コンピュータアーキテクチャ

第3回　2020年10月21日（水）
第4回　2020年10月28日（水）

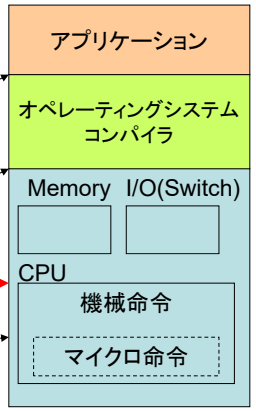
福井大学大学院工学研究科　情報・メディア工学専攻
森眞一郎

[主な講義内容]
- コンピュータの基本構成
  - … 基本構成と動作原理、性能指標、アムダールの法則
- 命令セットアーキテクチャ
  - … 命令型式とアドレス修飾、命令語の種類と機械語の構成
- メモリアーキテクチャ
  - … 記憶階層、キャッシュメモリ、仮想記憶
- 入出力アーキテクチャ
  - … 割り込みと入出力処理、バス、補助記憶装置
- プロセッサアーキテクチャ
  - … 文字・数字の表現、 四則演算回路と高速化、実行制御
- コンピュータの高性能化
  - … パイプライン処理、並列処理

[復習]コンピュータアーキテクチャとは？ (Computer Architecture)

プログラマから見えるコンピュータの論理仕様

- 「見る」レベルを変えると色々なアーキテクチャが定義できる
  - OS（レベル）アーキテクチャ
  - PMS(Processor-Memory-Switch)（レベル）アーキテクチャ
  - 命令セット（レベル）アーキテクチャ
  - マイクロアーキテクチャ

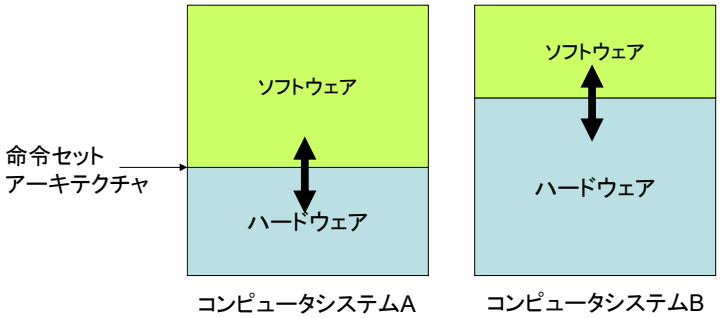


[復習]コンピュータアーキテクチャとは？ (Computer Architecture)

プログラマから見えるコンピュータの論理仕様

コンピュータシステムにおける
ハードウェアとソフトウェア
のトレードオフ（適切なバランスの決定）

→ コンピュータアーキテクチャ設計



命令セットアーキテクチャ ([ISA] Instruction Set Architecture)

命令セットアーキテクチャとは？

コンピュータのハードウェアとソフトウェアのインタフェースであり、
コンピュータアーキテクチャの最も基本となるものである。　(Amdahlの定義)

「コンピュータアーキテクチャ」、内田、小柳、オーム社
「コンピュータアーキテクチャ」、馬場敬信、オーム社

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能が必要かをハードウェア設計者に教えるものである。

「コンピュータアーキテクチャ」、坂井修一、コロナ社

コンパイラやOSなどの基本ソフトウェア（システムプログラム）機能を作成する際に、
そのプログラマが意識するハードウェア機能あるいはハードウェア構成方式である。

「コンピュータアーキテクチャ」、柴山潔、オーム社

命令セットアーキテクチャ ([ISA] Instruction Set Architecture)

用語

命令：コンピュータの動作を決めるもの
(Instruction)

命令セット：コンピュータの全ての命令の集まり
(Instruction Set)

機械語：ハードウェアが理解できる0と1の集合として命令を表現したもの
(Machine Language)

アセンブリ言語：　機械語と1対1で、人間が分かりやすい表現で命令を表現したもの
(Assembly Language)

アセンブラ：アセンブリ言語を機械語に変換するプログラム
(Assembler)

命令セットアーキテクチャ

命令語の一般的な構成

命令コード (操作コード)	オペランド1	オペランド2	オペランド3	・・・	AUX

- 命令コード(OP Code: Operation Code)
  命令の種類を表す
- N個のオペランド(Operand)　（通常　N　＝　3 or 2 [or 1 or 0]）
  命令が使用するデータを指定する
  （レジスタや主記憶に格納された変数や定数）
- その他の付加的な情報(AUX: AUXiliary field)

命令セットアーキテクチャ

命令セットが備えるべき基本的な機能

例えばComet IIでは

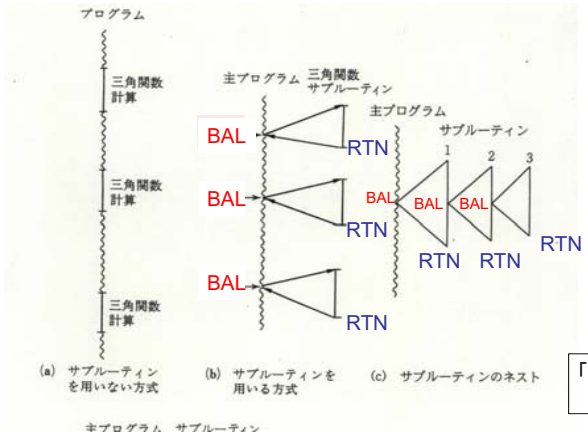
- 演算命令
  四則演算、論理演算、シフト演算、など　ADD*,SUB*.AND,OR, XOR, SL*, SR* 等
- データ転送命令
  主記憶からレジスタへの転送命令「LOAD命令」　LD
  レジスタから主記憶への転送命令「STORE命令」　ST
- プログラム制御命令
  特定のアドレスの命令に分岐する「分岐命令」　JPL,JNZ,JUMP等
  サブルーチン(関数)を呼び出す「手続き呼出し命令」　CALL
  サブルーチンから復帰する「リターン命令」　RET
  など
- その他の命令
  入出力命令、OS呼び出しなどのシステム制御用命令　SVC

[補足] 教科書では　無条件分岐命令、条件分岐命令のみを分岐命令と定義しているが、授業ではプログラム制御命令全体を分岐命令として扱います。

命令セットアーキテクチャ

命令セットが備えるべき基本的な機能

サブルーチン(関数)を呼び出す「手続き呼出し命令(BAL)」
サブルーチンから復帰する「リターン命令(RTN)」



「コンピュータアーキテクチャI」
富田眞治　著、丸善　より引用

注意: H24年度までの教科書ではリターンアドレスの退避先として$r31が指定されているがこれはあくまでも一例にすぎない。以下の例は、M(Rs)という「ある番地」にリターンアドレスを退避(保存)する例

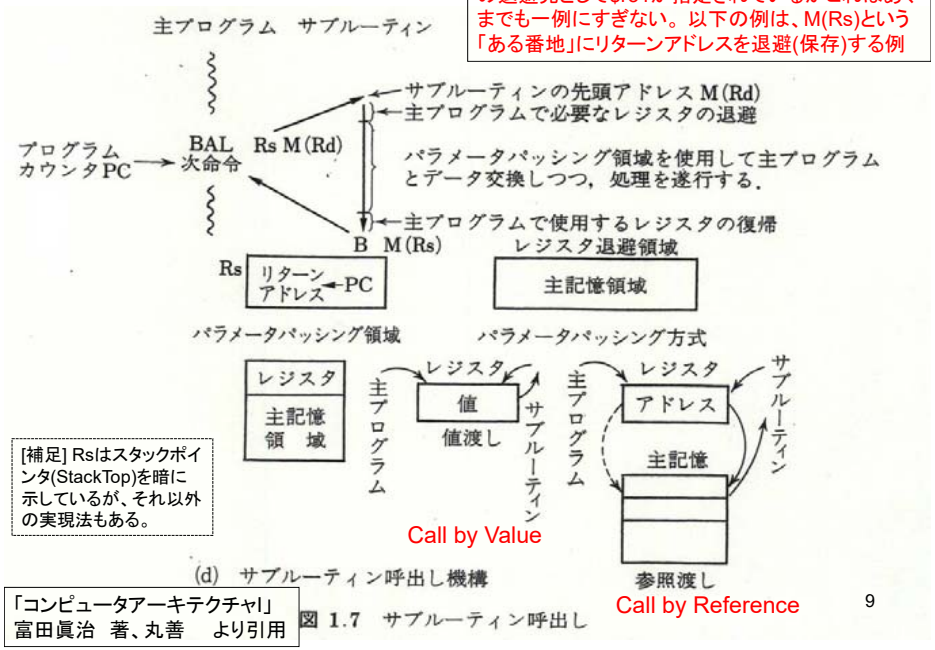


図 1.7 サブルーティン呼出し

[補足] Rsはスタックポインタ(StackTop)を暗に示しているが、それ以外の実現法もある。

「コンピュータアーキテクチャI」
富田眞治　著、丸善　より引用

命令セットアーキテクチャ

命令型式(オペランドの数による分類)

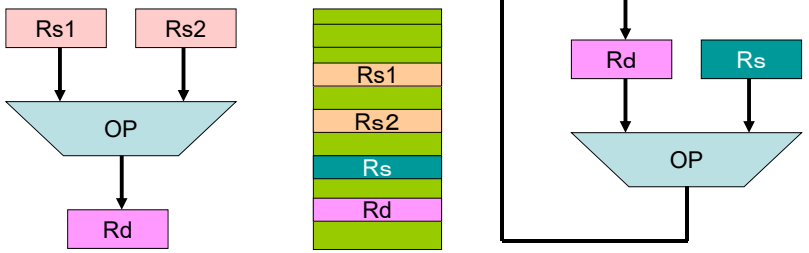
- 3アドレス方式
- 2アドレス方式
- 1アドレス方式(Accumulator Machine)
- 0アドレス方式(Stack Machine)

教科書には、さらに多くの種類が書かれていますが、通常は上記4つで十分です。

命令セットアーキテクチャ

命令型式(オペランドの数による分類)

- 3、2アドレス方式に対応する演算器周りのデータパス



レジスタあるいはメモリ

Rd ← Rs1 OP Rs2

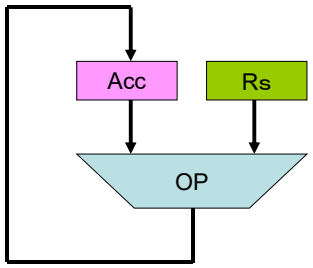
Rd ← Rd OP Rs

命令セットアーキテクチャ

命令型式(オペランドの数による分類)

- 1アドレス方式に対応する演算器周りのデータパス

デスティネーションならびにオペランドの一方が専用のレジスタ(アキュムレータAccumulator)に限定される。



一般の演算命令

Acc ← Acc OP Rs
Accの内容と オペランドのデータ との演算を行い結果をAccに格納

データ転送命令

Rd ← Acc
Accの内容を オペランド に格納

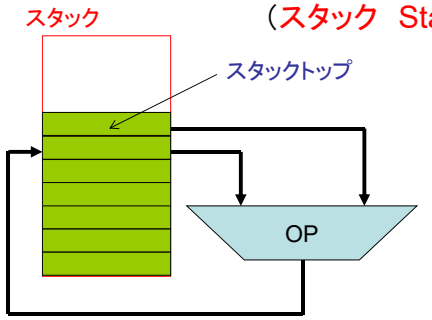
ACC ← Rs
オペランドのデータ を Acc に格納

命令セットアーキテクチャ

命令型式（オペランドの数による分類）

- 0アドレス方式に対応する演算器周りのデータパス

全てのオペランドが専用の記憶領域
（スタック　Stack）に限定される。



スタックの先頭（スタックトップ）から2つの
データをソースオペランドとして取り出し、
演算した結果を、スタックの先頭に格納する

メモリとスタックとの間のデータ転送命令
（PUSH, POP)のみオペランドとなる
アドレス指定が必要

スタックメモリ（Last In First Out)　⇔　FIFOメモリ（First In First Out）

命令セットアーキテクチャ

スタックマシンの動作原理

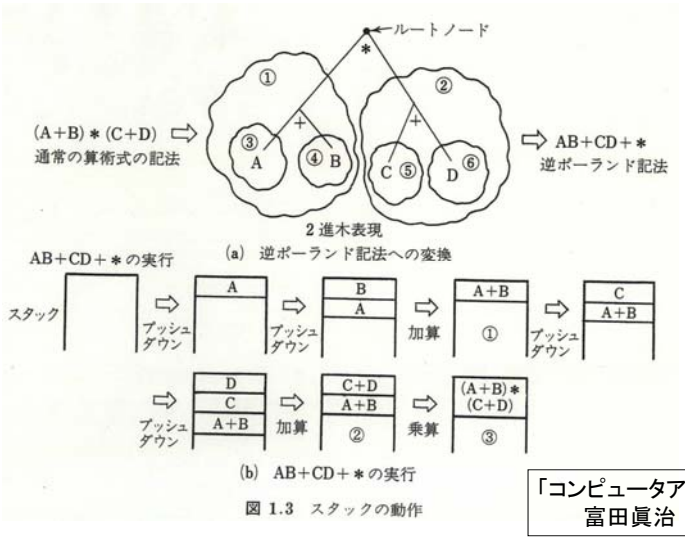


図 1.3 スタックの動作

「コンピュータアーキテクチャI」
富田眞治　著、丸善　より引用

命令セットアーキテクチャ

命令型式（オペランドの数による分類）

アドレス型式の違いによるプログラムの違い
E = (A+B) － (C+D)

3アドレス方式

```
ADD T1,A,B      //  T1 ←A + B
ADD E,C,D       //   E  ←C + D
SUB E,T1,E      //   E  ←T1 - E
```

2アドレス方式

```
LOAD T1,A       // T1←A
ADD   T1,B      // T1←T1 + B
LOAD T2,C       // T2←C
ADD   T2,D      // T2←T2 + D
SUB   T1,T2     // T1←T1 – T2
STORE E,T1      //  E ←T1
```

(4命令まで短くできます)

(余談)　数学での A+B-C-Dと
計算機上での (A+B) – (C+D) は
必ずしも一致しません。[有効桁数]

(注)ここで使っている命令セットは一例にすぎません。

命令セットアーキテクチャ

命令型式（オペランドの数による分類）

アドレス型式の違いによるプログラムの違い
E = (A+B) － (C+D)

1アドレス方式

```
LOAD   A        // ACC←A
ADD    B        // ACC←ACC+B
STORE T1        // T1   ←ACC
LOAD   C        // ACC←C
ADD    D        // ACC←ACC+D
STORE T2        // T2   ← ACC
LOAD   T1       //ACC ← T1
SUB    T2       //ACC ←ACC –T2
STORE E         //E    ←ACC
```

(もっと短くできます)

(注)ここで使っている命令セットは
一例にすぎません。

0アドレス方式

```
PUSH   C    // CをスタックにPUSH
PUSH   D    //  DをスタックにPUSH
ADD         // スタックの先頭2要素を
            // POPし加算したのち結果
            // をスタックにPUSH
PUSH   A    // AをスタックにPUSH
PUSH   B    // BをスタックにPUSH
ADD         // スタックの先頭2要素を
            // POPし加算したのち結果
            // をスタックにPUSH
SUB         // スタックの先頭2要素を
            // POPし減算したのち結果
            // をスタックにPUSH
POP    E    //スタックの先頭1要素を
            // POPし  E  に保存
```

命令セットアーキテクチャ

3(or 2)アドレス方式における命令型式の分類

オペランドがレジスタかメモリかによる分類

- R-R型
  すべてのオペランドがレジスタである命令型式
- S-S型（旧教科書ではM-M型）
  すべてのオペランドがメモリである命令型式
- R-S型（旧教科書ではR-M型）
  オペランドがレジスタのものと、メモリのものが混在する命令型式

・もっと詳細に分類することもあります（RR,RX,RS,SI,SS）
・一般にメモリをオペランドに指定すると、命令長が長くなるとともに、命令の実行に要する時間も長くなる。

命令セットアーキテクチャ

- 分岐のあるプログラムの動作(概念的な説明)

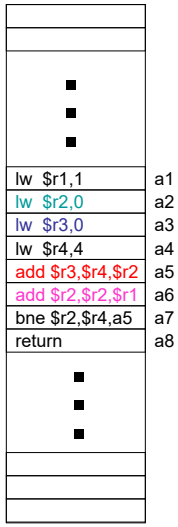
```
int i,s;

s=0;

for(i=0;i<4;i++){
    s=s+i;
}
```

```
    lw  $r1, 1
    lw  $r2, 0
    lw  $r3, 0
    lw  $r4, 4

L1:
    add   $r3,$r3,$r2
    add   $r2,$r2,$r1
    bne   $r2, $r4, L1
    return
```



この時、プログラムカウンタの値は

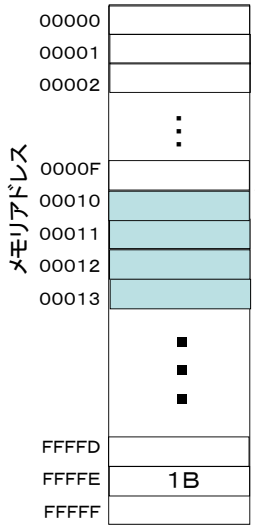
a1 a2 a3 a4|a5 a6 a7|a5 a6 a7|a5 a6 a7|a5 a6 a7|a8

の順で変化する

命令セットアーキテクチャ

アドレッシング

- メモリ上のデータ配置



CPU内部で扱うデータの基本単位は　ワード（語, Word）
一般に　32bitプロセッサであれば32bit（4B），
64bitプロセッサであれば64bit（8B）

語（Word）境界

語（Word）長

メモリアドレスは
1バイト（1B）単位に
割り当てられている

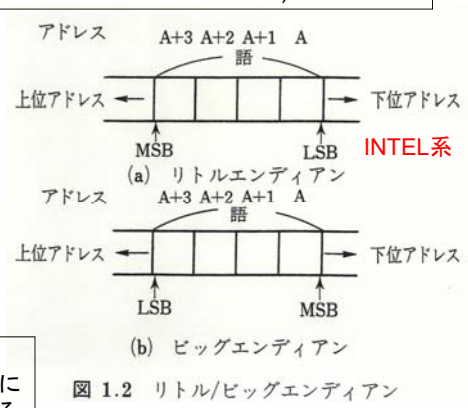
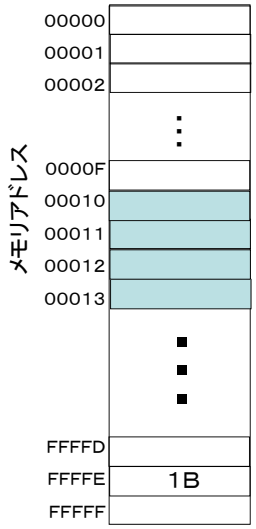


図 1.2　リトル/ビッグエンディアン

命令セットアーキテクチャ

アドレッシング

- メモリ上のデータ配置



命令もメモリ上に格納されるが、個々の命令語の長さが

同じ場合　⇒ 固定長命令
Fixed length instruction

異なる場合　⇒ 可変長命令
Variable length instruction

CISCが主流の時代は可変長命令が多用されていたが、
RISCが主流の現在は固定長命令が主流

さて、1語32ビットのコンピュータにおいては、
アドレス空間も32bit（以上）

(複数の)メモリアドレスを、固定長（例えば1Word）
の命令内に直接格納するのは不可能！！

⇒ **メモリアドレスを指定する何らかの仕組みが必要**

命令セットアーキテクチャ
アドレッシング

- アドレッシング

主なアドレッシング方式

直接アドレッシング(direct addressing)

間接アドレッシング(indirect addressing)
メモリ間接、レジスタ間接

即値アドレッシング(immediate addressing)

相対アドレッシング(relative addressing)
メモリ相対、レジスタ相対、PC相対

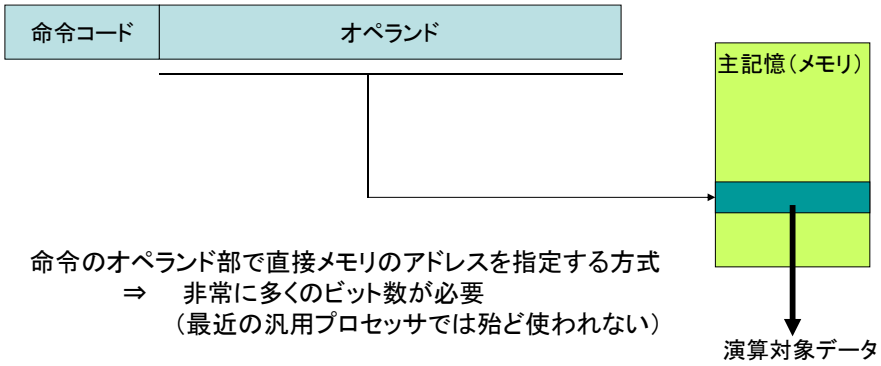
インデックス修飾(index modification)

絶対アドレス (Absolute address)

[アドレス修飾後の] 実効アドレス (Effective address)

命令セットアーキテクチャ
アドレッシング

- 直接アドレッシング(direct addressing)

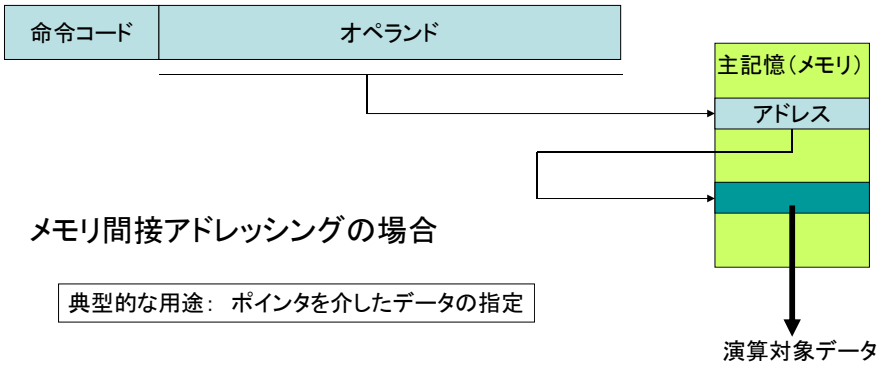


命令のオペランド部で直接メモリのアドレスを指定する方式
⇒ 非常に多くのビット数が必要
(最近の汎用プロセッサでは殆ど使われない)

命令セットアーキテクチャ
アドレッシング

- 間接アドレッシング(indirect addressing)

命令のオペランド部をアドレスとして一度メモリ(あるいはレジスタ)から値を
読み出し、読み出した内容をオペランドのアドレスとしてメモリのアドレスを指定



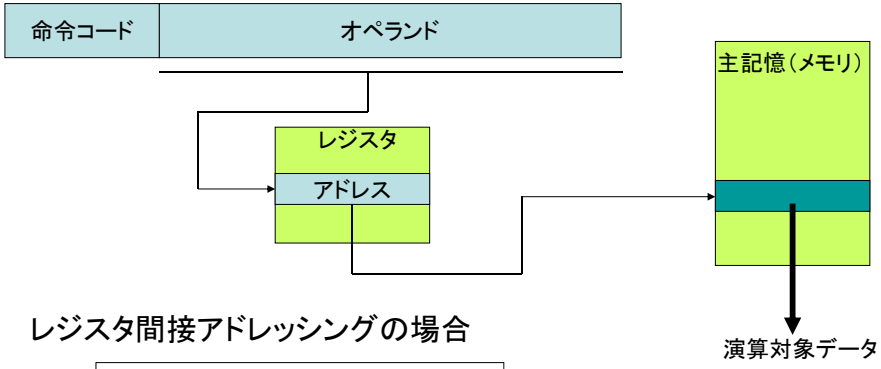
メモリ間接アドレッシングの場合

典型的な用途: ポインタを介したデータの指定

命令セットアーキテクチャ
アドレッシング

- 間接アドレッシング(indirect addressing)

命令のオペランド部をアドレスとして一度メモリ(あるいはレジスタ)から値を
読み出し、読み出した内容をオペランドのアドレスとしてメモリのアドレスを指定



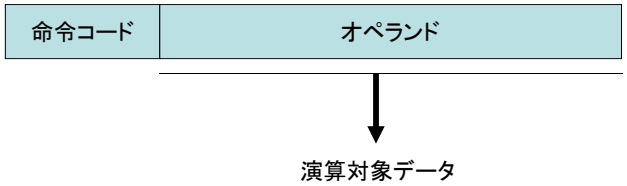
レジスタ間接アドレッシングの場合

典型的な用途: リターンアドレスの指定

命令セットアーキテクチャ
アドレッシング

- 即値アドレッシング(immediate addressing)

命令のオペランド部で直接オペランドそのものを指定する

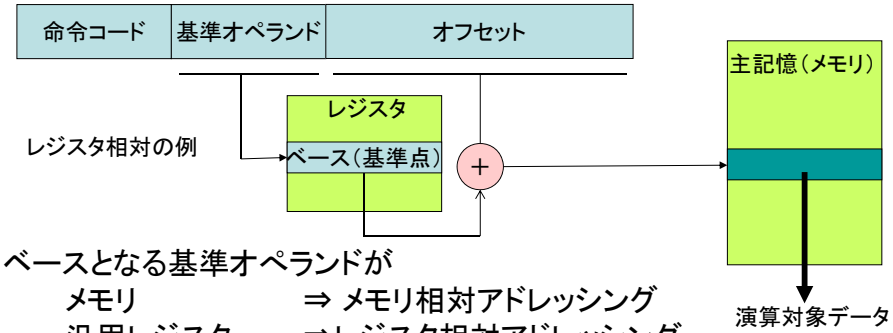


典型的な用途： 定数の指定

命令セットアーキテクチャ
アドレッシング

- 相対アドレッシング(relative addressing)

命令の基準オペランド部で指定されるレジスタあるいはメモリの内容を基準点とし、それにオフセットを加えたものをアドレスとしてメモリのアドレスを指定



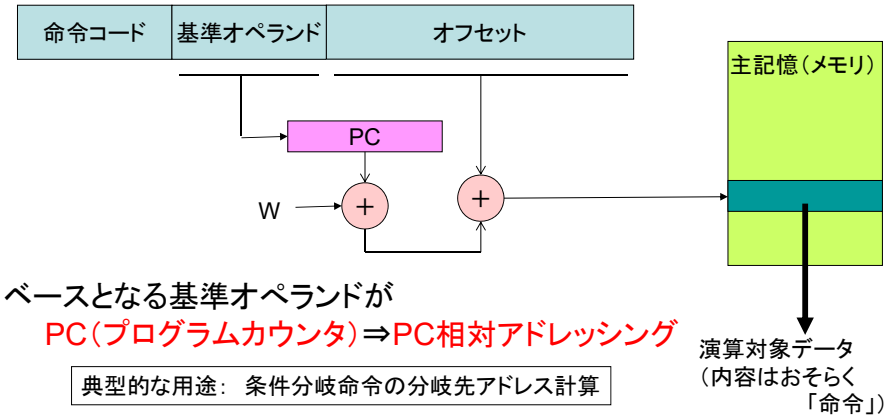
ベースとなる基準オペランドが
メモリ ⇒ メモリ相対アドレッシング
汎用レジスタ ⇒レジスタ相対アドレッシング

典型的な用途： メモリ上に構築したスタックやFIFOなどのポインタ操作、構造体アクセス

命令セットアーキテクチャ
アドレッシング

- 相対アドレッシング(relative addressing)

命令の基準オペランド部で指定されるレジスタあるいはメモリの内容を基準点とし、それにオフセットを加えたものをアドレスとしてメモリのアドレスを指定



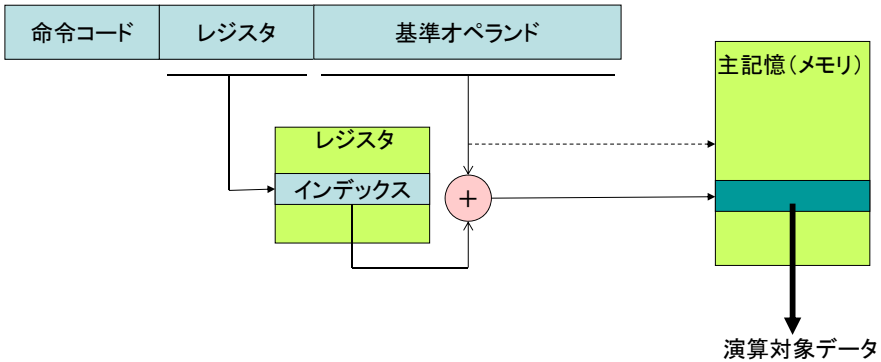
ベースとなる基準オペランドが
PC(プログラムカウンタ)⇒PC相対アドレッシング

典型的な用途： 条件分岐命令の分岐先アドレス計算

命令セットアーキテクチャ
アドレッシング

- インデックス修飾(index modification)

命令の基準オペランド部で指定されるメモリ内の基準点と、それに対するオフセットをインデックスとして加えたものをアドレスとしてメモリのアドレスを指定

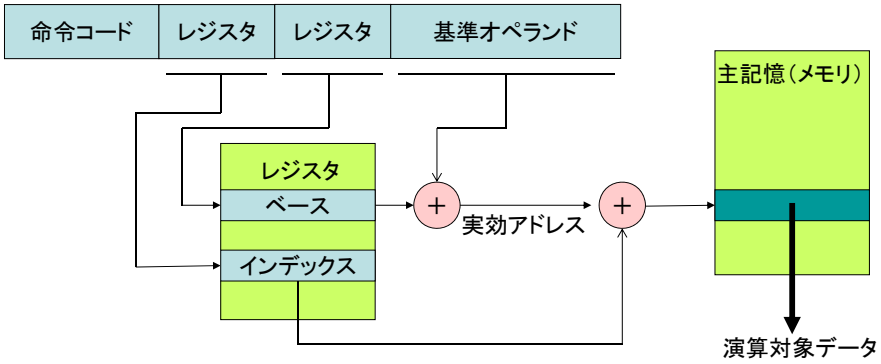


典型的な用途： 配列の要素指定(基準オペランドが配列の先頭アドレス)

命令セットアーキテクチャ
アドレッシング

- インデックス修飾(index modification)

レジスタ相対アドレッシングにインデックス修飾した例

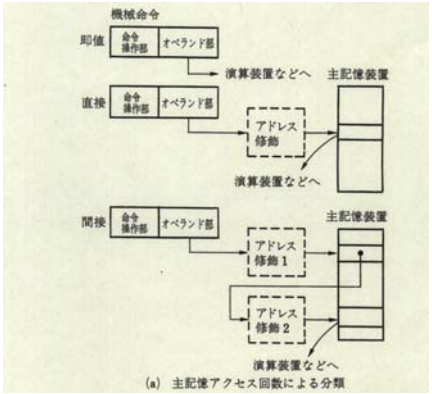


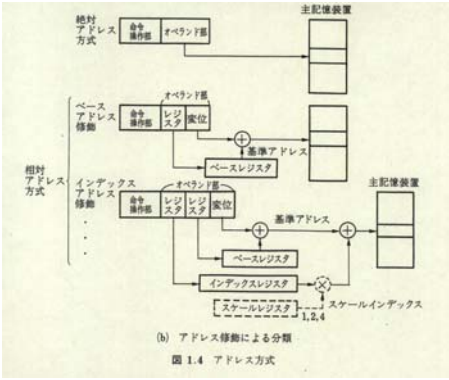
典型的な用途： 配列の要素指定（基準オペランドが配列の先頭アドレス）

命令セットアーキテクチャ
アドレッシング

- 違った視点での分類

（1）即値，直接，間接　　（2）絶対，相対





教. P64

命令セットアーキテクチャの例

CISC と RISC

Complex Instruction Set Computer

命令セットを大きくし、アドレッシング方式を数多く用意することで、目的とする操作をできるだけ1命令で実現し、プログラムの実行効率向上を目指す方式

Reduced Instruction Set Computer

命令は良く使われるものに限定し、アドレッシング方式も必要最小限に絞ってハードウェアを簡略化し、命令実行時間を短くすることで高速化を図る方式。メモリとCPUとの間のデータの受け渡しはロード命令とストア命令のみ。

・初期のコンピュータは基本的に CISC
・1980年代後半にRISCの概念が提案されて以降は（CPUの高速化とともに）RISCが優勢
・Intel の Pentium系CPUも(1995以降は) ユーザに見せるISAは過去のCPUとの互換性維持のためCISCであるが、CPUの内部でCISC命令をRISC命令に変換（分解）して実行

「コンピュータアーキテクチャ」、坂井修一、コロナ社

命令セットアーキテクチャの例

MIPSアーキテクチャ
- ・32bitのレジスタが32個
  （レジスタ指定に5bit、 但し r0 はゼロレジスタ)
- ・主記憶は語長32bitでアドレス空間は2^{32}
- ・基本的な命令形式(R-R型)
- ・メモリとの間のデータ転送はロードストア時のみ

⇒ **ロードストアアーキテクチャ**

- R形式：演算を表す命令の型式

OP [6bit]	rs	rt	rd	AUX(shamt,funct) [5+6 bit]

- I形式：データ転送命令の形式

OP [6bit]	rs	rt	Immediate/displacement (address) [16bit]

- J形式（A形式）：ジャンプ用命令形式

OP [6bit]	Address (Immediate/displacement) [26bit]

命令セットアーキテクチャの例

MIPSアーキテクチャの命令の種類

・演算命令

命令の種類	アセンブリ言語の表記	動作
加算	add $r1,$r2,$r3	$r1 = $r2 + $r3
減算	sub $r1,$r2,$r3	$r1 = $r2 - $r3
乗算	mult $r1,$r2,$r3	$r1 = $r2 * $r3
除算	div $r1,$r2,$r3	$r1 = $r2 / $r3
論理積	and $r1,$r2,$r3	$r1 = $r2 & $r3
論理和	or $r1,$r2,$r3	$r1 = $r2 | $r3
排他的論理和	xor $r1,$r2,$r3	$r1 = $r2 ^ $r3
左シフト	sll $r1,$r2, n	$r1 = $r2 << n
右シフト	srl $r1,$r2, n	$r1 = $r2 >> n

・データ転送命令

命令の種類	アセンブリ言語の表記	動作
load word	lw $r1, n($r2)	$r1 = MM[$r2 + n]
store word	sw $r1, n($r2)	MM[$r2 + n] = $r1
load byte	lb $r1, n($r2)	$r1 = MM[$r2 + n]
store byte	sb $r1, n($r2)	MM[$r2 + n] = $r1

・即値命令

命令の種類	アセンブリ言語の表記	説明
add immediate	addi $r1, $r2, n	$r1 = $r2 + n
and immediate	andi $r1, $r2, n	$r1 = $r2 & n
or immediate	ori $r1, $r2, n	$r1 = $r2 | n
xor immediate	xori $r1, $r2, n	$r1 = $r2 ^ n
set less than immediate	slti $r1, $r2, n	$r2 < nのとき$r1=1

命令セットアーキテクチャの例

MIPSアーキテクチャの命令の種類

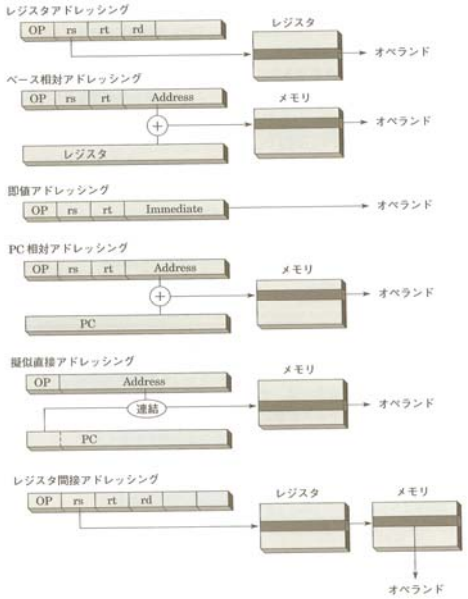
・プログラム制御命令

命令の種類	アセンブリ言語の表記	動作
条件分岐（==）	beq $r1, $r2, Label	$r1==$r2 のとき分岐
条件分岐（!=）	bne $r1, $r2, Label	$r1!=$r2 のとき分岐
無条件分岐	j Label	Label に分岐
無条件レジスタ分岐	jr $r1	$r1 の指す番地に分岐
テスト	slt $r1,$r2,$r3	$r2 < $r3 のとき $r1=1
手続き呼出し	jal Procedure	Procedure に分岐 戻り番地を $r31 に格納

条件	命令	動作
<=	slt $r1, $r3, $r2	$r3 < $r2 のとき $r1=1
	beq $r1, $zero, L1	$r1==0($r2 <= $r3) のとき L1 に分岐
<	slt $r1, $r2, $r3	$r2 < $r3 のとき $r1=1
	bne $r1, $zero, L1	$r1!=0($r2 < $r3) のとき L1 に分岐
>=	slt $r1, $r2, $r3	$r2 < $r3 のとき $r1=1
	beq $r1, $zero, L1	$r1==0($r2 >= $r3) のとき L1 に分岐
>	slt $r1, $r3, $r2	$r3 < $r2 のとき $r1=1
	bne $r1, $zero, L1	$r1!=0($r2 > $r3) のとき L1 に分岐

命令セットアーキテクチャの例

MIPSアーキテクチャの命令がサポートするアドレッシングモード



教2.4をまとめると

命令セットの設計指針

H20中間
命令セットアーキテクチャを設計する際に考慮すべき点を3つ列挙せよ。

・コンパイラが最適化しやすい命令セットを備えること
（「命令の機能を高める」ことと必ずしも等価ではない！！）

・命令セットを簡潔にまとめあげ、個々の命令の解釈・実行をハードウェアがやりやすくする

・使用頻度の高い命令を重点的に高速化・最適化する

	RISC	CISC
命令の種類	少ない	多い
アドレッシングの種類	少ない	多い
命令長	固定長	可変長
主記憶アクセス	ロードストア命令のみ可能	多くの命令で可能
レジスタ数	多い	少ない

設計指針の具体的なHW実装において 「規則性」「直交性」「プリミティブ性」などを考慮する