計算機構成論 第6回
―命令セットアーキテクチャ(3)―

大連理工大学・立命館大学 国際情報ソフトウェア学部

大森 隆行

講義内容

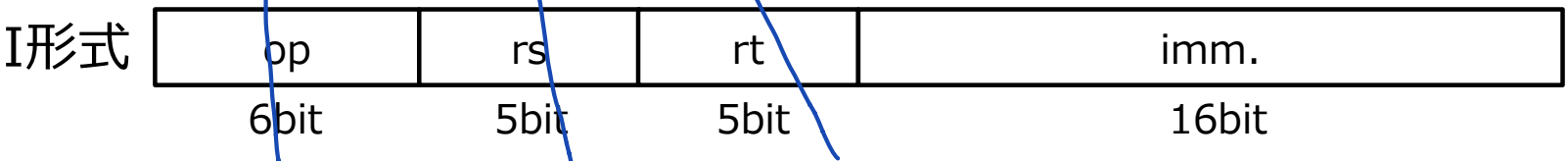
- 命令形式
  - R形式、I形式とは
  - 命令と機械語の対応
  - 配列の使用に対応するアセンブリコード
  - 分岐処理に対応するアセンブリコード
  - 無条件分岐とJ形式
  - 大小比較命令
  - アドレシング・モード

分岐処理

- C言語の条件判定、ループ(if, while, for等)を実現する際に使用

beq $s0, $s1, L1 …$s0と$s1の値が等しいとき、ラベルL1が付いた命令へジャンプ (branch on equal)

bne $s0, $s1, L1 …$s0と$s1の値が異なるとき、ラベルL1が付いた命令へジャンプ (branch on not equal)



beq (rs), (rt), (imm.)

bne (rs), (rt), (imm.)

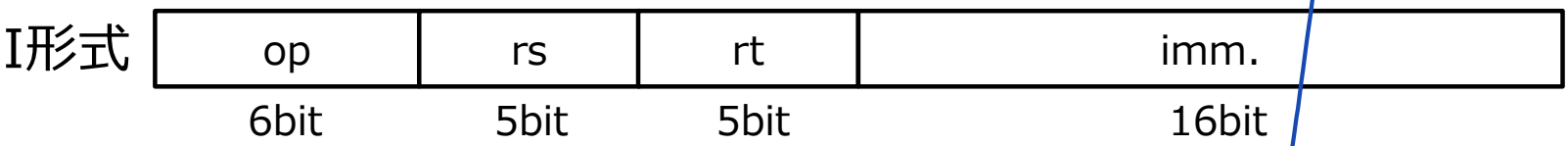
符号付き16bit整数で相対アドレス指定

分岐処理

- beq命令、bne命令は相対アドレス指定

```
1000        beq $s0, $s1, L1
  ⋮          ...
1024   L1:
```

+24
(命令としては6つ先)



- op: 命令操作コード 4
- rs: 第1ソースオペランド 16_{10}
- rt: 第2ソースオペランド 17_{10}
- immediate: 即値オペランド 5

プログラムカウンタを20+4進める

000100 10000 10001 0000000000000101

分岐処理

- プログラムカウンタ
  - PC: program counter 程序计数器
  - 現在実行中の命令が格納されている アドレスを保持するレジスタ
    - 汎用レジスタ32種類の中にはない 不在32种寄存器中在之内
  - MIPSの命令は32ビット(4バイト)なので、 現在のPCが2048なら、次の命令の番地は…2052
  - 通常、1命令実行時に、PC=PC+4という 動作を行っている 指向下一条
  - ただし、J命令が来ると、PCの下位 28ビットを指定された値で置き換える

講義内容

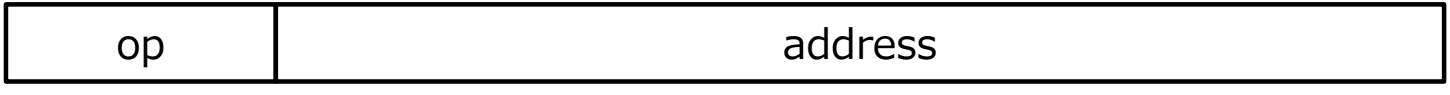
- 命令形式
  - R形式、I形式とは
  - 命令と機械語の対応
  - 配列の使用に対応するアセンブリコード
  - 分岐処理に対応するアセンブリコード
  - 無条件分岐とJ形式
  - 大小比較命令
  - アドレシング・モード

無条件分岐

- 無条件でジャンプ 无条件跳转

  `j L1` …無条件でラベルL1が付いた命令へジャンプ

- J形式

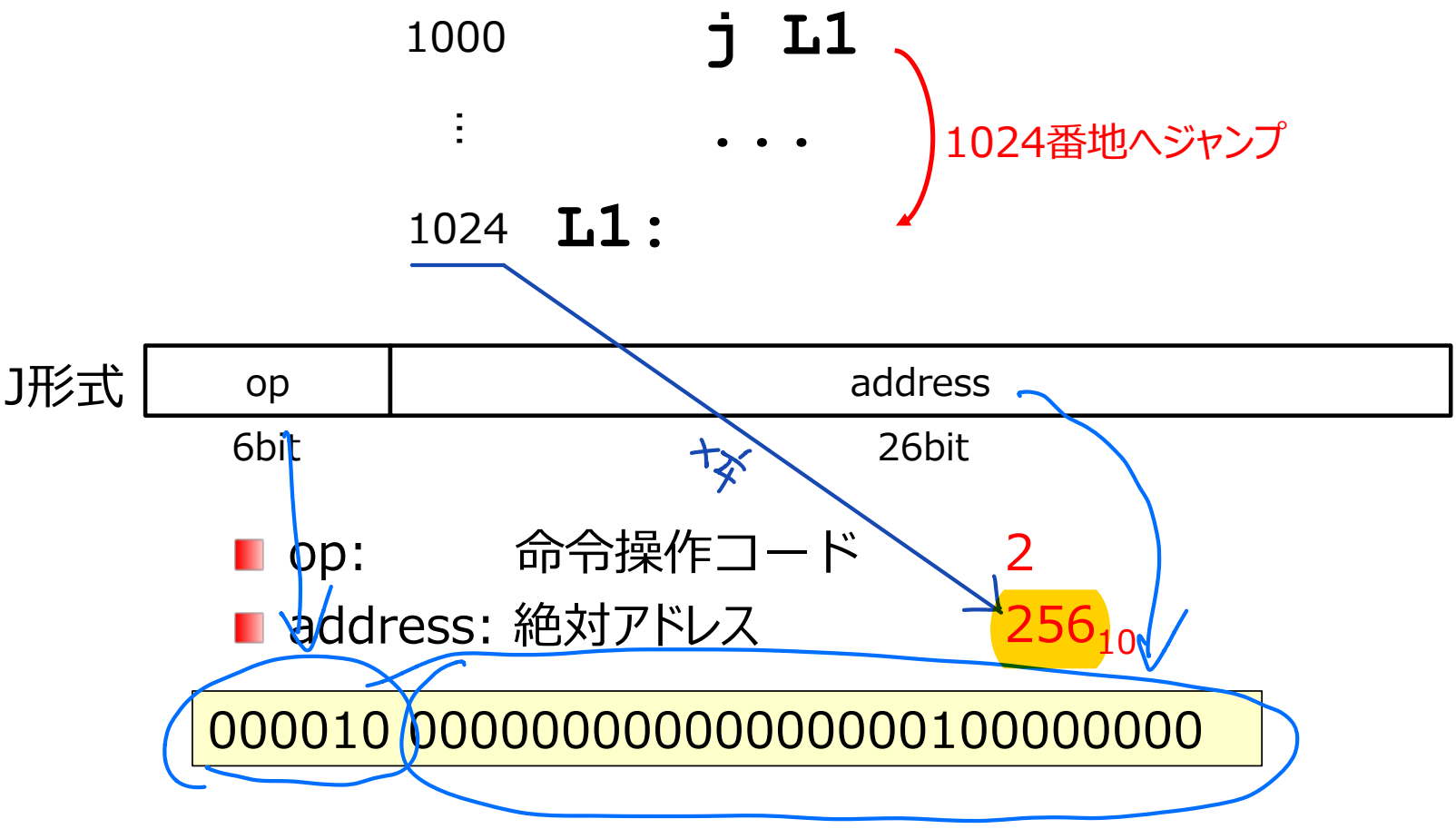


  - op: 命令操作コード
  - address: 絶対アドレス 绝对地址
    ただし、実際のアドレスの1/4の值 (何番目の命令かを示す)

这个地址是字地址 不是字节地址 ∴ ×1/4

实际就相当于第几条指令

無条件分岐


1000 j L1
... ...
1024 L1:
1024番地へジャンプ
J形式
op
address
6bit
26bit
op: 命令操作コード
address: 絶対アドレス
2
256₁₀
000010 00000000000000000100000000

分岐処理に対応するアセンブリコード

(例) `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:
```

f: $s0
g: $s1
h: $s2
i: $s3
j: $s4

確認問題

以下のようにwhile文と、それに対応するアセンブリコードがある。アセンブリコード中の空欄を埋めよ。

(例) `while(i!=a[j]) { i=i+1; }`

```
Loop: (1)   $t0, $s2, 2
      add   $t0, $t0, $s1
      lw    $t1, 0($t0)
      (2)   $s0, $t1, Exit
      (3)   $s0, $s0, (4)
      (5)   (6)
Exit:
```

i: $s0
a: $s1
j: $s2

確認問題

ラベルLoopの付いた命令文のアドレスは768_{10}だと仮定する。これらの命令を2進数で表したとき、赤字で示したExit, Loopに対応する値は何になるか。それぞれ、16ビット、26ビットの2進数で答えよ。

768

```
Loop:  beq   $s0, $s1, Exit
       addi  $s0, $s0, 1
       j     Loop
Exit:
```

i: $s0
j: $s1

0000 0000 0000 0010

768 ÷ 4 = 192

講義内容

- 命令形式
  - R形式、I形式とは
  - 命令と機械語の対応
  - 配列の使用に対応するアセンブリコード
  - 分岐処理に対応するアセンブリコード
  - 無条件分岐とJ形式
  - 大小比較命令
  - アドレシング・モード

大小比較

しょうなり

- < に対応する比較命令

slt $t0, $s0, $s1 …$s0が$s1より小さいとき、$t0を1に設定。そうでなければ0に設定。

若 s0 < s1
即) t0 = 1

slti $t0, $s0, 5 …$s0が5より小さいとき、$t0を1に設定。そうでなければ0に設定。

s0<5
t0就是1
不則 t0是0

slt (rd), (rs), (rt) R形式

slti (rt), (rs), (imm.) I形式

大小比較

(例) `while(i>=0) i=i-1;`

```
Loop: slt  $t0, $s1, $zero
      bne  $t0, $zero, Exit
      addi $s1, $s1, -1
      j    Loop
Exit:
```

0に常に

大小比較

- slt, sltiではなぜ直接分岐しないのか 为什么不直接在slt，slti中分支
  - → 命令が複雑になりすぎるから 因为说明太复杂了
  - → クロックサイクル時間の延長などが必要になる 因为说明太复杂了
- >, >=等に対応する命令は？
  - 擬似命令として用意されている 它被准备为伪指令
    - アセンブラが同じ働きの命令列に変換 它被准备为伪指令
  - blt (branch less than), bgt (- greater than), ble (branch less or equal), bge (- greater -)

講義内容

- 命令形式
  - R形式、I形式とは
  - 命令と機械語の対応
  - 配列の使用に対応するアセンブリコード
  - 分岐処理に対応するアセンブリコード
  - 無条件分岐とJ形式
  - 大小比較命令
  - アドレシング・モード

アドレシング・モード

寻址模式

不同指令，操作数的解释不同

操作数的解释因指令而异

- 命令によってオペランドの解釈のしかたが異なる
  →命令が操作の対象とするもの、
  (e.g., add命令の場合の加算する値)
  命令が必要とするアドレス
  (e.g., j命令のジャンプ先アドレス)
  が命令によって異なる

  (例如，在添加指令的情况下要添加的值)

  (例如j指令的跳转目标地址)

  寻址模式

  →指令的操作目标，指令所需的地址根据指令而有所不同
- それらの解釈の方法をアドレシング・モードと呼ぶ

  我们将这些解释方法称为寻址模式
- アドレシング・モードの種類

  寻址模式的类型
  - レジスタ・アドレシング (register addressing)

    寄存器寻址（寄存器寻址）
  - ベース相対アドレシング (base addressing)

    基址相对寻址（基址寻址）
  - 即値アドレシング (immediate addressing)

    立即寻址（立即寻址）
  - PC相対アドレシング (PC-relative addressing)

    PC相对寻址（PC相对寻址）
  - 擬似直接アドレシング (pseudo direct addressing)

    伪直接寻址（伪直接寻址）

レジスタ・アドレシング 寄存器寻址模式

- 指定したレジスタの中身をオペランドにする

即 操作数是寄存器

```
add $t0, $s0, $s1
```

op	rs	rt	rd	shamt	funct
000000	10000	10001	01000	00000	100000
add	$s0	$s1	$t0		add

ベース相対アドレシング

基址寻址模式

- 指定したレジスタの中身＋定数
  のメモリアドレスの内容をオペランドにする

```
lw $t0, 12($s0)
```

基址寄存器上加一个偏移量 所对地址
为操作数

op	rs	rt	immediate
100011	10000	01000	0000000000001100
lw	$s0	$t0	+12

即値アドレシング

即値数字址模式

- 指定した定数をオペランドにする

```
andi $s1, $s0, 24
```

$s0 和24 兩個取and 所得結果到$1

op	rs	rt	immediate
001100	10000	10001	0000000000011000
andi	$s0	$s1	+24

PC相対アドレシング

- (PC+4)+(指定した定数×4) のアドレスを示す

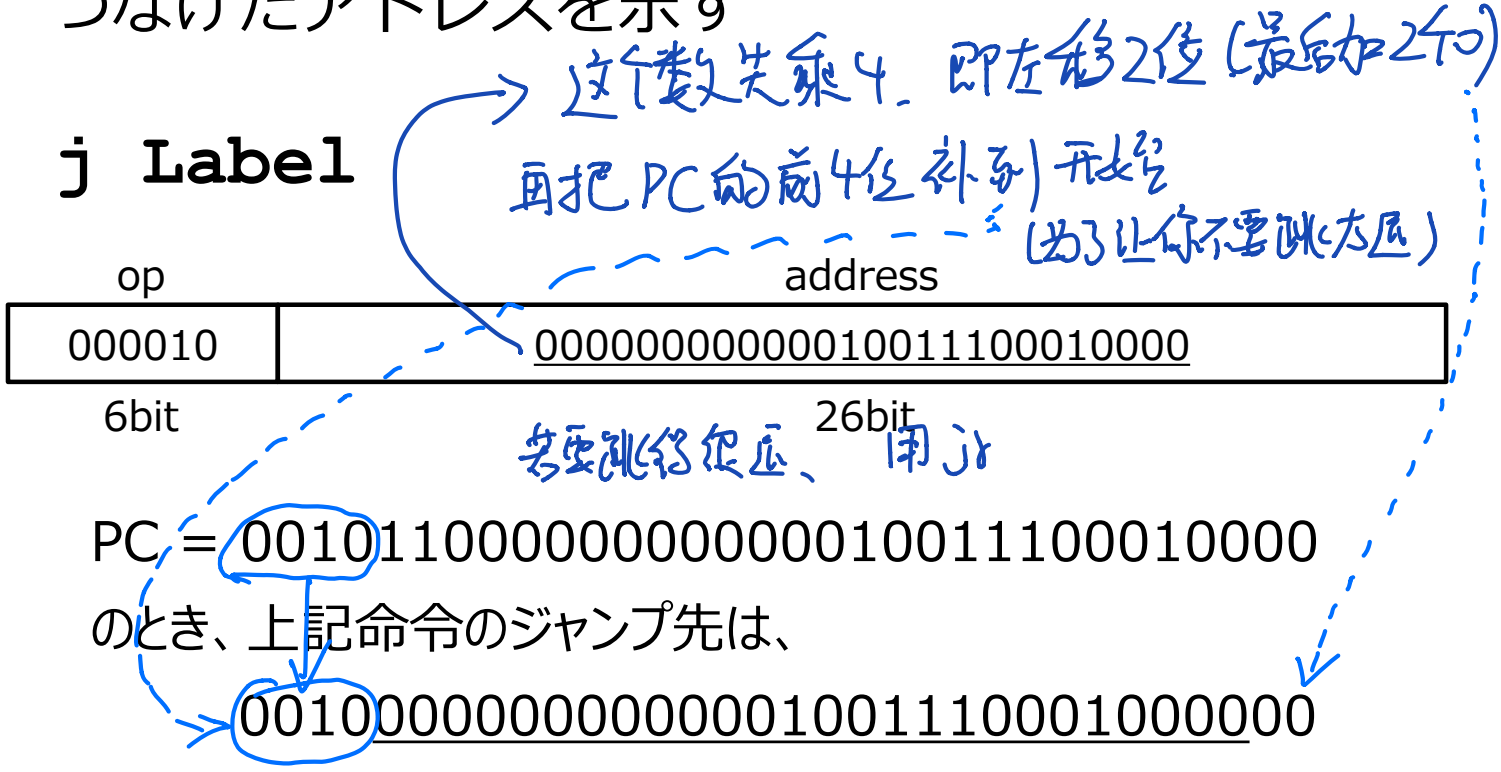
```
bne $t0, $s0, Label
```

op	rs	rt	address
000101	010000	10000	0000000000001100
bne	$t0	$s0	+12

擬似直接アドレシング

- PCの上位4ビット と 指定した定数×4 を つなげたアドレスを示す

j Label



32ビットの即値オペランド

- 命令長は32ビット
  →32ビットのオペランドは扱えないが…

0000 0000 0011 1101 0000 1001 0000 0000

上位16ビット = 61_{10}　　下位16ビット = 2304_{10}

- 上位、下位に分けて、2命令で読み込む

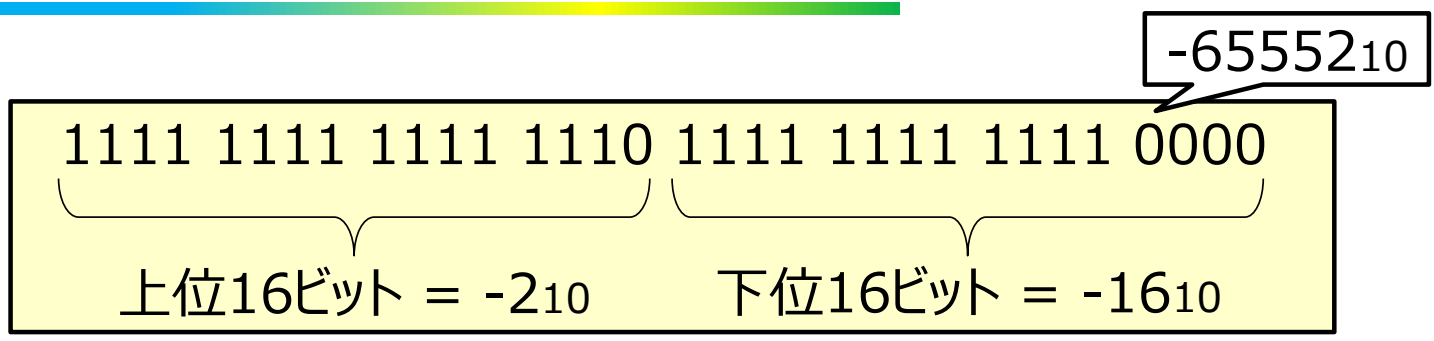
lui $s0, 61 …$s0の上位16ビットに定数を読み込む (load upper immediate)

0000 0000 0011 1101 0000 0000 0000 0000

ori $s0, $s0, 2304 …or演算 (or immediate)

0000 0000 0011 1101 0000 1001 0000 0000

32ビットの即値オペランド




```
lui $s0, -2
```

…$s0の上位16ビットに定数を読み込む (load upper immediate)

1111 1111 1111 1110 0000 0000 0000 0000

```
ori $s0, $s0, -16
```

…or演算 (or immediate)

1111 1111 1111 1110 1111 1111 1111 0000

確認問題

- それぞれの文はどのアドレシングモードを説明したものか答えよ。
  - (1) 指定したレジスタの中身＋定数のメモリアドレスの内容をオペランドにする
  - (2) (PC+4)+(指定した定数×4) のアドレスを示す
  - (3) 指定したレジスタの中身をオペランドにする
  - (4) PCの上位4ビット と 指定した定数×4 をつなげたアドレスを示す
  - (5) 指定した定数をオペランドにする

参考文献

- コンピュータの構成と設計 上 第5版
  David A.Patterson, John L. Hennessy 著、
  成田光彰 訳、日経BP社
- 山下茂 「計算機構成論１」講義資料