

計算機構成論 第4回

—命令セットアーキテクチャ(1)—

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

■ 命令セットアーキテクチャ

➡ ■ 概説

■ MIPS

- レジスタと基本的な演算命令

- メモリアクセス命令

ソースコードから機械語へ 再掲

C言語

```
int a,b,c,d;  
a = b + c;  
d = a * b;  
...
```

そのままでは
実行できない

アセンブリ言語

```
lw    $9, 4($1)  
lw    $10, 8($1)  
add   $8, $9, $10  
...
```

コンパイラ
により変換

機械語

```
01001101  
00100100  
01001000  
00100000
```

そのまま回路上
で実行可能

- 高級言語のプログラムは、コンパイラ、アセンブラ等により機械語のコードに変換される
- アセンブリ言語と機械語の命令は1対1対応

コンピュータの言葉

■ 命令 (instruction) :

■ コンピュータ言語の言葉

- そのコンピュータがどのような言葉を解釈できるか

■ 命令セット (instruction set) :

■ コンピュータの語彙 (一揃いの命令)

RISCとCISC

■ 命令セットの設計方針

- 1命令でできることを増やす
- 1命令でできることを単純化する

■ RISC (reduced instruction set computer)

- 1命令でできることは少ないが、単純
- 1命令あたりの実行速度が速い

■ CISC (complex instruction set computer)

- 1命令でできることは多いが、複雑
- 1命令あたりの実行速度が遅い

■ 近年では、両者の相違はなくなりつつある

MIPS

- MIPSコンピュータシステムズ
(現MIPSテクノロジーズ)社が
開発した命令セット
 - NINTENDO64, PlayStationなどに採用
 - 1命令が32bit (最新版は64bit)
 - Word: コンピュータで扱うデータ量の単位
→ MIPSでは32ビット
= レジスタやALUで一度に扱うデータのビット幅

命令

各"変数"の値は、
実際には**レジスタ**に格納される

add a, b, c ... $b+c$ の結果をaに格納
sub d, e, f ... $e-f$ の結果をdに格納

MIPSアセンブリの命令 → 機械語と1対1対応で翻訳できる

- 各命令の演算対象のことをオペランド (operand)と呼ぶ

■ add a, b, cであれば、オペランドは3つ (a, b, c)

- 3つの数値を足すなら、2つの命令に分けなければならない

add a, b, c **#a ← b+c**
add a, a, d **#a ← a+d**

設計原則(1) :
単純性は規則性に
つながる

命令セットの違い(例)

■ MIPSの加減算命令

■ add a, b, c

#b+cの結果をaに格納

■ sub a, b, c

#b-cの結果をaに格納

■ x86の加減算命令

■ add a, b

#a+bの結果をaに格納

■ sub a, b

#a-bの結果をaに格納

レジスタ

`add a, b, c`



`add $s0, $s1, $s2`

各"変数"の値は、
実際には**レジスタ**に格納される

レジスタ\$s1と\$s2の値の
和を\$s0に格納

■ レジスタ

- CPU内で演算対象のデータを格納するための記憶領域
- 高速にアクセス可能

■ MIPSでは、レジスタは32bit

- 語(word) : レジスタやALUで一度に扱うビット幅を示すデータ量の単位(32bit)

■ レジスタは32本(+a)ある

確認問題

- 次の説明に合う用語を答えなさい。
- (1) コンピュータ言語の言葉
- (2) コンピュータの語彙(一揃いの命令)
- (3) 1命令でできることは少ないが、
命令が単純となるような命令セットの設計方針
- (4) 1命令でできることが多いが、
命令が複雑となるような命令セットの設計方針
- (5) MIPSテクノロジーズ社(旧MIPSコンピュータ
システムズ)が開発した命令セット
- (6) CPU内で演算対象のデータを格納するための
記憶領域
- (7) レジスタやALUで一度に扱うビット幅を示すデータ量の単位
- (8) 命令の演算対象



講義内容

■ 命令セットアーキテクチャ

■ 概説

■ MIPS

➡ ■ レジスタと基本的な演算命令

■ メモリアクセス命令

レジスタ 再掲

`add a, b, c`



`add $s0, $s1, $s2`

各"変数"の値は、
実際にはレジスタに格納される

レジスタ\$s1と\$s2の値の
和を\$s0に格納

■ レジスタ

- CPU内で演算対象のデータを格納するための記憶領域
- 高速にアクセス可能

■ MIPSでは、レジスタは32bit

- 語(word) : レジスタやALUで一度に扱うビット幅を示すデータ量の単位(32bit)

■ レジスタは32本ある

レジスタの種類

\$zero	0	常にゼロ
\$at	1	アセンブラが一時的に使用
\$v0-v1	2-3	戻り値用
\$a0-a3	4-7	引数用
\$t0-t9	8-15, 24-25	一時レジスタ (一時変数用)
\$s0-s7	16-23	退避レジスタ (変数用)
\$k0-k1	26-27	OSカーネル用に予約
\$gp	28	グローバルポインタ
\$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

レジスタの数

- レジスタは高速
- ならば、レジスタの数は増やせば増やすほど良いか…？
- レジスタを増やすと…
 - 回路が大きくなる
 - 配線が長くなって信号の遅延が増える
 - クロック周波数を落とさざるを得ない



- 設計原則2：小さいほど高速

(例) レジスタを使用した C言語の代入文のコンパイル

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



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

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

- 変数の値は退避レジスタ(\$s0-\$s7)に格納
- 一時的な演算結果は一時レジスタ(\$t0-\$t9)に格納

基本的な演算命令

■ 加算

- 命令 : `add $s1, $s2, $s3`

- 意味 : $\$s1 = \$s2 + \$s3$

■ 減算

- 命令 : `sub $s1, $s2, $s3`

- 意味 : $\$s1 = \$s2 - \$s3$

基本的な演算命令

■ 論理演算 and

- 命令 : `and $s1, $s2, $s3`

- 意味 : $\$s1 = \$s2 \ \& \ \$s3$

■ 論理演算 or

- 命令 : `or $s1, $s2, $s3`

- 意味 : $\$s1 = \$s2 \ | \ \$s3$

■ 論理演算 nor

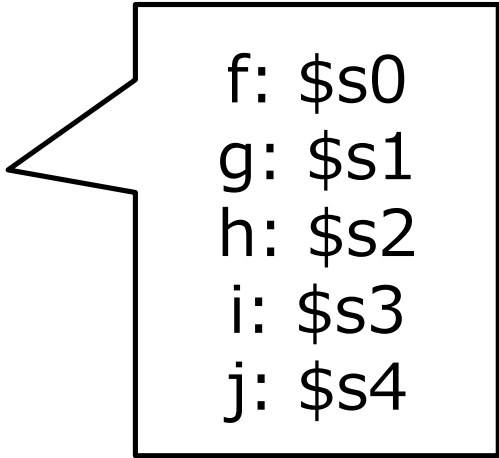
- 命令 : `nor $s1, $s2, $s3`

- 意味 : $\$s1 = \sim(\$s2 \ | \ \$s3)$

確認問題

- 次のアセンブリ命令に対応するCステートメントを示せ。

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



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

※ f,g,h,i,jはあらかじめ
定義されている
int型変数と考えてよい



講義内容

■ 命令セットアーキテクチャ

■ 概説

■ MIPS

■ レジスタと基本的な演算命令

➡ ■ メモリアクセス命令

メモリアクセス

- レジスタに収まらない値をメモリから取得したり、メモリに格納する

$$g = h + \underline{A[8]}$$



```
lw $t0, 32($s3)  
# load word
```

$$\underline{A[8]} = g + h$$



```
sw $t0, 32($s3)  
# store word
```

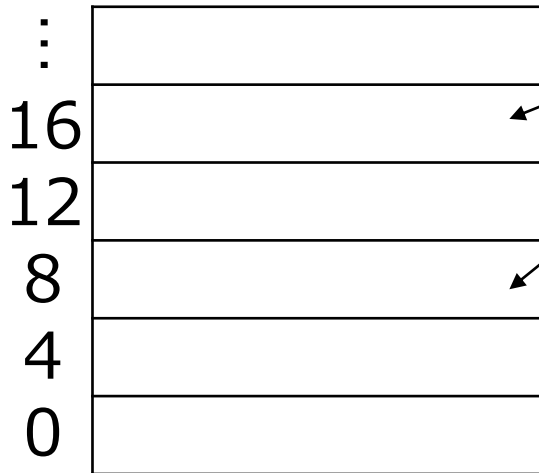
メモリアクセス

■ **lw \$s1, 8(\$s2)**

■ **sw \$s1, 8(\$s2)**

オフセット(offset)

ベースレジスタ(base register)



メモリ

■ メモリセル(1バイト)ごとにメモリ
アドレスが割り振られる

■ 1語 = 4バイト単位

■ 一般的にはDRAMが使用される

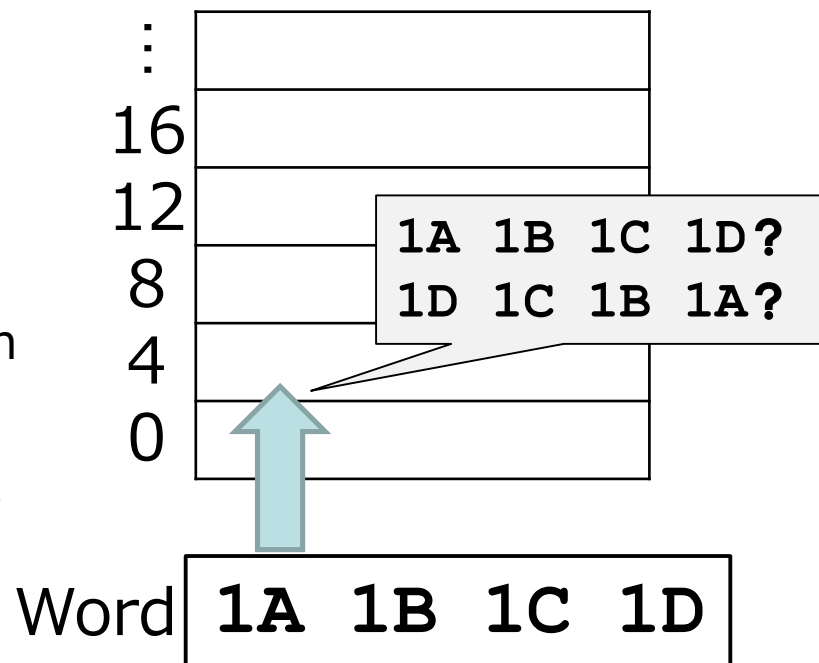
メモリへのワードの格納

■ 整列化制約

- MIPSでは、メモリアクセスの高速化のため、ワードは4の倍数の番地が先頭となるように配置しなければならない

■ 格納方法

- ビッグエンディアン big-endian
 - 上位バイトから順に格納
- リトルエンディアン little-endian
 - 下位バイトから順に格納
- MIPSではビッグエンディアン



メモリアクセス命令

- load word (メモリからレジスタへ転送)
 - 命令 : lw \$s1 20(\$s2)
 - 意味 : $\$s1 = \text{メモリ}[\$s2+20]$
- store word (レジスタからメモリへ転送)
 - 命令 : sw \$s1 20(\$s2)
 - 意味 : $\text{メモリ}[\$s2+20] = \$s1$

プログラム内蔵方式

- コンピュータで処理を行うための命令やデータを、あらかじめメモリ内に格納しておくという方式
- 今日のすべてのコンピュータの基本概念
- ノイマン型とも言われる

C言語

```
int a,b,c,d;  
a = b + c;  
d = a * b;  
...
```

アセンブリ言語

```
lw  $9, 4($1)  
lw  $10, 8($1)  
add $8,$9,$10  
...
```

機械語

```
01001101  
00100100  
01001000  
00100000
```

コンパイラ
により変換

アセンブラ
により変換

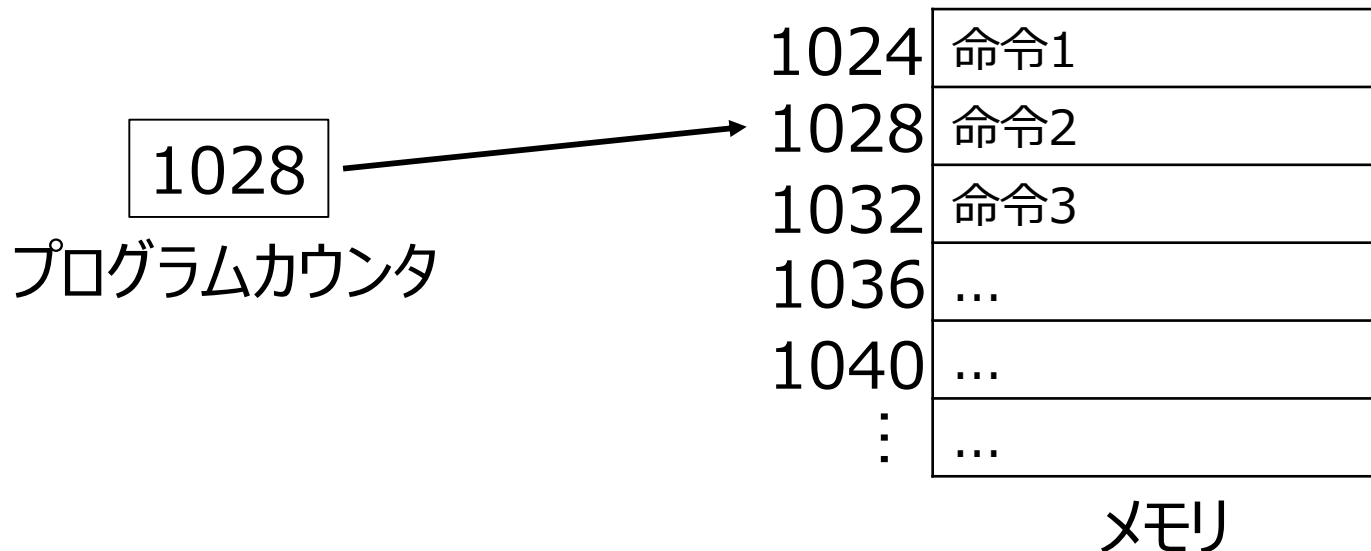


⋮	
16	...
12	...
8	...
4	...
0	01001101 0010...

メモリ

プログラムカウンタ

- 現在実行中の命令のメモリアドレスを示すレジスタ
 - 命令実行のタイミングによっては、次の命令のメモリアドレスを示す
- program counter, PCとも書く



確認問題

- 次の処理に対応するMIPSの演算命令を答えよ
 - 加算
 - 減算
 - メモリから1語読み出す
 - メモリに1語書き込む
- メモリアクセスは何バイト単位で行われるか
- 語を上位バイトから順にメモリに格納する方法を何と呼ぶか
- 語を下位バイトから順にメモリに格納する方法を何と呼ぶか
- 次に実行するプログラムのメモリアドレスを示すものを何と呼ぶか
- 処理を行うためのデータやプログラムをあらかじめメモリに格納しておく方式を何と呼ぶか



確認問題

- (1) MIPSにおいてint型の配列A[100]を宣言した。A[0]が格納されているメモリアドレスが1000のとき、A[50]が格納されているメモリセルの先頭アドレスを答えよ。

■ Hint: int型→4バイト(32ビット)

- (2) MIPSはbig-endianである。
上記(1)の状況で $A[0] = \text{FF332211}_{16}$ のとき、主記憶のアドレス1000, 1001, 1002, 1003の内容をそれぞれ2進数で答えよ。

1000	(A[0]の値)
	(A[1]の値)
	...
?	(A[50]の値)
	...
	...

1000	FF332211
	(A[1]の値)
	...
	(A[50]の値)
	...
	...



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