工学部講義

コンピュータハードウェア（3）

坂井　修一
東京大学大学院 情報理工学系研究科 電子情報学専攻
東京大学　工学部　電子情報工学科／電気工学科

- はじめに
- 命令セットアーキテクチャ

はじめに

- 本講義の目的
  - コンピュータアーキテクチャの基本を学ぶ
- 時間・場所
  - 火曜日 10:15 - 11:45、工3－31
- ホームページ（ダウンロード可能）
  - url:　http://www.mtl.t.u-tokyo.ac.jp/~sakai/hard/
- 教科書
  - 坂井修一『コンピュータアーキテクチャ』（コロナ社、電子情報レクチャーシリーズC-9）

  教科書通りやります
- 参考書
  - D. Patterson and J. Hennessy, Computer Organization & Design、2nd Ed.（邦訳『コンピュータの構成と設計』（第２版)上下 (日系ＢＰ)）
  - 馬場敬信『コンピュータアーキテクチャ』（改訂２版）、オーム社
  - 富田眞治『コンピュータアーキテクチャⅠ』、丸善
- 予備知識：　論理回路
  - 坂井修一『論理回路入門』、培風館
- 成績
  - 試験（＋出席）

講義の概要と予定（1／2）

1．コンピュータアーキテクチャ入門
　ディジタルな表現、負の数、実数、加算器、ＡＬＵ，フリップフロップ、レジスタ、計算のサイクル
2．　データの流れと制御の流れ
　主記憶装置、メモリの構成と分類、レジスタファイル、命令、命令実行の仕組み、実行サイクル、算術論理演算命令、シーケンサ、条件分岐命令
3．命令セットアーキテクチャ
　操作とオペランド、命令の表現形式、アセンブリ言語、命令セット、算術論理演算命令、データ移動命令、分岐命令、アドレシング、サブルーチン、RISCとCISC
4．パイプライン処理（1）
　パイプラインの原理、命令パイプライン、オーバヘッド、構造ハザード、データハザード、制御ハザード
5．パイプライン処理（2）
　フォワーディング、遅延分岐、分岐予測、命令スケジューリング
6．キャッシュ
　記憶階層と局所性、透過性、キャッシュ、ライトスルーとライトバック、ダイレクトマップ型、フルアソシアティブ型、セットアソシアティブ型、キャッシュミス

講義の概要と予定（2／2）

7．仮想記憶
　仮想記憶、ページフォールト、TLB、物理アドレスキャッシュ、仮想アドレスキャッシュ、メモリアクセス機構
8．命令レベル並列処理（1）
　並列処理、並列処理パイプライン、VLIW、スーパスカラ、並列処理とハザード
9．命令レベル並列処理（2）
　静的最適化、ループアンローリング、ソフトウェアパイプライニング、トレーススケジューリング
10．アウトオブオーダ処理
　インオーダーとアウトオブオーダー、フロー依存、逆依存、出力依存、命令ウィンドウ、リザベーションステーション、レジスタリネーミング、マッピングテーブル、リオーダバッファ、プロセッサの性能
11．入出力と周辺装置
　周辺装置、ディスプレイ、二次記憶装置、ハードウェアインタフェース、割り込みとポーリング、アービタ、DMA、例外処理

試験：　7月後半

３．命令セットアーキテクチャ

- 内容
  - 命令の表現形式とアセンブリ言語
    - 操作とオペランド
    - 命令の表現形式
    - アセンブリ言語
  - 命令セット
    - 算術論理演算命令
    - データ移動命令
    - 分岐命令
  - アドレシング
    - アドレシングの種類
    - バイトアドレシングとエンディアン
    - ゼロレジスタと定数の生成
  - サブルーチンの実現
    - サブルーチンの基本
    - スタックによるサブルーチンの実現
    - サブルーチンのプログラム
  - 練習問題

命令セットとは何か？

- 命令セット
  - 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

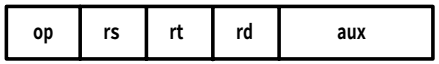
操作とオペランド

- 命令 ＝ 操作 op ＋ 対象
  - 対象 ＝ オペランド
    - ソースオペランド
    - デスティネーションオペランド
  - オペランドとなるもの
    - データレジスタ
    - メモリ語
    - プログラムカウンタ
    - その他のレジスタ
    - 即値

op：	$\log_2$(対象とするコンピュータの命令セットの大きさ)
rs, rt, rd：	$\log_2$(レジスタファイルに含まれるレジスタの数)
aux, imm/dpl, addr：	（命令長）－（他のフィールドのビット数の総和）

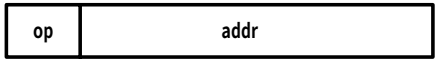
図 3.3 命令フィールドの大きさ

命令の表現形式



(1) R型

(2) I型



(3) A型

op: 操作コード、 rs, rt, rd: オペランドレジスタ、 aux: 実行細則、
imm/addr: 即値または変位、addr: メモリアドレス

命令フィールドの大きさ

op：	$\log_2$（対象とするコンピュータの命令セットの大きさ）
rs, rt, rd：	$\log_2$（レジスタファイルに含まれるレジスタの数）
aux, imm/dpl, addr：	（命令長）－（他のフィールドのビット数の総和）

図 3.3 命令フィールドの大きさ

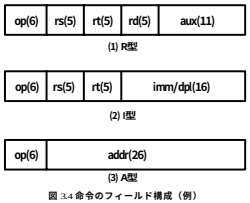
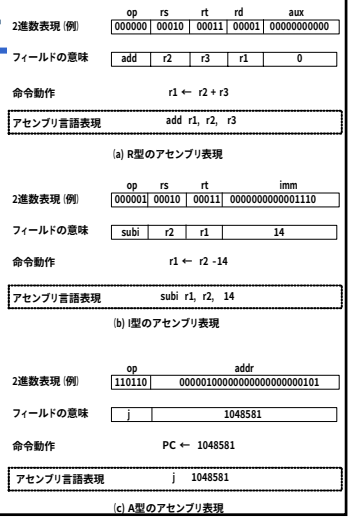


図 3.4 命令のフィールド構成（例）

↑ 命令語が32ビット、命令セットの大きさが64、レジスタ数が32

アセンブリ言語

- プログラムの表記
  - 機械語： 読みにくい
  - アセンブリ言語
    - 英語に近い記号で表記
    - 機械語と1対1対応



命令セット

- 命令の分類 (復習)
  - 算術論理演算命令
  - データ移動命令
  - 分岐命令
- 算術論理演算命令

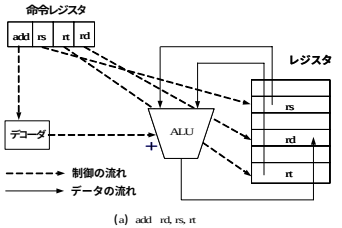
表 3.1 算術演算命令

	整数演算命令		浮動小数点演算命令
	R 型	I 型	R 型
加算	add	addi	fadd
減算	sub	subi	fsub
乗算	mul	muli	fmul
除算	div	divi	fdiv
剰余	rem	remi	
絶対値	abs		fabs
算術左シフト	sla		
算術右シフト	sra		

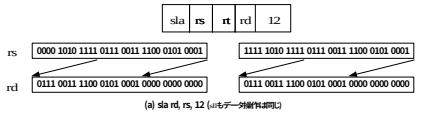
表 3.2 論理演算命令

	R 型	I 型
論理積	and	andi
論理和	or	ori
否定	not	
NOR	nor	nori
NAND	nand	nandi
排他的論理和	xor	xori
EQUIV	eq	eqi
論理左シフト	sll	
論理右シフト	srl	

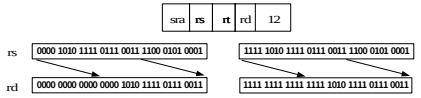
算術論理演算命令の動作例



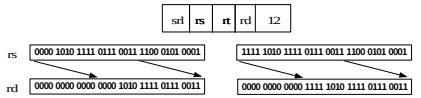
シフト命令



(a) sla rd, rs, 12 (srもデータ操作は同じ)



(b) sra rd, rs, 12



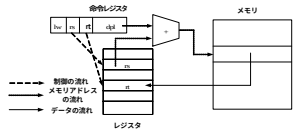
(c) srl rd, rs, 12

図 3.7 シフト命令

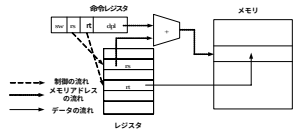
データ移動命令

表 3.3 データ移動命令

移動量	メモリ ⇒ レジスタ		レジスタ ⇒ メモリ	
64ビット	ld	load double word	sd	store double word
32ビット	lw	load word	sw	store word
16ビット	lh	load half word	sh	store half word
8ビット	lb	load byte	sb	store byte



(a) lw rt dpl(rs)



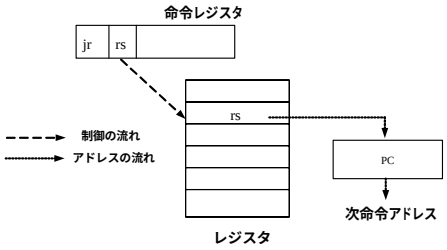
(b) sw rt dpl(rs)

図 3.8 データ移動命令の動作

分岐命令 (1)： 無条件分岐命令

表 3.4 無条件分岐命令

命令	意味	形式	アセンブリ言語の表現	動作
j	jump	A	j addr	pc ← addr
jr	jump register	R	jr rs	pc ← (rs)
jal	jump and link	A	jal addr	r31 ← (pc) +4; pc ← addr



分岐命令 (2)：条件分岐命令

表 3.5 条件分岐命令

命令	意味	形式	アセンブリ言語の表現	動作
beq	branch on equal	I	beq rs, rt, dpl	rs = rt ならば pc = (pc) +4+dpl
bne	branch on not equal	I	bne rs, rt, dpl	rs<>rt ならば pc = (pc) +4+dpl
blt	branch on less than	I	blt rs, rt, dpl	rs<rt ならば pc = (pc) +4+dpl
ble	br. on less than or eq.	I	ble rs, rt, dpl	rs<=rt ならば pc = (pc) +4+dpl

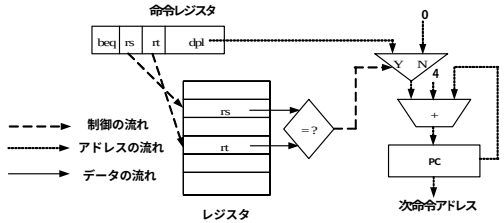
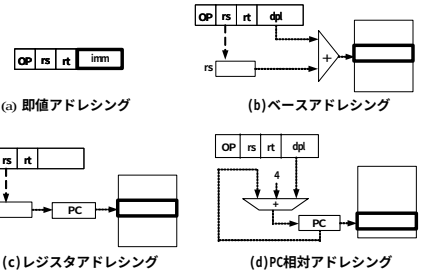


図 3.10 条件分岐命令の動作

アドレシング

表 3.6 アドレシング

アドレシング方式	命令の例（アセンブリ言語）	生成されるアドレス
即値アドレシング	addi rt, rs, imm	（直接値 imm を生成）
ベース相対アドレシング	lw rt, dpl(rs)	(rs) + dpl
レジスタ・アドレシング	j rs	(rs)
PC 相対アドレシング	beq rs, rt, dpl（分岐するとき）	(pc)+4+dpl



(a) 即値アドレシング　(b)ベースアドレシング

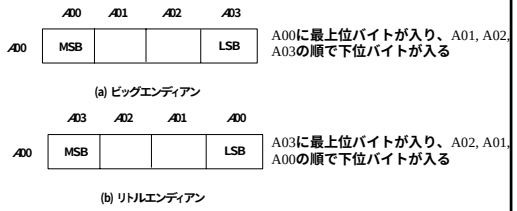
(c)レジスタアドレシング　(d)PC相対アドレシング

図 3.11　アドレシング

バイトアドレシングとエンディアン

◆バイトアドレシング：　アドレシングの単位を「語」ではなく「バイト」とするアドレシング
普通のメモリアドレシングはバイトアドレシング

◆エンディアン：　語の中のバイトの配列順を定めたもの



MSB: Most Significant Byte （最上位バイト）、LSB: Least Significant Byte （最下位バイト）

図 3.12 ビッグエンディアンとリトルエンディアン

ゼロレジスタと定数の生成

- ゼロレジスタ
  - 恒常的に"0"が入っているレジスタ（定数）
  - この授業ではr0がゼロレジスタとする

```
addi r1, r0, 28
```
(a) 定数の生成（16 ビット）

```
addi r1, r0, 0101010101010101
sla r1 r1 16
ori r1, r1, 0000000011111111
```
(b) 定数の生成（32 ビット）

```
eq r1 r0 r1
```
(c) ビット反転

図 3.13　ゼロレジスタによる定数の生成とビット反転

サブルーチン

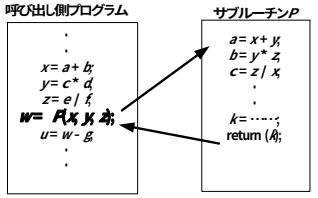
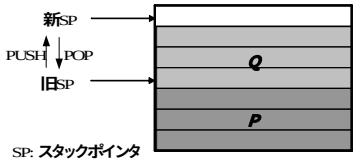


図 3.14 サブルーチンの基本形

(1) レジスタ値の待避
(2) 戻り番地（次の命令番地）の待避
(3) サブルーチンの先頭番地へのジャンプ
(4) サブルーチン本体の実行
(5) 戻り番地へのジャンプ
(6) レジスタ値の復帰
(7) もとの命令列の実行再開

図 3.15　サブルーチンの手順

スタックによるサブルーチンの実現



スタック

スタック
＝ LIFO型メモリ

LIFO = Last In First Out

P→ *Q*→ *R*の順でサブルーチンが呼ばれたとき

図 3.16 スタックとサブルーチン

```
sw r1, 0(sp)          sub sp sp 4
add sp sp 4           lw r1, 0(sp)
(a) プッシュ           (b) ポップ
```

図 3.17 スタック操作のプログラム

サブルーチンのプログラム

```
        sw r1 0(sp)        ;   レジスタ値の待避（必要なだけ）始め
        sw r2 4(sp)
        .....
        sw rk 4k(sp)       ;   レジスタ値の待避終わり
        add sp 4k+4
        jal address
        sub sp 4k+4
        lw r1 0(sp)        ;   レジスタ値の復帰始め
        lw r2 4(sp)
        .....
        lw rk 4k(sp)       ;   レジスタ値の復帰終わり
        もとの仕事の続き
        ..............
```

```
address:  .....          サブルーチン本体
          .....
          .....
          jr r31
```

図 3.18 サブルーチンのアセンブラプログラム

RISCとCISC

- プロセッサの分類
  - RISC: Reduced Instruction Set Computer
  - CISC: Complex Instruction Set Computer

	RISC	CISC
命令数	少ない	多い
命令形式	一語固定長	可変長
個々の命令動作 (アドレシングモード)	単純	複雑
レジスタ数	多い	少ない
例	Sun Sparc MIPS R10000 IBM PowerPC Comaq Alpha	Intel X86 Motorola M68000

RISC vs. CISC

- 歴史的な考察
  - CISCが先にあった（1960年代頃～）
    - レジスタは高価
    - 命令の種類（特にアドレッシングの種類）は多数あればユーザの要求に応えられると考えられた
  - CISCへの反省
    - じっさいの計算では、ほとんどが単純な命令
    - 複雑な命令
      - コンパイラが出力するのが難しい
      - 単純な命令の組合せで実現可能
  - RISCの発案と展開
    - 1980年代の潮流： Cocke (IBM, Turing Award Winner), Patterson(UCB), Hennessy(Stanford)ら
    - 「RISCはCISCより速い」は真実！
    - IntelにおいてもCISC命令をRISCに解釈し直して実行している