命令セット・命令の実行

命令セット

機械語とアセンブリ言語

命令・命令セット・命令セットアーキテクチャ

- 命令（instruction）
  - コンピュータの動作を指示するもの。コンピュータは命令にしたがって実行する（命令の実行）
- 命令セット（instruction set）
  - コンピュータのすべての命令の集まり
- 命令セットアーキテクチャ
  - コンピュータで使われる命令の表現形式と各命令の動作を定めたもの
  - コンピュータに何ができるかをユーザに示し、どのようなハードウェア機構が必要であるかを設計者に教える

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、メモリと入出力機器間
- 分岐命令
  - 無条件分岐命令、条件分岐命令

命令・機械語・アセンブリ言語

- 命令は2進数（ビット列）で符号化され、１つの命令は1語の命令メモリに格納されている
  - 1語はCPUが扱うデータの最小単位。通常32ビット（4バイト）か64ビット（8バイト）。CPUに依存
- １命令を１命令語とも呼ぶ
- このようなビット列で「命令」（「命令語」）を表す言語を「機械語」と呼ぶ
  - CPUという機械が理解できる言語（CPUという機械向けの言語）＝機械語
- 機械語は人間が理解できない⇒アセンブリ言語
  - 人間が読める、英語に近い表記
  - 機械語と１対１対応（疑似命令などで例外はある）

言語間の関係

Cなど高級言語

y=x;

コンパイラ

命令セット

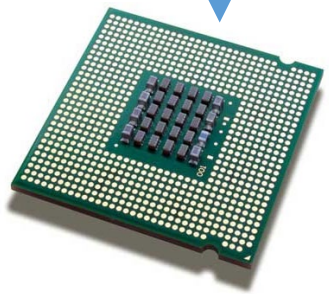
アセンブラ

アセンブリ言語
（低水準言語）

```
la $t0, x
lw $s0, 0($t0)
la $t1, y
sw $s0, 0($t0)
```

機械語（低水準言語、
CPUへの命令）

```
00111011000010000001000000000001
10001100000100000000000000000000
00111011000000010001000000000001
00110100001010010000000000000100
10101101001100000000000000000000
```



アセンブラとは

- アセンブリ言語プログラムを機械語に変換するソフトウエア
- アセンブリ言語・アセンブラは命令セット依存なので、CPUによって異なる
- アセンブラはアセンブリ言語を機械語に単純に置き換えるだけでなく、多数の有用な機能を持っている。たとえば、
  - ラベルを使うだけで、アセンブラはそれに対応するアドレスを決定してくれる（例：Label: … j Label）
  - 疑似命令がある（MIPSの場合）。実際の命令は単純な動作なので、これを使うことにより、複数の処理ができ、プログラムが書きやすくなる。実際の命令ではないので、アセンブラで実際の命令に変換する必要がある

MIPS32アーキテクチャ（1/2）

- ミップス・コンピュータシステムズ（現ミップス・テクノロジーズ）が開発した32ビットマイクロプロセッサの命令セットアーキテクチャである
- 命令セットが非常にきれいなので、米国ではコンピュータ・アーキテクチャを学校で教えるときに教材としてMIPSアーキテクチャを使うことが多い
- 対して、x86, x64アーキテクチャ（インテル社CPUなど）の命令セットは複雑で、教材としては適さない

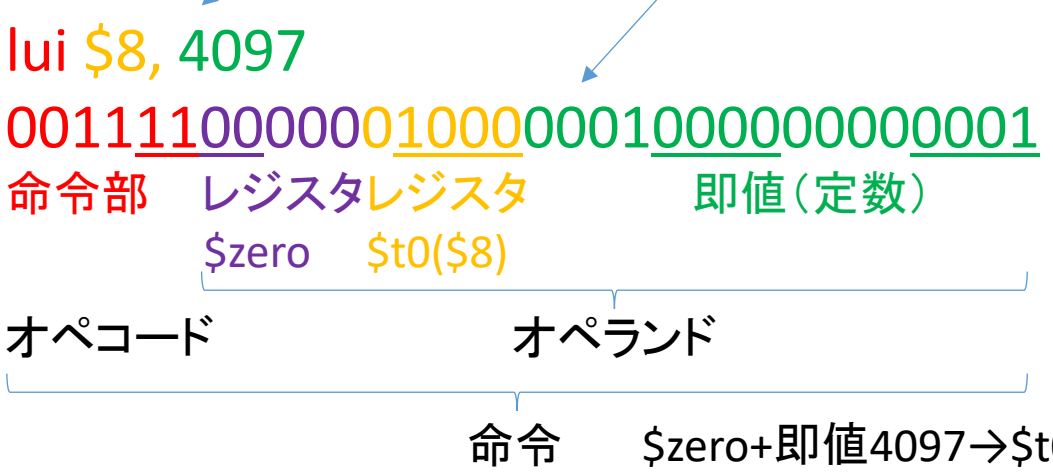
MIPS32アーキテクチャ(2/2)

- ワークステーション:MIPSプロセッサを使ったワークステーションシステムを製造していた企業として、SGI、DEC、NEC、ソニーなどがあった
- OS分野:MIPSアーキテクチャ上に移植されたオペレーティングシステムとして、Windows CE、Linux、BSDなどがあった
- 機器組み込み分野:コンピュータネットワーク(CISCOのルーター)、ゲーム機(NINTENDO64・ソニーのPlayStation)、プリンター、デジタルテレビ、DVDレコーダ、携帯情報端末などに広く採用されいる

C言語・アセンブリ言語・機械語の具体例

C言語	アセンブリ言語（疑似命令あり）	実際の命令	機械語（16進数で表記。実際は二進数）

y=x;
//変数代入
	la $t0, x　#アドレスをロード
	lw $s0, 0($t0) #語をロード
	la $t1, y　#アドレスをロード
	sw $s0, 0($t1) #語をストア
		lui $8, 4097
		lw $16, 0($8)
		lui $1, 4097
		ori $9, $1, 4
		sw $16, 0($9)
			0x3C081001
			0x8D100000
			0x3C011001
			0x34290004
			0xAD300000



命令形式（Instruction Format）

- 命令語は32ビットの固定長
- Rタイプ, Iタイプ, Jタイプの3種類に分類

type	フォーマット（ビット数）					
R	命令部（6）	レジスタ（5）	レジスタ（5）	レジスタ（5）	シフト量（5）	機能（6）
I	命令部（6）	レジスタ（5）	レジスタ（5）	即値（オフセット）（16）		
J	命令部（6）	アドレス（26）				

↑
フィールド

Rタイプ命令

000000	10000	10001	01000	00000	100000
add/sub /sll/srl…	$s0	$s1	$t0	シフト演算にのみ使用。シフト量を指定	add

大まかな分類

細かな指定（100010ならsub 000000ならsll）

add $t0, $s0, $s1

$t0 ← $s0 + $s1

レジスタ名	ビット列	レジスタ名	ビット列
$t0	01000	$s0	10000
$t1	01001	$s1	10001
…	…	…	…
$t7	01111	$s7	10111

Iタイプ命令

001000	10000	01000	0000000000001100
addi	$s0	$t0	+12

(addiu: 001001)

16ビット符号つき整数
（2の補数表現）
-32768～+32767

addi $t0, $s0, 12

$t0 ← $s0 + 12

レジスタ名	ビット列	レジスタ名	ビット列
$t0	01000	$s0	10000
$t1	01001	$s1	10001
…	…	…	…
$t7	01111	$s7	10111

Iタイプ命令

100011	10000	01000	0000000000001100
lw	$s0	$t0	+12

(sw: 101011)

16ビット符号つき整数
（2の補数表現）
-2^{15}〜$+2^{15}$

lw $t0, 12($s0)

アドレス$s0+12の語の内容をレジスタ$t0にロードする

レジスタ名	ビット列	レジスタ名	ビット列
$t0	01000	$s0	10000
$t1	01001	$s1	10001
…	…	…	…
$t7	01111	$s7	10111

Jタイプ命令

110110	00000100000000000000000101
j	1048581

j Label　(アセンブラがアドレス計算を行い
　　上記のような機械語を生成する)

Labelにジャンプする
（上記機械語の場合：PC ← 1048581）

命令の設計について:ビット数の決め方その一

- nビットが2^n通り表現できる
- 命令セット中の命令数で命令部(+機能部)の最小ビット数を決める
- レジスタの数でレジスタの最小ビット数を決める
- 残りのフィールドのビット数も必要に応じてその最小ビット数を決める
- 命令語のビット数は固定なので、上記すべてを満足するように、設計を行う

命令の設計について:ビット数の決め方その二

- 上記のように、命令→レジスタ→その他の順に各々のフィールドの最小ビットを計算して設計する方法のほか、複数のフィールドを総合的に考え、それぞれのビット数を決める方法も考えられる

命令部（?bit）	レジスタ（?bit）	レジスタ（?bit）	即値（オフセット）（?bit）

manaba小テスト：演習13-1

- 15分
- 7点

命令の実行

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:

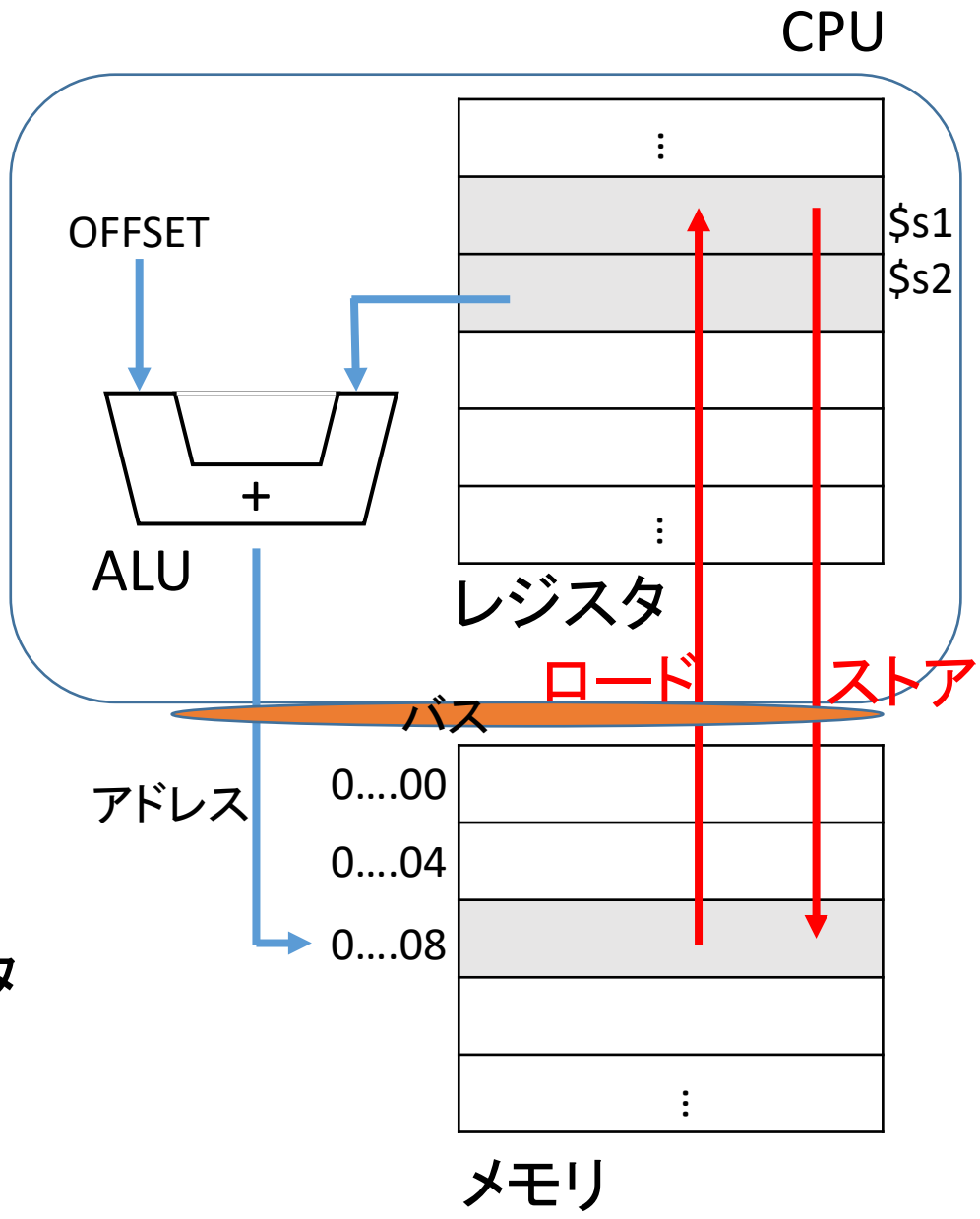
lw $s1, OFFSET($s2)

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



算術論理演算命令

加算:

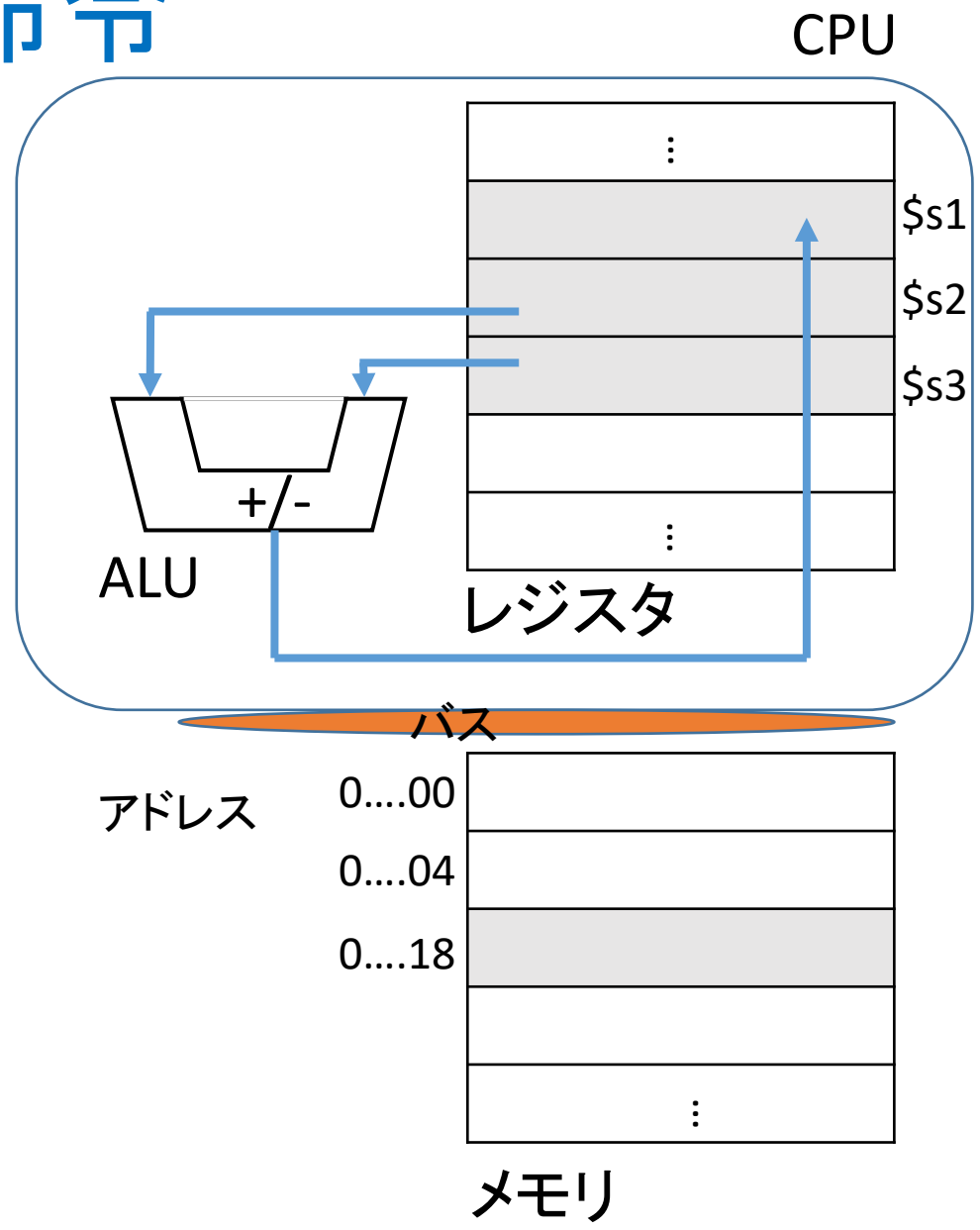
add $s1, $s2, $s3

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



符号拡張について

- 定数加算では、たとえば16ビットの即値（定数）を32ビットに符号拡張する必要がある
- ここで4 ビットの符号付きの数 0011 を 8 ビットに拡張することを考える。0011は正の整数なので、00000011 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 1100 を 8 ビットに 拡張するにはどうしたらよいか
  - 答えは 11111100 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）

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

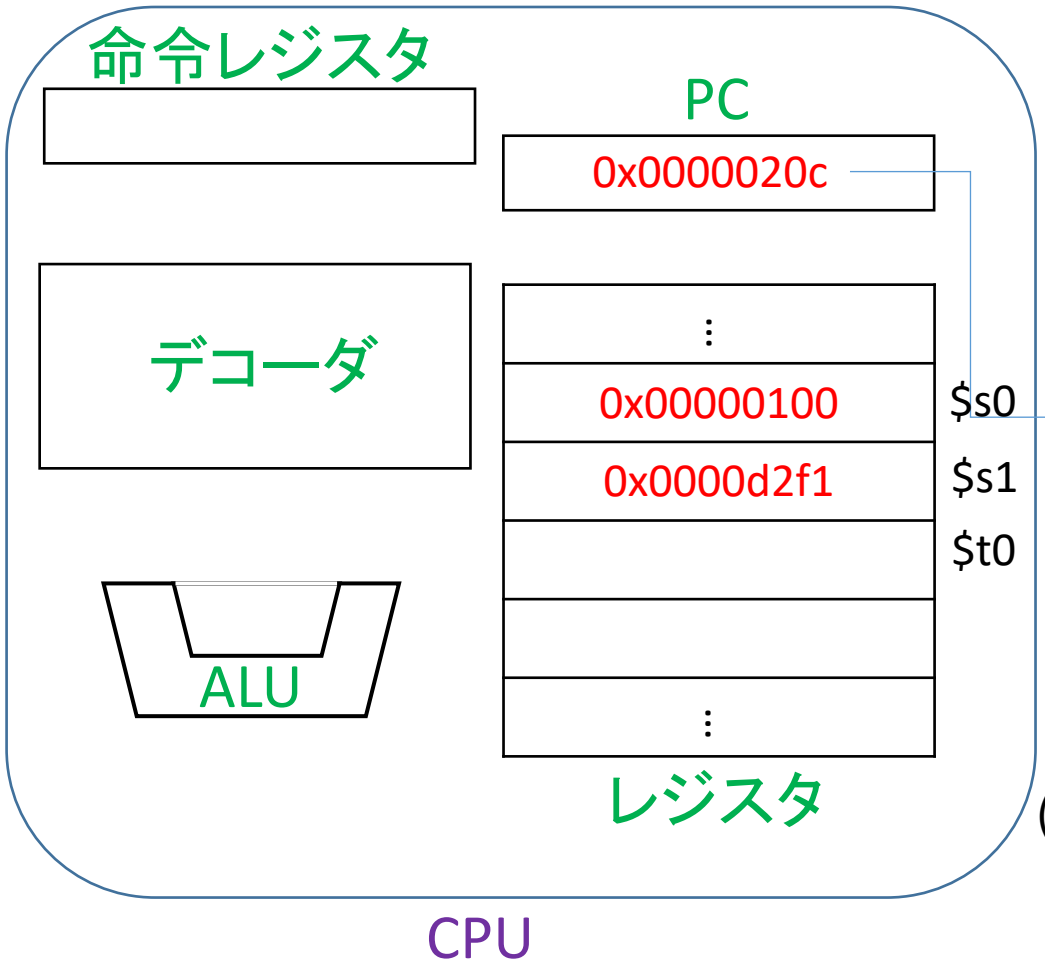
分岐命令：条件分岐

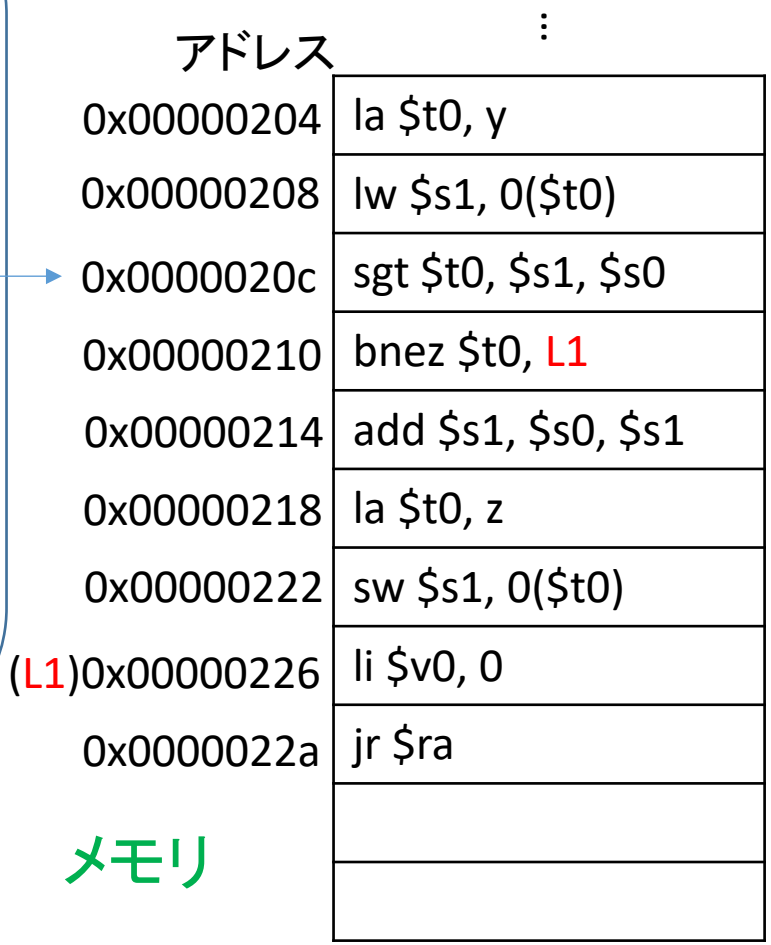
比較命令	分岐命令
比較= seq $r1, $r2, $r3 比較≠ sne $r1, $r2, $r3 比較< slt $r1, $r2, $r3 比較> sgt $r1, $r2, $r3 : :	条件= beqz $r1, Label $r1=0ならLabelに分岐する 条件≠ bnez $r1, Label $r1≠0ならLabelに分岐する

先頭のs: $r1に値1をset（比較条件を満足すれば）

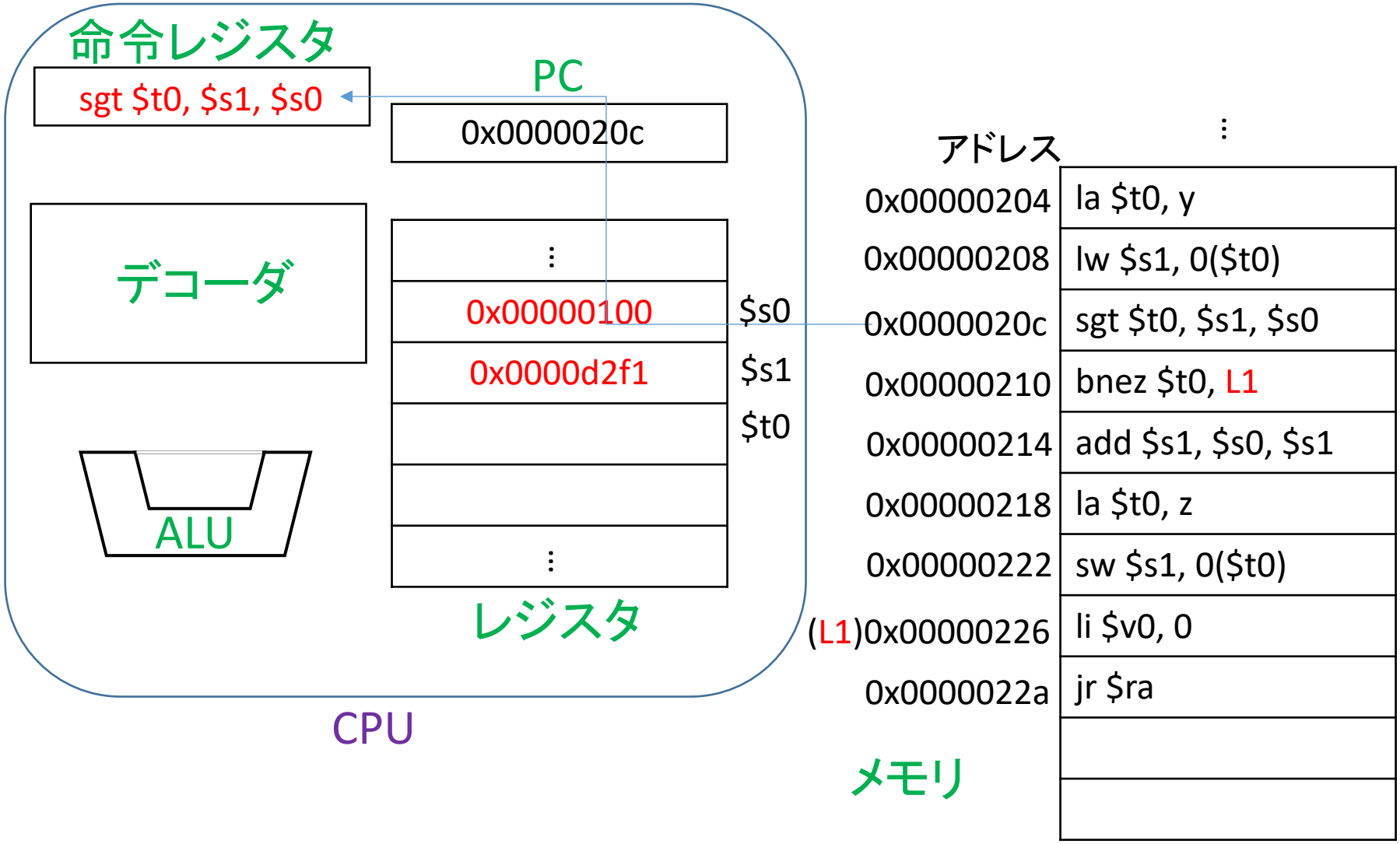
beqz: branch if equal 0
bnez: branch if not equal 0

分岐命令：(1)比較

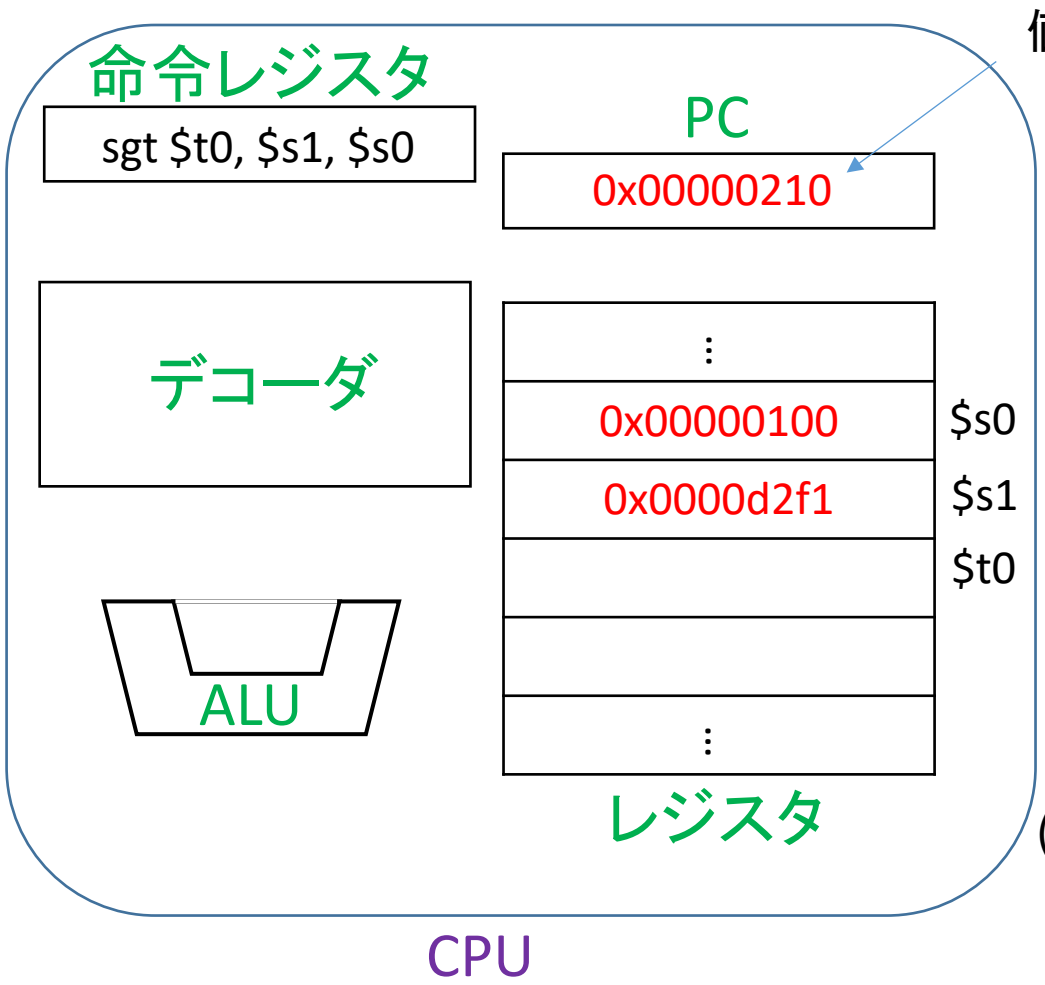

命令レジスタ
PC
0x0000020c
デコーダ
0x00000100
$s0
0x0000d2f1
$s1
$t0
ALU
レジスタ
CPU

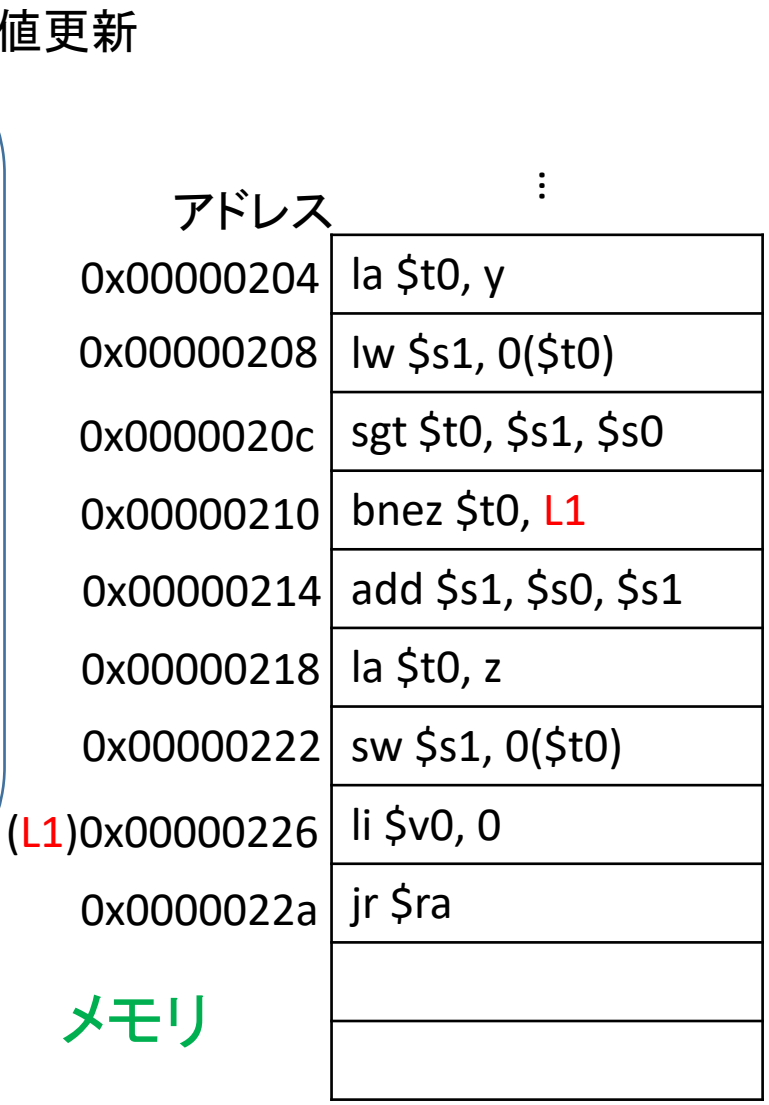

アドレス
0x00000204 la $t0, y
0x00000208 lw $s1, 0($t0)
0x0000020c sgt $t0, $s1, $s0
0x00000210 bnez $t0, L1
0x00000214 add $s1, $s0, $s1
0x00000218 la $t0, z
0x00000222 sw $s1, 0($t0)
(L1)0x00000226 li $v0, 0
0x0000022a jr $ra
メモリ

分岐命令：(1)比較

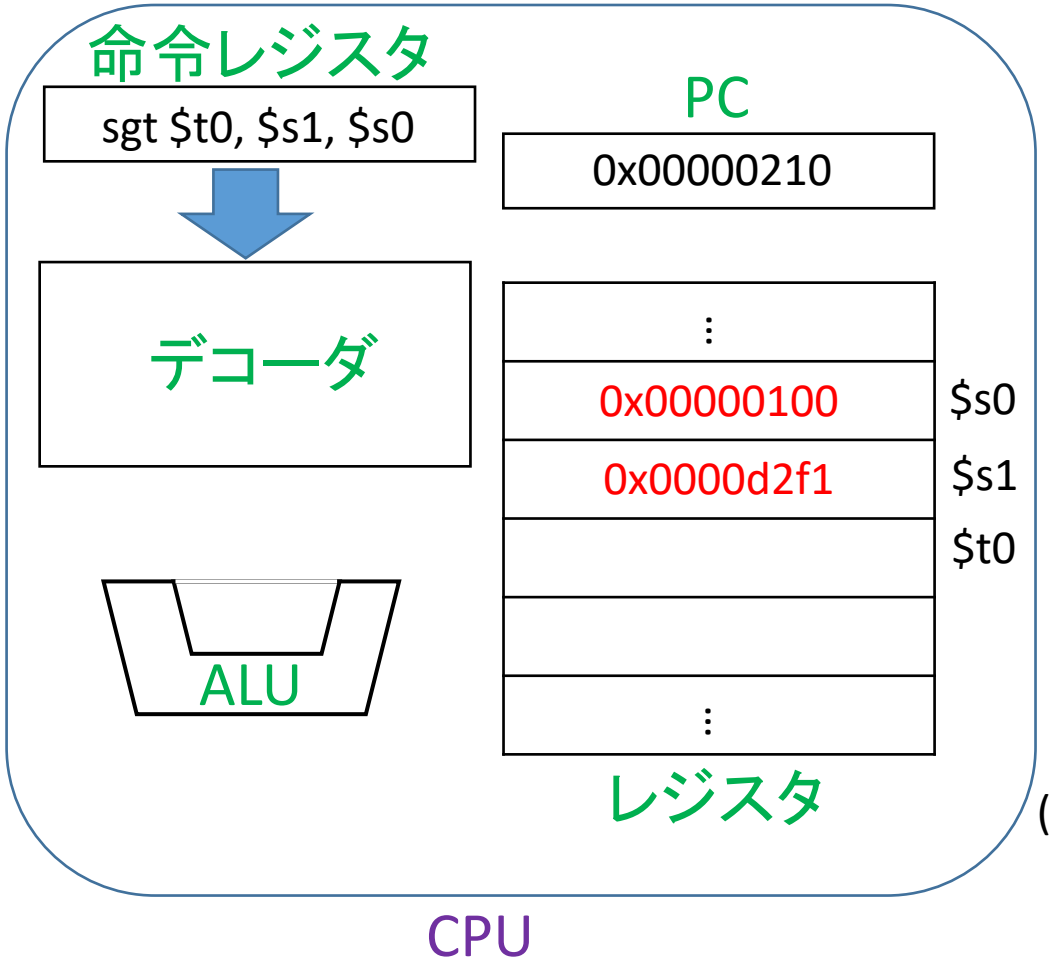

命令レジスタ
sgt $t0, $s1, $s0
PC
0x0000020c
デコーダ
ALU
0x00000100
0x0000d2f1
$s0
$s1
$t0
レジスタ
CPU
アドレス
0x00000204
0x00000208
0x0000020c
0x00000210
0x00000214
0x00000218
0x00000222
(L1)0x00000226
0x0000022a
la $t0, y
lw $s1, 0($t0)
sgt $t0, $s1, $s0
bnez $t0, L1
add $s1, $s0, $s1
la $t0, z
sw $s1, 0($t0)
li $v0, 0
jr $ra
メモリ

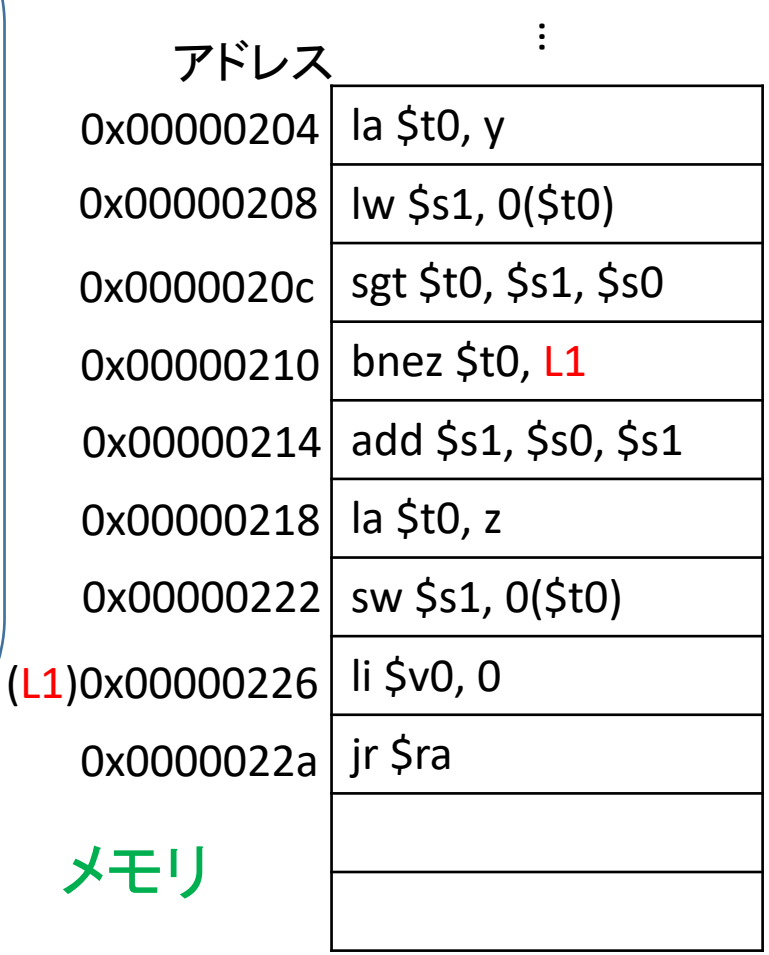
分岐命令：(1)比較


命令レジスタ
sgt $t0, $s1, $s0
PC
0x00000210
値更新
デコーダ
ALU
⋮
0x00000100
$s0
0x0000d2f1
$s1
$t0
⋮
レジスタ
CPU

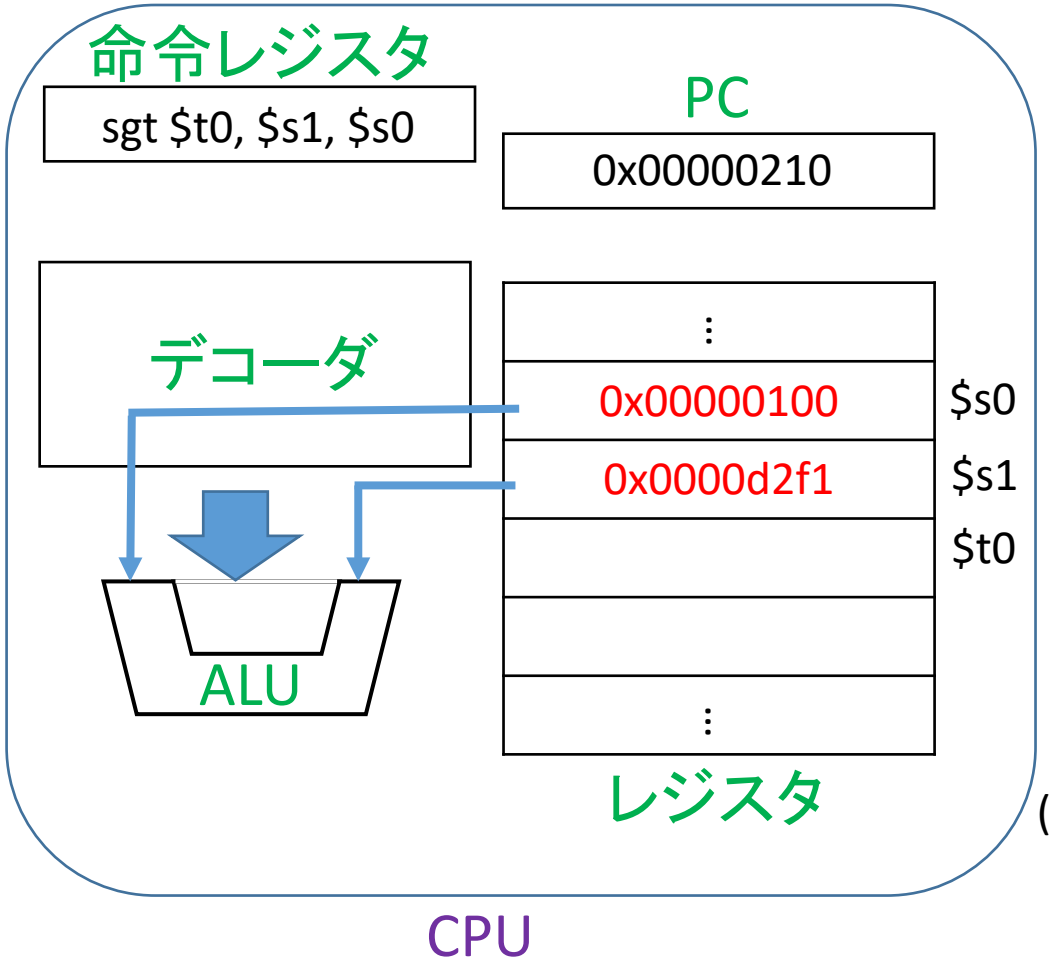

⋮
アドレス
0x00000204 la $t0, y
0x00000208 lw $s1, 0($t0)
0x0000020c sgt $t0, $s1, $s0
0x00000210 bnez $t0, L1
0x00000214 add $s1, $s0, $s1
0x00000218 la $t0, z
0x00000222 sw $s1, 0($t0)
(L1)0x00000226 li $v0, 0
0x0000022a jr $ra
メモリ

分岐命令：(1)比較


命令レジスタ
sgt $t0, $s1, $s0
PC
0x00000210
デコーダ
0x00000100
$s0
0x0000d2f1
$s1
$t0
ALU
レジスタ
CPU


アドレス
0x00000204 la $t0, y
0x00000208 lw $s1, 0($t0)
0x0000020c sgt $t0, $s1, $s0
0x00000210 bnez $t0, L1
0x00000214 add $s1, $s0, $s1
0x00000218 la $t0, z
0x00000222 sw $s1, 0($t0)
(L1)0x00000226 li $v0, 0
0x0000022a jr $ra
メモリ

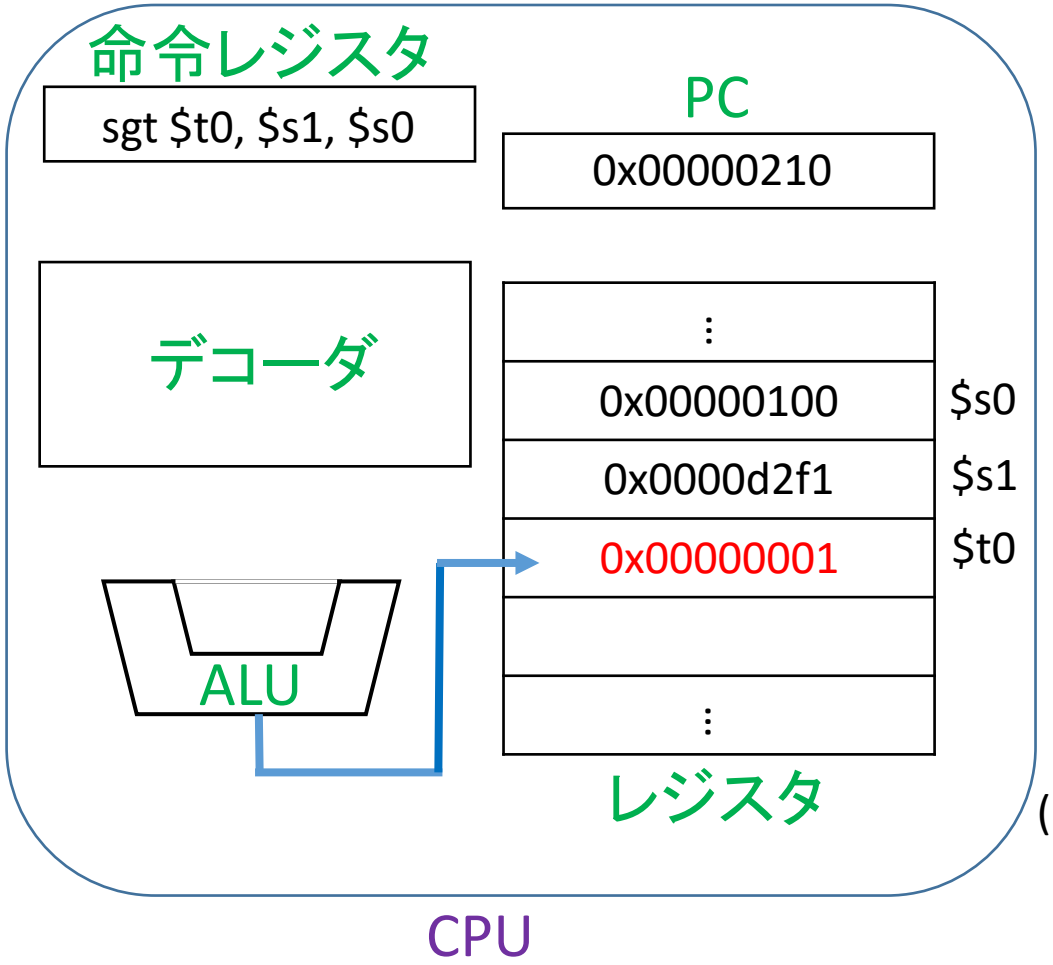
分岐命令：(1)比較



アドレス	
0x00000204	la $t0, y
0x00000208	lw $s1, 0($t0)
0x0000020c	sgt $t0, $s1, $s0
0x00000210	bnez $t0, L1
0x00000214	add $s1, $s0, $s1
0x00000218	la $t0, z
0x00000222	sw $s1, 0($t0)
(L1)0x00000226	li $v0, 0
0x0000022a	jr $ra

メモリ

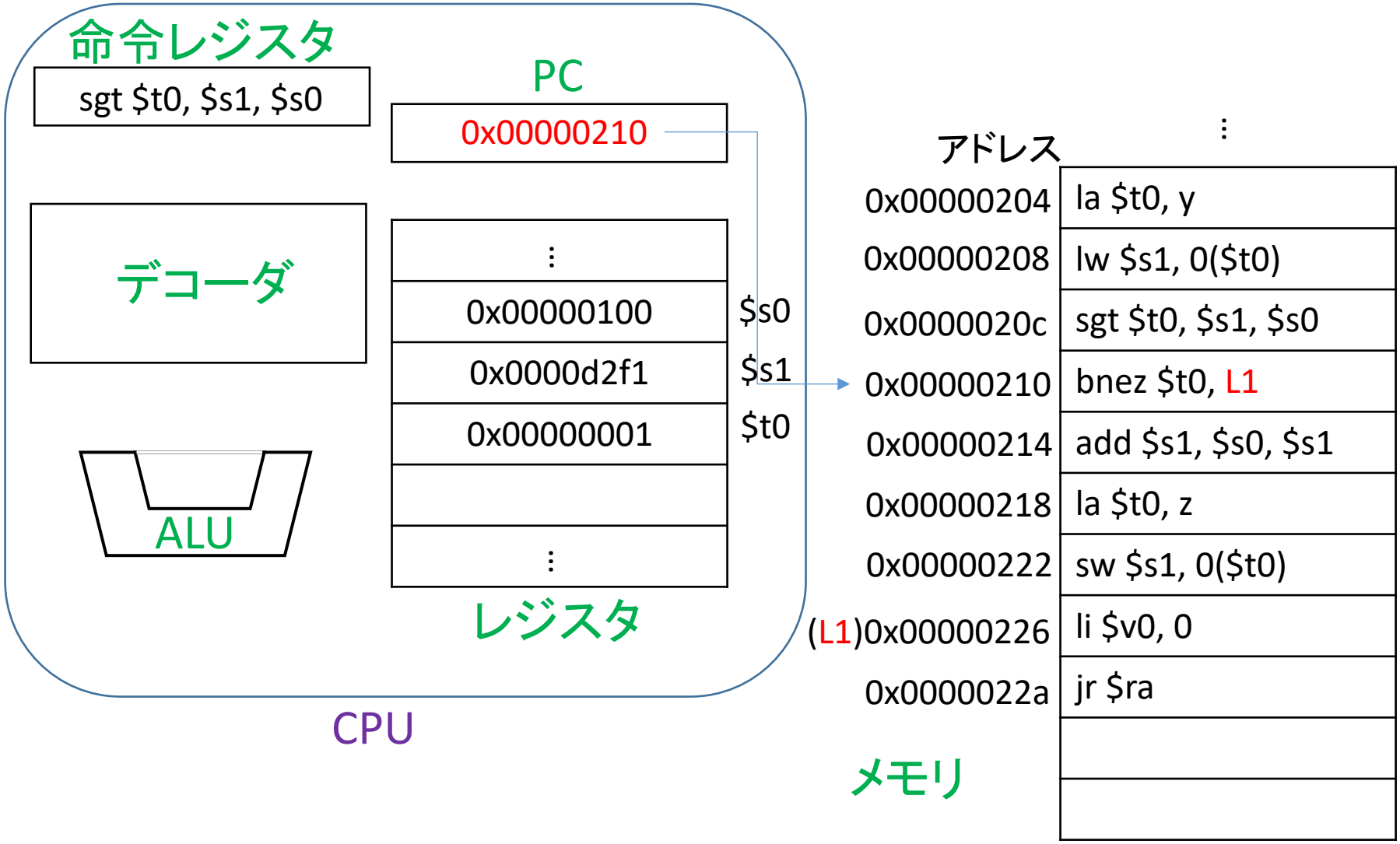
分岐命令：(1)比較



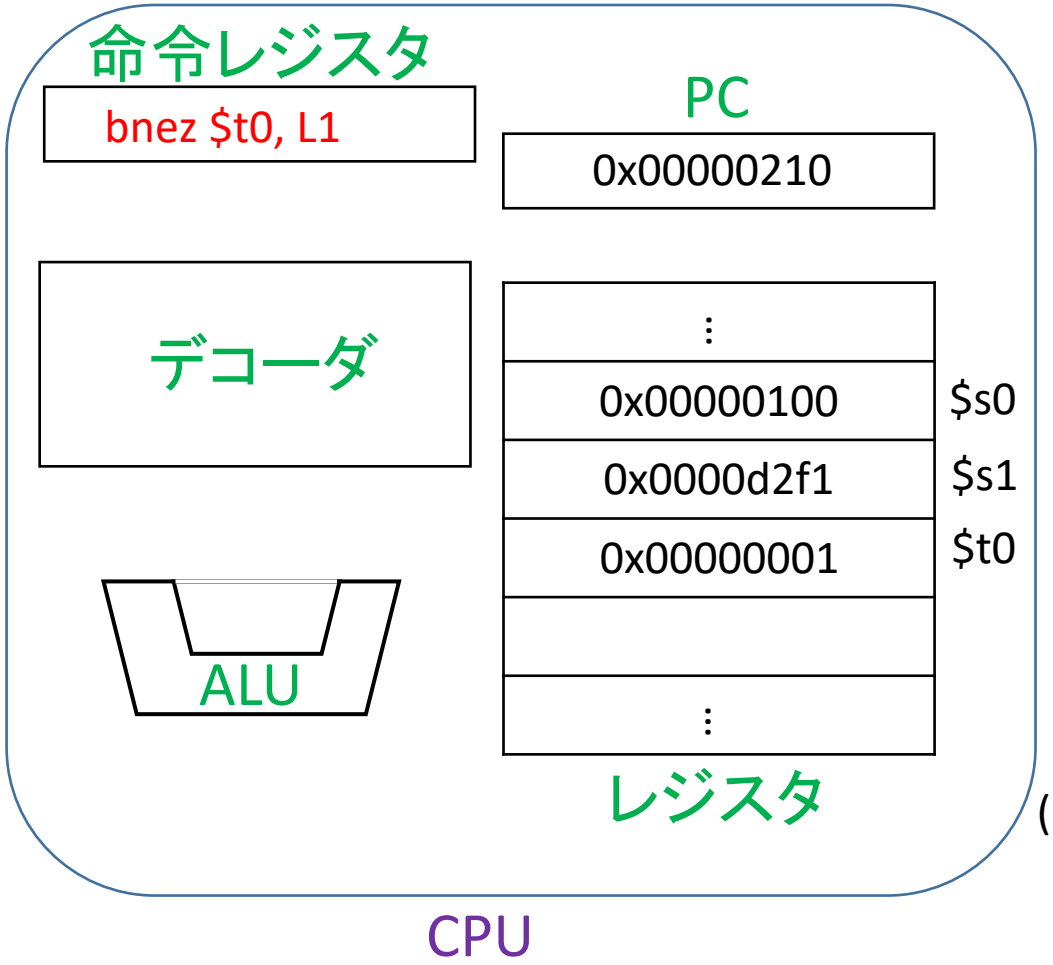
アドレス	
	⋮
0x00000204	la $t0, y
0x00000208	lw $s1, 0($t0)
0x0000020c	sgt $t0, $s1, $s0
0x00000210	bnez $t0, L1
0x00000214	add $s1, $s0, $s1
0x00000218	la $t0, z
0x00000222	sw $s1, 0($t0)
(L1)0x00000226	li $v0, 0
0x0000022a	jr $ra

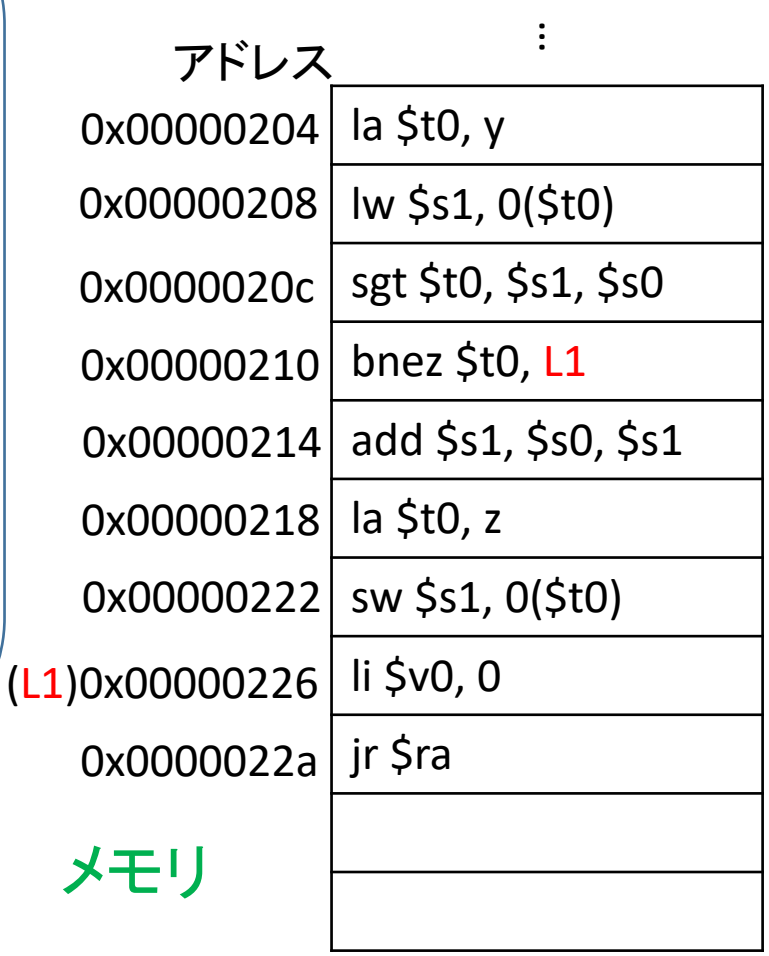
メモリ

分岐命令:(2)条件分岐

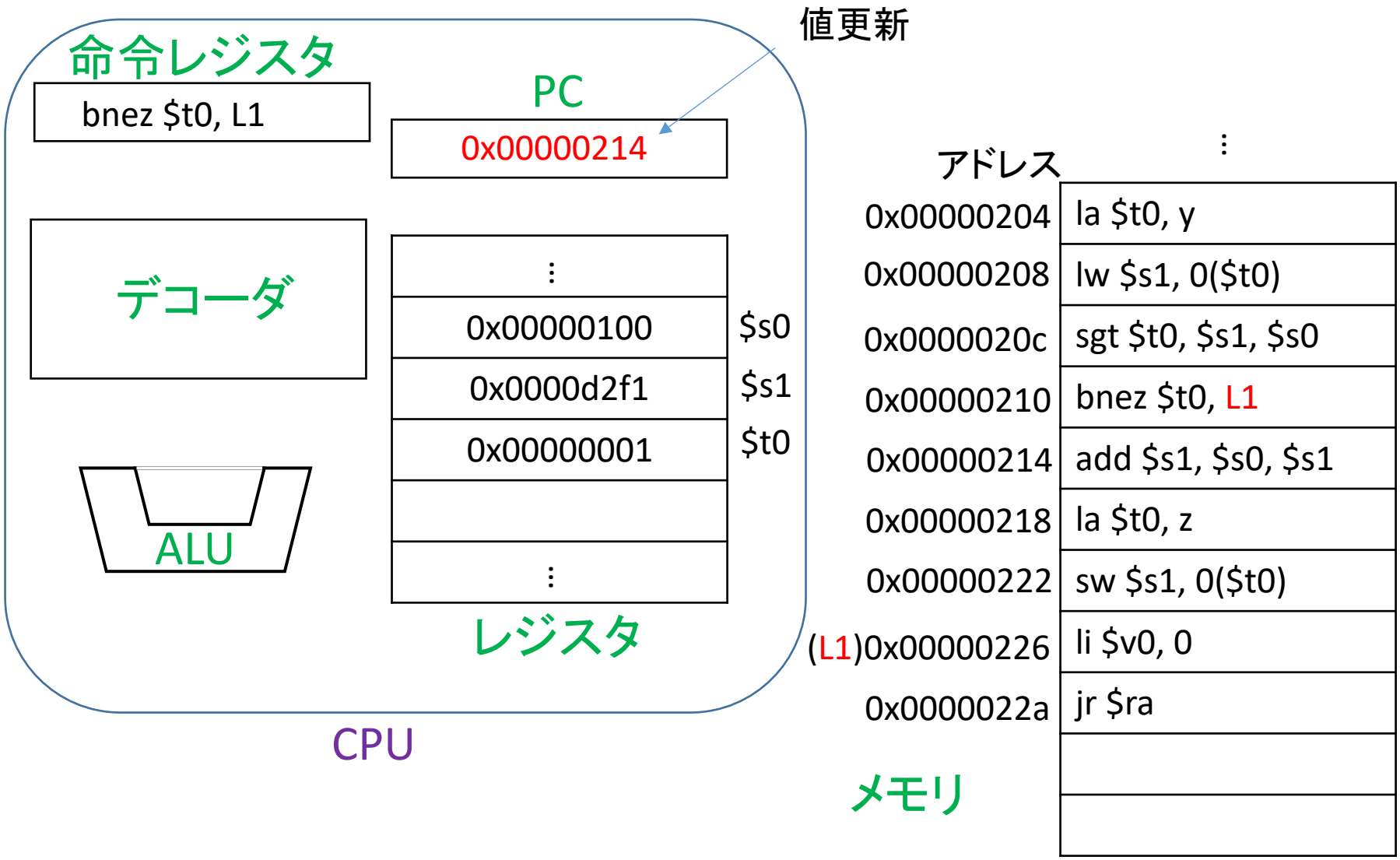

命令レジスタ
sgt $t0, $s1, $s0
PC
0x00000210
デコーダ
ALU
⋮
0x00000100
0x0000d2f1
0x00000001
$s0
$s1
$t0
⋮
レジスタ
CPU
アドレス
⋮
0x00000204
la $t0, y
0x00000208
lw $s1, 0($t0)
0x0000020c
sgt $t0, $s1, $s0
0x00000210
bnez $t0, L1
0x00000214
add $s1, $s0, $s1
0x00000218
la $t0, z
0x00000222
sw $s1, 0($t0)
(L1)0x00000226
li $v0, 0
0x0000022a
jr $ra
メモリ

分岐命令:(2)条件分岐

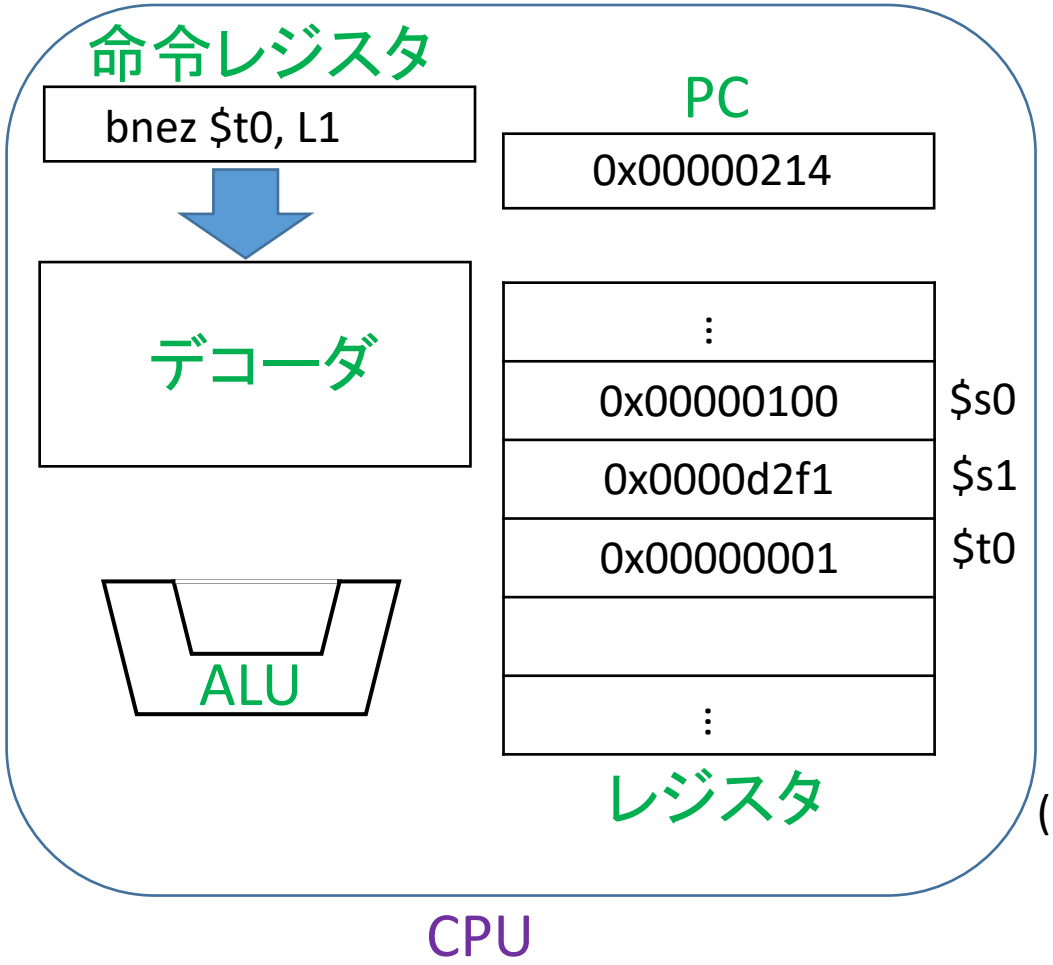

命令レジスタ
bnez $t0, L1
PC
0x00000210
デコーダ
⋮
0x00000100
$s0
0x0000d2f1
$s1
0x00000001
$t0
⋮
ALU
レジスタ
CPU

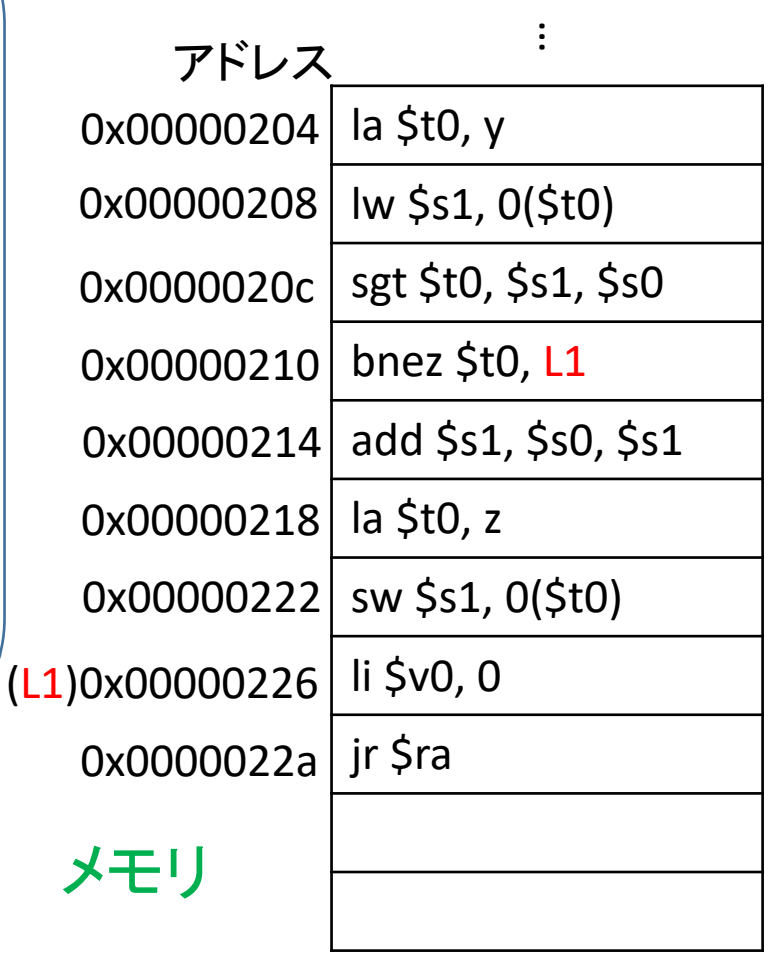

⋮
アドレス
0x00000204 la $t0, y
0x00000208 lw $s1, 0($t0)
0x0000020c sgt $t0, $s1, $s0
0x00000210 bnez $t0, L1
0x00000214 add $s1, $s0, $s1
0x00000218 la $t0, z
0x00000222 sw $s1, 0($t0)
(L1)0x00000226 li $v0, 0
0x0000022a jr $ra
メモリ

分岐命令:(2)条件分岐

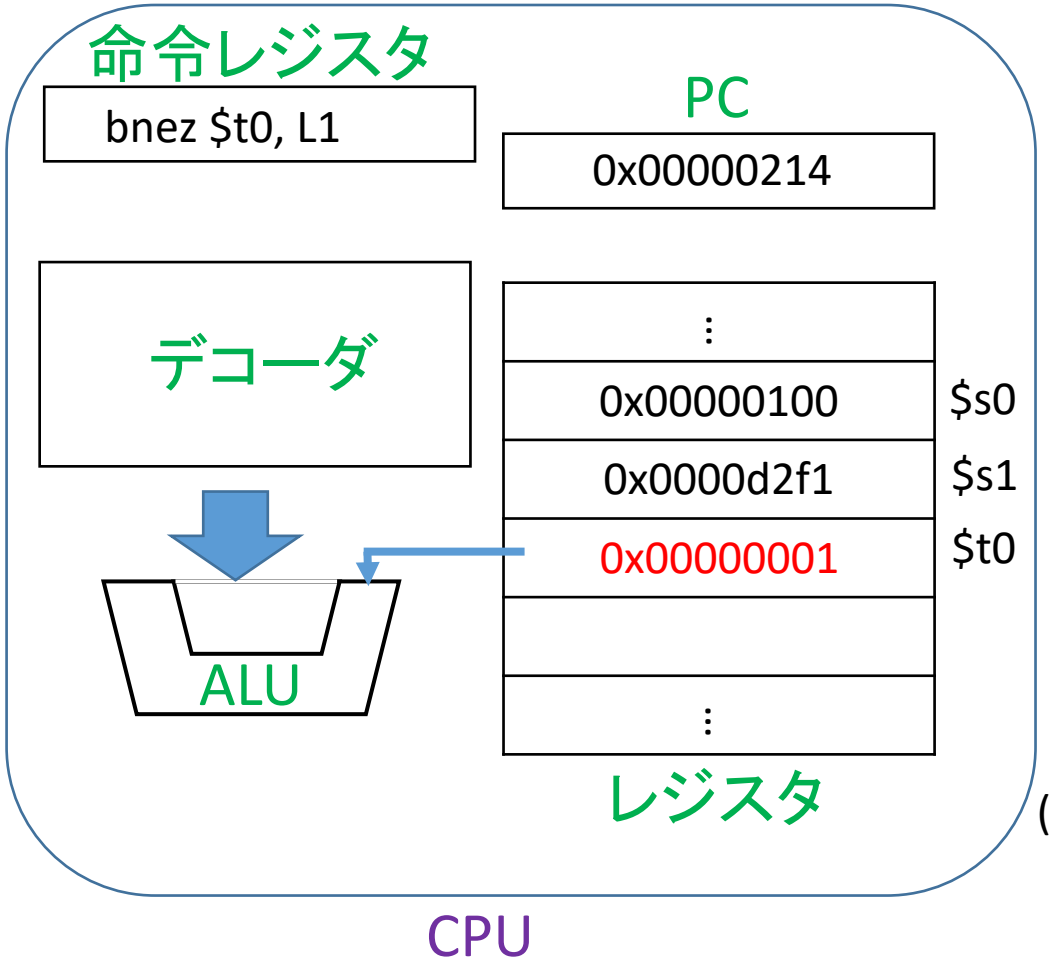

値更新
命令レジスタ
bnez $t0, L1
PC
0x00000214
デコーダ
ALU
⋮
0x00000100
$s0
0x0000d2f1
$s1
0x00000001
$t0
⋮
レジスタ
CPU
⋮
アドレス
0x00000204 la $t0, y
0x00000208 lw $s1, 0($t0)
0x0000020c sgt $t0, $s1, $s0
0x00000210 bnez $t0, L1
0x00000214 add $s1, $s0, $s1
0x00000218 la $t0, z
0x00000222 sw $s1, 0($t0)
(L1)0x00000226 li $v0, 0
0x0000022a jr $ra
メモリ

分岐命令:(2)条件分岐


命令レジスタ
bnez $t0, L1
PC
0x00000214
デコーダ
⋮
0x00000100
$s0
0x0000d2f1
$s1
0x00000001
$t0
⋮
ALU
レジスタ
CPU


⋮
アドレス
0x00000204 la $t0, y
0x00000208 lw $s1, 0($t0)
0x0000020c sgt $t0, $s1, $s0
0x00000210 bnez $t0, L1
0x00000214 add $s1, $s0, $s1
0x00000218 la $t0, z
0x00000222 sw $s1, 0($t0)
(L1)0x00000226 li $v0, 0
0x0000022a jr $ra
メモリ

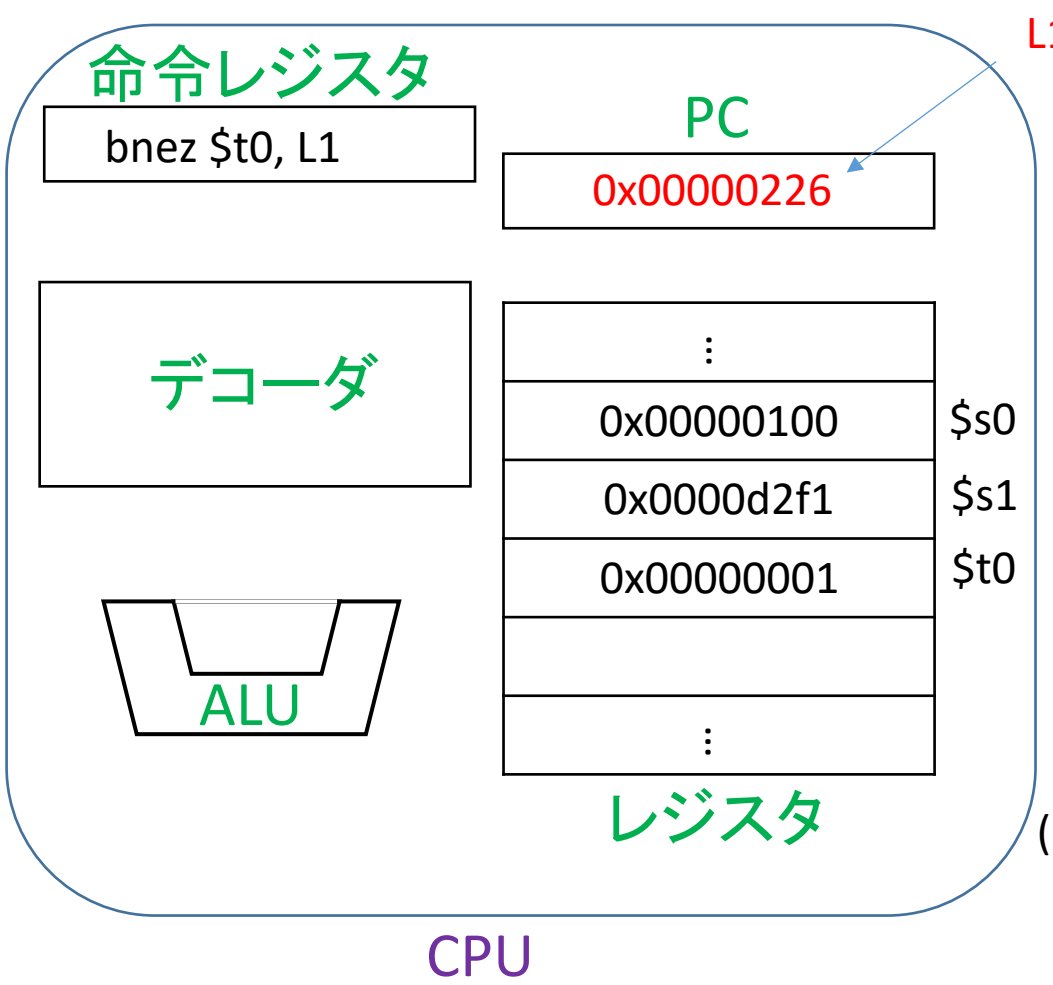
分岐命令：(2)条件分岐



アドレス	
	⋮
0x00000204	la $t0, y
0x00000208	lw $s1, 0($t0)
0x0000020c	sgt $t0, $s1, $s0
0x00000210	bnez $t0, L1
0x00000214	add $s1, $s0, $s1
0x00000218	la $t0, z
0x00000222	sw $s1, 0($t0)
(L1)0x00000226	li $v0, 0
0x0000022a	jr $ra

メモリ

分岐命令:(2)条件分岐

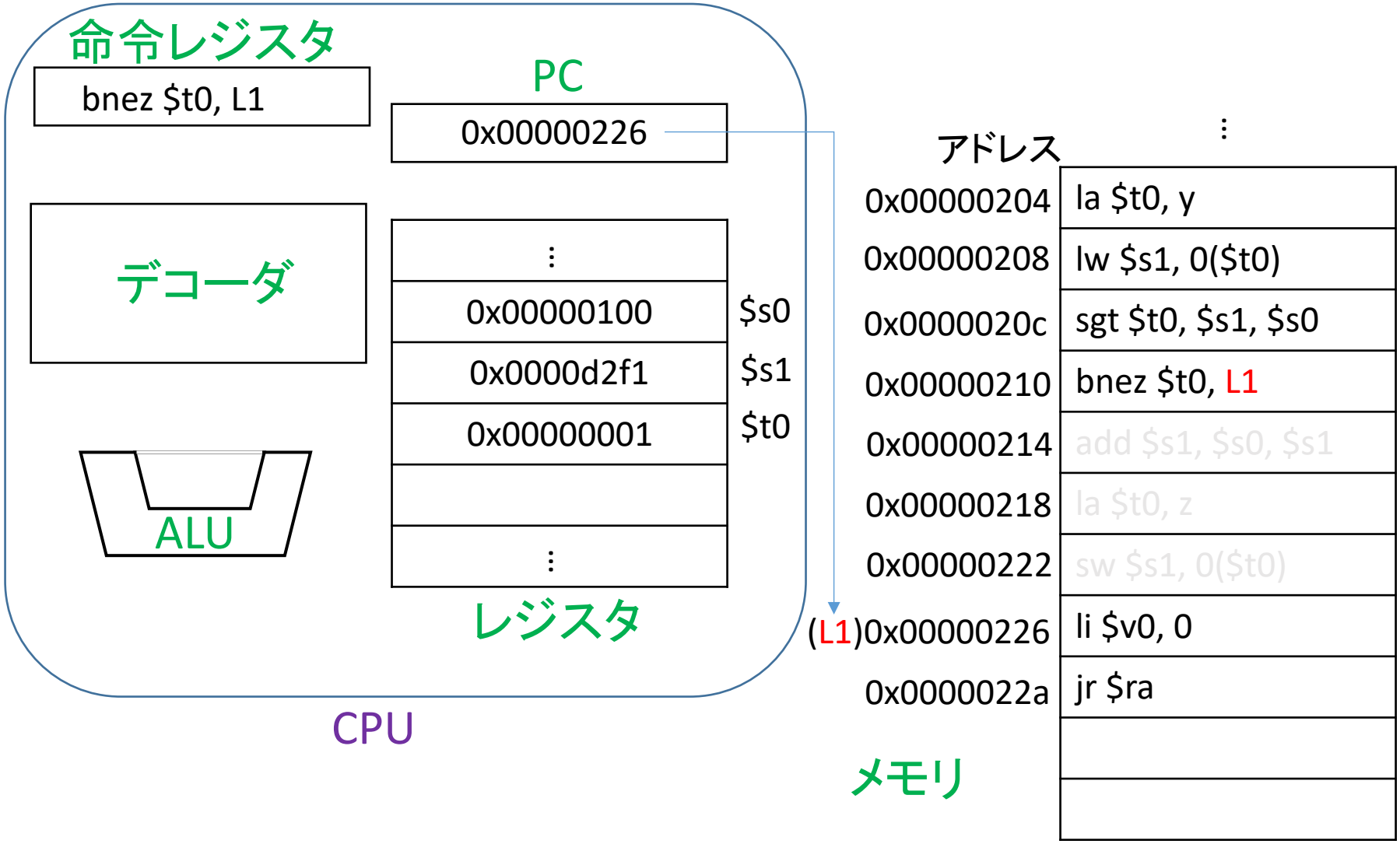


L1で値更新

アドレス	
0x00000204	la $t0, y
0x00000208	lw $s1, 0($t0)
0x0000020c	sgt $t0, $s1, $s0
0x00000210	bnez $t0, L1
0x00000214	add $s1, $s0, $s1
0x00000218	la $t0, z
0x00000222	sw $s1, 0($t0)
(L1)0x00000226	li $v0, 0
0x0000022a	jr $ra

メモリ

分岐命令:(2)条件分岐


命令レジスタ
bnez $t0, L1
PC
0x00000226
デコーダ
ALU
0x00000100
$s0
0x0000d2f1
$s1
0x00000001
$t0
レジスタ
CPU
アドレス
0x00000204
la $t0, y
0x00000208
lw $s1, 0($t0)
0x0000020c
sgt $t0, $s1, $s0
0x00000210
bnez $t0, L1
0x00000214
add $s1, $s0, $s1
0x00000218
la $t0, z
0x00000222
sw $s1, 0($t0)
(L1)0x00000226
li $v0, 0
0x0000022a
jr $ra
メモリ

manaba小テスト：演習14-2

- 10分
- 4点

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する 足し算であれば (i)レジスタの値をALUで演算し、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
  - 順序回路（ラッチ、フリップフロップ、カウンタ）