MIPSのまとめ

（教科書3.7節～3.8節，4.1節～4.4節）

レジスタ

MIPS のレジスタ

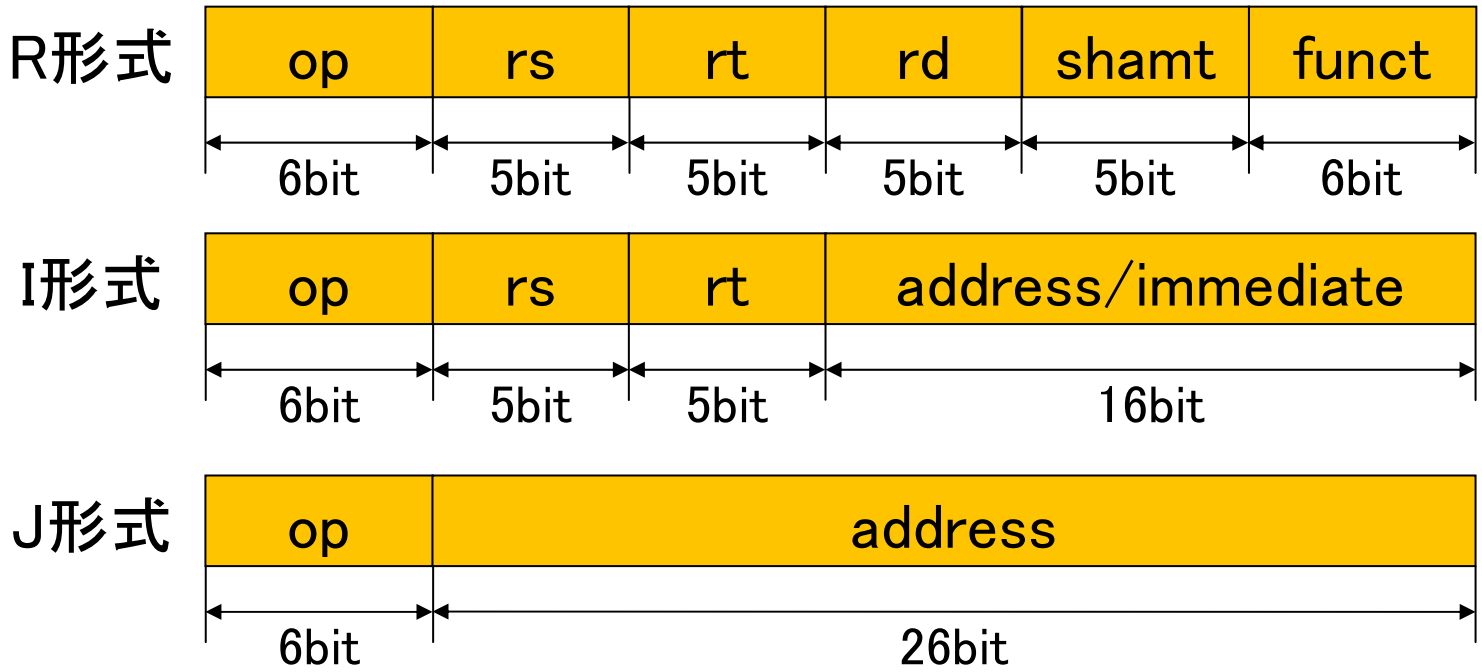
レジスタ名	レジスタ番号	用途	関数呼出時のレジスタ破壊
$zero	0	定数値 0，不要な結果の破棄	(NA)
$v0～$v1	2～3	関数の返り値	可
$a0～$a3	4～7	引数用	不可
$t0～$t7	8～15	一時保存用	可
$s0～$s7	16～23	変数用	不可
$t8～$t9	24～25	一時保存用	可
$sp	29	スタックポインタ(後述)	不可
$fp	30	フレームポインタ(後述)	不可
$ra	31	戻り番地	不可

プログラムカウンタ PC：次に実行する命令の番地を格納．非分岐命令実行ごとに +4 され，また分岐命令実行時に更新される．

命令形式

命令形式: 命令語のフィールド構成.

- MIPS は命令語を32bit 幅で統一している.
- MIPS の命令形式は, R形式, I形式, J形式の3種類がある.
- どの命令形式かは, op フィールド(**命令操作コード**)で判別できる.



命令セット

- 算術論理演算
  - 加減算命令（R形式，I形式）
  - 比較命令（R形式，I形式）
  - 論理演算命令（R形式，I形式）
  - シフト命令（R形式）
- データ転送
  - メモリロード命令（I形式）
  - メモリストア命令（I形式）
  - 即値ロード命令（I形式）
- 分岐
  - 条件分岐命令（I形式）
  - 無条件分岐命令（R形式，J形式）

加減算命令(1)

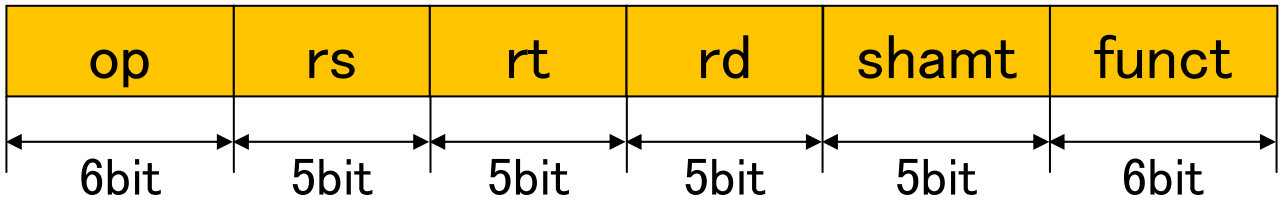
レジスタ±レジスタ → レジスタ

add	$○, $△, $□	# $△ + $□ → $○
sub	$○, $△, $□	# $△ − $□ → $○
addu	$○, $△, $□	# $△ + $□ → $○
subu	$○, $△, $□	# $△ − $□ → $○

- add/sub 命令は，2の補数表現された符号つき整数の加減算を行う.
- addu/subu 命令は，符号なし整数の加減算を行う.
- add/sub 命令と addu/subu 命令はオーバーフロー発生時の振る舞いが違う.（計算については同じことを行う.）

加減算命令(2)

R形式



命令	op	rs	rt	rd	shamt	funct
add $○, $△, $□	000000	$△	$□	$○	00000	100000
addu $○, $△, $□	000000	$△	$□	$○	00000	100001
sub $○, $△, $□	000000	$△	$□	$○	00000	100010
subu $○, $△, $□	000000	$△	$□	$○	00000	100011

加減算命令(3)

レジスタ + 定数 → レジスタ

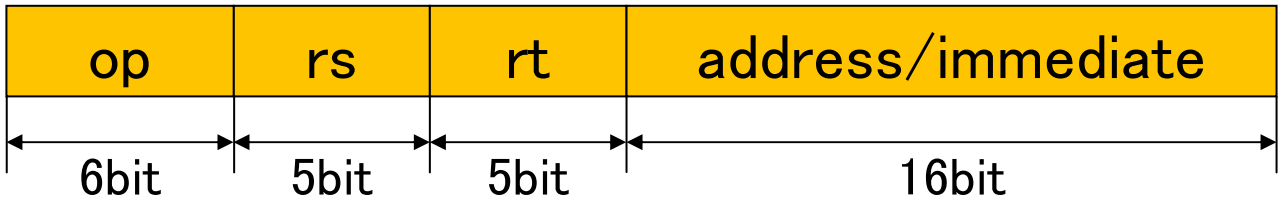
```
addi    $○, $△, n      # $△ + n → $○
addiu   $○, $△, n      # $△ + n → $○
```

即値オペランド

- addi/addiu 命令では，定数 n に 16bit の2の補数表現の符号つき整数(−32768～32767)を指定できる.
- 定数 n は 32bit に符号拡張されてから加算される.
- addi 命令と addiu 命令はオーバーフロー発生時の振る舞いが違う.（計算については同じことを行う.）
- 「subi」「subiu」は存在しない.（∵addi, addiu で代用すればよい.）

加減算命令(4)

I形式



命令	op	rs	rt	address/immediate
addi $○, $△, n	001000	$△	$○	n
addiu $○, $△, n	001001	$△	$○	n

加減算命令(5)

例）$s1 に 65539 が格納されているとき，

addi　$s0, $s1, 13

	00000000 00000001	00000000 00000011	$s1: $65539_{(10)}$
+)	00000000 00000000	00000000 00001001	$13_{(10)}$
	00000000 00000001	00000000 00001100	$s0: $65552_{(10)}$

addi　$s0, $s1, −13

	00000000 00000001	00000000 00000011	$s1: $65539_{(10)}$
+)	11111111 11111111	11111111 11110111	$-13_{(10)}$
	00000000 00000000	11111111 11111010	$s0: $65526_{(10)}$

比較命令(1)

レジスタ < レジスタ → レジスタ

```
slt     $○, $△, $□     # $△ < $□ → $○
sltu    $○, $△, $□     # $△ < $□ → $○
```

レジスタ < 定数 → レジスタ(定数は 32bit に符号拡張)

```
slti    $○, $△, n      # $△ < n → $○
sltiu   $○, $△, n      # $△ < n → $○
```

- slt/slti 命令は，第二オペランドと第三オペランドを 2 の補数表現された符号つき整数とみなして，比較結果を $○ に書き込む．
- sltu/sltiu 命令は，第二オペランドと第三オペランドを符号なし整数とみなして，比較結果を $○ に書き込む．
- slti/sltiu 命令では，定数 n は 32bit に符号拡張されてから比較される．

論理演算命令(1)

レジスタ and/or レジスタ → レジスタ

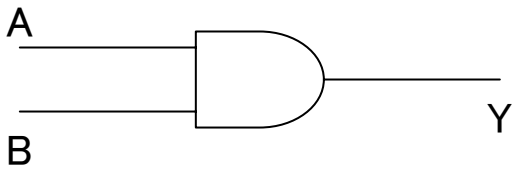
```
and     $○, $△, $□     # $△ & $□ → $○
or      $○, $△, $□     # $△ | $□ → $○
xor     $○, $△, $□     # $△ ^ $□ → $○
```

レジスタ and/or 定数 → レジスタ(定数は 32bit にゼロ拡張)

```
andi    $○, $△, n      # $△ & n → $○
ori     $○, $△, n      # $△ | n → $○
xori    $○, $△, n      # $△ ^ n → $○
```

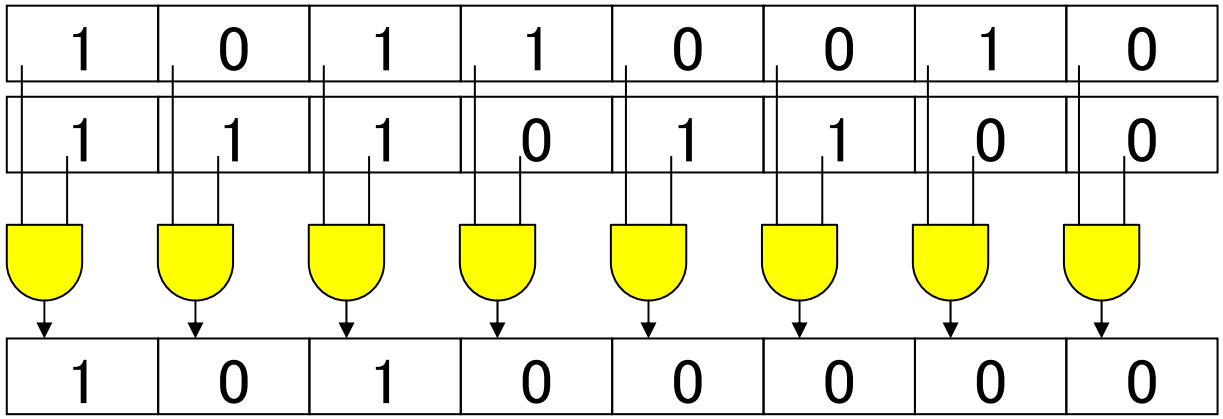
論理演算命令(2)

ANDゲート: 入力が全て 1 のときのみ出力が 1.



A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

AND演算(&): オペランドのビットごとに AND をとる.



シフト命令（1）

レジスタを n ビット論理左右シフト → レジスタ

```
sll     $○, $△, n      # $△ を n ビット論理左シフト → $○
srl     $○, $△, n      # $△ を n ビット論理右シフト → $○
```

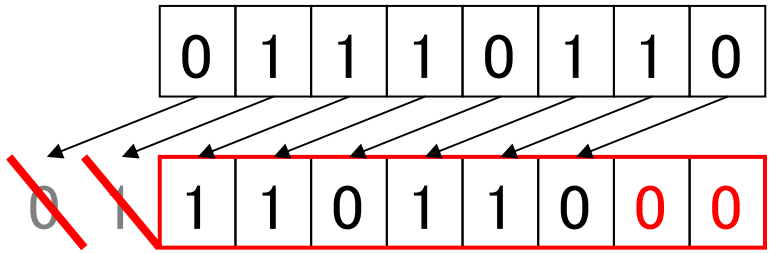
レジスタを n ビット算術右シフト → レジスタ

```
sra     $○, $△, n      # $△ を n ビット算術右シフト → $○
```

シフト命令(2)

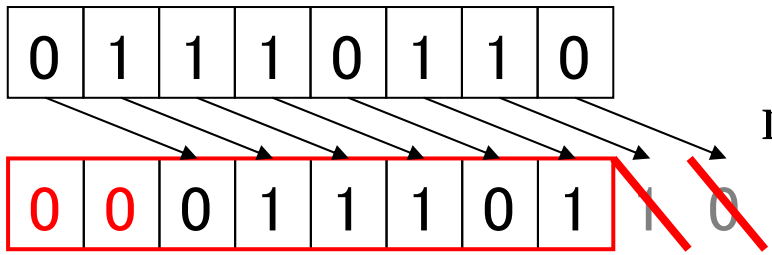
n ビット論理シフト演算: n ビット分を左右にずらし，はみ出したビットを捨てて，空いたビットに 0 を補填する.

n ビット左シフト:



n=2の場合

n ビット右シフト:

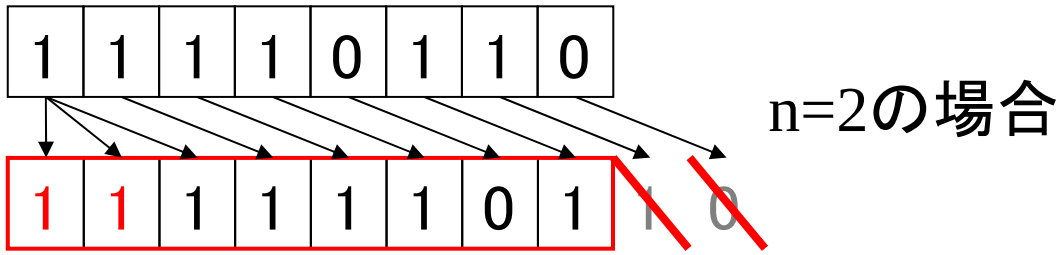


n=2の場合

シフト命令（3）

n ビット算術右シフト演算: n ビット分を右にずらし，はみ出したビットを捨てて，空いたビットに 最上位ビット（符号ビット） を補填する.

n ビット右シフト:



データ転送命令(1)

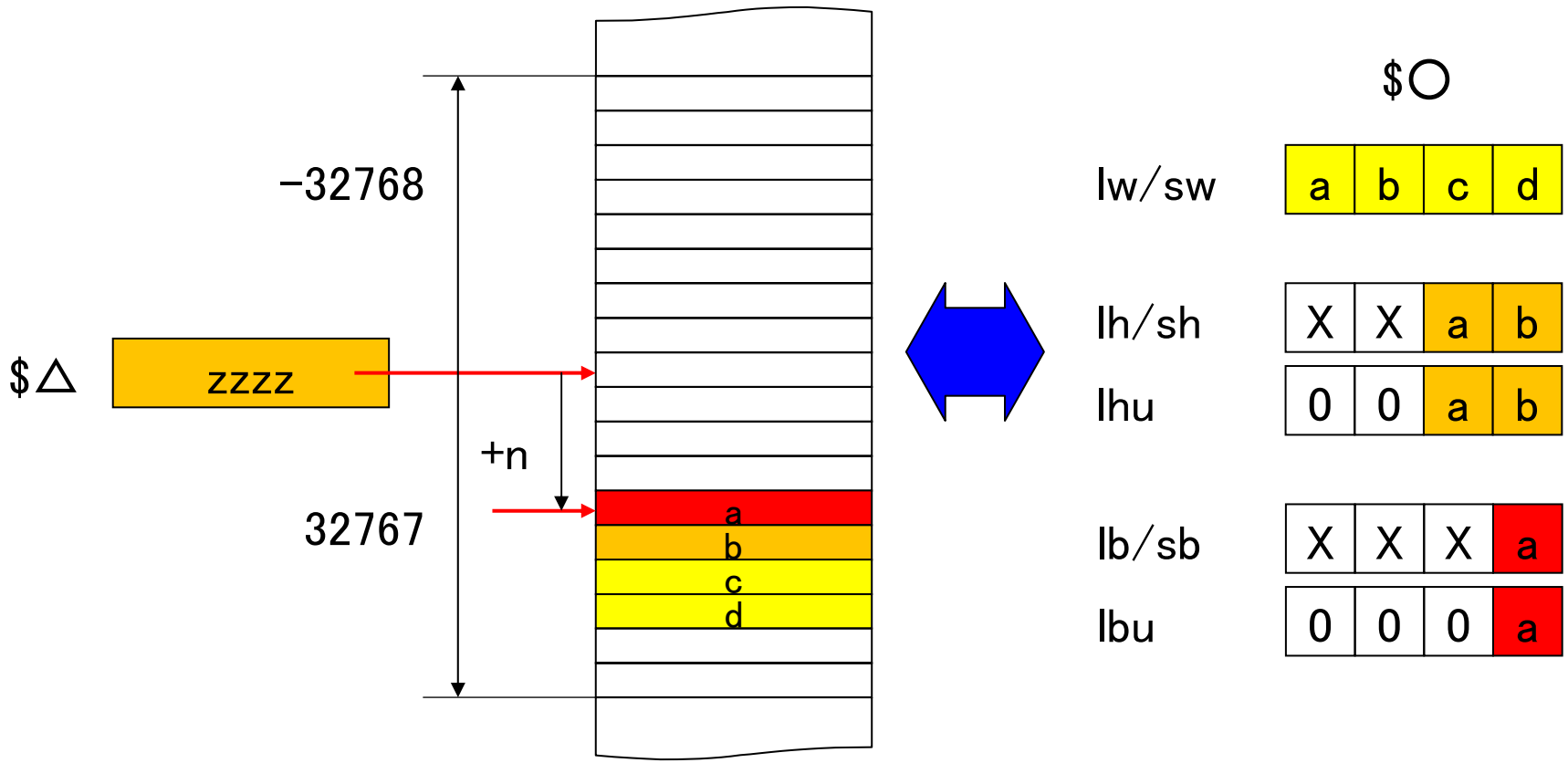
レジスタ／メモリ間(符号つき)

lw	$○, n($△)	# ($△ + n)番地 → $○, 1ワード(32bit)
lh	$○, n($△)	# ($△ + n)番地 → $○, 1ハーフワード(16bit)
lhu	$○, n($△)	# ($△ + n)番地 → $○, 1ハーフワード(16bit)
lb	$○, n($△)	# ($△ + n)番地 → $○, 1バイト(8bit)
lbu	$○, n($△)	# ($△ + n)番地 → $○, 1バイト(8bit)
sw	$○, n($△)	# $○ → ($△ + n)番地, 1ワード(32bit)
sh	$○, n($△)	# $○ → ($△ + n)番地, 1ハーフワード(16bit)
sb	$○, n($△)	# $○ → ($△ + n)番地, 1バイト(8bit)

- オフセット n には, 16bit の2の補数表現の整数(−32768～32767)を指定できる.
- オフセット n は 32bit に符号拡張されてからベースレジスタに加算される.
- lw/sw 命令の対象番地は4の倍数, lh/sh 命令の対象番地は2の倍数でなければならない.(整列化制約)
- lb/lh 命令では読み込むデータの符号拡張, lbu/lhu 命令ではゼロ拡張が行われる.

データ転送命令(2)

lw/lh/lb/lhu/lbu/sw/sh/sb　$○, n($△)



データ転送命令（4）

即値→レジスタ

```
lui     $○, n      # $○の上位 16bit に n を格納
                   # $○の下位 16bit に 0 を格納
```

32bit 即値（nm）の転送

```
lui     $○, n      # $○の上位 16bit に n を格納
addi $○, m         # $○の下位 16bit に m を格納（符号拡張あり）
```

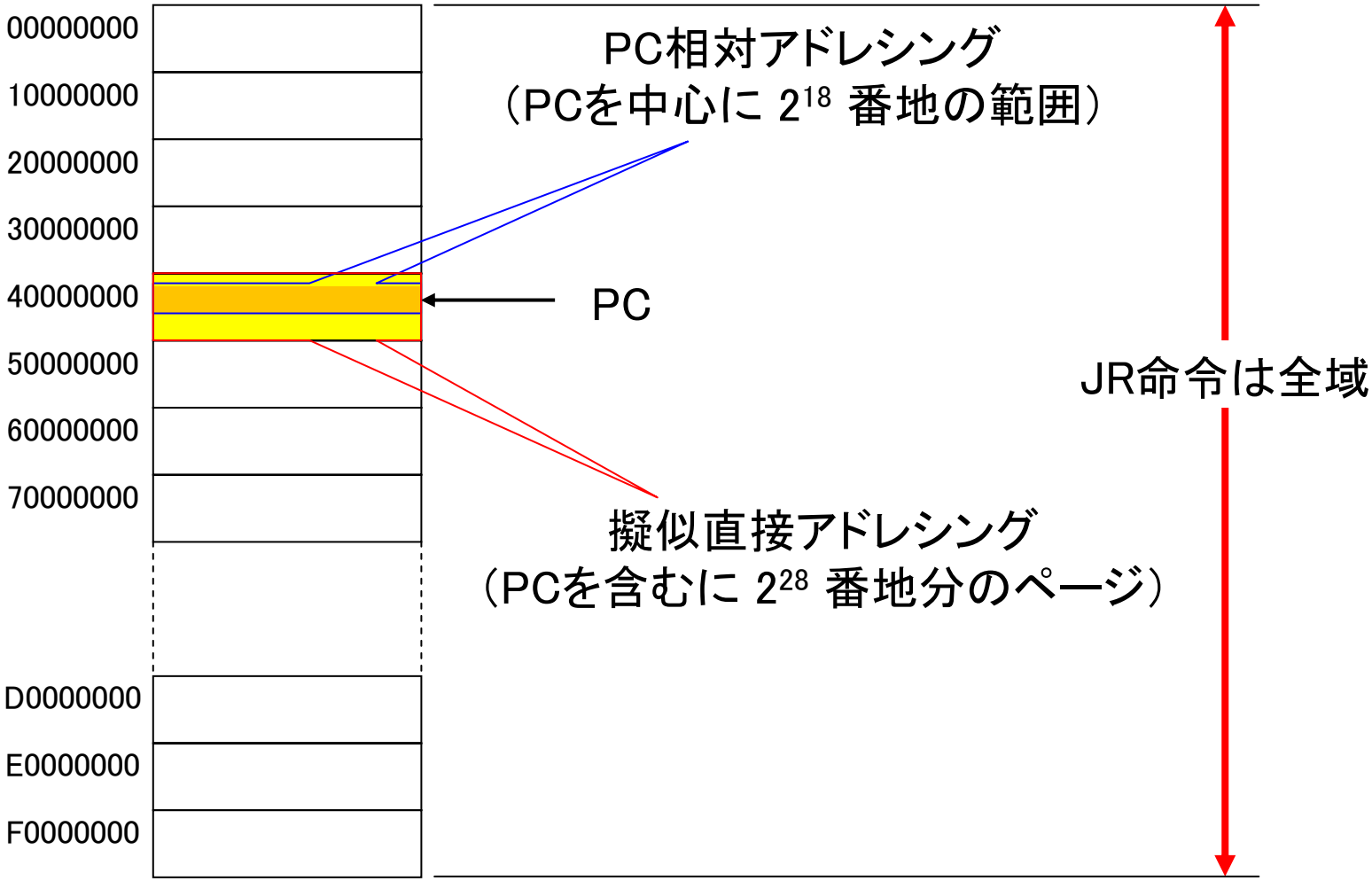
```
lui     $○, n      # $○の上位 16bit に n を格納
ori     $○, m      # $○の下位 16bit に m を格納（符号拡張なし）
```

分岐命令(1)

j	Label	# Label に無条件分岐
jal	Label	# Label に無条件分岐(関数呼出用)
jr	$〇	# $〇の番地に無条件分岐
beq	$△, $□, Label	# $△ = $□ ならば Label に分岐
bne	$△, $□, Label	# $△ ≠ $□ ならば Label に分岐

- 命令によってジャンプできる範囲に違いがある.
  - j, jal(J形式):擬似直接アドレシング
  - jr(R形式):レジスタ値をそのままPCに設定
  - beq, bne(I形式):PC相対アドレシング

分岐命令の守備範囲


00000000
10000000
20000000
30000000
40000000
50000000
60000000
70000000
D0000000
E0000000
F0000000
PC相対アドレシング
（PCを中心に 2^{18} 番地の範囲）
PC
JR命令は全域
擬似直接アドレシング
（PCを含むに 2^{28} 番地分のページ）