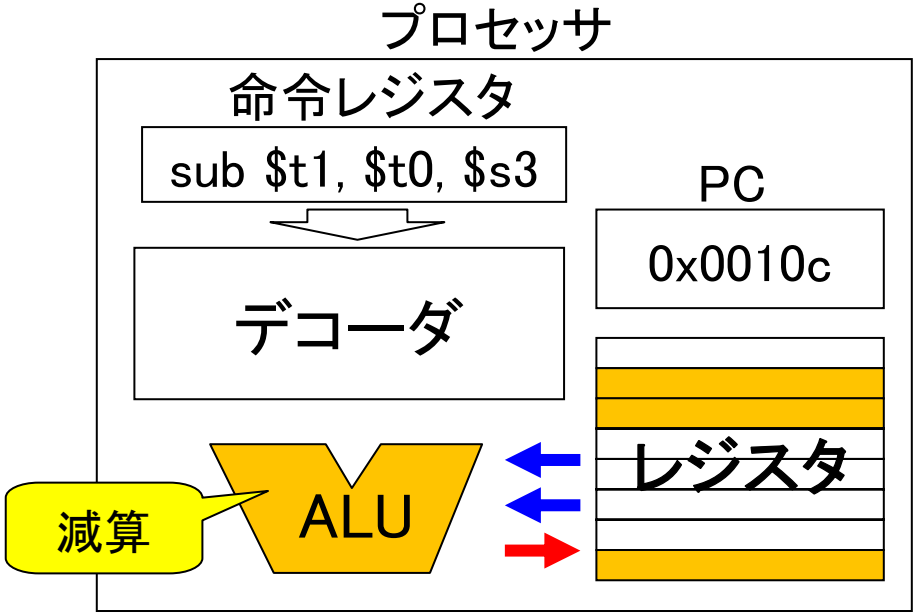
1.命令の取得:PCが指すアドレスの値をメモリから命令レジスタへ取得. 同時に, 次命令の取得に備えてPCの値を更新(例えば＋4)

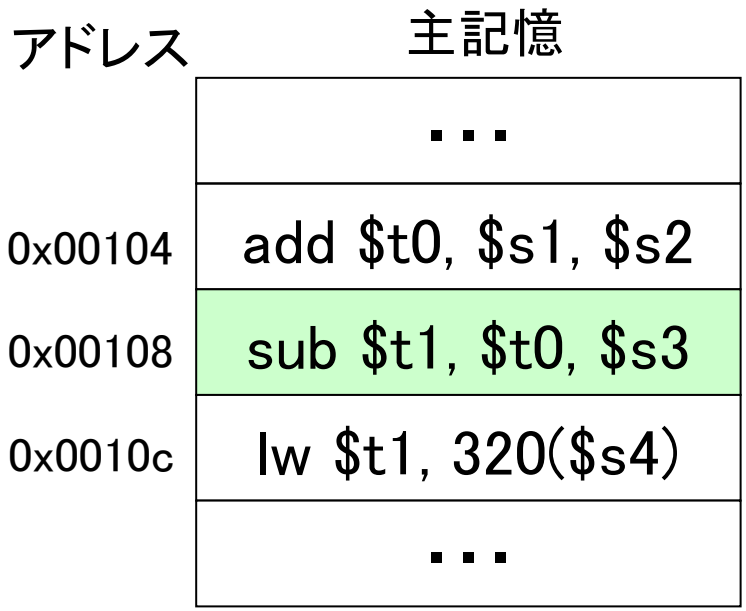
実際には, 各命令は2進表現でメモリに格納されている

2.命令の解読:命令レジスタの値をデコード

3.命令の実行:解読結果に従って実行

命令の実行手順





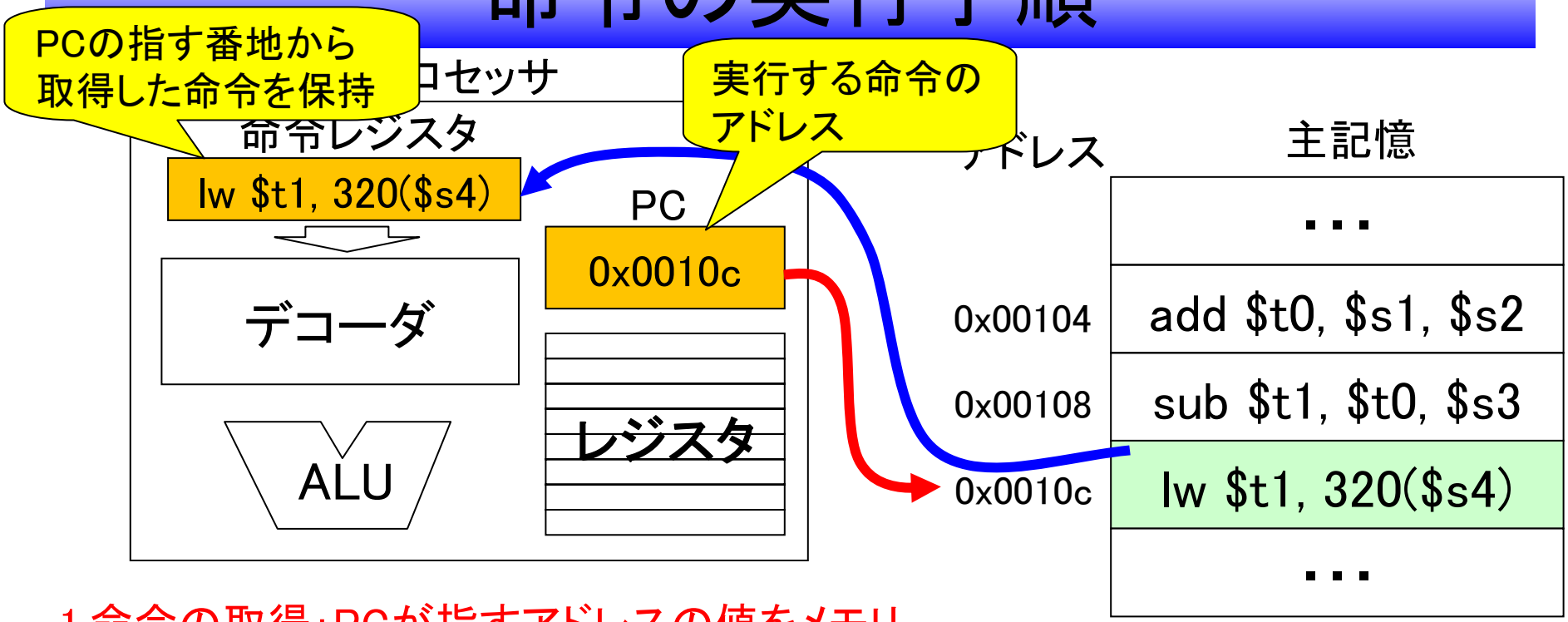
1.命令の取得：PCが指すアドレスの値をメモリから命令レジスタへ取得．同時に，次命令の取得に備えてPCの値を更新（例えば＋4）

2.命令の解読：命令レジスタの値をデコード

3.命令の実行：解読結果に従って実行

実際には，各命令は2進表現でメモリに格納されている

命令の実行手順



1.命令の取得：PCが指すアドレスの値をメモリから命令レジスタへ取得．同時に，次命令の取得に備えてPCの値を更新（例えば＋4）

2.命令の解読：命令レジスタの値をデコード

3.命令の実行：解読結果に従って実行

実際には，各命令は2進表現でメモリに格納されている

以降，繰り返し・・・

逐次制御だけで十分なのか？

- 命令実行に関する基本動作
  - 主記憶に格納された命令を1個ずつ，順番に実行する（逐次制御）
- 全ての場合に対応できるのだろうか？

```
A[80] = (g + A[2]) - (h + A[i]);
```

```
lw  $t0, 8 ($s4)
add $t0, $t0, $s1
add $t1, $s3, $s3
add $t1, $t1, $t1
add $t1, $t1, $s4
lw  $t1, 0 ($t1)
add $t1, $t1, $s2
sub $t1, $t0, $t1
sw  $t1, 320 ($s4)
```

逐次制御で問題なし！

```
if (i == j)
  f = g + h;
else
  f = g - h;
```

???

逐次制御だけでは対応できない！
条件に応じて分岐する必要がある！

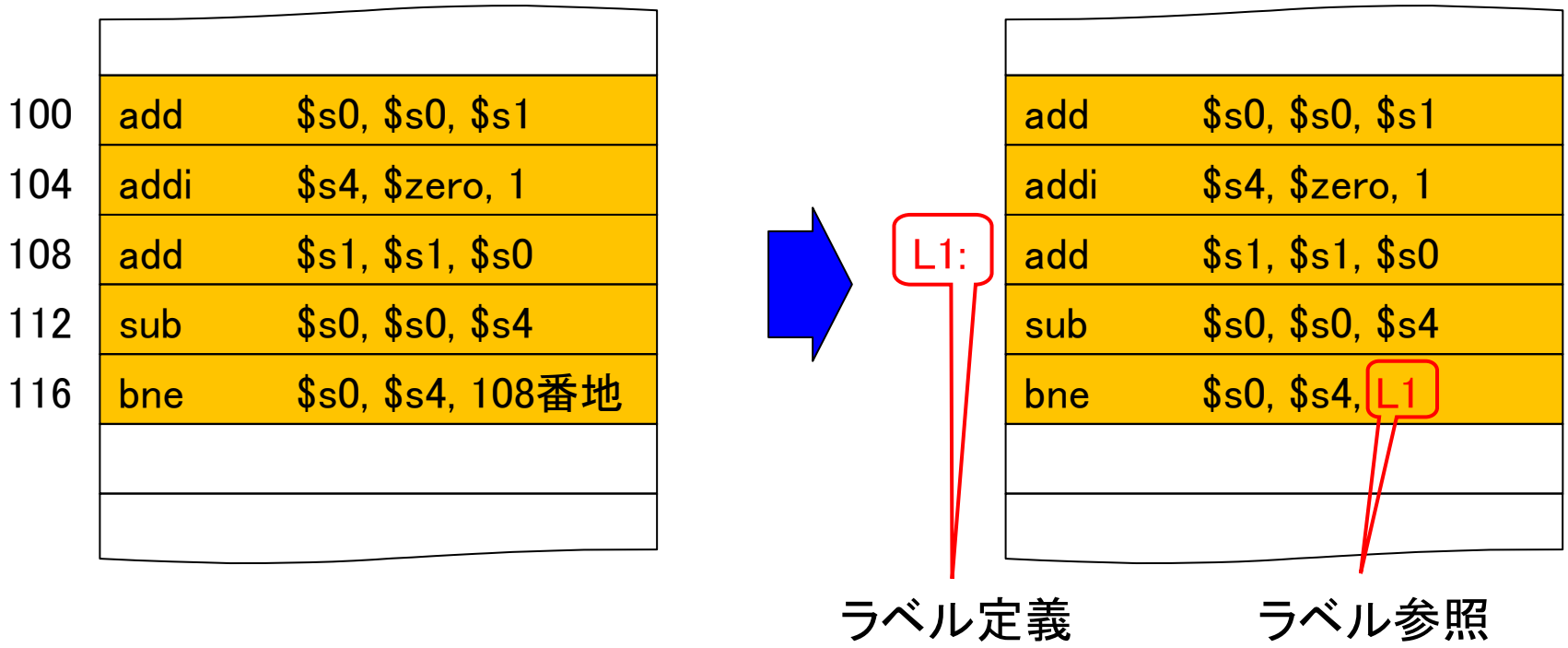
MIPSの命令（一部）

命令区分	命令	例	意味
算術演算	add	add $s1, $s2, $s3	$s1 = $s2 + $s3
	Subtract	sub $s1, $s2, $s3	$s1 = $s2 − $s3
データ転送	load word	lw $s1, 100($s2)	$s1に，メモリの[$s2+100]番地のワードデータを読込み
	store word	sw $s1, 100($s2)	メモリの[$s2+100]番地に，$s1のワードデータを書込み
条件分岐	branch on equal	beq $s1, $s2, L	もし、$s1==$s2ならLへ分岐
	branch on not equal	bne $s1, $s2, L	もし、$s1!=$s2ならLへ分岐
	set on less than	slt $s1, $s2, $s3	もし、$s2<$s3なら$s1=1，以外なら$s1=0
無条件ジャンプ	jump	j L	Lにジャンプ
	jump register	jr $s1	$s1の値が示すアドレスにジャンプ

$s1～$s3は汎用レジスタ

ラベル名

ラベル名: 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.



ラベル名はアセンブラによって実際の番地に変換される.

条件分岐命令(1)

条件分岐命令がない場合

① add $s1, $s2, $s3
② add $t0, $s1, $s4
③ lw $t1, 8 ($s5)
④ sub $t1, $t1, $t0
⋮

条件分岐命令がある場合

① add $s1, $s2, $s3 ①
② beq $s1, $s6, GoTo ②
add $t0, $s2, $s4 ③
sw $t0, 8 ($s5) ④
⋮

GoTo:
③ add $s4, $t4, $t3
④ sw $s4, 8 ($s5)

命令実行の流れ

②で分岐条件が成立した場合

分岐条件が不成立の場合

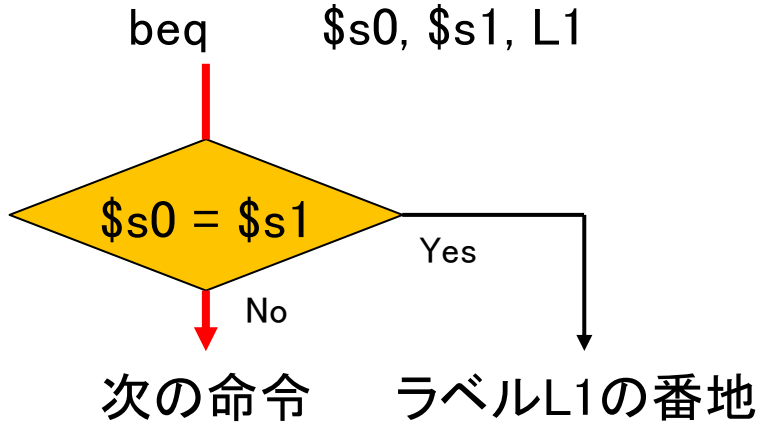
条件分岐命令(2)

例)

beq $s0, $s1, L1

レジスタ$s0の値と$s1の値を比較して，同じであればラベルL1へ分岐(PCの値を設定)

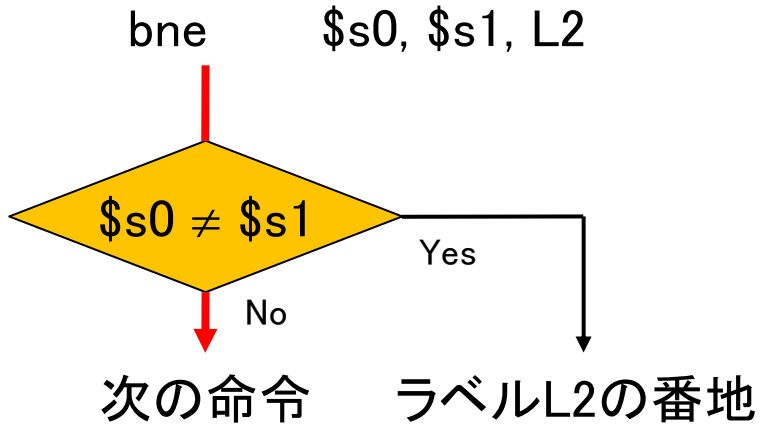
beq: Branch on EQual



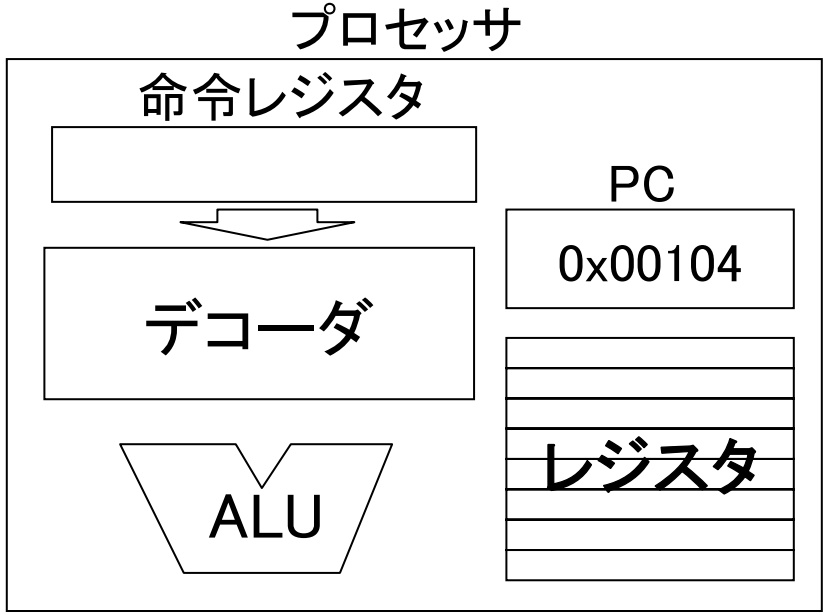
bne $s0, $s1, L1

レジスタ$s0の値と$s1の値を比較して，同じでなければラベルL1へ分岐(PCの値を設定)

bne: Branch on Not Equal



条件分岐命令(3)



アドレス	主記憶
	...
0x00104	beq $t0, $s1, L1
0x00108	add $t1, $t0, $s3
0x0010c	lw $t1, 320($s4)
	...
L1 0x00400	lw $t1, 120($s4)
	...

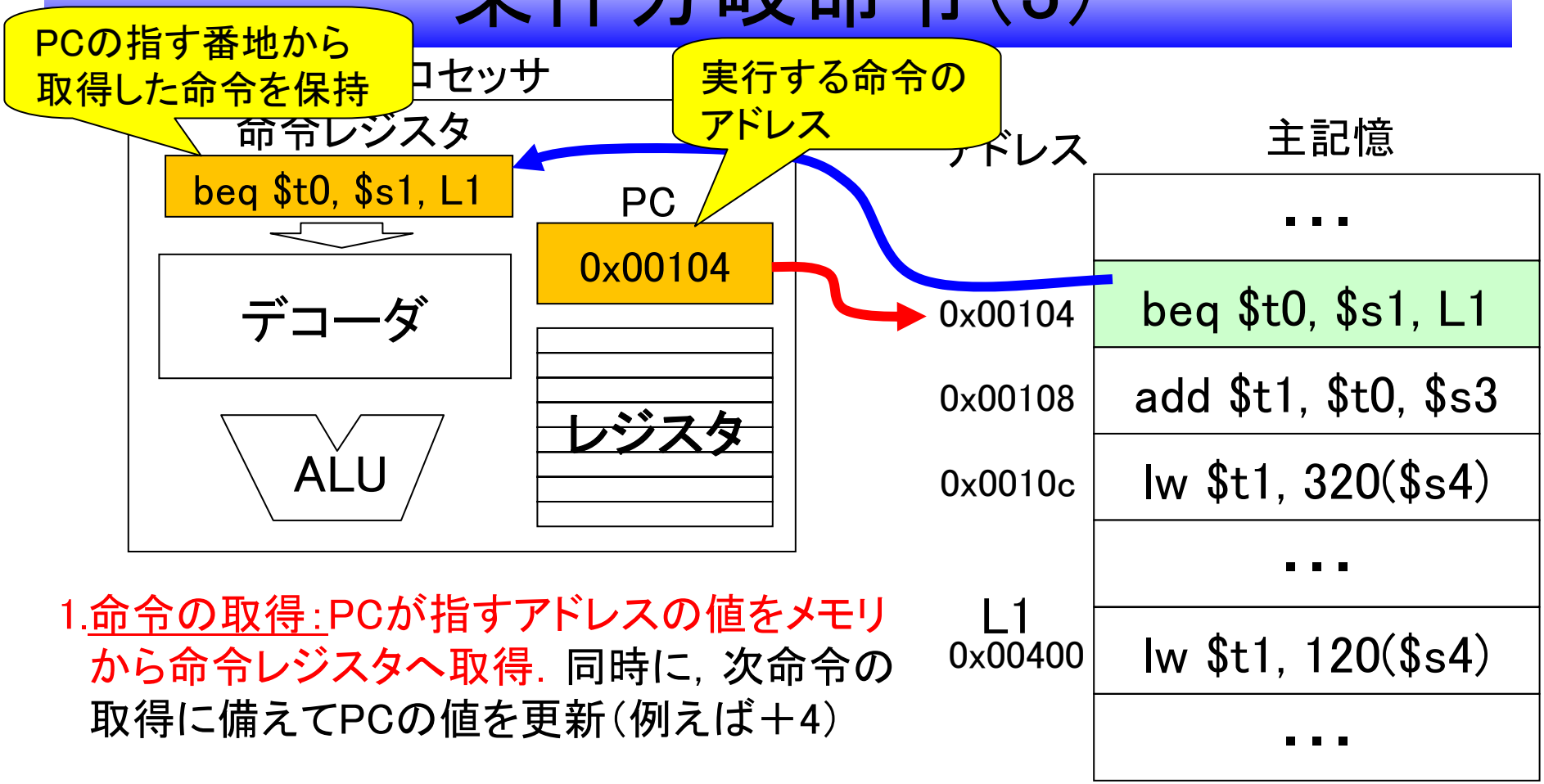
1.命令の取得:PCが指すアドレスの値をメモリから命令レジスタへ取得．同時に，次命令の取得に備えてPCの値を更新(例えば＋4)

2.命令の解読:命令レジスタの値をデコード

3.命令の実行:解読結果に従って実行

実際には，各命令は2進表現でメモリに格納されている

条件分岐命令(3)



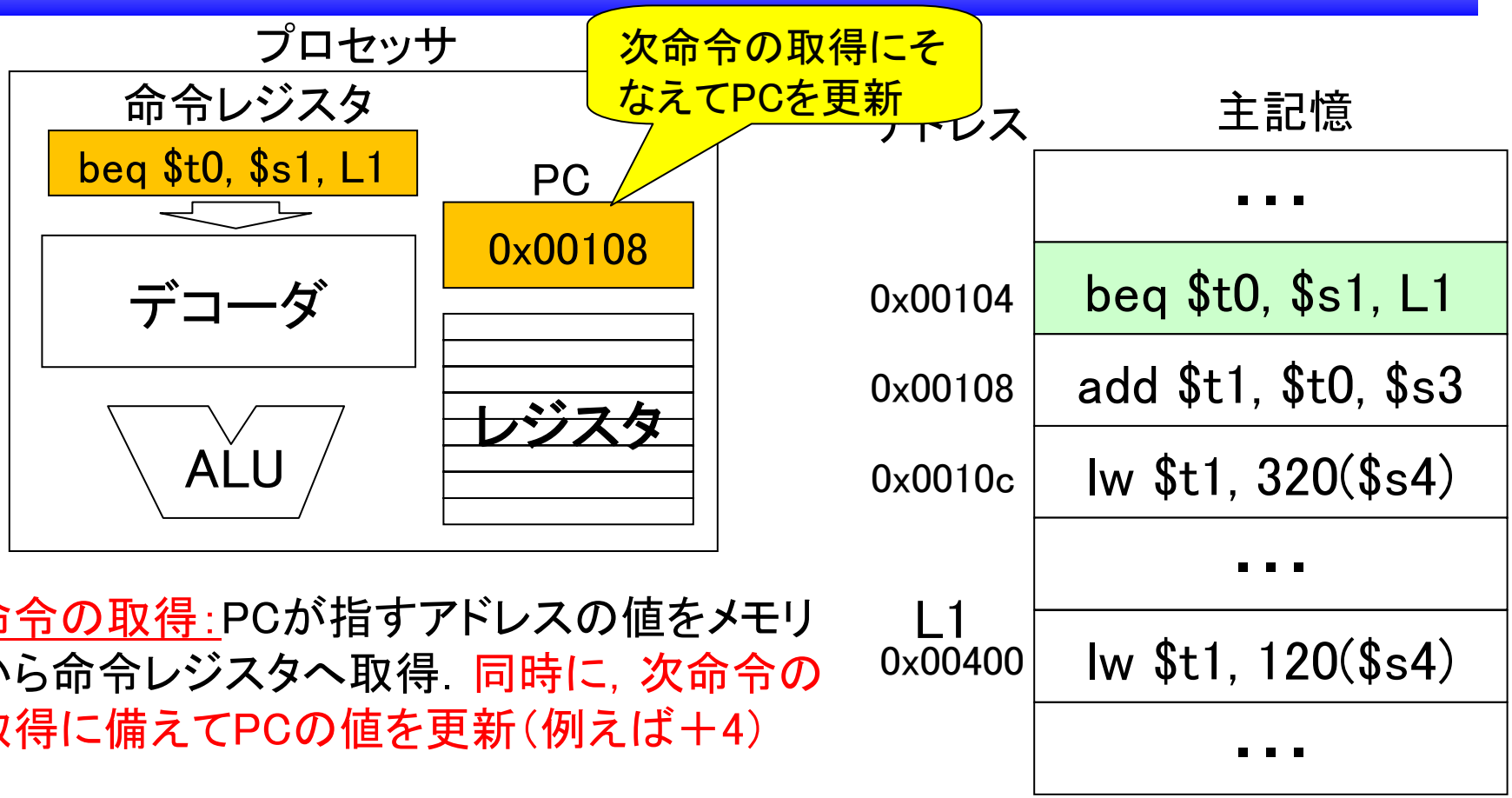
1.命令の取得:PCが指すアドレスの値をメモリから命令レジスタへ取得. 同時に, 次命令の取得に備えてPCの値を更新(例えば＋4)

2.命令の解読:命令レジスタの値をデコード

3.命令の実行:解読結果に従って実行

実際には, 各命令は2進表現でメモリに格納されている

条件分岐命令(3)



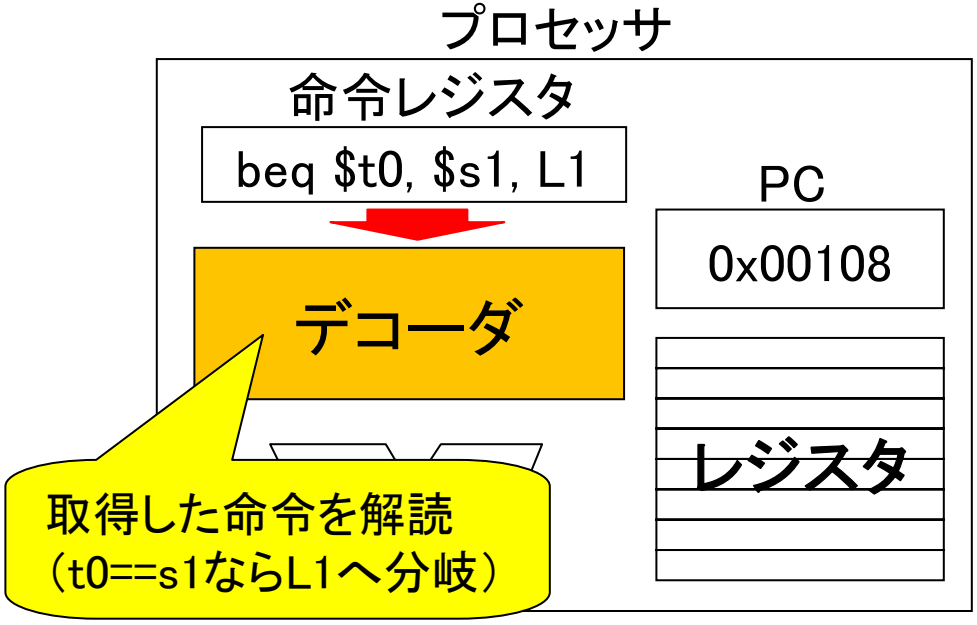
1.命令の取得:PCが指すアドレスの値をメモリから命令レジスタへ取得. 同時に, 次命令の取得に備えてPCの値を更新(例えば＋4)

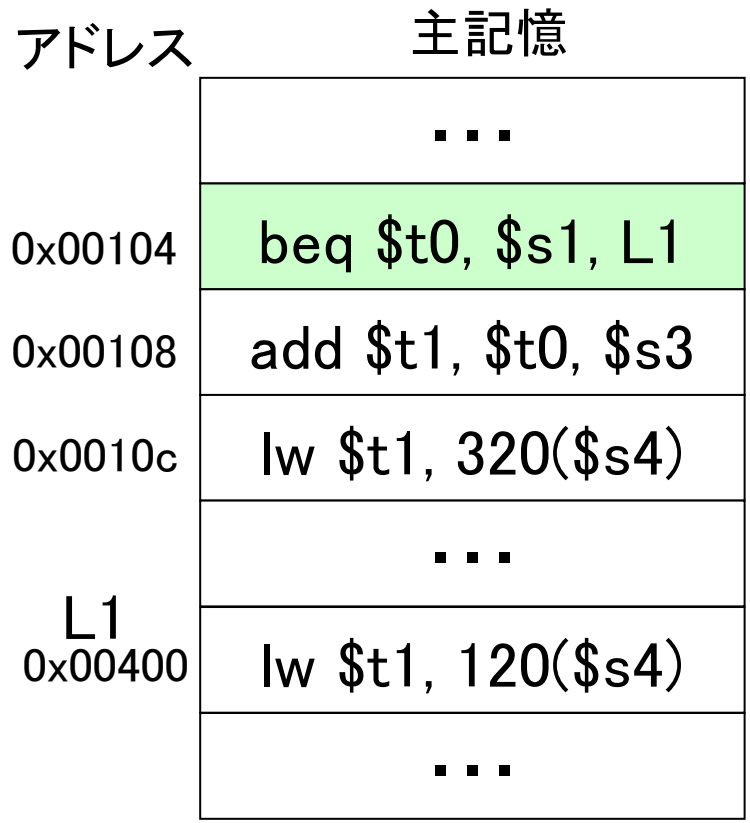
2.命令の解読:命令レジスタの値をデコード

3.命令の実行:解読結果に従って実行

実際には, 各命令は2進表現でメモリに格納されている

条件分岐命令(3)





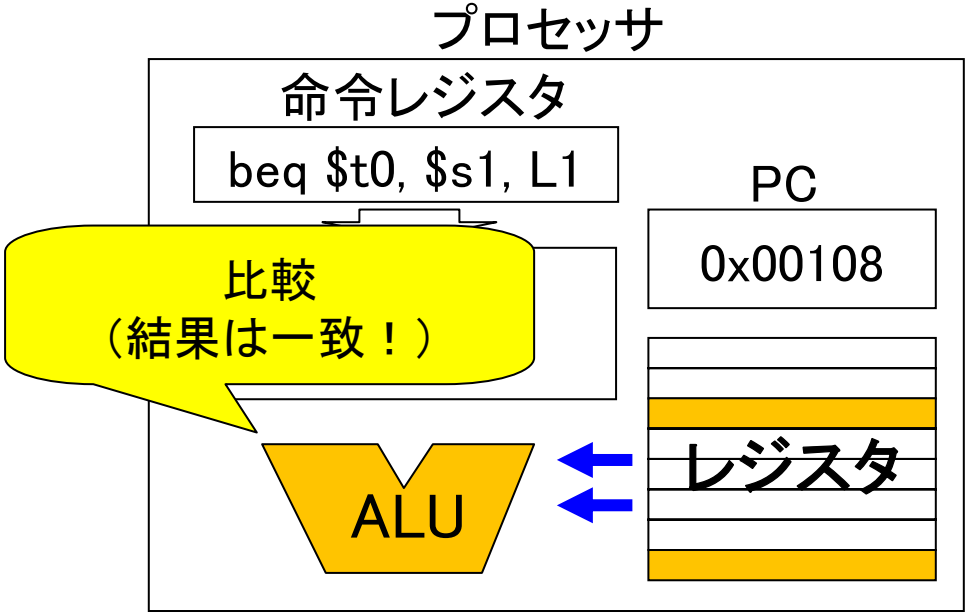
1.命令の取得:PCが指すアドレスの値をメモリから命令レジスタへ取得．同時に，次命令の取得に備えてPCの値を更新(例えば＋4)

2.命令の解読:命令レジスタの値をデコード

3.命令の実行:解読結果に従って実行

実際には，各命令は2進表現でメモリに格納されている

条件分岐命令(3)



アドレス	主記憶
	...
0x00104	beq $t0, $s1, L1
0x00108	add $t1, $t0, $s3
0x0010c	lw $t1, 320($s4)
	...
L1 0x00400	lw $t1, 120($s4)
	...

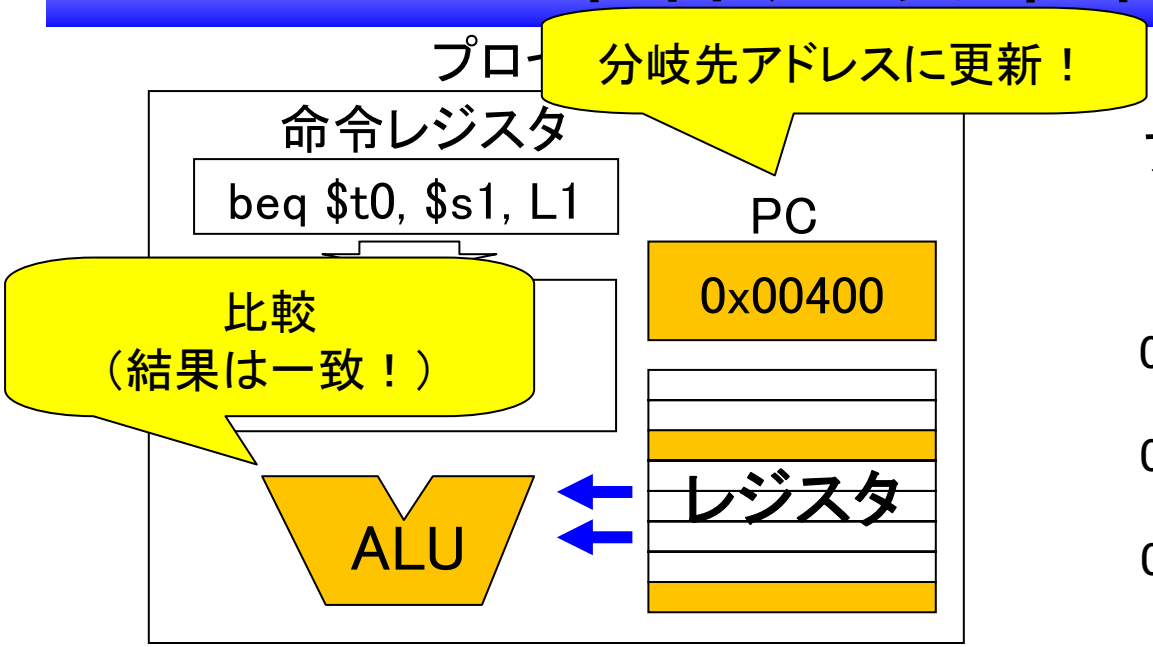
1.命令の取得：PCが指すアドレスの値をメモリから命令レジスタへ取得．同時に，次命令の取得に備えてPCの値を更新（例えば＋4）

2.命令の解読：命令レジスタの値をデコード

3.命令の実行：解読結果に従って実行

実際には，各命令は2進表現でメモリに格納されている

条件分岐命令(3)



アドレス	主記憶
	...
0x00104	beq $t0, $s1, L1
0x00108	add $t1, $t0, $s3
0x0010c	lw $t1, 320($s4)
	...
L1 0x00400	lw $t1, 120($s4)
	...

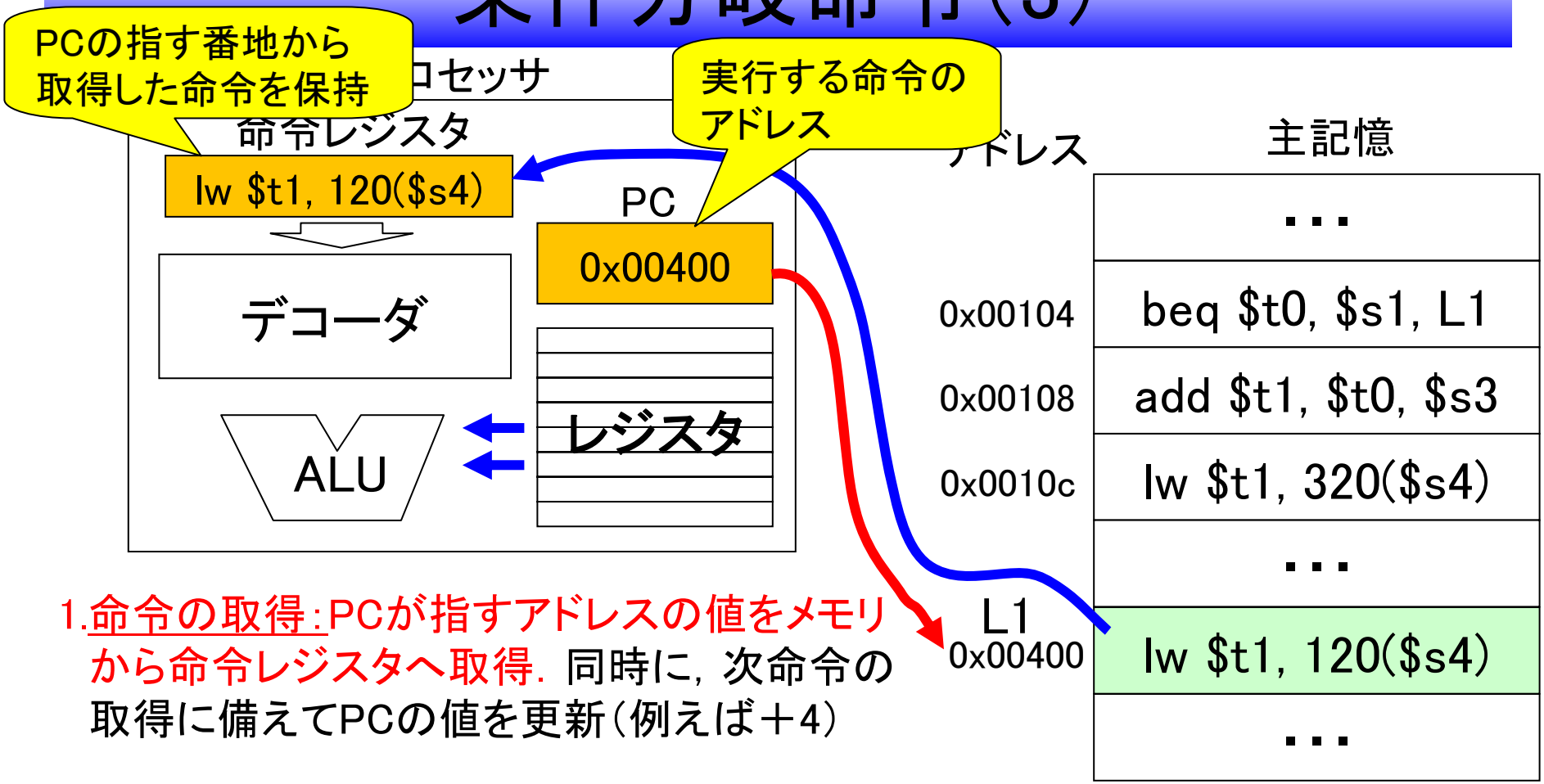
1.命令の取得：PCが指すアドレスの値をメモリから命令レジスタへ取得．同時に，次命令の取得に備えてPCの値を更新（例えば＋4）

2.命令の解読：命令レジスタの値をデコード

3.命令の実行：解読結果に従って実行

実際には，各命令は2進表現でメモリに格納されている

条件分岐命令(3)



1.命令の取得:PCが指すアドレスの値をメモリから命令レジスタへ取得. 同時に, 次命令の取得に備えてPCの値を更新(例えば＋4)

2.命令の解読:命令レジスタの値をデコード

3.命令の実行:解読結果に従って実行

実際には, 各命令は2進表現でメモリに格納されている

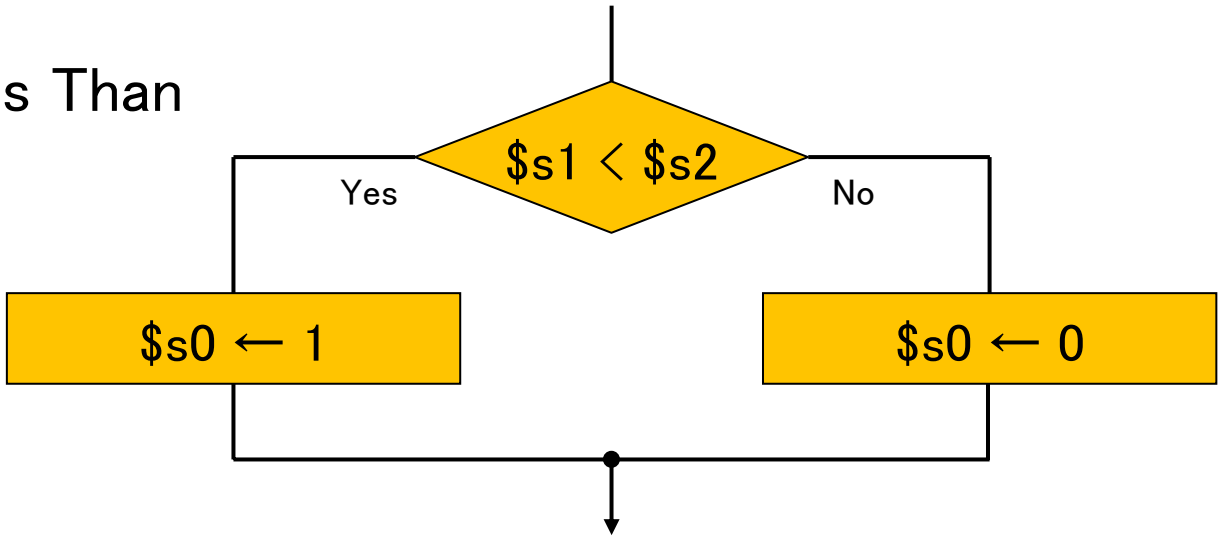
条件分岐命令(4)

例)

slt　$s0, $s1, $s2

レジスタ$s1の値と$s2の値を比較して，$s1<$s2であれば$s0に値「1」を，そうでなければ値「0」を格納(分岐条件の設定に利用)

slt: Set on Less Than



無条件分岐命令(1)

分岐命令がない場合

① add $s1, $s2, $s3
② add $t0, $s1, $s4
③ lw $t1, 8 ($s5)
④ sub $t1, $t1, $t0
⋮

無条件分岐命令がある場合

① add $s1, $s2, $s3
② j GoTo
add $t0, $s2, $s4
sw $t0, 8 ($s5)
⋮

GoTo:
③ add $s4, $t4, $t3
④ sw $s4, 8 ($s5)

命令実行の流れ

常に分岐先を実行

無条件分岐命令(2)

例)

j　L1

常にラベルL1へ分岐(PCの値を設定)

j: Jump

jr　$s0

常に$s0に格納されている番地へ分岐(PCの値を設定)

jr: Jump Register

アセンブリ言語プログラムを書いてみよう！

問題：以下，4つの場合に関して，slt命令，beq命令，または，bne命令を用いてアセンブリ言語プログラムを作成せよ．なお，常にゼロの値を保持するレジスタ（$zero）を用いて良い．

①「$s0 < $s1」ならラベルLへ分岐
②「$s0 > $s1」ならラベルLへ分岐
③「$s0 <= $s1」ならラベルLへ分岐
④「$s0 >= $s1」ならラベルLへ分岐

アセンブリ言語プログラムを書いてみよう！

問題：以下，4つの場合に関して，slt命令，beq命令，または，bne命令を用いてアセンブリ言語プログラムを作成せよ．なお，常にゼロの値を保持するレジスタ（$zero）を用いて良い．

①「$s0 < $s1」ならラベルLへ分岐
②「$s0 > $s1」ならラベルLへ分岐
③「$s0 <= $s1」ならラベルLへ分岐
④「$s0 >= $s1」ならラベルLへ分岐

解答①

$s0<$s1なら$t0←1，
それ以外なら$t0←0

```
slt     $t0, $s0, $s1
bne     $t0, $zero, L
```

$t0≠$Zero（つまり$t0=1）
ならLへ分岐

解答②

$s1<$s0なら$t0←1，
それ以外なら$t0←0

```
slt     $t0, $s1, $s0
bne     $t0, $zero, L
```

$t0≠$Zero（つまり$t0=1）
ならLへ分岐

アセンブリ言語プログラムを書いてみよう！

問題：以下，4つの場合に関して，slt命令，beq命令，または，bne命令を用いてアセンブリ言語プログラムを作成せよ．なお，常にゼロの値を保持するレジスタ（$zero）を用いて良い．

①「$s0 < $s1」ならラベルLへ分岐
②「$s0 > $s1」ならラベルLへ分岐
③「$s0 <= $s1」ならラベルLへ分岐　⇒$s0>$s1でない
④「$s0 >= $s1」ならラベルLへ分岐　⇒$s0<$s1でない

解答③

$s1<$s0なら$t0←1，
それ以外なら$t0←0

```
slt     $t0, $s1, $s0
beq     $t0, $zero, L
```

$t0＝$Zero（つまり$t0=0）
ならLへ分岐

解答④

$s0<$s1なら$t0←1，
それ以外なら$t0←0

```
slt     $t0, $s0, $s1
beq     $t0, $zero, L
```

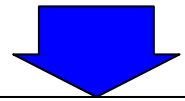
$t0＝$Zero（つまり$t0=0 ）
ならLへ分岐

（2006年度）

コード例: C言語の goto 文

- ラベル名で指定するところに飛ぶ.

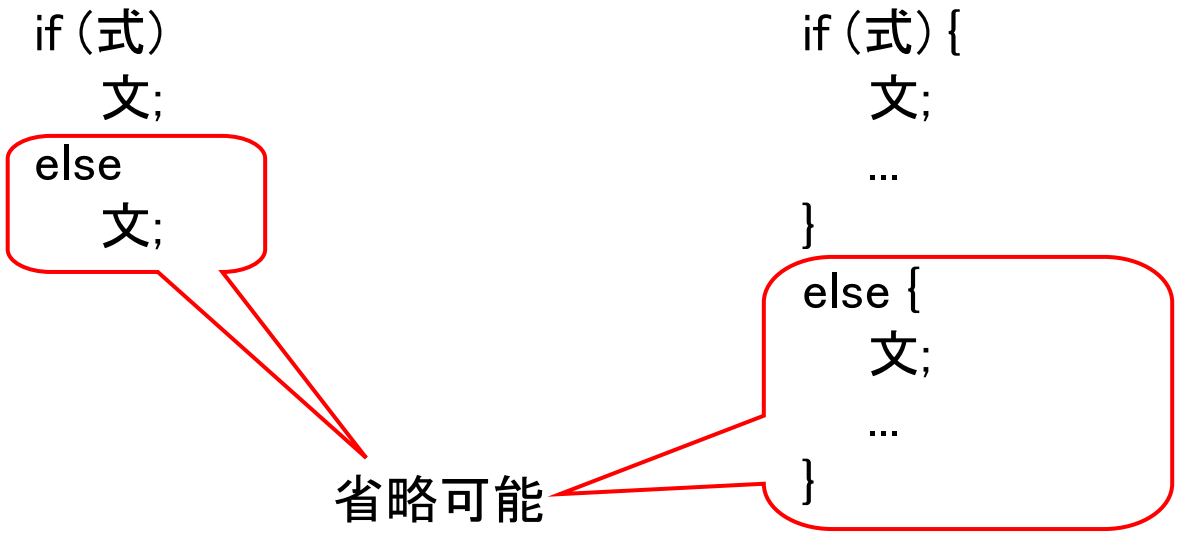
```
            …
            goto L100;
            …
L100:       xxxx
```



```
            j       L100
            …
L100:       xxxx
```

コード例: C言語の if 文(1)

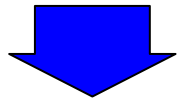
- 式の値が真(0以外の整数)ならば else の前を実行する. 式の値が偽(0)ならば else の後を実行する. 文が複数ある場合は中括弧でくくる.



- 式の値が偽のときに特にすることがなければ, else 以降は省略してもよい.

コード例: C言語の if 文(2)

```
        if (i == j) goto L1;
        f = g + h;
L1:     f = f - i;
```

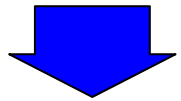


```
        beq     $s3, $s4, L1
        add     $s0, $s1, $s2
L1:     sub     $s0, $s0, $s3
```

変数名	レジスタ
f	$s0
g	$s1
h	$s2
i	$s3
j	$s4

コード例: C言語の if 文(3)

```
if (i == j)
        f = g + h;
else
        f = g - h;
```

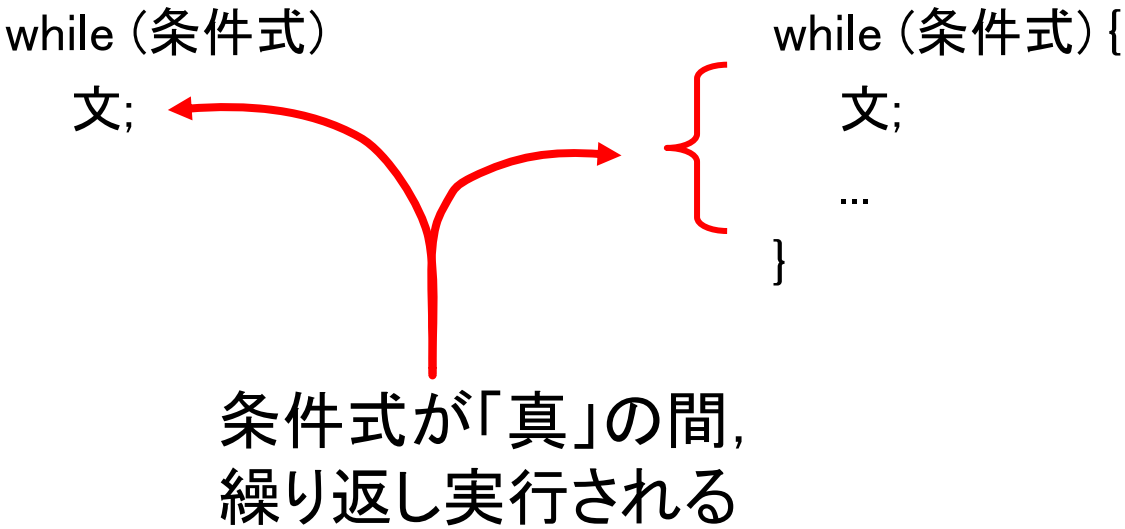


```
        bne     $s3, $s4, ELSE
        add     $s0, $s1, $s2
        j       EXIT
ELSE:   sub     $s0, $s1, $s2
EXIT:   …
```

変数名	レジスタ
f	$s0
g	$s1
h	$s2
i	$s3
j	$s4

コード例: C言語の while 文（1）

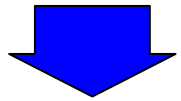
1. 条件式を評価する.
2. 条件式の値が真（0以外の整数）ならば文または中括弧の中を実行する. 偽（0）ならば文または中括弧の中を実行せずに次に進む（＝ループを終了する）.
3. 文または中括弧の中を実行した後は 1. に戻る.



コード例: C言語の while 文(2)

```
while (save[i] == k)
        i = i + j;
```

save[i] の番地 = $s6 + $s3 * 4

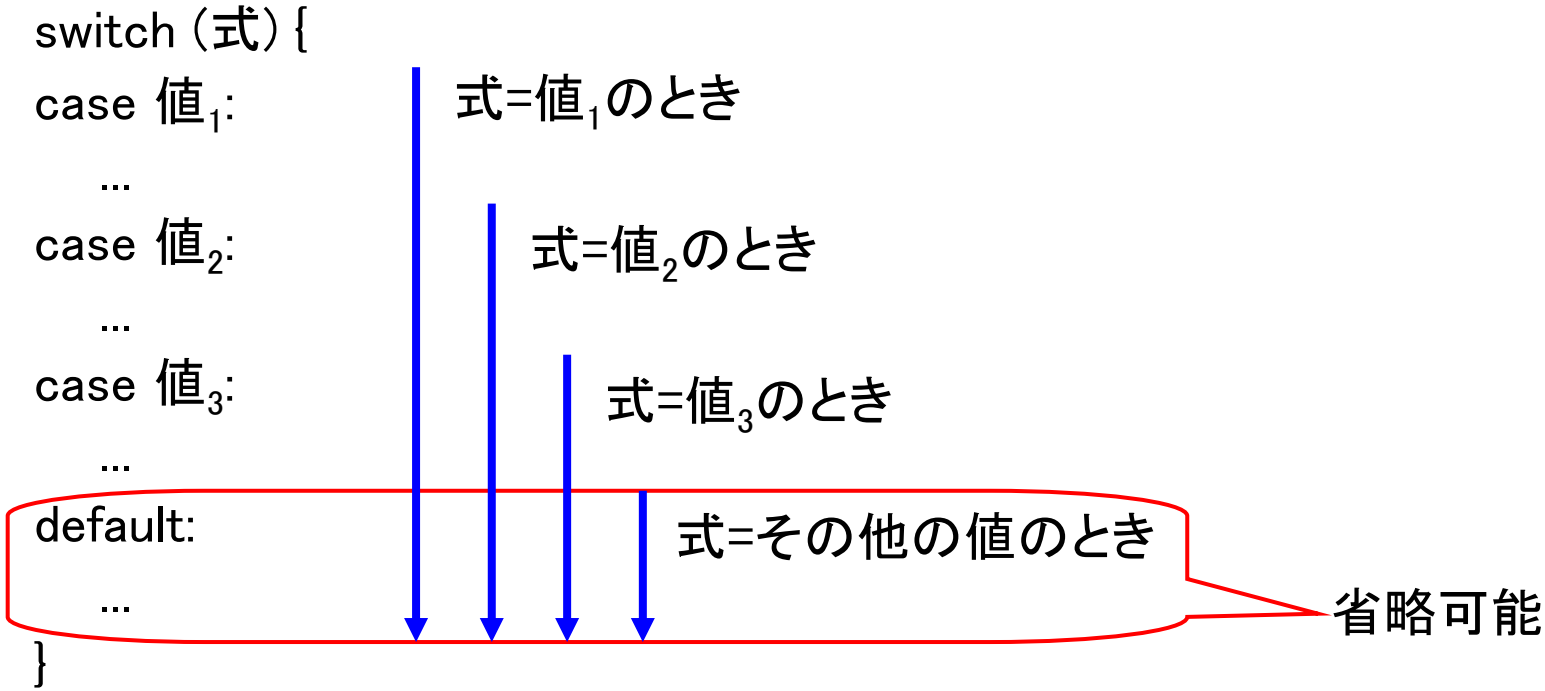


```
WHILE:  add     $t0, $s3, $s3
        add     $t0, $t0, $t0
        add     $t0, $t0, $s6
        lw      $t1, 0($t0)
        bne     $t1, $s5, EXIT
        add     $s3, $s3, $s4
        j       WHILE
EXIT:   …
```

変数名	レジスタ
i	$s3
j	$s4
k	$s5
save[] 先頭番地	$s6

コード例: C言語の switch 文(1)

- 式の値と同じ値の case のところに飛ぶ. 該当する case がない場合は default のところに飛ぶ.

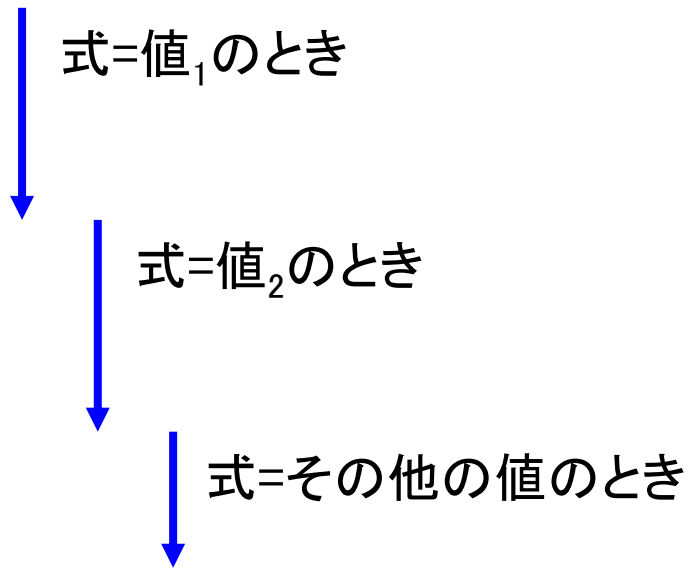


- default 以降は不要ならば省略してもよい.

コード例: C言語の switch 文(2)

- 通常，switch 文の各 case のあとには break 文を入れる．
- switch 文の中括弧の中で break 文を実行すると，中括弧の残りの実行を打ち切り，switch 文の外に飛ぶ．

```
switch (式) {
case 値1:
    ...
    break;
case 値2:
    ...
    break;
default:
    ...
}
```

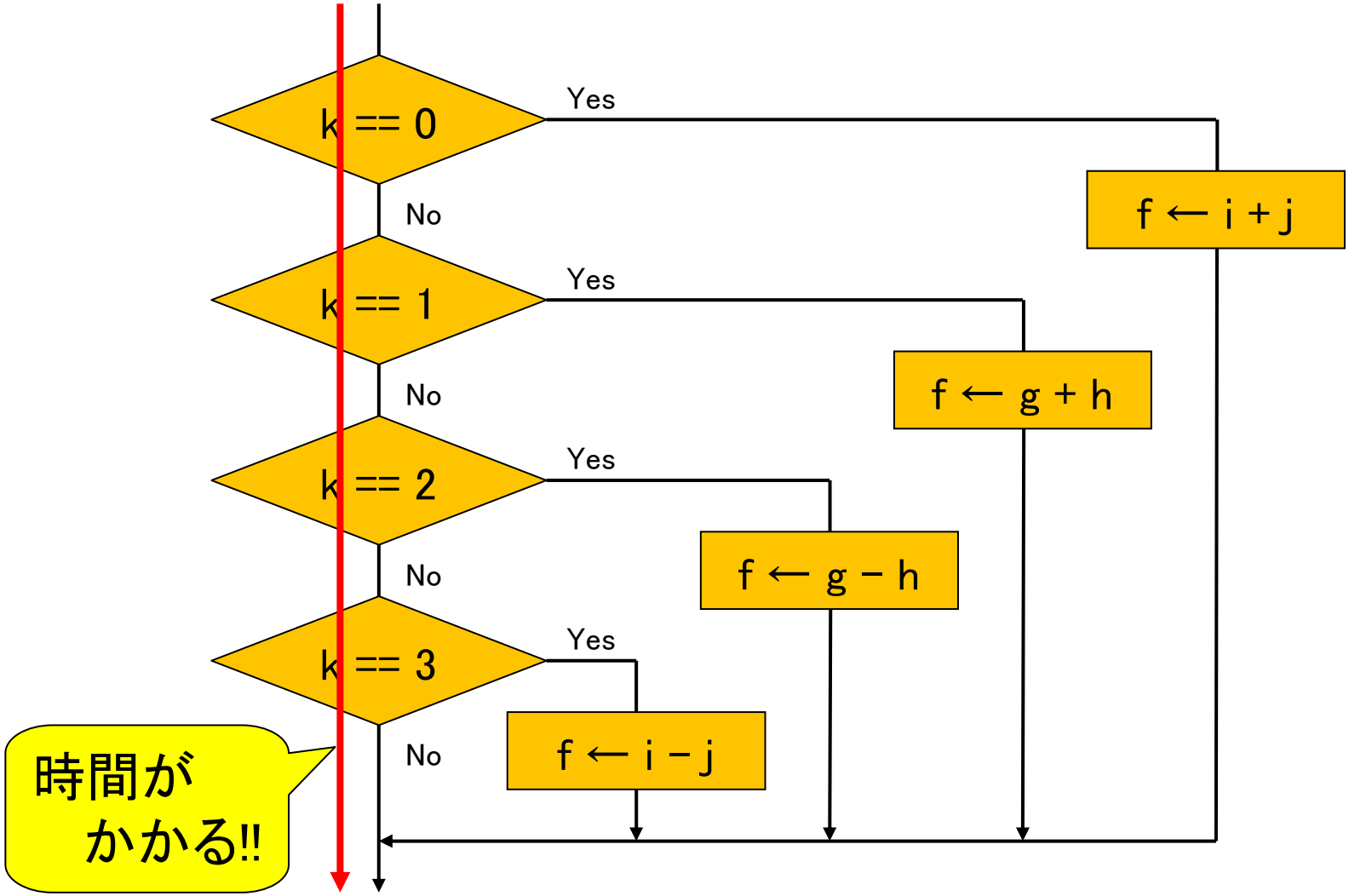


コード例: C言語の switch 文(3)

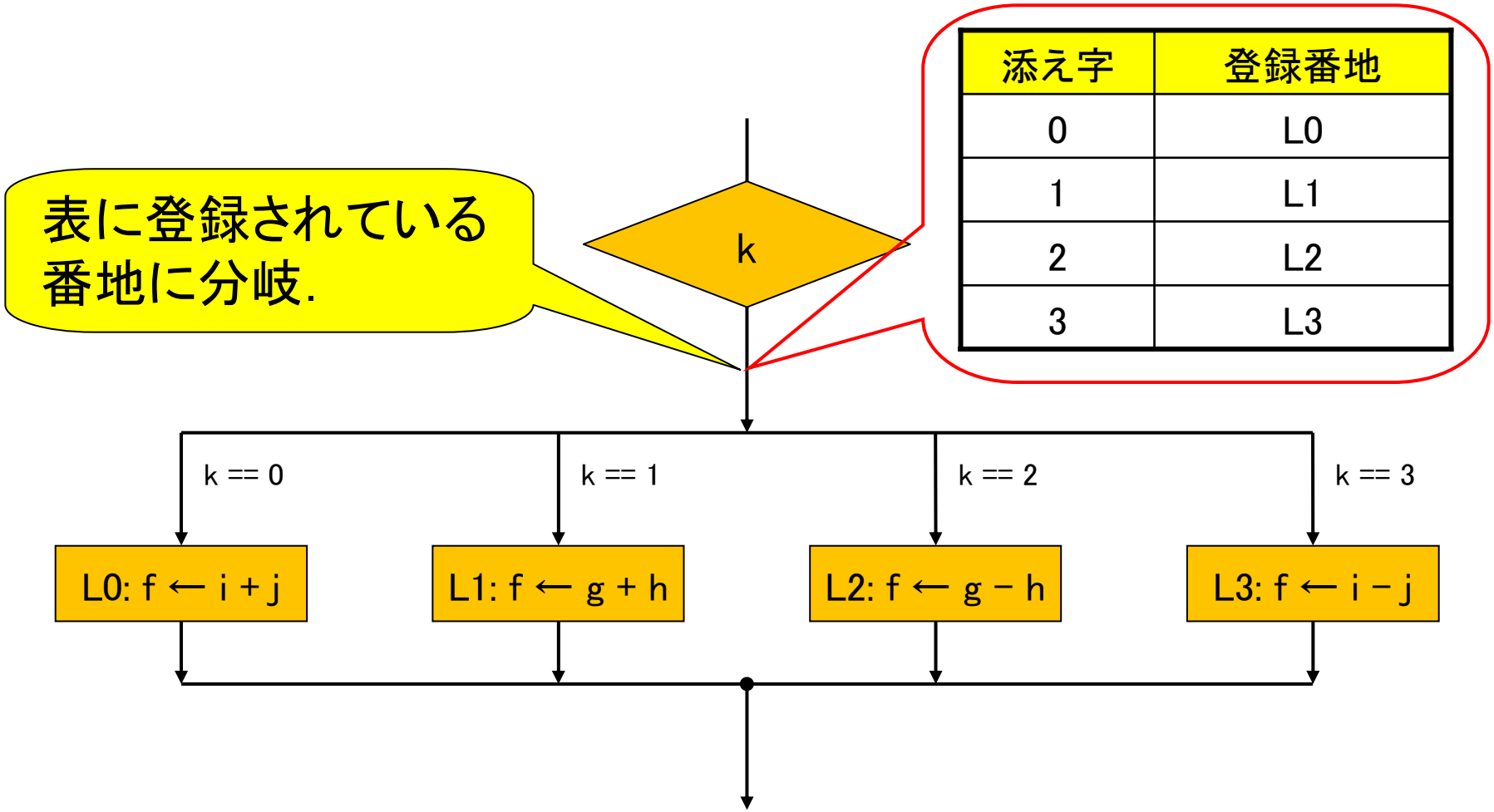
```
switch (k) {
case 0:
        f = i + j;
        break;
case 1:
        f = g + h;
        break;
case 2:
        f = g − h;
        break;
case 3:
        f = i − j;
        break;
}
```

・kが「0」の場合
  ・f=i+j;だけが実行される
・kが「1」の場合
  ・f=g+h;だけが実行される
・kが「2」の場合
  ・f=g−h;だけが実行される
・kが「3」の場合
  ・f=i−j;だけが実行される
・kが「0,1,2,3以外の場合」
  ・switch本体は実行されない

コード例: C言語の switch 文(4)



コード例: C言語の switch 文(5)



コード例: C言語の switch 文(6)

```
範囲検査    slt   $t3, $s5, $zero
            bne   $t3, $zero, EXIT
            slt   $t3, $s5, $t2
            beq   $t3, $zero, EXIT
多岐分岐    add   $t0, $s5, $s5
            add   $t0, $t0, $t0
            add   $t0, $s6, $t0
            lw    $t1, 0($t0)
            jr    $t1
L0:         add   $s0, $s3, $s4
            j     EXIT
L1:         add   $s0, $s1, $s2
            j     EXIT
L2:         sub   $s0, $s1, $s2
            j     EXIT
L3:         sub   $s0, $s3, $s4
EXIT:       …
```

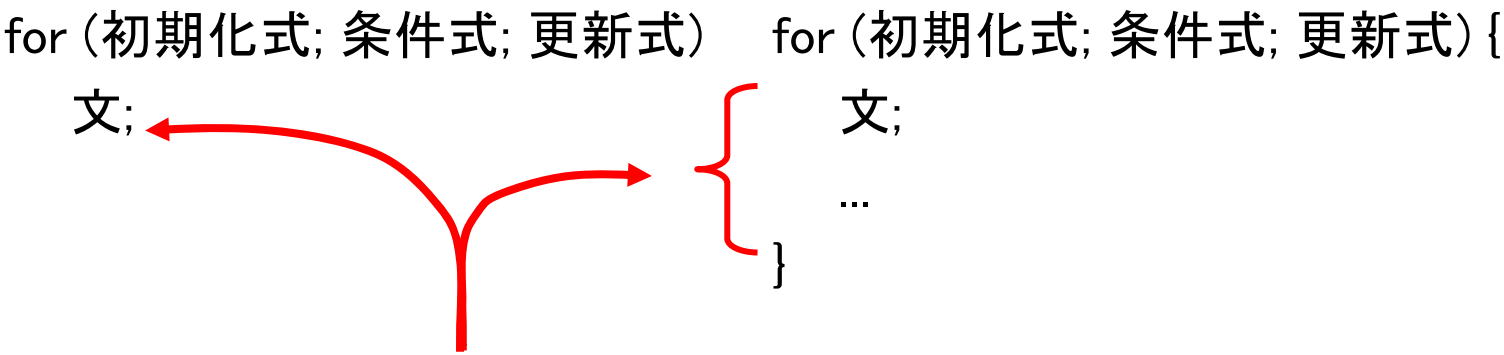
変数名	レジスタ
f	$s0
g	$s1
h	$s2
i	$s3
j	$s4
k	$s5
番地表先頭番地	$s6
定数4	$t2

$s6 →

添え字	登録番地
0	L0
1	L1
2	L2
3	L3

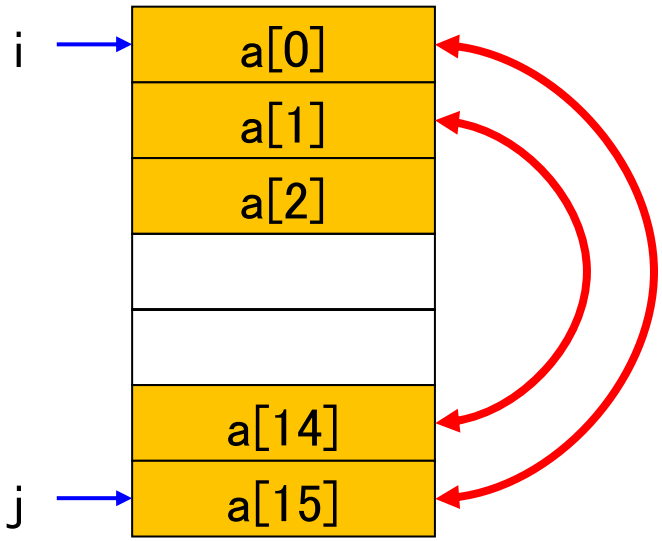
コード例: C言語の for 文(1)

1. 初期化式を評価する.
2. 条件式の値が真(0以外の整数)ならば文または中括弧の中を実行する. 偽(0)ならば文または中括弧の中を実行せずに次に進む(＝ループを終了する).
3. 文または中括弧の中を実行した後は, 更新式を評価し, 2. に戻る.



コード例: C言語の for 文(2)

```
for (i = 0, j = 15; i < j; i++, j--) {
        tmp = a[i];
        a[i] = a[j];
        a[j] = tmp;
}
```



コード例: C言語の for 文(3)

```
初期化式          add   $s0, $zero, $zero
                  add   $s1, $t4, $zero
条件式   FOR:     slt   $t0, $s0, $s1
                  beq   $t0, $zero, EXIT
                  add   $t1, $s0, $s0
                  add   $t1, $t1, $t1
tmp = a[i]        add   $t1, $s3, $t1
                  lw    $s2, 0($t1)
                  add   $t2, $s1, $s1
                  add   $t2, $t2, $t2
a[i] = a[j]       add   $t2, $s3, $t2
                  lw    $t3, 0($t2)
                  sw    $t3, 0($t1)
a[j] = tmp        sw    $s2, 0($t2)
更新式            add   $s0, $s0, $t5
                  sub   $s1, $s1, $t5
                  j     FOR
          EXIT:   …
```

変数名	レジスタ
i	$s0
j	$s1
tmp	$s2
a[] 先頭番地	$s3
定数15	$t4
定数1	$t5