命令の実行
アセンブリ言語プログラム

再掲

命令の種類

- 算術論理演算命令
  - 加算、減算、乗算、除算、剰余、絶対値など
  - 論理積、論理和、否定など
- データ転送命令
  - レジスタ間、メモリとレジスタ間、メモリ・レジスタと入出力機器間
- 分岐命令
  - 無条件分岐命令、条件分岐命令

データ転送命令

ロード:

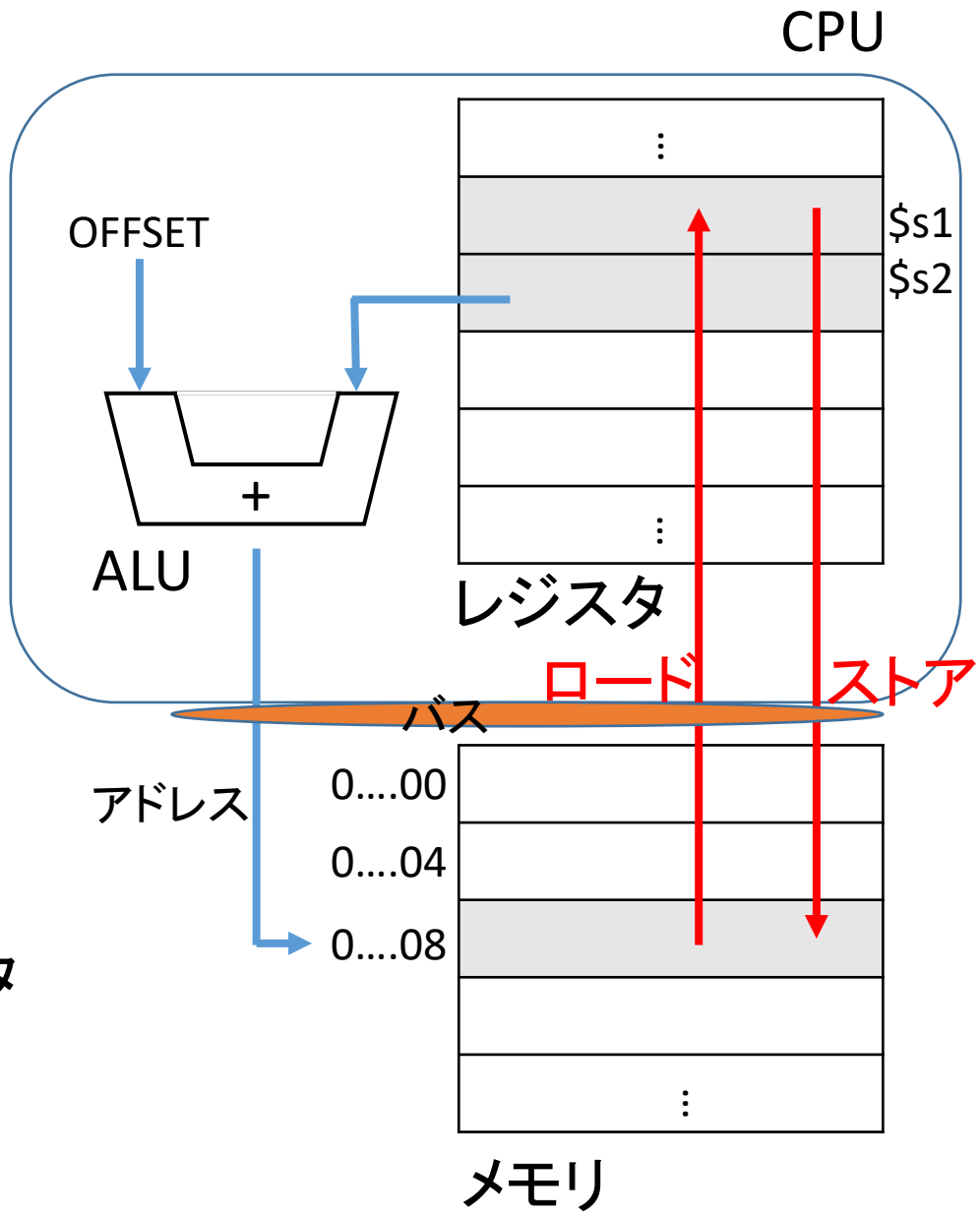
lw $s1, OFFSET($s2)

レジスタ$s2の値にOFFSETを加えてアドレスを得る。このアドレスで指定されるメモリのデータ（語）をレジスタ$s1に格納する

ストア

sw $s1, OFFSET($s2)

レジスタ$s2の値にOFFSETを加えてアドレスを得る。このアドレスで指定されるメモリにレジスタ$s1の値（語）を書き込む



データ転送命令（詳細）

- 以下、$r1, $r2などは任意のレジスタを表すものとし、実際には$s0や$t0などのレジスタ名を指定する

- 語のロード

  lw $r1, OFFSET($r2)

  レジスタ$r2の値を基準アドレスとして、OFFSETバイト離れたアドレスがロード対象のメモリアドレスとなる。OFFSETは変位とも呼ぶ

- アドレス値のロード

  la $r1, Label

  ラベルLabelのアドレス値をレジスタ$r1にロードする（疑似命令）

データ転送命令（詳細）

- 定数のロード

  li $r1, Imm

  定数Immをレジスタ$r1にロードする。Immを即値（immediate）ともいう

- 語のストア

  sw $r1, OFFSET($r2)

  $r1の内容（語）を$r2+OFFSETをアドレスとするメモリにストアする

- レジスタ間転送

  move $r1, $r2

  $r2の内容を$r1に転送する（疑似命令）

  addを使う方法は後に述べる

データ転送命令（詳細）

- メモリ・レジスタと入出力装置間のデータ転送は、入出力機器（のレジスタ）にメモリアドレスを割り振って、ロード／ストア命令によってデータ転送を行う

プログラム例

C言語

```
int x = 10;

int y;

int main(void)
{
    y=x;
    return 0;
}
```

アセンブリ言語

```
        .data       # データ領域
        .globl      x
x:      .word       10

        .globl      y
y:      .word       0

        .text       # プログラム領域
        .globl      main
main:
        la          $t0, x          # xのアドレスをロード
        lw          $s0, 0($t0)     # xの値をロード
        la          $t0, y          # yのアドレスをロード
        sw          $s0, 0($t0)     # yのメモリにxの値をストア
        li          $v0, 0  # system call番号をリセット。なくてもよい
        jr          $ra             # 復帰アドレス$raにジャンプ
# 今の場合そこにnop(「何もしない」)が入っているので終了
```

アセンブリ言語の書式

プログラムは行を単位として作成され、各行は以下の4種類に分類される

- コメント
  - #から開始するもの
- 命令
  - la $t0, x
- アセンブラ指示子
  - ドット.で開始する、データ領域の開始（.data）やプログラム領域の開始（.text）指示などや、メモリへの値の配置(.word)、ラベルxの外部参照可能宣言（.globl x）など
- ラベル
  - 行の最初から書き始め、最後にコロン:をつける。命令やデータのアドレスを他から参照するためのもの。なお、ラベルのアドレスはアセンブラによって自動獲得する。
  - 上記例では、x, y, mainがラベル

算術論理演算命令

加算:

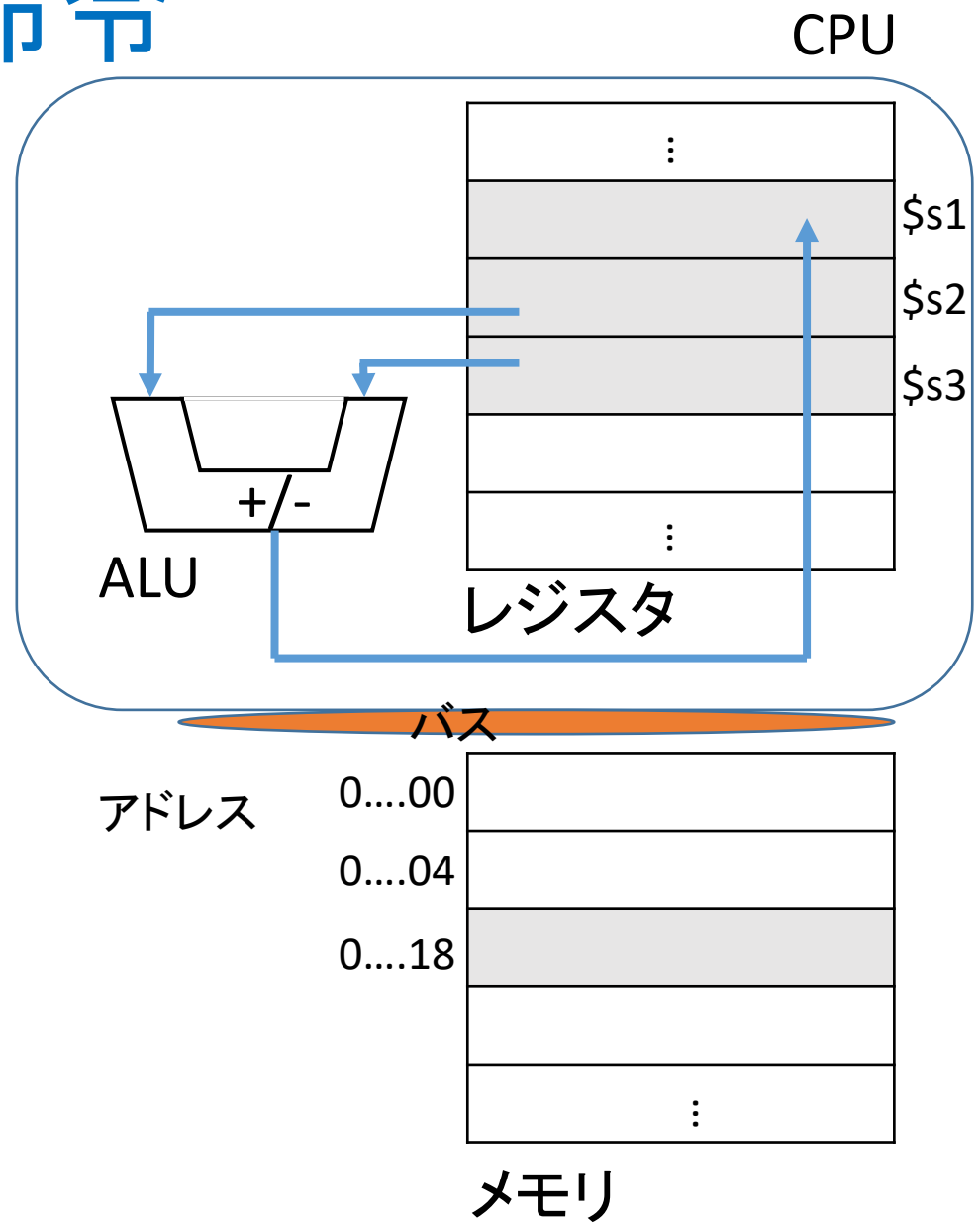
add $s1, $s2, $s3

レジスタ$s2の値とレジスタ$s3の値を加算して、レジスタ$s1に格納する

減算

sub $s1, $s2, $s3

レジスタ$s2の値からレジスタ$s3の値を減算して、レジスタ$s1に格納する



算術論理演算命令(詳細)

- 加算

  add $r1, $r2, $r3

  $r1 ← $r2 + $r3

- 減算

  sub $r1, $r2, $r3

  $r1 ← $r2 - $r3

- 定数加算

  addi $r1, $r2, Imm

  $r1 ← $r2 + Imm

  即値Immは16ビットの符号つき数で、32ビットに符号拡張される。符号拡張とは、

  ① 第0,…,15ビットに元の16ビットの値をそのまま入れる

  ② 第16,…,31ビットには正の値ならすべてに0を入れる。負の値ならすべて1を入れる

  ③ 詳細はP25

算術論理演算命令（詳細）

- 加減算命令の活用
  - レジスタ間のデータ転送

    add $r1, $r2, $zero　　# $r1 ← $r2

    （moveという疑似命令はあるが）
  - 符号反転

    ?　　# $r1 ← -$r2

- 定数加算命令の活用
  - レジスタへのデータセット

    ?　　# $r1 ← 50
  - 定数の減算（これは「活用」ほどでもない…）

    addi $r1, $r2, -50　　# $r1 ← $r2 - 50

算術論理演算命令(詳細)

- 論理積

  and $r1, $r2, $r3

  $r1 ← $r2 & $r3

- 論理和

  or $r1, $r2, $r3

  $r1 ← $r2 | $r3

- 論理否定

  not $r1, $r2

  $r1 ← ~$r2 # not $r2

プログラム例

C言語

```
int x=10;
int y=20;
int z;

int main(void)
{
    z=x+y;
    return 0;
}
```

アセンブリ言語

```
        .data # データ領域
        .globl  x
x:      .word   10
        .globl  y
y:      .word   20
        .globl  z
z:      .word   0

        .text # プログラム領域
        .globl  main
main:
        la      $t0, x
        lw      $s0, 0($t0)

        la      $t0, y
        lw      $s1, 0($t0)

        add     $s1, $s0, $s1

        la      $t0, z
        sw      $s1, 0($t0)

        li      $v0, 0
        jr      $ra
```

演習

- 上記プログラムをx-y+100計算に変更しなさい

符号拡張について

- ここで4 ビットの符号付きの数 0010 を 8 ビットに拡張することを考える。0010 は正の整数なので、00000010 とすればよいと簡単にわかる
- では、2 の補数で表現された 4 ビットの負の数 1110 を 8 ビットに 拡張するにはどうしたらよいか
  - 答えは 11111110 である。
  - 実は、正負の符号 (正なら 0、負なら 1) を 必要な数だけ左にコピーすれば、簡単にビット幅を広げることができる。
  - ビット幅を広げる際のこの手法を符号拡張という。

manaba小テスト：演習14-1

- 10分
- 3点（6点から換算）

演習

1. 前出の符号付き2進数1110 と 11111110 をそれぞれ 10 進数で表しなさい
2. 符号付き2進数0100を 8 ビットに拡張しなさい。これは 10 進数でいくつか
3. 符号付き2進数1000を 8 ビットに拡張しなさい。これは 10 進数でいくつか

再掲

命令の種類

- 算術論理演算命令
  - 加算、減算、乗算、除算、剰余、絶対値など
  - 論理積、論理和、否定など
- データ転送命令
  - レジスタ間、メモリとレジスタ間、メモリ・レジスタと入出力機器間
- 分岐命令
  - 無条件分岐命令、条件分岐命令

分岐命令：無条件分岐

- j Label

  常にLabelへ分岐

- jr $r1

  常に$r1に格納されているアドレスへ分岐

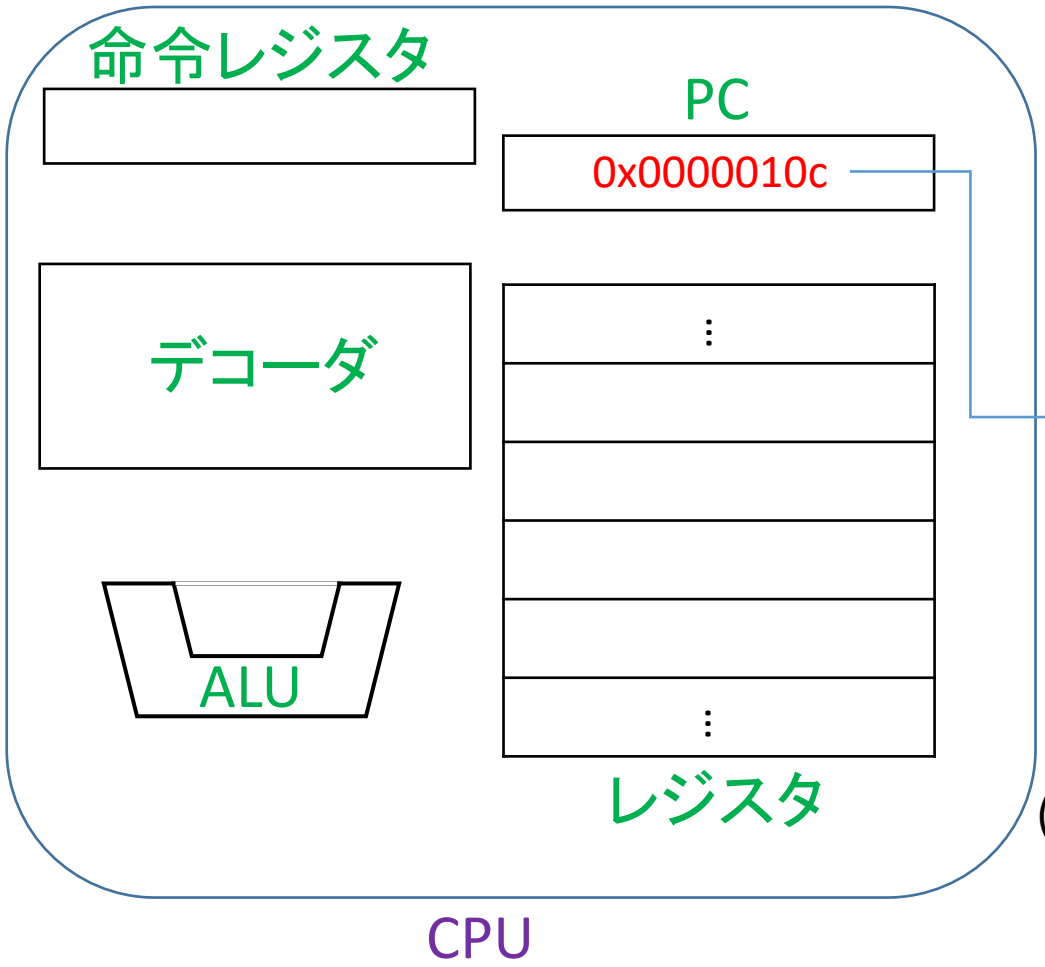
分岐命令：条件分岐

比較命令	分岐命令
比較= seq $r1, $r2, $r3 比較≠ sne $r1, $r2, $r3 比較< slt $r1, $r2, $r3 比較> sgt $r1, $r2, $r3 : :	条件= beqz $r1, Label $r1=0ならLabelに分岐する 条件≠ bnez $r1, Label $r1≠0ならLabelに分岐する

先頭のs: $r1に値1をset（比較条件を満足すれば）

beqz: branch if equal 0
bnez: branch if not equal 0

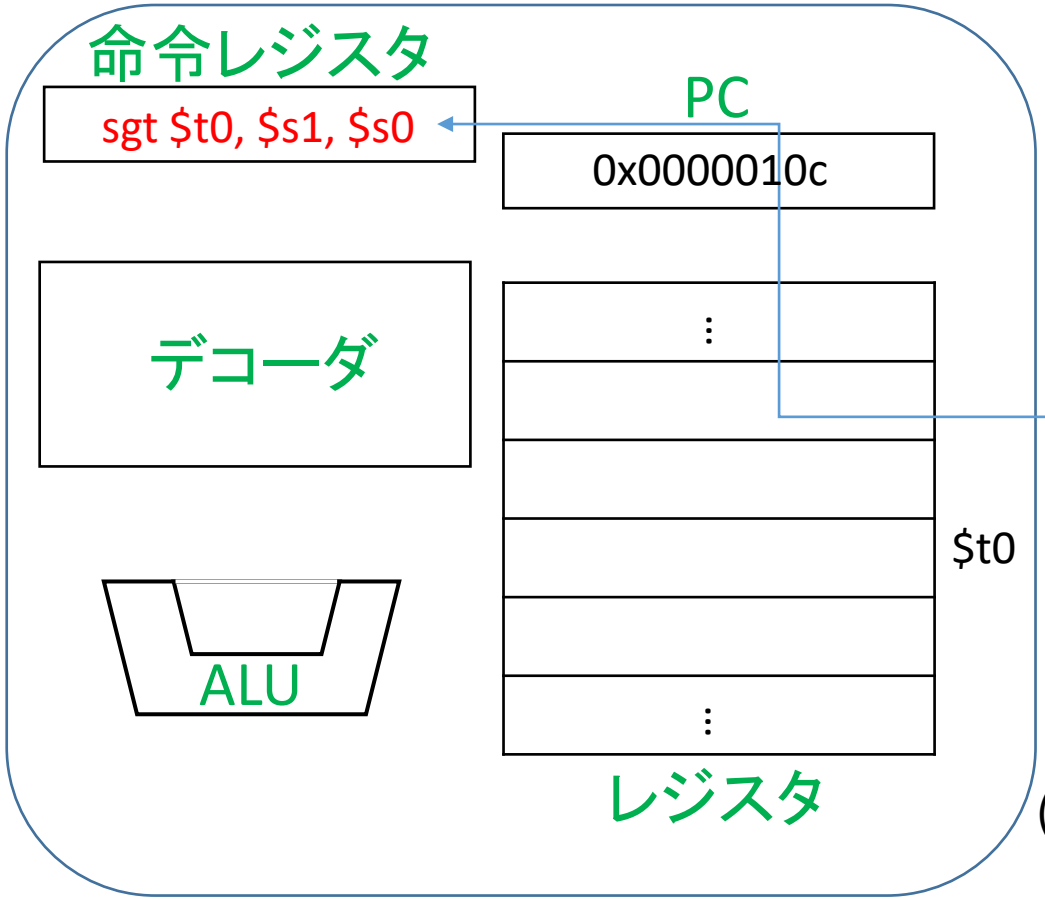
分岐命令：(1)比較

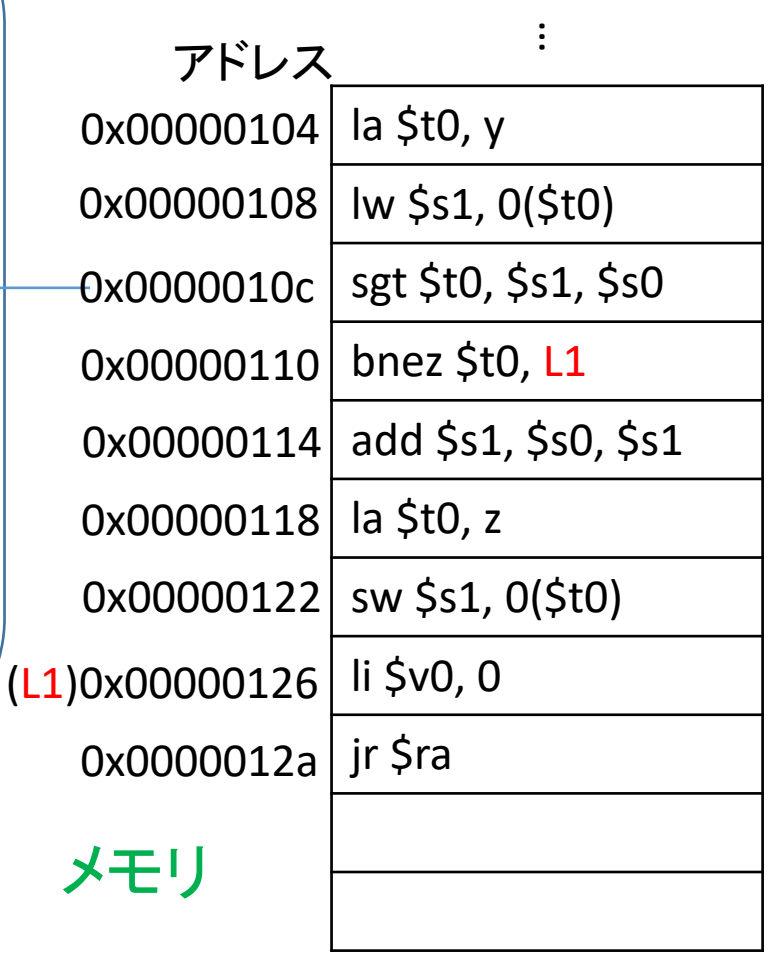


アドレス	
0x00000104	la $t0, y
0x00000108	lw $s1, 0($t0)
0x0000010c	sgt $t0, $s1, $s0
0x00000110	bnez $t0, L1
0x00000114	add $s1, $s0, $s1
0x00000118	la $t0, z
0x00000122	sw $s1, 0($t0)
(L1)0x00000126	li $v0, 0
0x0000012a	jr $ra

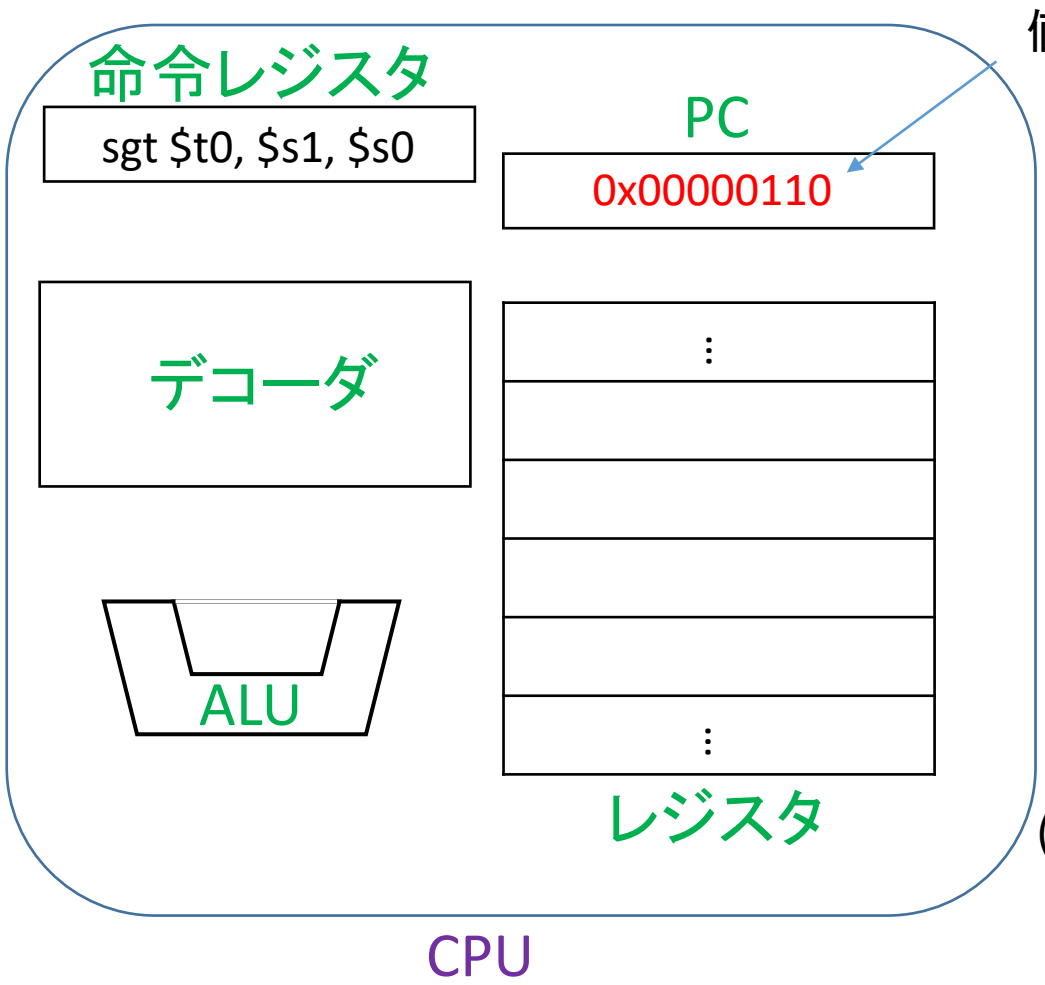
メモリ

分岐命令：(1)比較


命令レジスタ
sgt $t0, $s1, $s0
PC
0x0000010c
デコーダ
ALU
$t0
レジスタ
CPU


アドレス
0x00000104 la $t0, y
0x00000108 lw $s1, 0($t0)
0x0000010c sgt $t0, $s1, $s0
0x00000110 bnez $t0, L1
0x00000114 add $s1, $s0, $s1
0x00000118 la $t0, z
0x00000122 sw $s1, 0($t0)
(L1)0x00000126 li $v0, 0
0x0000012a jr $ra
メモリ

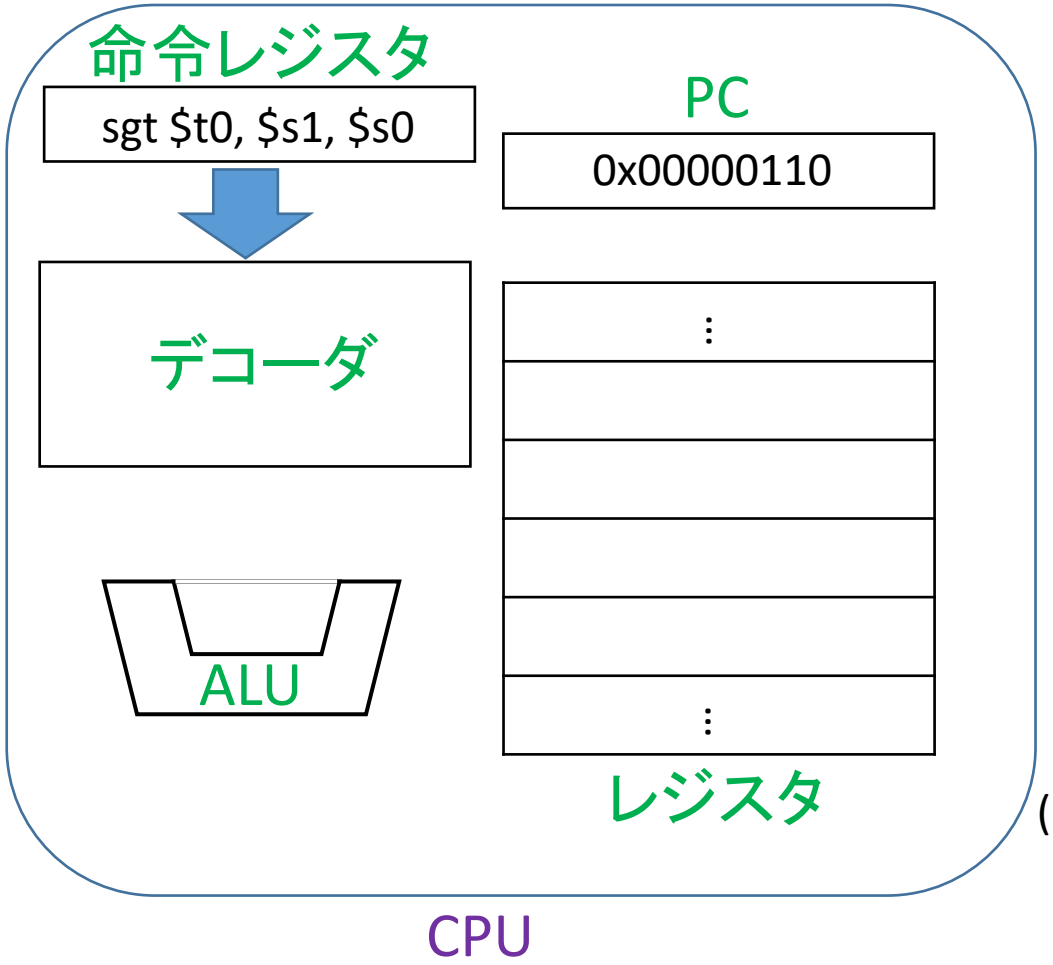
分岐命令：(1)比較



アドレス	
0x00000104	la $t0, y
0x00000108	lw $s1, 0($t0)
0x0000010c	sgt $t0, $s1, $s0
0x00000110	bnez $t0, L1
0x00000114	add $s1, $s0, $s1
0x00000118	la $t0, z
0x00000122	sw $s1, 0($t0)
(L1)0x00000126	li $v0, 0
0x0000012a	jr $ra

メモリ

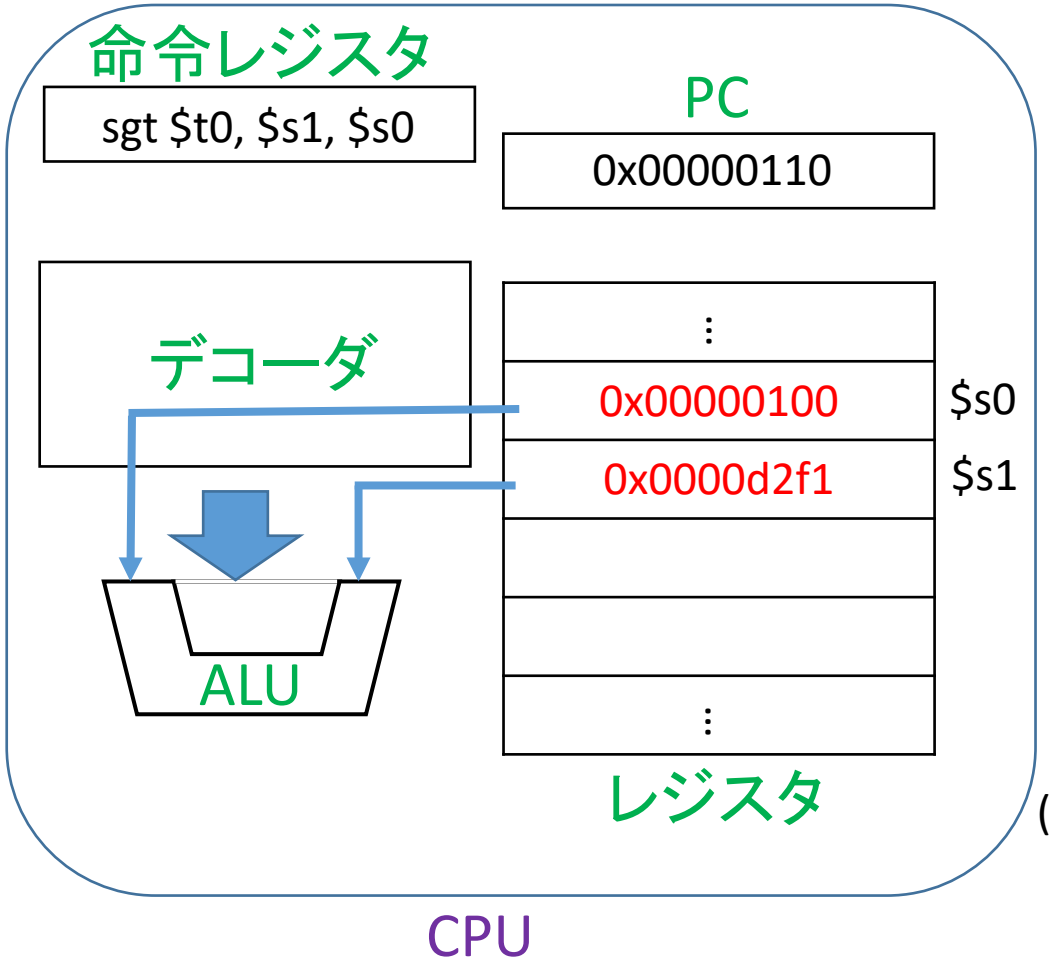
分岐命令：(1)比較



アドレス	
	⋮
0x00000104	la $t0, y
0x00000108	lw $s1, 0($t0)
0x0000010c	sgt $t0, $s1, $s0
0x00000110	bnez $t0, L1
0x00000114	add $s1, $s0, $s1
0x00000118	la $t0, z
0x00000122	sw $s1, 0($t0)
(L1)0x00000126	li $v0, 0
0x0000012a	jr $ra

メモリ

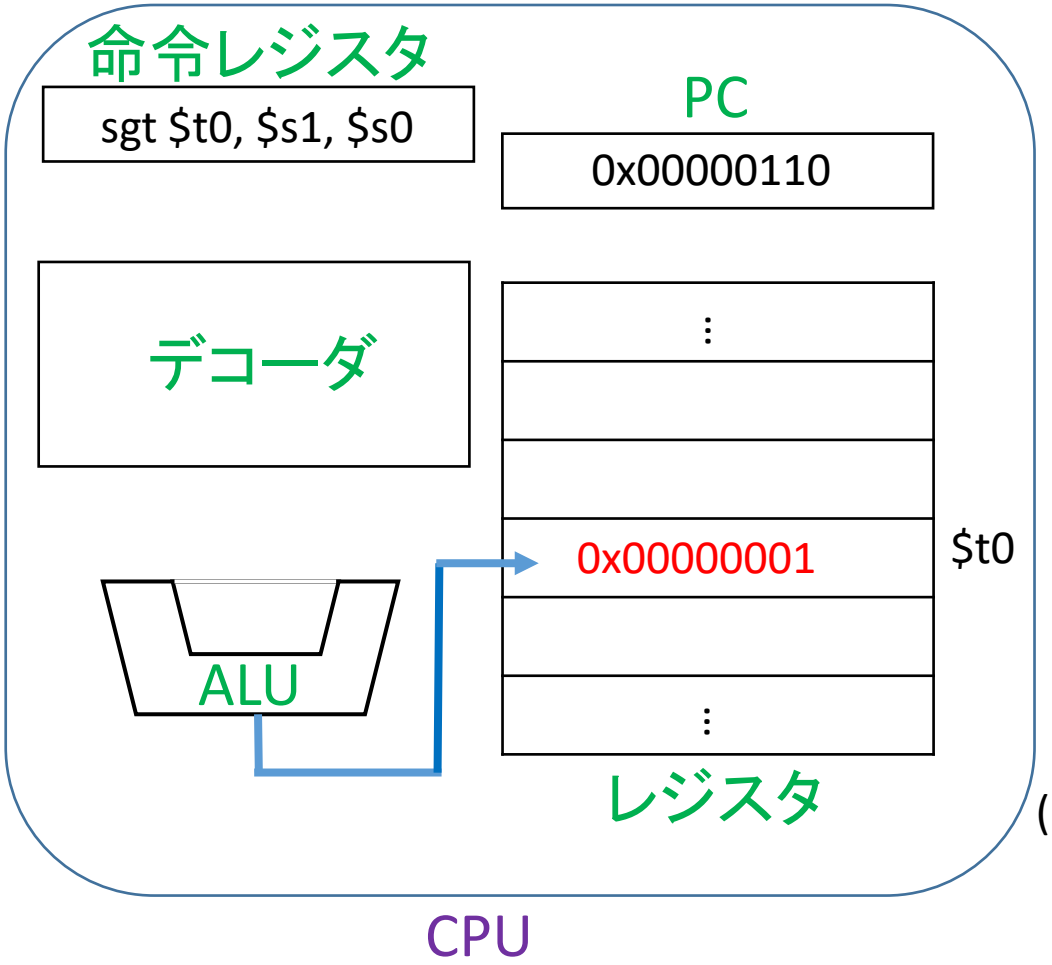
分岐命令：(1)比較



アドレス	
0x00000104	la $t0, y
0x00000108	lw $s1, 0($t0)
0x0000010c	sgt $t0, $s1, $s0
0x00000110	bnez $t0, L1
0x00000114	add $s1, $s0, $s1
0x00000118	la $t0, z
0x00000122	sw $s1, 0($t0)
(L1)0x00000126	li $v0, 0
0x0000012a	jr $ra

メモリ

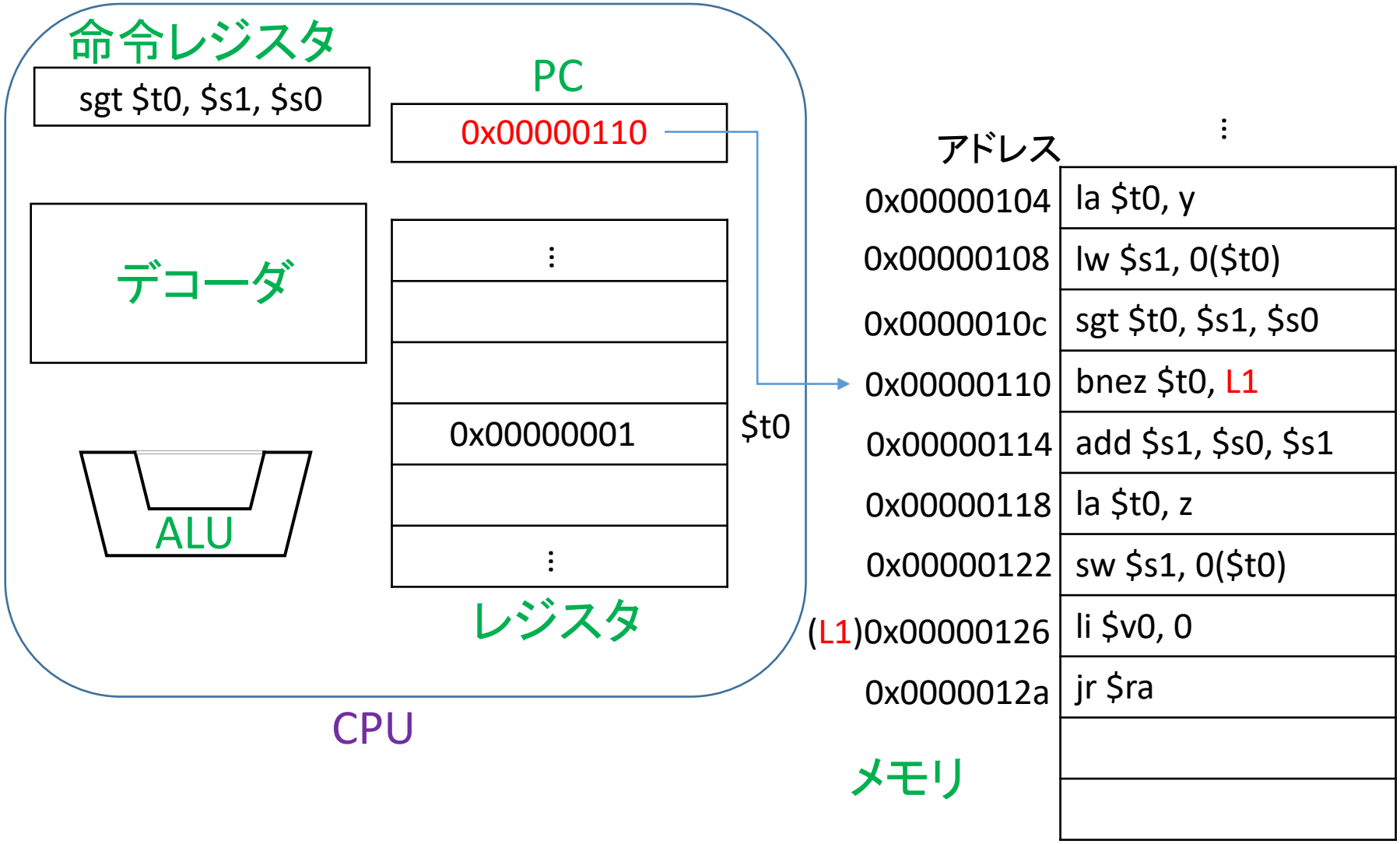
分岐命令：(1)比較



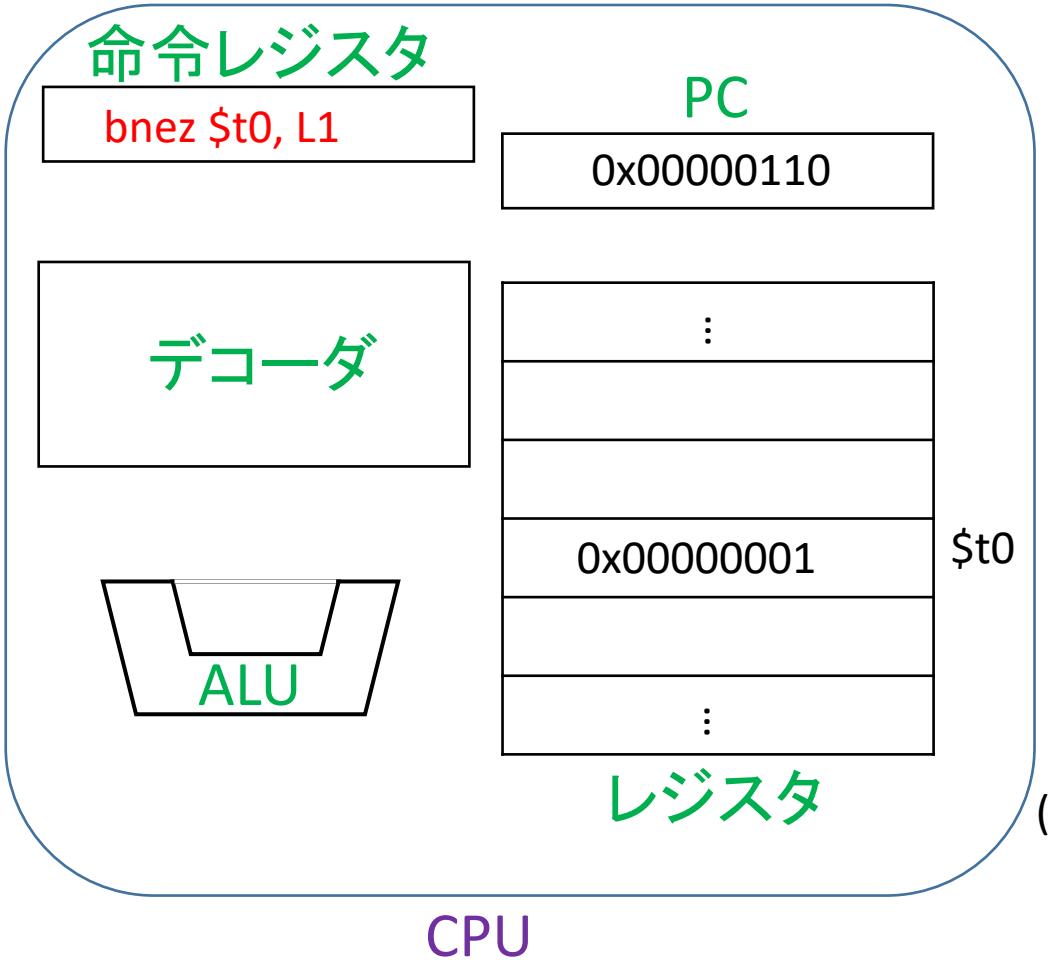
アドレス	
0x00000104	la $t0, y
0x00000108	lw $s1, 0($t0)
0x0000010c	sgt $t0, $s1, $s0
0x00000110	bnez $t0, L1
0x00000114	add $s1, $s0, $s1
0x00000118	la $t0, z
0x00000122	sw $s1, 0($t0)
(L1)0x00000126	li $v0, 0
0x0000012a	jr $ra

メモリ

分岐命令：(2)条件分岐


命令レジスタ
sgt $t0, $s1, $s0
PC
0x00000110
デコーダ
ALU
0x00000001
$t0
レジスタ
CPU
アドレス
0x00000104
0x00000108
0x0000010c
0x00000110
0x00000114
0x00000118
0x00000122
(L1)0x00000126
0x0000012a
la $t0, y
lw $s1, 0($t0)
sgt $t0, $s1, $s0
bnez $t0, L1
add $s1, $s0, $s1
la $t0, z
sw $s1, 0($t0)
li $v0, 0
jr $ra
メモリ

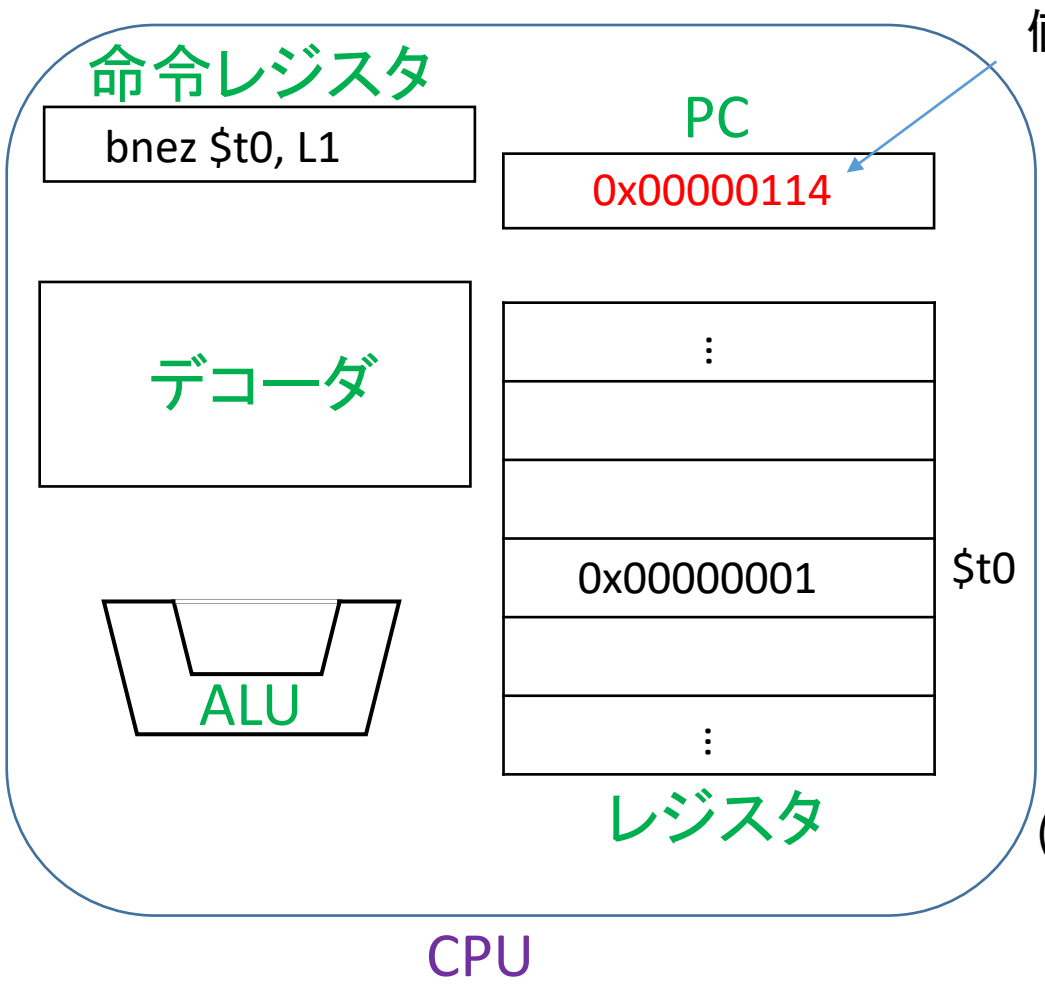
分岐命令：(2)条件分岐

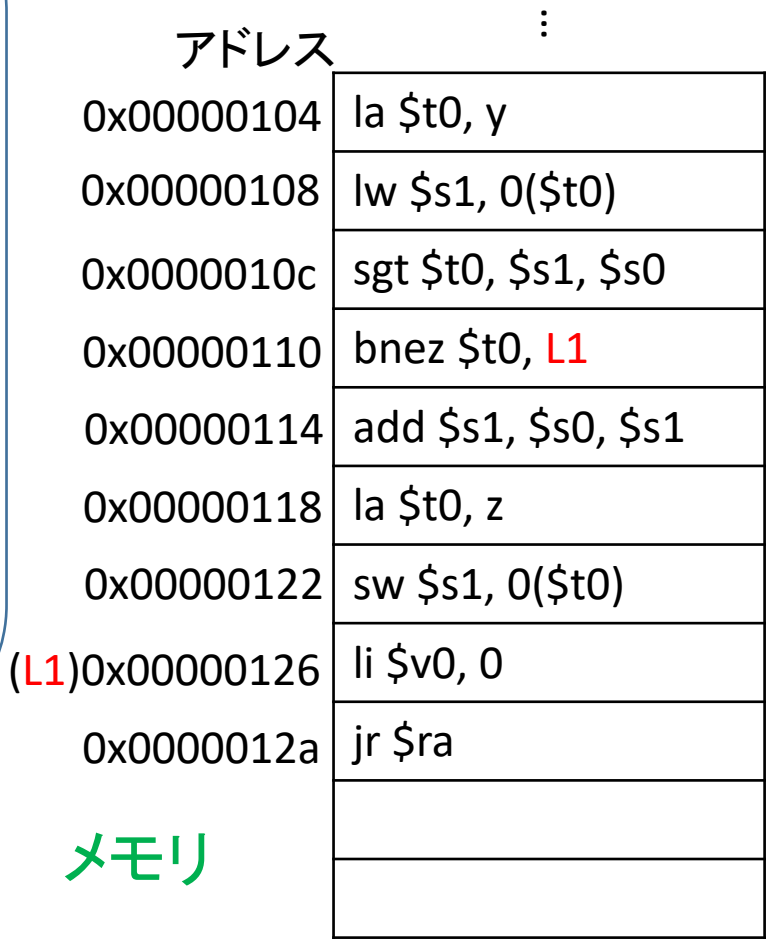


アドレス	
0x00000104	la $t0, y
0x00000108	lw $s1, 0($t0)
0x0000010c	sgt $t0, $s1, $s0
0x00000110	bnez $t0, L1
0x00000114	add $s1, $s0, $s1
0x00000118	la $t0, z
0x00000122	sw $s1, 0($t0)
(L1)0x00000126	li $v0, 0
0x0000012a	jr $ra

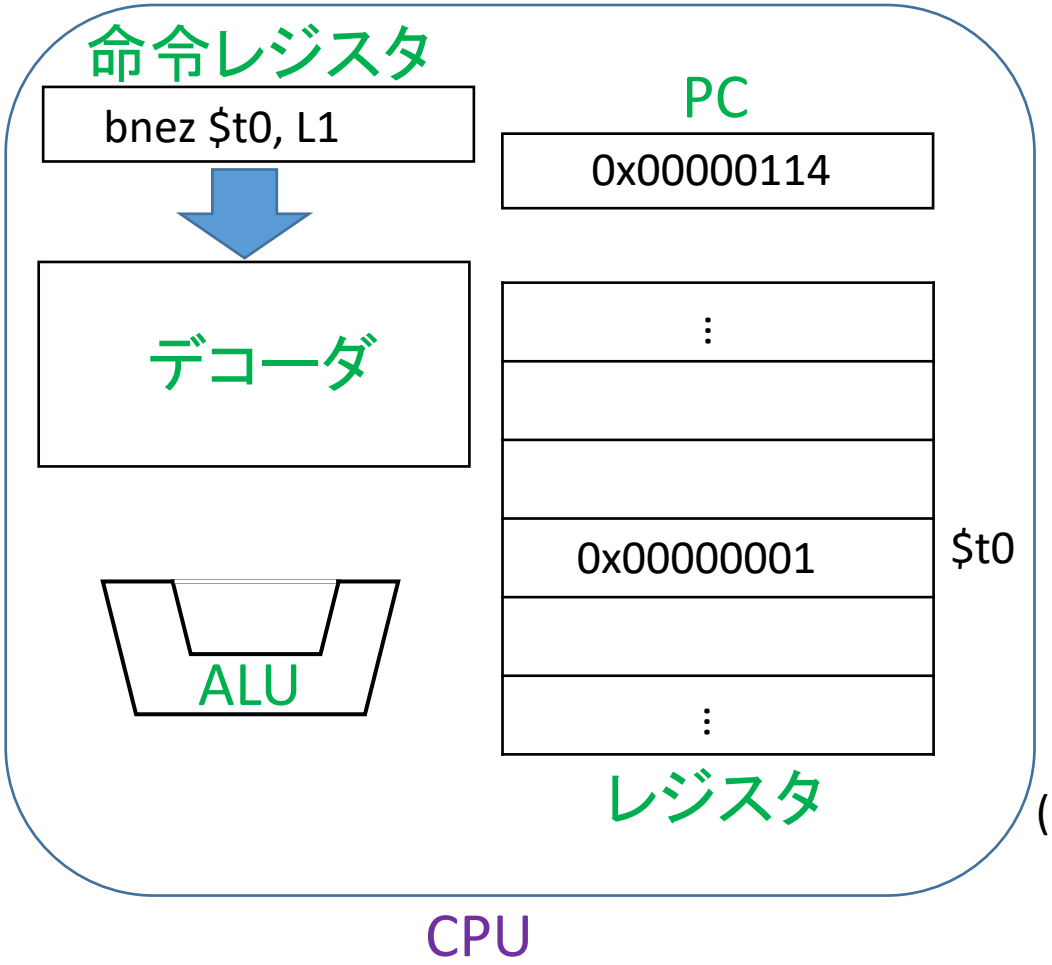
メモリ

分岐命令：(2)条件分岐


値更新
命令レジスタ
bnez $t0, L1
PC
0x00000114
デコーダ
ALU
0x00000001
$t0
レジスタ
CPU


アドレス
0x00000104 la $t0, y
0x00000108 lw $s1, 0($t0)
0x0000010c sgt $t0, $s1, $s0
0x00000110 bnez $t0, L1
0x00000114 add $s1, $s0, $s1
0x00000118 la $t0, z
0x00000122 sw $s1, 0($t0)
(L1)0x00000126 li $v0, 0
0x0000012a jr $ra
メモリ

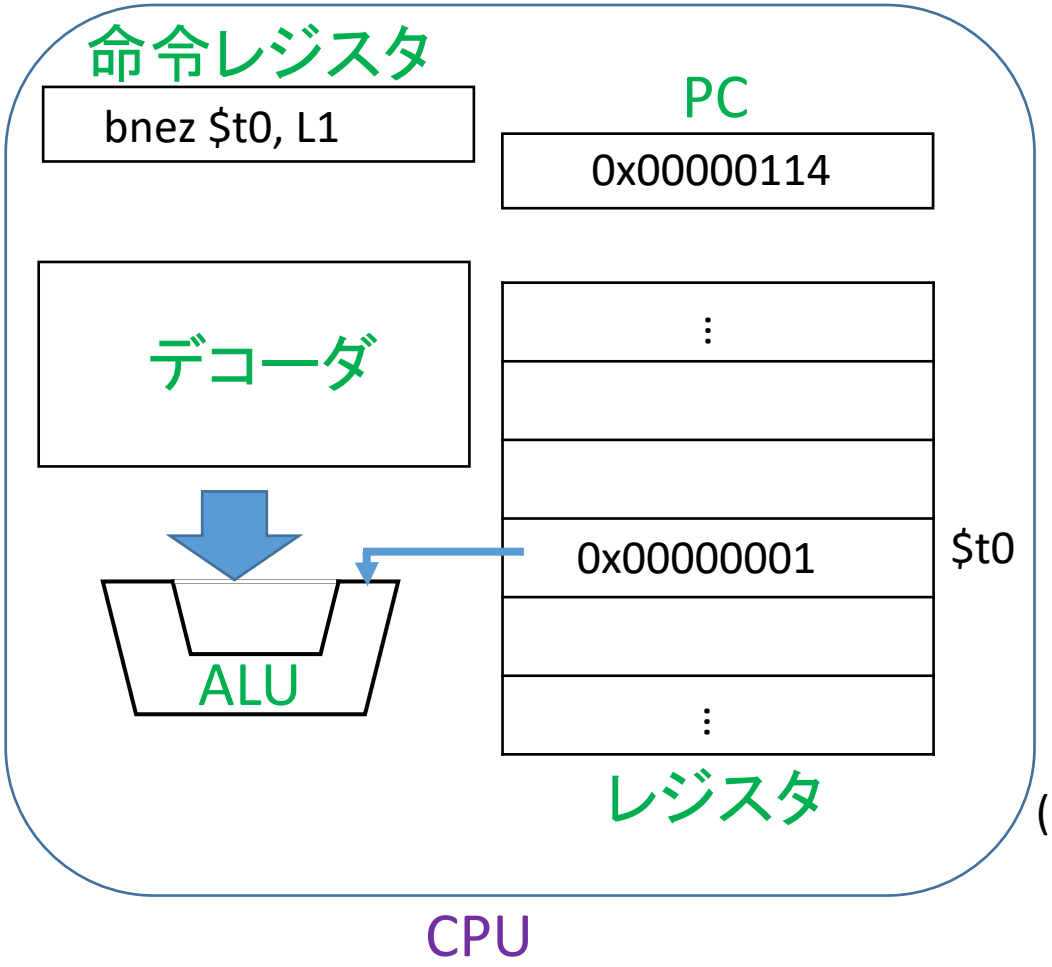
分岐命令：(2)条件分岐



アドレス	
0x00000104	la $t0, y
0x00000108	lw $s1, 0($t0)
0x0000010c	sgt $t0, $s1, $s0
0x00000110	bnez $t0, L1
0x00000114	add $s1, $s0, $s1
0x00000118	la $t0, z
0x00000122	sw $s1, 0($t0)
(L1)0x00000126	li $v0, 0
0x0000012a	jr $ra

メモリ

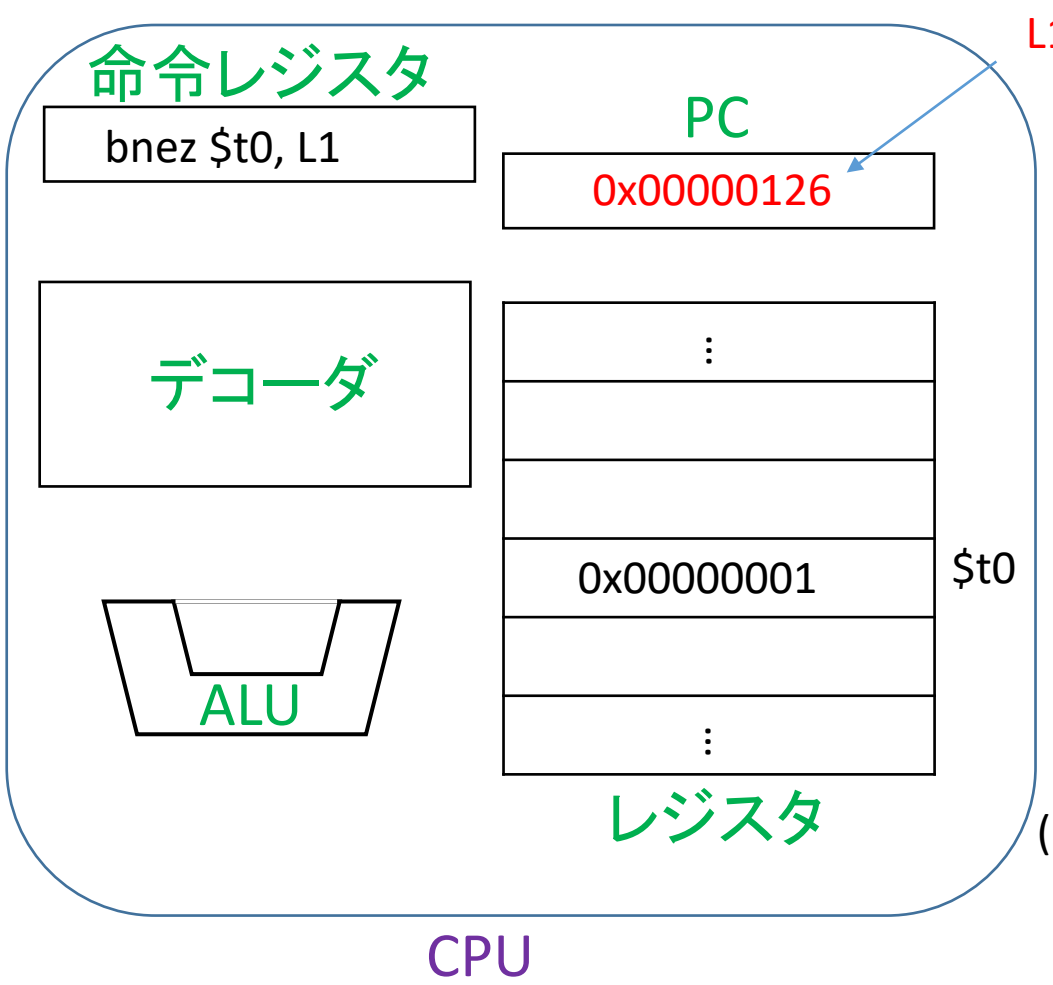
分岐命令：(2)条件分岐



アドレス	
0x00000104	la $t0, y
0x00000108	lw $s1, 0($t0)
0x0000010c	sgt $t0, $s1, $s0
0x00000110	bnez $t0, L1
0x00000114	add $s1, $s0, $s1
0x00000118	la $t0, z
0x00000122	sw $s1, 0($t0)
(L1)0x00000126	li $v0, 0
0x0000012a	jr $ra

メモリ

分岐命令：(2)条件分岐

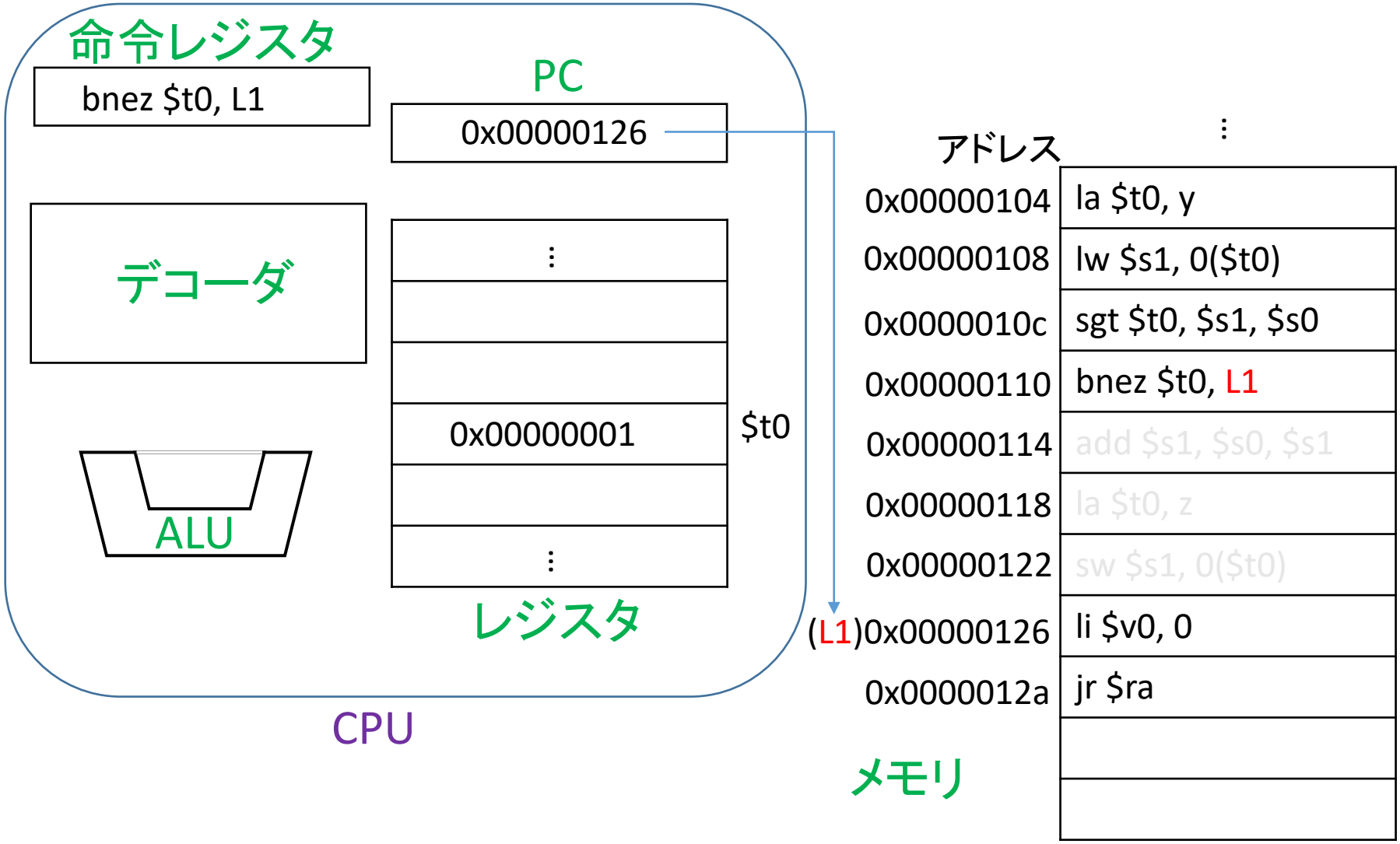


L1で値更新

アドレス	
0x00000104	la $t0, y
0x00000108	lw $s1, 0($t0)
0x0000010c	sgt $t0, $s1, $s0
0x00000110	bnez $t0, L1
0x00000114	add $s1, $s0, $s1
0x00000118	la $t0, z
0x00000122	sw $s1, 0($t0)
(L1)0x00000126	li $v0, 0
0x0000012a	jr $ra

メモリ

分岐命令：(2)条件分岐


命令レジスタ
bnez $t0, L1
PC
0x00000126
デコーダ
ALU
0x00000001
$t0
レジスタ
CPU
アドレス
0x00000104
0x00000108
0x0000010c
0x00000110
0x00000114
0x00000118
0x00000122
(L1)0x00000126
0x0000012a
la $t0, y
lw $s1, 0($t0)
sgt $t0, $s1, $s0
bnez $t0, L1
add $s1, $s0, $s1
la $t0, z
sw $s1, 0($t0)
li $v0, 0
jr $ra
メモリ

manaba小テスト：演習14-2

- 10分
- 4点

manaba小テスト：演習14-2

1．0x0000010cにある命令を実行した結果、レジスタ$t0に0x00000000が入ったとする。

- 次に実行すべき命令を格納しているメモリアドレスを0x.....で答えなさい。
- さらにその次に実行すべき命令を格納しているメモリアドレスを0x.....で答えなさい。

2．0x0000010cにある命令を実行した結果、レジスタ$t0に0x00000001が入ったとする。

- 次に実行すべき命令を格納しているメモリアドレスを0x.....で答えなさい。
- さらにその次に実行すべき命令を格納しているメモリアドレスを0x.....で答えなさい。

命令の実行サイクル

	動作	命令実行の仕組み
①命令フェッチ	命令を取得	(i)PCの値（アドレス）をアドレスデコーダで解読する (ii)指定されたアドレスの命令をレジスタに移す (iii)PCの値を1語分（1語のバイト数分）増やす
②命令デコード	命令を解読	(i)レジスタの命令を命令デコーダで解読する
③命令実行	命令を実行	解読された命令を実行（演算・メモリ制御）する 足し算であれば (i)メモリからデータをレジスタに移し、ALUで演算を実行、結果をレジスタに転送する 分岐であれば (i)レジスタ間の値を比較し、PCの値をジャンプ先のアドレスにセットするなど

期末定期試験について

- 内容的には、manaba小テスト、manabaレポート課題、授業中の演習・質問（Question）が中心。ただし、問題の出し方はこれらと一致しない（つまりこれらをこのまま使わない）ので、注意してください

- 以下の内容からは出題しない
  - カルノー図
  - 順序回路（ラッチ、フリップフロップ）
  - 命令そのもの（ただし、命令の設計や大まかな構成などは重要で出題範囲である！）
  - アセンブリ言語のプログラム