第2章 命令:
コンピュータの言葉(1)

大阪大学 大学院 情報科学研究科
今井 正治

E-mail: arch-2014@vlsilab.ics.es.osaka-u.ac.jp

講義内容(1)

- コンピュータ・ハードウェアの演算
- コンピュータ・ハードウェアのオペランド
- 符号付き数と符号なし数
- コンピュータ内での命令の表現
- 論理演算
- 条件判定用の命令
- コンピュータ・ハードウェア内での手続きのサポート

講義内容(2)

- 人との情報交換
- 32ビットの即値およびアドレスに対するMIPSのアドレッシング方式
- 並列処理と命令:同期
- プログラムの翻訳と起動
- Cプログラムの包括的な例題解説
- 配列とポインタの対比
- 誤信と落とし穴

MIPS

- MIPS Technologies 社が1980年代に開発したRISC方式のマイクロプロセッサ
- MIPS = Microprocessor without Interlocked Pipeline Stages(パイプライン・ステージがインターロックされないマイクロプロセッサ)
- 32ビットの汎用レジスタ32個から構成されるレジスタファイル

アセンブリ言語での表現

- 加算命令
  - `add a, b, c`
  - 2つの変数 b と cを加えて，その和を a に格納する
- MIPSの演算命令
  - 各演算命令で実行できる演算は1つだけ
  - 変数を3つ指定しなければならない
- 設計原則1：単純性は規則性につながる

a, b, c はオペランド（operand）

アセンブリ言語での表現

- C言語の文

  a = b + c + d + e;

  に対応するアセンブリ言語の命令列

```
add a, b, c     # b と c の和を a に格納
add a, a, d     # 結果として，b, c, d の和が
                # a に格納される
add a, a, e     # 結果として，b, c, d, e の和が
                # a に格納される
```


MIPSのオペランド

- レジスタ
  - `$s0 - $s7, $t0 - $t7, $zero`
    `$a0 - $a3, $v0 - $v1`
    `$gp, $fp, $sp, $ra, $at`
- メモリ
  - データ転送命令のみによってアクセスされる
  - バイト・アドレス方式（アドレスはバイト単位）
  - 1語（word）は32ビット（4バイト）
  - メモリの大きさは2^{30}ワード

MIPSのアセンブリ言語（1）

区分	命令	例	意味	備考
算術演算	add	`add $s1, $s2, $s3`	$s1 = $s2 + $s3	3オペランド，データはレジスタ中
	subtract	`sub $s1, $s2, $s3`	$s1 = $s2 - $s3	3オペランド，データはレジスタ中
	add immediate	`addi $s1, $s2, 20`	$s1 = $s2 + 20	定数を加算
データ転送	load word	`lw $s1, 20($s2)`	$s1 = mem[$s2+20]	メモリからレジスタへ転送
	store word	`sw $s1, 20($s2)`	mem[$s2+20] = $s1	レジスタからメモリへ転送
	load half	`lh $s1, 20($s2)`	$s1 = mem[$s2+20]	メモリからレジスタへ半語を転送
	load half unsigned	`lhu $s1, 20($s2)`	$s1 = mem[$s2+20]	メモリからレジスタへ半語を転送
	store half	`sh $s1, 20($s2)`	mem[$s2+20] = $s1	レジスタからメモリへ半語を転送
	load byte	`lb $s1, 20($s2)`	$s1 = mem[$s2+20]	メモリからレジスタへバイトを転送
	load byte unsigned	`lbu $s1, 20($s2)`	$s1 = mem[$s2+20]	メモリからレジスタへバイトを転送
	store byte	`sb $s1, 20($s2)`	mem[$s2+20] = $s1	レジスタからメモリへバイトを転送
	load linked word	`ll $s1, 20($s2)`	$s1 = mem[$s2+20]	不可分なスワップの前半
	store condition word	`sc $s1, 20($s2)`	Mem[$s2+20]=$s1	不可分なスワップの後半
	load upper immediate	`Lui $s1, 20`	$s1 = 20*$2^{16}$	定数を上位16ビットにロード

MIPSのアセンブリ言語(2)

区分	命令	例	意味	備考
論理演算	and	`and $s1, $s2, $s3`	$s1 = $s2 & $s3	3オペランド, ビット単位のAND
	or	`or $s1, $s2, $s3`	$s1 = $s2 | $s3	3オペランド, ビット単位のOR
	nor	`nor $s1, $s2, 20`	$s1 = ~($s2 | $s3)	3オペランド, ビット単位のNOR
	and immediate	`andi $s1, $s2, 20`	$s1 = $s2 & 20	レジスタと定数のビット単位AND
	or immediate	`or $s1, $s2, $s3`	$s1 = $s2 | 20	レジスタと定数のビット単位OR
	shift left logical	`sll $s1, $s2, 10`	$s1 = $s2 << 10	定数分左へシフト
	shift right logical	`srl $s1, $s2, 10`	$s1 = $s2 >> 10	定数分右へシフト
条件分岐	branch on equal	`beq $s1, $s2, 25`	if($s1 = $s2) goto PC+4+100	等しいときにPC相対分岐
	branch on not equal	`bne $s1, $s2, 25`	if($s1 != $s2) goto PC+4+100	等しくないときにPC相対分岐
	set on less than	`slt $s1, $s2, $s3`	if($s1 < $s2) $s1 = 1; else $s1 = 0	より小さいかの判定, be, bne用
	set on less than unsigned	`sltu $s1, $s2, $s3`	if($s1 < $s2) $s1 = 1; else $s1 = 0	より小さいかの判定, 符号なし整数用

MIPSのアセンブリ言語(3)

区分	命令	例	意味	備考
条件分岐	set on less than immediate	`slt $s1, $s2, 20`	if($s1 < 20) $s1 = 1; else $s1 = 0	定数より小さいかの判定, 2の補数
	set on less than immediate unsigned	`sltu $s1, $s2, 20`	if($s1 < 20) $s1 = 1; else $s1 = 0	定数より小さいかの判定, 符号なし整数用
無条件ジャンプ	jump	`j 2500`	go to 10000	目的のアドレスへジャンプ
	jump register	`jr $ra`	go to $ra	switch文や手続き呼び出しからの戻りに利用
	jump and link	`jal 2500`	$ra = PC+4; go to 10000	手続き呼び出し用

単純な代入文のコンパイル

- C言語の文

```
a = b + c;
d = a – e;
```

- MIPSのアセンブリ言語の文

```
add a, b, c
sub d, a, e
```


複雑な代入文のコンパイル

- C言語の文

```
f = (g + h) – (i + j);
```

- MIPSのアセンブリ言語の文

```
add t0, g, h
add t1, i, j
sub f, t0, t1

# t0, t1 はコンパイラによって生成される一時変数
```

講義内容(1)

- コンピュータ・ハードウェアの演算
- コンピュータ・ハードウェアのオペランド
- 符号付き数と符号なし数
- コンピュータ内での命令の表現
- 論理演算

レジスタ

- MIPSでは，算術命令のオペランドは，ハードウェアに直接組み込まれている特殊な記憶領域の中から選んで使用する
- この記憶領域はレジスタ(register)と呼ばれる
- MIPSのレジスタは32個
- 各レジスタは32ビット = 4バイト = 1語(word)
- 設計原則2: 小さければ小さいほど高速になる

レジスタを使用した代入文のコンパイル

- C言語の文

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

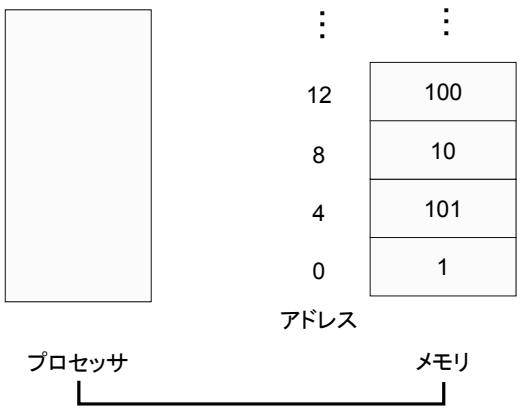
  - f, g, h, i, j をレジスタ $s0, $s1, $s2, $s3, $s4 に割り当てる
- コンパイル結果

```
add $t0, $s1, $s2
add $t1, $s3, $s4
sub $s0, $t0, $t1
```


データ転送命令

- メモリとレジスタ間でのデータ転送(data transfer)を行う命令
- ロード(load)命令
  - メモリからレジスタへのデータ転送
  - 転送単位: ワード, ハーフワード, バイト
- ストア(store)命令
  - レジスタからメモリへのデータ転送
  - 転送単位: ワード, ハーフワード, バイト

メモリア・ドレスとその内容



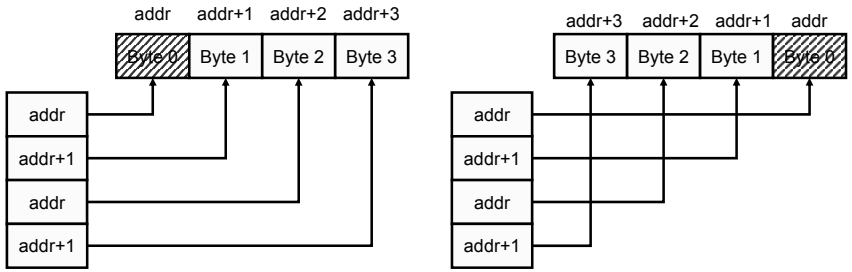
MIPSのメモリ構成

- バイト・アドレッシング
- 1ワード = 4バイト
- ビッグ・エンディアン（big endian）
- ワードアドレスは4の倍数に制約（alignment restriction）

アドレス	データ			
28	B28	B29	B30	B31
24	B24	B25	B26	B27
20	B20	B21	B22	B23
16	B16	B17	B18	B19
12	B12	B13	B14	B15
8	B8	B9	B10	B11
4	B4	B5	B6	B7
0	B0	B1	B2	B3

エンディアン（endianness）

- ビッグ・エンディアン（big endian）
  - 語のアドレスは左端のバイトのアドレス
- リトル・エンディアン（little endian）
  - 語のアドレスは右端のバイトのアドレス



オペランドがメモリ中にあるときの代入文のコンパイル

- C言語の文

```
g = h + A[8];
```

- コンパイル結果

```
lw  $t0, 32($s3)
add $s1, $s2, $t0
```

  - データ転送命令中の定数：オフセット(offset)
  - アドレスを得るために加えるレジスタ：ベース・レジスタ(base register)

ロードとストアが使用されているコードのコンパイル

- C言語の文

```
A[12] = h + A[8];
```

- コンパイル結果

```
lw  $t0, 32($s3)
add $t0, $s2, $t0
sw  $t0, 48($s3)
```


定数または即値のオペラント

- レジスタに定数を加算
  - 定数をメモリから得る方法

```
lw  $t0, AddrConstant4($s1)
add $s3, $s3, $t0
```

  - 即値加算命令を使用する方法

```
addi $s3, $s3, 4
```

- 設計原則3: 一般的な場合を高速化する

講義内容(1)

- コンピュータ・ハードウェアの演算
- コンピュータ・ハードウェアのオペランド
- 符号付き数と符号なし数
- コンピュータ内での命令の表現
- 論理演算

n 進法

- 2進法: 2種類の記号を使用
  - '0' と '1' で表現 (bit)
- 一般に N 進法では、N 種類の記号を用いて情報を表現
  - 2進法: '0', '1'
  - 8進法: '0', '1', '2', '3', '4', '5', '6', '7'
  - 10進法: '0', '1', '2', '3', '4', '5', '6', '7', '8', '9'
  - 16進法: '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E', 'F'

2進数

- N 桁の2進数: $X = (x_{N-1}, x_{N-2}, \ldots, x_1, x_0)$
  - x_{N-1}: MSB（most significant bit）
  - x_0: LSB（least significant bit）
- 2^N 種類の情報を表現可能
- 対応する10進数
  - 符号なし数の場合
    $0 \sim 2^N - 1$

(0, 0, …, 0, 0)
(0, 0, …, 0, 1)
(0, 0, …, 1, 0)
:
(1, 1, …, 1, 1)

2進化10進数
（BCD: Binary Coded Decimal）

- 4桁の2進数で1桁の10進数を表現

x_3	x_2	x_1	x_0	X
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7

x_3	x_2	x_1	x_0	X
1	0	0	0	8
1	0	0	1	9
1	0	1	0	--
1	0	1	1	--
1	1	0	0	--
1	1	0	1	--
1	1	1	0	--
1	1	1	1	--

16進数（Hexadecimal）

- 4桁の2進数で1桁の16進数を表現

a_3	a_2	a_3	a_3	N
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7

a_3	a_2	a_3	a_3	N
1	0	0	0	8
1	0	0	1	9
1	0	1	0	A (a)
1	0	1	1	B (b)
1	1	0	0	C (c)
1	1	0	1	D (d)
1	1	1	0	E (e)
1	1	1	1	F (f)

ビット，ニブル，バイト

- ビット（bit）
  - 1桁の2進数
  - 0，1
- ニブル（nibble）
  - 4桁の2進数
    1桁の16進数
  - 表現方法
    $(0000)_2 \sim (1111)_2$
    $(0)_{16} \sim (F)_{16}$
    $X'0 \sim X'F$
- バイト（byte）
  - 8桁の2進数
    （2桁の16進数）
  - 表現方法
    $(00000000)_2 \sim (11111111)_2$
    $(00)_{16} \sim (FF)_{16}$
    $X'00 \sim X'FF$

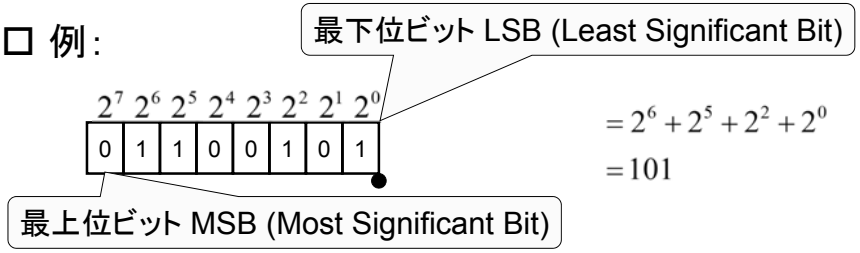
正整数の2進表現

□ 対応関係 $X = (x_{n-1}, x_{n-2}, \Lambda\ , x_1, x_0)$

χ

$$X = \sum_{i=0}^{n-1} x_i \times 2^i = x_{n-1} \times 2^{n-1} + \Lambda\ + x_1 \times 2 + x_0$$

□ 例:



$$= 2^6 + 2^5 + 2^2 + 2^0$$
$$= 101$$

小数の2進表現(1)

□ 対応関係 $X = (x_{-1}, x_{-2}, \Lambda\ , x_{-n+1}, x_{-n})$

χ

$$X = \sum_{i=1}^{n} x_{-i} \times 2^{-i} = x_{-1} \times 2^{-1} + x_{-2} \times 2^{-2} + \Lambda\ + x_{-n} \times 2^{-n}$$

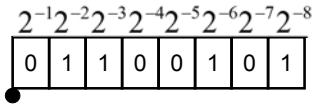
□

$$2^{-1} = \frac{1}{2} = 0.5, \quad 2^{-2} = \frac{1}{4} = 0.25$$

$$2^{-3} = \frac{1}{8} = 0.125, \quad 2^{-4} = \frac{1}{16} = 0.0625$$

小数の2進表現(2)

□ 例: $X = (0, 1, 1, 0, 0, 1, 0, 1)$



$$X = \frac{1}{4} + \frac{1}{8} + \frac{1}{64} + \frac{1}{256}$$
$$= 0.39453125$$

負数の表現方法

□ 必要な(望ましい)性質
- 正負の判定が容易
- 一意に表現可能
- 補数演算が容易 ($X \Rightarrow -X$)
- 加算が容易

□ 表現方法
- ゲタばき表現 (2^n余り表現)
- 符号付き絶対値表現
- 1の補数表現
- 2の補数表現

負の整数の表現方法

数値	ゲタばき表現	符号＋絶対値	1の補数	2の補数
7	1111	0111	0111	0111
6	1110	0110	0110	0110
5	1101	0101	0101	0101
4	1100	0100	0100	0100
3	1011	0011	0011	0011
2	1010	0010	0010	0010
1	1001	0001	0001	0001
0	1000	0000	0000	0000
-0	----	1000	1111	----
-1	0111	1001	1110	1111
-2	0110	1010	1101	1110
-3	0101	1011	1100	1101
-4	0100	1100	1011	1100
-5	0011	1101	1010	1011
-6	0010	1110	1001	1010
-7	0001	1111	1000	1001
-8	0000	----	----	1000

ゲタばき(biased)表現

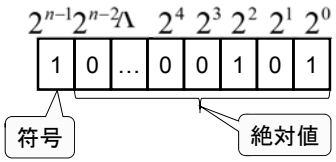
- 表現: 数値＋ 2^{n-1}
- 表現範囲: $-2^{n-1} \sim 2^{n-1}-1$
- 特徴
  - 非負(正または0)の場合にMSB = 1
  - 0の表現は一意(- 0はない)
  - 補数演算は，1の補数＋1で実現可能
  - 加減算の結果は補正が必要

符号付き絶対値 (sign and magnitude)表現

- 表現: 符号と数値の絶対値で表現
- 表現範囲:

$$-2^{n-1}-1 \sim 2^{n-1}-1$$

- 特徴:
  - 負数の場合、MSB = 1
  - 0 の表現が2通り
  - 補数演算は容易 (符号の反転)
  - 加減算が複雑



1の補数(one's complement)表現

- 表現: 正数はそのまま，負数を1の補数で表現

- 表現範囲: $-2^{n-1}-1 \sim 2^{n-1}-1$
- 特徴
  - 負数の場合、MSB = 1
  - 0 の表現は2通りある。- 0 と + 0
  - 補数演算は容易(0と1の反転)
  - 負の最小値の絶対値 = 正の最大値の絶対値
  - 加減算の結果の補正が必要

1の補数（1's Complement）

- 定義：各桁の値（x_i）をその補数（$\overline{x_i}$）で置き換えて得られる値
  - $X = (x_{n-1}, \ldots, x_2, x_1, x_0)$ の1の補数を $Y = (y_{n-1}, \ldots, y_2, y_1, y_0)$ とすると，
    $y_i = \overline{x_i} = 1 - x_i, 0 \leq {}^{\forall}i \leq n-1$
  - 例：

	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
$A =$	0	1	1	0	0	1	0	1
$B =$	1	0	0	1	1	0	1	0

2の補数（two's complement）表現

- 表現：正数はそのまま，負数を2の補数で表現

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0		2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
0	1	1	0	0	1	0	1	⟷	1	0	0	1	1	0	1	1

- 表現範囲：$-2^{n-1} \sim 2^{n-1} - 1$
- 特徴
  - 負数の場合、MSB = 1
  - 0 の表現は1通りだけ
  - 補数演算は「1の補数＋1」演算が必要
  - 加減算の結果は補正が不要
  - 負の最小値の絶対値 ≠ 正の最大値の絶対値

2の補数（2's Complement）

- n 桁の2進数 A に対し，次式を満たす n 桁の2進数 B を A の2の補数と定義する．

$$A + B = 2^n$$

- 例：01100101 の2の補数は 10011011

$$\begin{array}{r} 01100101 \\ +\ 10011011 \\ \hline 100000000 \end{array}$$

- 1の補数と2の補数の関係：
  2の補数は1の補数+1

2の補数の計算方法

- 2の補数 = 1の補数 + 1
- 例

元の数：	01100101
	⬇ 0と1を反転
1の補数：	10011010
+1：	1
2の補数：	10011011
	⬇ 0と1を反転
1の補数：	01100100
+1：	1
元の数：	01100101

2の補数表現での注意点

- 正の最大値の絶対値≠負の最大値の絶対値
- 負の最大値の2の補数は負の最大値になる

X(2進表現)	X(10進表現)	Xの2の補数（2進表現）	Xの2の補数（10進表現）
0 1 1 1 1 1 1 1	127	1 0 0 0 0 0 0 1	-127
0 1 1 1 1 1 1 0	126	1 0 0 0 0 0 1 0	-126
0 0 0 0 0 0 0 1	1	1 1 1 1 1 1 1 1	-1
0 0 0 0 0 0 0 0	0	0 0 0 0 0 0 0 0	0
1 1 1 1 1 1 1 1	-1	0 0 0 0 0 0 0 1	1
1 0 0 0 0 0 0 1	-127	0 1 1 1 1 1 1 1	127
1 0 0 0 0 0 0 0	-128	1 0 0 0 0 0 0 0	-128

表現化方法の比較

要請＼表現方法	ゲタばき表現	符号＋絶対値	1の補数	2の補数
正負の判定が容易	○	○	○	○
一意に表現可能	○	+0と-0が存在	+0と-0が存在	○
補数演算が容易	加算が必要	○	○	加算が必要
加算が容易	結果の補正が必要	場合分けが必要	結果の補正が必要	○

符号拡張（sign extention）

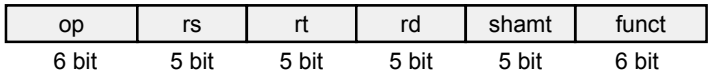
- n ビットの2進数をそれより大きいビット数の2進数に変換する方法
  - 符号付き数の符号はMSB
  - 変換される数の上位のビットを符号ビットで置き換える
- 例
  - 0000 0000 0000 0010
    ⇒ 0000 0000 0000 0000 0000 0000 0000 0010
  - 1111 1111 1111 1110
    ⇒ 1111 1111 1111 1111 1111 1111 1111 1110

講義内容（1）

- コンピュータ・ハードウェアの演算
- コンピュータ・ハードウェアのオペランド
- 符号付き数と符号なし数
- コンピュータ内での命令の表現
- 論理演算

MIPSプロセッサでの命令の表現

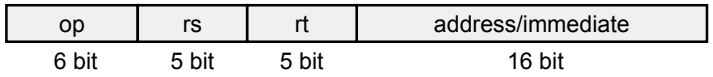
- 命令長: 32ビット固定
- 命令フォーマット(instruction format)
  - R フォーマット



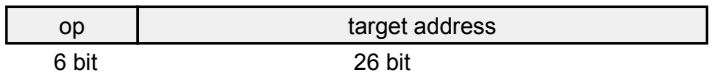
  - op: 命令操作コード(operation code)
  - rs: 第1ソース・オペランドのレジスタ(source)
  - rt: 第2ソース・オペランドのレジスタ(target)
  - rd: デスティネーション(destination)レジスタ
  - shamt: シフト量(shift amount)
  - funct: 機能コード(function code)

MIPSプロセッサでの命令の表現(2)

  - I フォーマット



    - address: アドレス(データ転送, 分岐で使用)
    - immediate: 即値(即値命令で使用)
  - J フォーマット



    - target address: ジャンプ命令で使用
- 設計原則4: 優れた設計には適度な妥協が必要

MIPSのレジスタ構成(1)

略号	番号	用途
zero	0	定数 0(値は常に0)
at	1	アセンブラ用に予約
v0	2	式の評価と関数の実行結果
v1	3	式の評価と関数の実行結果
a0	4	引数1
a1	5	引数2
a2	6	引数3
a3	7	引数4

MIPSのレジスタ構成(2)

略号	番号	用途
t0	8	一時変数(呼び出し側で保存不要)
t1	9	一時変数(呼び出し側で保存不要)
t2	10	一時変数(呼び出し側で保存不要)
t3	11	一時変数(呼び出し側で保存不要)
t4	12	一時変数(呼び出し側で保存不要)
t5	13	一時変数(呼び出し側で保存不要)
t6	14	一時変数(呼び出し側で保存不要)
t7	15	一時変数(呼び出し側で保存不要)

MIPSのレジスタ構成(3)

略号	番号	用途
s0	16	一時変数(呼び出し側で保存必要)
s1	17	一時変数(呼び出し側で保存必要)
s2	18	一時変数(呼び出し側で保存必要)
s3	19	一時変数(呼び出し側で保存必要)
s4	20	一時変数(呼び出し側で保存必要)
s5	21	一時変数(呼び出し側で保存必要)
s6	22	一時変数(呼び出し側で保存必要)
s7	23	一時変数(呼び出し側で保存必要)

MIPSのレジスタ構成(4)

略号	番号	用途
t8	24	一時変数(呼び出し側で保存不要)
t9	25	一時変数(呼び出し側で保存不要)
k0	26	OSカーネル用に予約
k1	27	OSカーネル用に予約
gp	28	グローバル領域へのポインタ
sp	29	スタックポインタ
fp	30	フレームポインタ
ra	31	リターンアドレス(関数呼出しに使用)

MIPS命令の符号化

命令	形式	op	rs	rt	rd	shamt	funct	addr
add	R	0	レジスタ	レジスタ	レジスタ	0	32	使用せず
sub	R	0	レジスタ	レジスタ	レジスタ	0	34	使用せず
addi	I	8	レジスタ	レジスタ	使用せず	使用せず	使用せず	定数
lw	I	35	レジスタ	レジスタ	使用せず	使用せず	使用せず	アドレス
sw	I	43	レジスタ	レジスタ	使用せず	使用せず	使用せず	アドレス

アセンブリ言語の機械語への翻訳

- アセンブリ言語のコード

```
lw  $t0, 8($s3)
add $t0, $s2, $t0
sw  $t0, 48($s3)
```

- 対応する機械語コード

アセンブリ・コード	op	rs	rt	address		
				rd	shamt	funct
lw $t0, 8($s3)	35	19	8		8	
add $t0, $s2, $t0	0	18	8	8	0	32
sw $t0, 48($s3)	43	19	8		48	

機械語コード

- 10進表現

アセンブリ・コード	op	rs	rt	address		
				rd	shamt	funct
`lw $t0, 8($s3)`	35	19	8	8		
`add $t0, $s2, $t0`	0	18	8	8	0	32
`sw $t0, 48($s3)`	43	19	8	48		

- 2進表現

アセンブリ・コード	op	rs	rt	address		
				rd	shamt	funct
`lw $t0, 8($s3)`	100011	10011	01000	0000 0000 0000 1000		
`add $t0, $s2, $t0`	000000	10010	01000	01000	00000	100000
`sw $t0, 48($s3)`	101011	11011	01000	0000 0000 0011 0000		

MIPSの機械語の例(1)

名前	命令形式	例						備考
`lw`	I	35	18	17	100			lw $s1, 100($s2)
`sw`	I	34	18	17	100			sw $s1, 100($s2)
`add`	R	0	18	19	17	0	32	add $s1, $s2, $s3
`sub`	R	0	18	19	17	0	34	sub $s1, $s2, $s3
`addi`	I	8	18	17	100			addi $s1, $s2, 100
`sll`	R	0	0	16	10	4	0	sll $t2, $s0, 4
`srl`	R	0	0	16	10	4	2	srl $t2, $s0, 4

MIPSの機械語の例(2)

名前	命令形式	例						備考
`beq`	I	4	17	18	25			beq $s1, $s2, 100
`bne`	I	5	17	18	25			bne $s1, $s2, 100
`slt`	R	0	18	19	17	0	42	slt $s1, $s2, $s3
`j`	J	2			2500			j 10000
`jr`	R	0	31	0	0	0	8	jr $ra
`jal`	J	3			2500			jal 10000

講義内容(1)

- コンピュータ・ハードウェアの演算
- コンピュータ・ハードウェアのオペランド
- 符号付き数と符号なし数
- コンピュータ内での命令の表現
- 論理演算

CおよびMIPSにおける論理演算の表現

論理演算	Cの演算子	Javaの演算子	MIPSの命令
左シフト	<<	<<	sll
右シフト	>>	>>>	srl
ビット単位のAND	&	&	and, andi
ビット単位のOR	|	|	or, ori
ビット単位のNOT	~	~	nor

NOR命令を用いた論理否定の実現

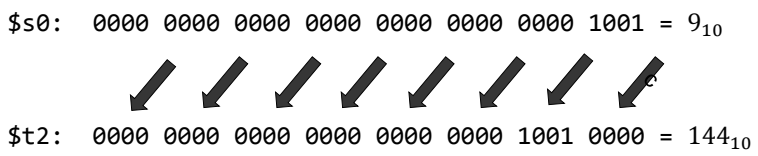
- NOR命令の一方のオペランドを0にする，もしくは同じ2つのオペランドに同じレジスタを指定することで，NOT演算が実現できる
- オペランドを0にするためには，$zero（ゼロレジスタ）を指定すれば良い

論理左シフト命令 sll

- sll: shift left logical

```
Sll  $t2, $s0, 4  # $t2 = $s0 << 4
```

op	rs	rt	rd	shamt	funct
0	0	16	10	4	0

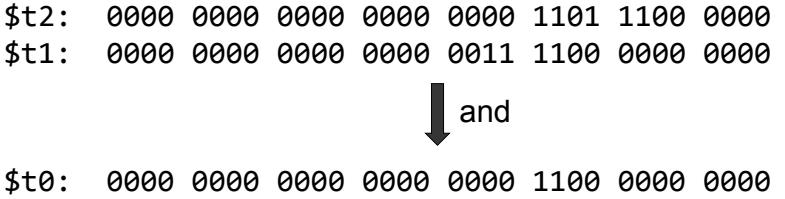


論理積 and

- and: logical and

```
and  $t2, $t1, $t0  # $t2 = $t1 & $t0
```

op	rs	rt	rd	shamt	funct
0	10	9	8	0	36



論理和 or

- or: logical or

```
or  $t2, $t1, $t0  # $t2 = $t1 | $t0
```

op	rs	rt	rd	shamt	funct
0	10	9	8	0	37

```
$t2:  0000 0000 0000 0000 0000 1101 1100 0000
$t1:  0000 0000 0000 0000 0011 1100 0000 0000
```


```
$t0:  0000 0000 0000 0000 0011 1101 1100 0000
```


nor命令を用いた論理否定の実現

- nor: logical nor (not or)

```
nor  $t2, $t1, $t0  # $t2 = ~($t1 | $t0)
```

op	rs	rt	rd	shamt	funct
0	10	9	8	0	39

```
$t2:  0000 0000 0000 0000 0000 1101 1100 0000
$t1:  0000 0000 0000 0000 0000 0000 0000 0000
```


```
$t0:  1111 1111 1111 1111 1111 0010 0011 1111
```