計算機アーキテクチャ講義
ノート１

平成19年10月１日配布
今瀬　真

質問
オフィスアワー：授業のある月曜日16：30から17：00
事前にメールまたは授業終了後に予約をとること
メール：imase@ist.osaka-u.ac.jp
http://www.ispl.jp/~imase/lecture/comp-arch/2007/

開講日：10月1日、10月15日、10月22日、10月29日、11月12日、
11月19日、11月26日、12月3日、12月10日、12月17日、
1月7日、1月21日、1月28日、1月30日
試験：　2月11日（多分、教務係りからのアナウンスに従う事）

計算機アーキテクチャ講義

講義の目的：汎用計算機の構造を理解する。

利点：

- システム設計や組み込みプログラミングを行う上での素養となる。
- 通常のプログラミングでも、厳しい性能要求や高度なデバッグなどはアーキテクチャをしらないとできない。
- オペレーティングを理解する上で、必須の知識

講義の主眼

- 計算機のメンタルイメージをつける
- アーキテクチャの基本を正しく把握することにより、新しい技術に対する理解力を養う。現状の計算機のアーキテクチャを理解するのが目的でなく、何故そうなっているかを理解できる素養を身につける。

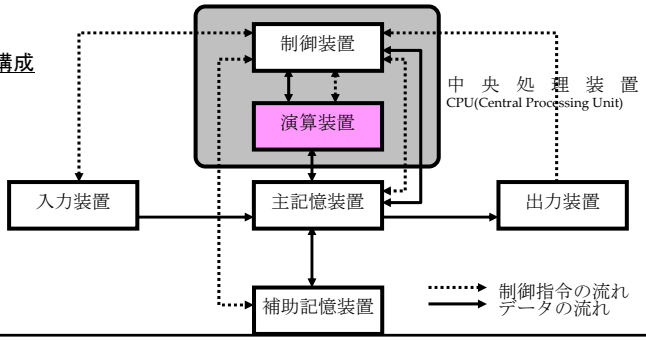
計算機アーキテクチャ講義

- 講義の進め方：下記の教科書に従って講義をすすめる。授業中に教科書を引用するので、受講の際は教科書を持参すること。
  - 「計算機アーキテクチャ」橋本昭洋著　昭晃堂 ISBN4-7856-2027-7
- 授業の内容はプリントを配布する。
- 講義の後は、教科書の該当する部分を読んで復習すること。教科書の内容をすべて説明しない。プリントで配布する部分の理解だけでよい。
- 成績：試験成績で判定（追試験は原則行わない）。病気などやむ経ない場合は実施するので、申し出ること。

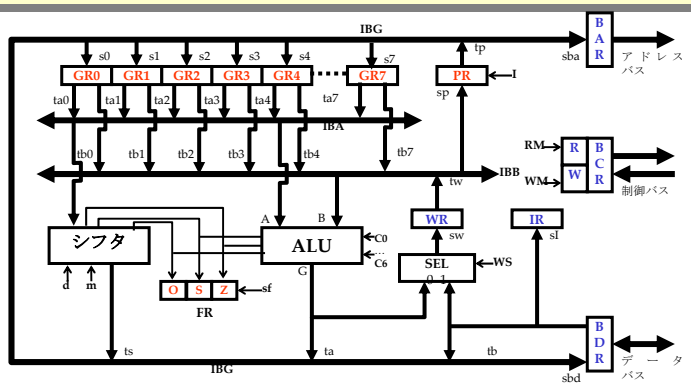
0. 計算機の構成（復習）

1. 入力装置：キーボードやマイク、カメラなど
2. 出力装置：ディスプレー、プリンターなど
3. 補助記憶装置：ハードディスク、フロッピィディスク、CD-ROMなど。
4. **主記憶装置**：メモリボード。語の1次元配列。機械語命令とデータを格納
5. **中央処理装置**：CPUボード。主記憶の機械語命令を取り出し制御装置に格納し、その指令に従い演算や主記憶装置と演算装置の間のデータのやりとりなどを実行する。

一般の計算機構成



計算機（中央処理装置）の演算装置の構成

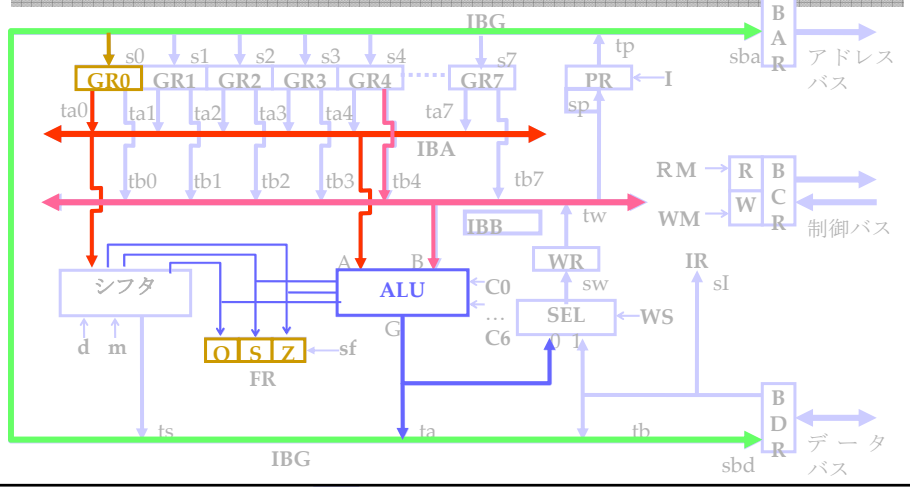


- 演算装置は、下記の要素で構成されている。
  - バス：データの通路（ビット単位の線、電圧で1，0を表現）
  - レジスタ：記憶素子（フリップフロップ（1ビット）の集合）
  - 専用回路（関数：入力に対して特定の出力をだす）
- 各要素へは、制御装置からその動作を指令する信号が送られる。

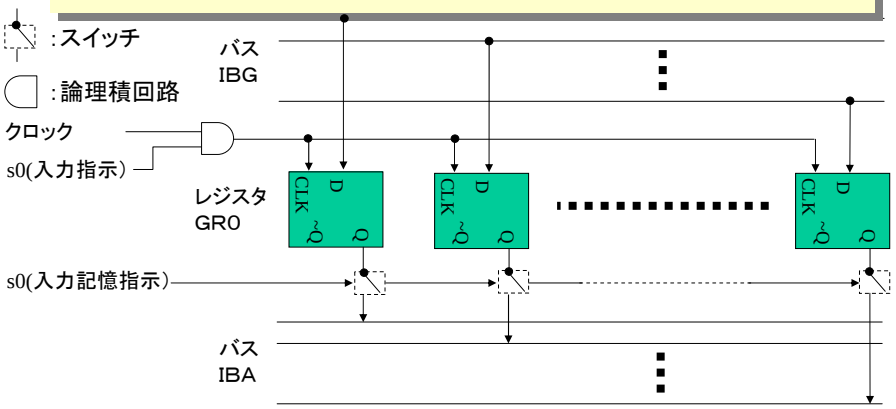
計算機の1クロックの動作例

- 制御装置から送られる制御信号により、動作する。

（例） GR0、FR←GR0＋GR4には下記の信号を1にすれば良い。　ta0　tb4　C0～C6　ta　sf s0



レジスタの構成例

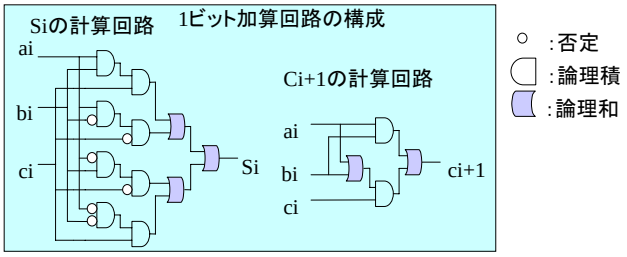


- 配布されるクロックと入力記憶指示の論理積をとることにより、指示があった時のみバスからデータをとりこむ。
- バスへの出力は、スイッチの開閉により行う。

専用回路（関数）

- ALU（算術演算回路）：AとBの信号から演算を行い結果をGに出す。（C0～C6で演算の種類を指定）
- SEL（選択回路）：2つの入力のどちらかを選択して出力する。
- シフタ（シフト回路）：入力をシフトして出力する。
  - これらは、論理回路（入力を変換して出力する関数の機能）で構成されており記憶素子は含まれていない。
  - 例えば1ビットの加算器は下記のように構成されている。

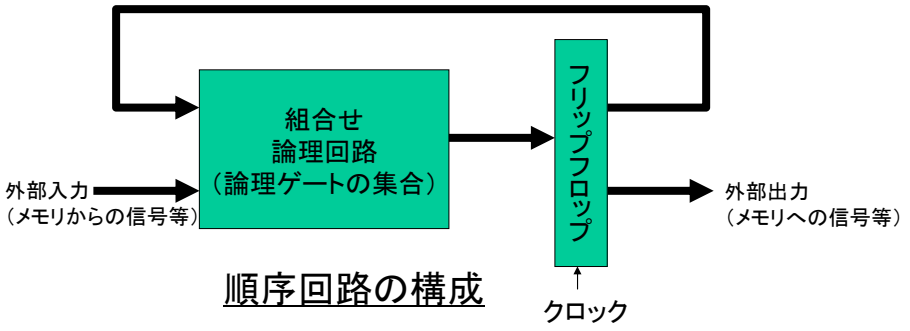
a_i	b_i	c_i	S_i	C_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1



同期式計算機のモデル

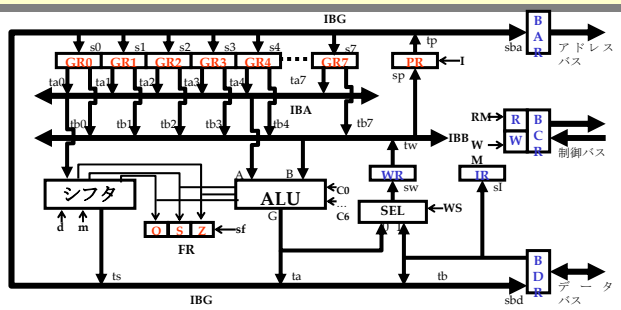
- 計算機は論理ゲートの集合（関数を実現する。一般に論理回路と呼ぶ。）とフリップフロップの集合（記憶）であり、各クロックの間に関数の計算が行われクロックの立ち上がりで状態が変更される。

  →一般に順序回路と呼ぶ



順序回路の構成

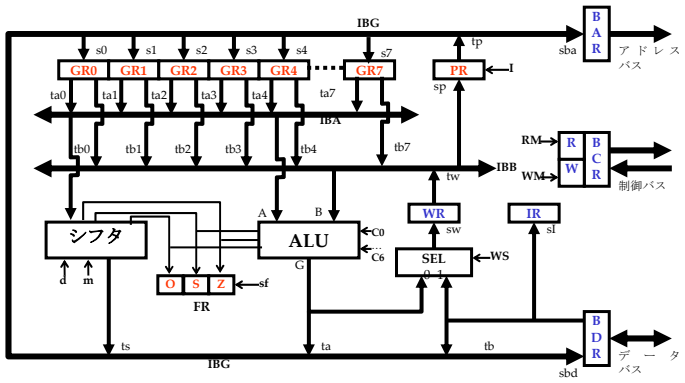
計算機の動作原理



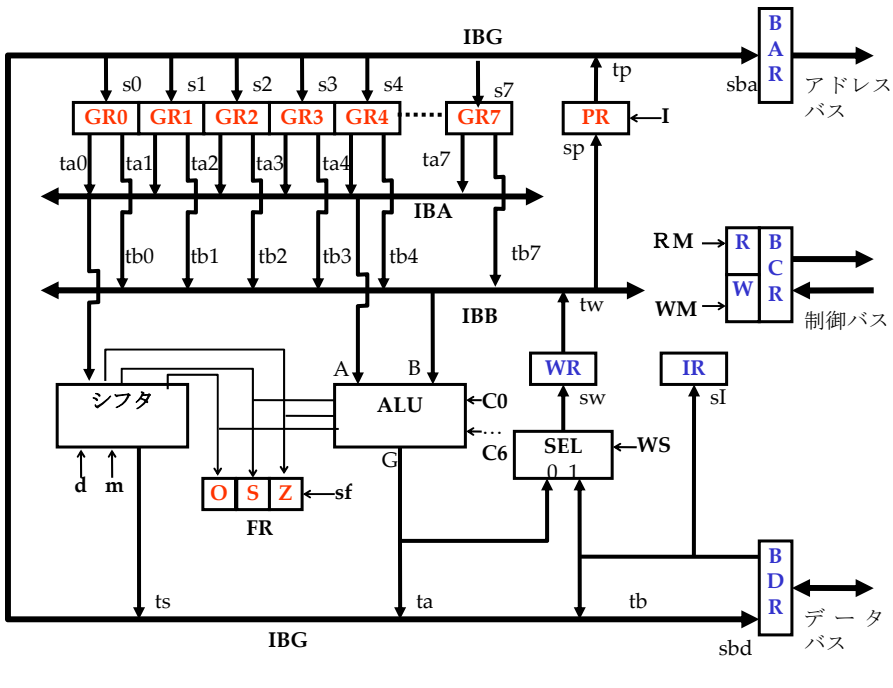
- 計算機は1クロックサイクルの間にレジスタ→バス→専用回路→バス→レジスタの動作を行い。レジスタの状態を更新する。
- 命令はこの繰り返しにより実現される。（複数のクロックサイクルで機械語1命令が実行される。

  実際の計算機は、高速化をはかるため、様々な工夫がなされている。（計算機アーキテクチャの授業で習う）。

計算機の動作原理



1命令を実行するのにいくつのクロックサイクルが必要となるか？→命令により異なる
以下でいくつかの命令の例を示す。



- 加減算の機械語命令の実行 (SUBA GR0,adr,GR1)
  - step1: BAR ← PR, PR ← PR+1, R ←1（命令の読み出し）

    `tp sba I RM` ← 上記の動作を実現するために1とする信号の集合
  - step 2: R=1| → step2,
    R=0| IR ← BDR, BAR ← PR, PR ← PR+1, R = 1
    （命令のアドレス部の読み出し ）

    `sI tp sba I RM  CLK=クロック∩R̄` ← R̄でクロックをとめて R＝1の時はNOPを実現
  - step 3: R=1| → step3,
    R=0| WR ← BDR （アドレス計算１）

    `sw WS(=1)  CLK=クロック∩R̄`
  - step 4: BAR ← WR+GR1, R←1 （アドレス計算２）

    `ta1 tw C0～C6 ta sba RM` ← C0～C6で加算を指定
  - step 5: R=1| → Phase 5, R=0| WR ← BDR

    `sw ws  CLK=クロック∩R̄` ← C0～C6で減算を指定
  - step 6: GR0・FR ← GR0 – WR

    `ta0 tw C0～C6 ta sf s0`

- ジャンプ命令やシフト命令の実行 (JUMP adr, GR3)
  - step 1: BAR ← PR, PR ← PR+1, R=1

    `tp sba I RM`
  - step 2: R=1| → 2,
    R=0| IR ← BDR, BAR ← PR, PR ← PR+1, R=1

    `sI tp sba I RM  CLK=クロック∩R̄` ← R̄でクロックをとめて R＝1の時はNOPを実現
  - step 3: R=1| → 3,
    R=0| WR ← BDR

    `WS sw  CLK=クロック∩R̄`
  - step 4: WR ← WR＋GR3

    `ta3 tw C0～C6 WS(=0) sw` ← C0～C6で加算を指定
  - step 5: PR ←WR, → 1

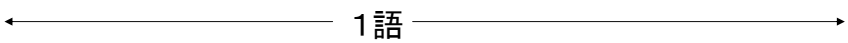
    `tw sp`

1　はじめに

- 講義では、説明しない。難しいが、講義が終了した時点で読めばわかるはず。*各自読んでおくこと。*
- 情報の人なら常識として、vonNeumann, stored program computer, ENIAC, UNIVAC, IBM360, CDC6600, Amdahl などは知っておきべき。

2．1　データ語の構成

- 語長（処理の単位）とアドレス付けとは単位が異なる。
- 語長の決定要素：1語でどれだけのデータ／命令を表現できるか、
  - 長い：CPUのコスト高、１命令の実行速度の低下、無駄なメモリ領域の増加
  - 短い：1語で表現できないデータの発生頻度が増加し、１つの演算を複数の命令で実行する必要がおこる（この場合は著しく速度が遅くなる。）



0番地	1番地	2番地	3番地
4番地	5番地	6番地	7番地
8番地	9番地	10番地	11番地

２．２　数の表現

１０進整数

- 4ビットで表現する。１６通りの表現ができるが、そのうちのどれで１０通りの数字をどう表現するかが問題。通常の計算機では提供されていない。
- BCD符号

  0→0000 1→0001 2→0010 3→0011 4→0100 5→0101

  6→0110 7→0111 8→1000 9→1001
- 3余り符号

  0→0011 1→0100 2→0101 3→0110 4→0111 5→1000

  6→1001 7→1010 8→1011 9→1100

  桁上げが通常の2進数加算器により自動的に生成

[問1] 3余り符号では、桁上げが、4ビットに通常の2進数加算器で自動的に生成されることを示せ。また、加算を行った結果を3余り符号とするためには、どのような補正が必要か？

演習（１０．２変更）

- 3余り符号系のコードで2＋$_2$3を計算する。ただし、＋$_2$単純な2進加算を意味する。

  2のビット表現は0101　3のビット表現は0110

  0101＋$_2$0110＝1011（8?）→ 1011-0011=1000(5)
- 3余り符号系のコードで5＋$_2$6を計算する。

  5のビット表現は1000　6のビット表現は1001

  1000＋$_2$1001＝10001（*?）→0001+0011=10100(1)

その桁の値：桁上がりがある場合は***3加算***、桁上がりがない場合は3減算

2．２　数の表現（正整数）

- 正整数：nビット（1語）で0～2^n-1の数を表現

  Xはnビットの系列$(x_{n-1},x_{n-2}, \ldots ,x_0)$ $x_i \in \{0,1\}$

  $(X)_2 \equiv \Sigma_{i=0\ldots n-1} [x_i \times 2^i]$

  000001→1　000010→2　……　111111→2^n-1
- $(X)_\alpha$はシステムαでのXが表現する数∈Iを表す。

  例えば、$(000010)_2 = 2$
- 逆に数a∈Iのシステムαでのnビット系列を$I_\alpha(a)$で表す。正整数表現は、下記のとおり表せる。

  $I_2(a) \equiv (x_{n-1},x_{n-2}, \ldots ,x_0)$

  such that　$a=\Sigma_{i=0\ldots n-1} x_i \times 2^i$　$x_i \in \{0,1\}$

2.2.1　数の表現（整数）

nビットで$-2^{n-1}-1$から$2^{n-1}-1$までを表現したい。

正負の整数はいくつかの表現法がある。

その基準：

- 正負の判定が容易（以下の例はすべて最上位ビットが0か1で正負が判定可能）
- 加（減）算回路の構成が簡単（加算回路と減算回路の2つを用意するのでなく一つの回路＋αで構成したい。できれば正整数の加算回路を流用したい。）

2進数の表現系(4ビットの場合)

数	8余り符号系	符号付き絶対値系	1の補数系	2の補数系
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	ーー	1000	1111	ーー
-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	1111	1001	1010
-7	0001	1111	1000	1001
-8	0000	ーー	ーー	1000

符号付絶対値系が素直な表現であるが、
ビット列でみた場合、負数は大小関係が逆転する。また、演算は他の方式に比べて場合分けが煩雑で演算回路が他の符号系に比べて複雑となる。

2.2.2 数の表現(整数)

2^{n-1}余り符号系 $(X)_{EX}$

- nビットで-2^{n-1}から$2^{n-1}-1$までを表現
- $(X=(x_{n-1},x_{n-2}, \ldots ,x_0))_{EX} \equiv (X)_2 - 2^{n-1} = \Sigma_{i=0\ldots n-1}[x_i \times 2^i] - 2^{n-1}$

  すなわち、$I_{EX}(a) \equiv I_2(a+2^{n-1})$
- 符号:正数は$x_{n-1}=1$　負数は$x_{n-1}=0$
- 加算:正整数の加算を行い、桁上げありの場合は2^{n-1}加算、桁上げなしの場合は2^{n-1}減算

以下では正整数の加算を*加算*$_2$と表現する。混同しない場合は省略することもある。

演習

2^{n-1}余り符号系 $(X)_{EX}$で下記の計算をせよ。
ただし、n=4

- ４＋３　　$1100+_2 1011 \rightarrow 10111$（桁上がりあり）
  $0111+_2 1000 \rightarrow 1111 = (7)_{EX}$
- ４＋（－２）　　$1100 +_2 0110 \rightarrow 10010$（桁上がりあり）
  $0010 +_2 1000 \rightarrow 1010 \ (2)_{EX}$
- ３＋（－５）　　$1011 +_2 0011 \rightarrow 01110$（桁上がりなし）
  $1110 -_2 1000 \rightarrow 0110 \ (-2)_{EX}$

2.2.2　数の表現（整数）

2^{n-1}余り符号系 $(X)_{EX}$の計算例

- ４＋３：$1100+_2 1011 \rightarrow 0111$（桁上がりあり）
  $0111+_2 1000 \rightarrow 1111 = (7)_{EX}$
- ４＋（－２）：$1100 +_2 0110 \rightarrow 0010$（桁上がりあり）
  $0010 +_2 1000 \rightarrow 1010 \ (2)_{EX}$
- ３＋（－５）：$1011 +_2 0011 \rightarrow 1110$（桁上がりなし）
  $1110 -_2 1000 \rightarrow 0110 \ (-2)_{EX}$

[用語]オーバフロー：演算結果が取り扱える範囲の数より大きくなる
アンダーフロー：演算結果が取り扱える範囲の数より小さくなる

[問2] 2^{n-1}余り符号系の加算演算でアンダーフローした場合(例えば4ビットの場合−7+(−6))、オーバフローした場合(例えば7+6)に、どのように検出できるか？

[問3]　2^{n-1}余り符号系 $(X)_{EX}$ の加算：正整数の加算を行い、桁上げありの場合は2^{n-1}加算、桁上げなしの場合は2^{n-1}減算で正しく計算できることを証明せよ。

2.2.2　数の表現（整数）

符号付絶対値系

- 最上位ビットで正負を表す
  0010→2　1010→−2
- 加算X+Y（一方が正で他方が負）
- XとYの絶対値を比較し、大きい方から小さい方を引き、大きい方の符号をつける。
- 7＋（−4）の例：
  0111と1100の絶対値を比較（前者が大きい）
  0111−$_2$0100→0011（大きい方−小さい方）
  0011＋$_2$0000→0011（大きい方の符号をつける）

絶対値の比較をするには、減算を行う必要があり、一つの加算減算に2回の演算が必要となるので、絶対値表現は計算機で使用されていない。

2.2.2　数の表現（整数）

1の補数系

- nビットで$-2^{n-1}+1$から$2^{n-1}-1$までを表現
- 1の補数：$(X)_2$の1の補数とは、$2^n-1-(X)_2$。すなわち、すべてのビットを反転してえられる。
- 1の補数系：負の数を1の補数表現する。
  $x_{n-1}=0$の時　$(X)_{1C}\equiv(X)_2$、
  $x_{n-1}=1$の時　$(X)_{1C}\equiv -(2^n-1)+_2 (X)_2$

すなわち $a\geqq 0$の時　$I_{1C}(a)\equiv I_2(a)$、　$a<0$の時　$I_{1C}(a)\equiv I_2(a+2^n-1)$

- 符号：正数は$x_{n-1}=0$　負数は$x_{n-1}=1$
- 加算（X＋Y）：*加算$_2$*を行い、最上位からの桁上げがあれば1加算
- 減算（X−Y）：減数のすべてのビットを反転し(Y)、X＋Yを計算

演習

- 3と−3を1の補数系で表現せよ（ただし、n=4）

  3：0011（ビットの反転）→ 1100 ：−3
  −3：1100 →（ビットの反転）0011 ：3

1の補数系で次の加算を行え（ただし、n=4）

- 3＋(−2)　0011+$_2$1101 →10000（桁上げあり）→0001：1
- 3＋2　0011+$_2$0010 →00101（桁上げなし）→0101：5
- −3＋(−2)　1100+$_2$1101 →11001（桁上げあり）→1010：-5
- −3＋2　1100 +$_2$0010 →01110（桁上げなし）→1110：-1

2.2.2　数の表現（整数）

1の補数系演算の例

- 補数化　3：0011（ビットの反転）→ 1100 ：−3
  −3：1100 →（ビットの反転）0011 ：3
- 加算例
  - 3＋(−2)：0011+$_2$1101 →0000（桁上げあり）→0001：1
  - 3＋2：0011+$_2$0010 →0101（桁上げなし）→0101：5
  - −3＋(−2)：1100+$_2$1101 →1001（桁上げあり）→1010：−5
  - −3＋2：1100 +$_2$0010 →1110（桁上げなし）→1110：−1

[問4] 1の補数系演算でアンダーフローした場合(例えば4ビットの場合−7+(−6))、オーバフローした場合(例えば7+6)に、どのように検出できるか？

[問5]1の補数系の加算が正しい結果をだすことを証明せよ。

2.2.2　数の表現（整数）

2の補数系

- nビットで-2^{n-1}から$2^{n-1}-1$までを表現
- 2の補数：$(X)_2$の2の補数とは、$2^n-(X)_2$。すなわち、すべてのビットを反転し1を*加算*$_2$してえられるX＋１。
- 2の補数系：負の数を2の補数表現する。

$x_{n-1}=0$の時　$(X)_{2C}\equiv(X)_2$、$x_{n-1}=1$の時　$(X)_{2C}\equiv(X)_2-2^n$

すなわち　$a\geqq 0$の時　$I_{2C}(a)\equiv I_2(a)$、　$a<0$の時　$I_{2C}(a)\equiv I_2(a+2^n)$

- 符号：正数は$x_{n-1}=0$　負数は$x_{n-1}=1$
- 加算：*加算*$_2$を行う。
- 減算（X－Y）：減数のすべてのビットを反転し(Y)、X＋Y＋１を計算する

演習

- 3と－3を2の補数系で表現せよ（ただし、n=4）

3：0011（ビットの反転）→ 1100（1加算）→1101：－３

－３：1101 →（ビットの反転）0010 →（1加算）0011：３

2の補数系で次の加算を行え（ただし、n=4）

- ３＋(−2)　　0011+$_2$1110 →0001：１
- ３＋２　　0011+$_2$0010 →0101 ：５
- －３＋(−2)　　1101+$_2$1110 →1011 :-5
- －３＋ ２　　1101 +$_2$ 0010 →1111:-1

2.2.2 数の表現(整数)

2の補数系演算の例

- 補数化 3:0011(ビットの反転)→ 1100 (1加算) →1101:－3
  －3:1101 →(ビットの反転) 0010 →(1加算) 0011:3
- 加算例
  - 3＋(−2):0011+$_2$1110 →0001:1
  - 3＋2:0011+$_2$0010 →0101 :5
  - －3＋(−2):1101+$_2$1110 →1011 :−5
  - －3＋ 2: 1101 +$_2$ 0010 →1111:−1

[問6] 2の補数系の加算演算でアンダーフローした場合(例えば4ビットの場合−7+(−6))、オーバフローした場合(例えば7+6)に、どのように検出できるか？

[問7]2の補数系の加算演算が正しく計算できていることを証明せよ。

2.2.2 数の表現(シフト)

シフト

- 論理シフト:nビットのビット系列を指定されたビット数だけ左右にシフト。空いた部分には0がはいる。
- 算術シフト:1ビット左シフトが2倍(kビット左シフトが2^k倍)、右シフトが1/2倍(kビット右シフトが$1/2^k$倍)となるようにシフト。
- **2の補数系の算術シフト** 符号ビットを残し、左シフトは下位14ビットを左にシフトし最下ビットに0を挿入。右シフトは下位15ビットを右にシフトし、最上位ビットに符号ビットと同じ値を挿入。(n＝16の時)

【問8】 1の補数系に場合、算術シフトの条件を示せ。

演習

- 次の5ビットの2の補数表現について算術シフトを行え
- 00101を左1ビットシフト（2倍）

  00101→01010
- 00101を右1ビットシフト（1/2倍）

  00101→00010
- 11101を左1ビットシフト（2倍）

  11101→11010
- 11101を右1ビットシフト（1/2倍）

  11101→11110

2.2.2 数の表現（シフト）

「算術シフトとなっていることの証明」

$X=(x_{n-1},x_{n-2},\ldots,x_0)$：2の補数系の数、

その左シフトを$X_L=(x_{n-1},x_{n-3},\ldots,x_0,x_L)$、右シフトの数を$X_R=(x_{n-1},x_R,x_{n-2},\ldots,x_0)$とする。

- $(X)_{2C}=-x_{n-1}\times 2^{n-1}+\Sigma_{i=0\ldots n-2}[x_i\times 2^i]$より、$(X_L)_{2C}=2\times(X)_{2C}$が成立する条件は、

  $-x_{n-1}\times 2^{n-1}+\Sigma_{i=0\ldots n-3}[x_i\times 2^{i+1}]+x_L=2\times\{-x_{n-1}\times 2^{n-1}+\Sigma_{i=0\ldots n-2}[x_i\times 2^i]\}$

  $(x_{n-1}-x_{n-2})\times 2^{n-1}+x_L=0$
- $n\geqq 2$より、$x_{n-1}=x_{n-2}$かつ$x_L=0$が成立する必要がある。これより、$x_{n-1}\neq x_{n-2}$の時は表現できないこと、および$x_L=0$となる。
- $(X_L)_{2C}=1/2\times(X)_{2C}$が成立する条件についても、同様に$x_{n-1}=x_R$かつ$x_0=0$が得られる。

2.2.2　数の表現（符号拡張とアドレス計算）

- **符号の拡張とアドレス計算**：
  2の補数系のnを拡大するためには、符号ビットと同じ値を上位ビットに付け足せばよい。

例1：0011（3）→00000011

例2：1010（−6）→11111010

[問9]（10110101）$_2$ ＝181に（1110）$_2$ ＝−2を加算せよ。

2.2.2　実数の表現（浮動小数点数）

浮動小数点による表現と誤差

- 6．02×10^{23}のような表現（6．02を仮数、23を指数、10を指数の底という。
- 一般には、下記のように表せる。計算機ではp 、e_{min} 、e_{max}の値は固定化していること（表現できる数が有限個）に注意。
  - $\pm d_0.d_1d_2d_3\ldots d_{p-1} \times \beta^e$　ただし、$0 \leqq d_i < \beta$　$e_{min} \leqq e \leqq e_{max}$
  - この実数値は、$\pm(d_0+d_1*\beta^{-1}+d_2*\beta^{-2}+\ +d_{p-1}*\beta^{-p+1})*\beta^e$
    →誤差の問題が主課題
- $d_0 \neq 0$の時、正規化されているという。
  - 正規化表現に限れば、一つの数の表現は一意に定まる。
  - ある値のeに対してに対して、$1\times\beta^e$以上$1\times\beta^{e+1}$より小さい実数の範囲で等間隔のβ^pの数が表現できる。実数をもっとも近い浮動小数点表現で近似する。

演習

次の浮動少数点表現を10進実数で表現せよ。

- 2.102×3^2 （$\beta=3$　e＝2　p＝4）

2.102×9
＝（　2　＋1・(1/3)＋0・(1/9)＋2・(1/27)）×9
＝　2・9　＋1・3　　＋0・1　　＋2・(1/3)
＝21＋2/3＝21.6666…

- 1.1101×2^1（$\beta=2$　e＝1　p＝5）

1.1101×2
＝（1＋1*(1/2)　＋1*(1/4)＋0*(1/8)＋1*(1/16)）×2
＝　2＋　1　　＋1*(1/2)＋　0　　＋　1*(1/8)
＝3＋5/8＝3.625

2.2.2　数の表現（浮動小数点数）

- 表現できる数の間隔はβ^{e+1-p}となり、
- これをulp(units in the last place)とよぶ。

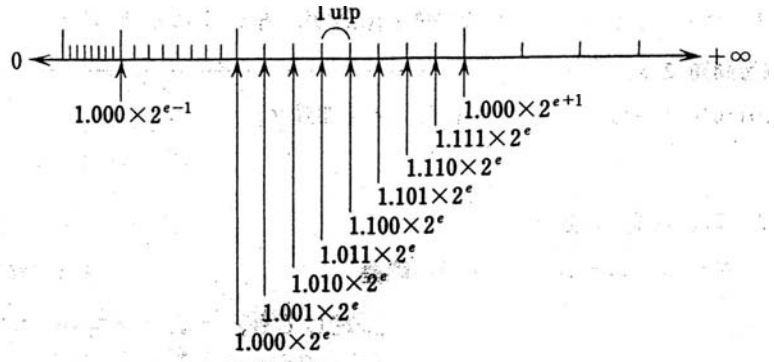


図 2.1　浮動小数点数で表せる数（$\beta=2$，$p=4$ の場合）

演習

次の浮動少数点表現のULPとその表す数の範囲を示せ。

- 2.102×3^{2}（$\beta=3$　$e=2$　$p=4$）

  ＵＬＰ＝（２．１０２－２．１０１）×９
  ＝０．００１×９＝　１×(1/27)×９＋＝１／３
  数の範囲:2.102×9−(1/3)･(1/2)＝21.5以上
  2.102×9＋(1/3)･(1/2)＝21.8333…より小さい

- 1.1101×2^{1}（$\beta=2$　$e=1$　$p=5$）

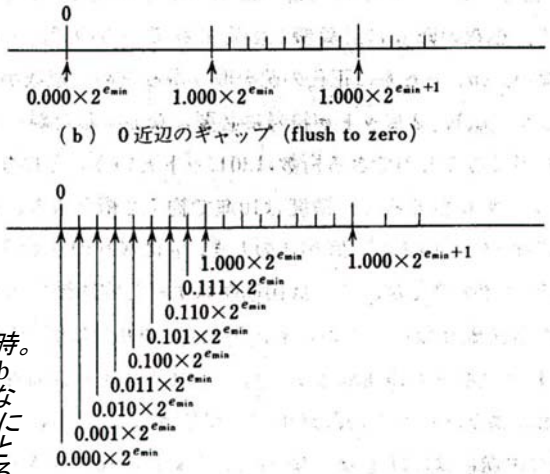
  ＵＬＰ＝（１．１１０１－１．１１００）×２
  ＝０．０００１×２＝　１×(1/16)×２＋＝１／８
  数の範囲：1.1101×2−(1/8)･(1/2)＝3.5625以上
  1.1101×2−(1/8)･(1/2)＝3.6875より小さい

2.2.2　数の表現（浮動小数点数）

- 絶対誤差＝$0.5\times\beta^{e+1-p}$
  - （元の実数値と表現された数の差の最大値）
- 相対誤差＝ $0.5\times\beta^{e+1-p}$／（元の実数値）
  ＜$0.5\times\beta^{e+1-p}$／$(1.000\ldots)\times\beta^{e}$
  ＝$(\beta/2)\times\beta^{-p}$（マシンエプシロン）
  - （元の実数値と表現された数の差の最大値）／（元の実数値）
  - β^{e} ＜（元の実数値）＜ β^{e+1}
- 表せる最小値はβ^{emin}、0が表せない。$e=e_{min}$の場合だけ正規化でない表現を許す（gradual underflow）ただし、相対誤差は0に近づくにつれて大きくなる。
- 【問10】 $\beta=10$、$p=3$の時、真の値12.35を1.24×10^{1}で表した時、絶対誤差はいくらか？また相対誤差はいくらか？

2.2.2 数の表現(浮動小数点数)

gradual underflow
e=eminの場合だけ正規化でない表現を許す
これでも相対誤差は0に近づくにつれて急速に大きくなる。



(b) 0近辺のギャップ (flush to zero)

(c) 緩やかなアンダフロー (gradual underflow)

図 2.2 浮動小数点数の値域とアンダフロー

•知見:bが非常に大きな数の時。a ÷b ×cとするよりは、a ×c ÷bとする方が相対誤差が小さくなる。浮動小数点表現はマシンにより異なり(emin、emax 、pのとり方)、同じプログラムを異なる計算機で実行した場合に、異なる結果がでることがある。

2.2.2 数の表現(浮動小数点数)

IBM System360の浮動小数点形式

- β =16、d_0=0、d_1≠0、p=7、e_{min}=−64、e_{max}=63を32ビットで表現。

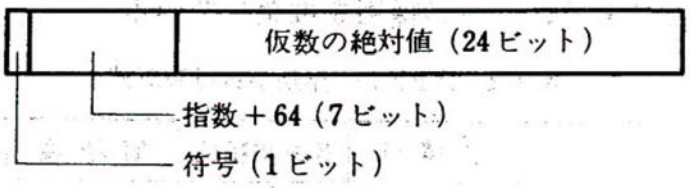


図 2.3 System 360 の浮動小数点数形式 (単精度)
仮数部は16進 6 桁で，小数点の位置は最上位桁の左側.
指数部は64余り符号で−64〜＋63の範囲の値を表せる.

2.2.2 数の表現(浮動小数点数)

IBM System360の浮動小数点形式

- 特徴:2つの数の絶対値の大小関係を、32ビット中の符号を除く31ビットを整数とみて比較することにより可能。
  - 本文では、βとして2をとるか16をとるかの利害得失について述べている。自習しておくように。

[問11] β=2,p=24とβ=16,p=6の場合についてマシンエプシロンを求めよ。

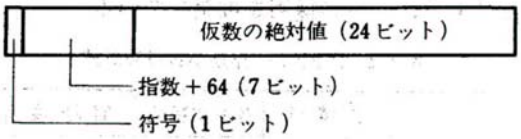


図 2.3 System 360 の浮動小数点数形式（単精度）
仮数部は16進6桁で，小数点の位置は最上位桁の左側．
指数部は64余り符号で−64〜+63の範囲の値を表せる．

2.2.2 数の表現(浮動小数点数)

IEEE標準の浮動小数点形式

- 単精度:β＝2、d_0＝1、p＝24、e_{min}＝−126、e_{max}＝127を32ビットで表現。
- 正規化浮動少数点のみを対象とする
  - 仮数部の精度は24ビットであるが23ビットで表現可能
- $|e_{min}| < e_{max}$
  - 小さい値2^{emin}の逆数がオーバフローしない
- 符号:1ビット＋指数:8ビット＋仮数部23ビット＝32ビット
- 指数は127余り符号で、−127と128は特殊な値を定義(表2．3)

2.2.2 数の表現(浮動小数点数)

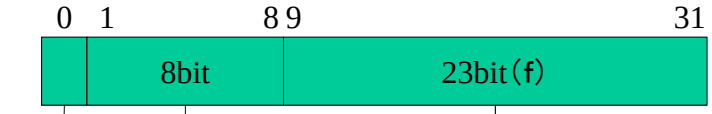
IEEE標準の浮動小数点形式 :特殊な表現を使う

表 2.3 IEEE 754 における特殊値の表現

指 数 部	仮 数 部	意 味
$e = e_{min} - 1$	$f = 0$	± 0
$e = e_{min} - 1$	$f \neq 0$	$0.f \times 2^{e_{min}}$
$e_{min} \leqq e \leqq e_{max}$	f	$1.f \times 2^{e}$
$e = e_{max} + 1$	$f = 0$	$\pm\infty$
$e = e_{max} + 1$	$f \neq 0$	NaN

2.2.2 数の表現(浮動小数点数)

IEEE標準の浮動小数点形式



符号部

仮数部:2進表現、23ビットであるが正規化浮動小数点を利用するので24ビットの精度がある(1.100011とするとfは100011となり最初の1は省略されている)

指数部 −127:±0の表現とgradual underflowの表現
−126〜127:指数を表現
128: ±∞とNaN(Not A Number)の表現

2.2.2 数の表現（浮動小数点数）

IEEE標準の浮動小数点形式

- NaN（表現不能の数）： 0÷0や√−1を求めることがあっても、処理を中断せずにすむ。
- ±∞： 0でない数を±0で割った場合に返される。
- ±0： ＋0＝－0の判定は成立と規定できる。1/x＝1/yが成立してもx＝yは必ずしも成立しない。1/（1/X）＝Xが成立。
- 0.f×2^{emin}： 0と1.0×2^{emin}の間の数について、1.f×2^{emin}と同じULPで表現

表 2.3 IEEE 754 における特殊値の表現

指 数 部	仮 数 部	意 味
$e = e_{min} - 1$	$f = 0$	±0
$e = e_{min} - 1$	$f \neq 0$	$0.f \times 2^{e_{min}}$
$e_{min} \leq e \leq e_{max}$	f	$1.f \times 2^{e}$
$e = e_{max} + 1$	$f = 0$	±∞
$e = e_{max} + 1$	$f \neq 0$	NaN

2.2.2 数の表現（浮動小数点数）

IEEE標準の浮動小数点形式

表 2.4 NaN を生じる演算（ただし REM は剰余を求める演算子）

演 算	例
＋	$\infty + (-\infty)$
×	$0 \times \infty$
／	$0 / 0$, ∞ / ∞
REM	xREM 0, ∞REMy
$\sqrt{\ }$	$\sqrt{x}$, $x < 0$

2.2.2　数の表現（浮動小数点数）

保護桁

- x－yの計算：桁合わせして計算。桁合わせして単純に有効桁数で引き算をすると誤差が大きくなる
  - 例：$x=1.00\times10^{0}$　$y=9.99\times10^{-1}$の場合
    - $1.00\times10^{0}-0.999\times10^{0}=0.001\times10^{0}=1.00\times10^{-3}$

      が正解なのに、有効桁数で計算すると
    - $1.00\times10^{0}-0.99\times10^{0}=0.01\times10^{0}=1.00\times10^{-2}$
      - 相対誤差は9（$1.00\times10^{-2}-1.00\times10^{-3}$）／ 1.00×10^{-3}）
- 保護桁：有効桁数を多くしたままで計算する。

2.2.2　数の表現（浮動小数点数）

保護桁

- 計算時の有効桁数をどれだけとれば良いか？

（例）

$x=1.00112\times10^{0}$，$y=1.11599\times10^{-3}$（$\beta=10$，$p=6$，したがって $\varepsilon=0.000005$）とする．

	（全桁の計算）	（1桁の保護桁）
	1.00112×10^{0}	1.00112×10^{0}
	-0.00111599×10^{0}	-0.001115×10^{0}
	1.00000401×10^{0}	1.000005×10^{0}
丸めた結果	1.00000×10^{0}	1.00001×10^{0}

相対誤差$=0.00000599<1.2\varepsilon$.

2.2.2　数の表現（浮動小数点数）

精度pの正規化された数$F_1=m_1\times 2^{e1}$ $F_2=m_2\times 2^{e2}$の差を求める場合、厳密な（偶数への）丸め計算を行うには、３ビットの保護桁があればよい。$F_1 > F_2$とする。

「証明」

1. $e_1-e_2=0$または1の時、桁合わせでシフトするのは高々1ビットであり、1桁の保護桁で厳密な丸め計算が行える。
2. $e_1-e_2\geqq 2$の時、差を求めた結果の左端2ビットがともに0とういことはない。従って、減算後の正規化のための左シフトは高々1ビットである。減算でp＋1桁が正しく求められれば、厳密な丸め計算を行える。
3. p＋1桁を正しく得るためには、p＋2桁を正しく求めればよい。
4. 大きい方の仮数部のp＋1桁目以降はすべて0であるから、差をp＋2桁まで正しく求めるには、以下のp＋3桁の演算を行えばよい。

2.2.2　数の表現（浮動小数点数）

精度pの正規化された数$F_1=m_1\times 2^{e1}$ $F_2=m_2\times 2^{e2}$の差を求める場合、厳密な（偶数への）丸め計算を行うには、３ビットの保護桁があればよい。$F_1 > F_2$とする。

「証明つづき」

従って、下記の手順で厳密な演算が行える。

1. 小さい方の数F_2をe_1-e_2ビット右にシフト
2. F_2のp＋3桁目以降がすべて0の時、p＋3桁目を0としてp＋3桁の減算
3. p＋3桁目以降に1が存在の時、p＋3桁目を1としてp＋3桁の減算
4. p＋2桁目を丸める。

2.2.2　数の表現（浮動小数点数）

（例）

$e_1 - e_2 = 5$ とする．

（全桁による計算）

GR

```
  1.000001100
 −0.00001011010001  （末尾の 3 ビットの OR をとる）
  0.11111010101111
+              1     丸めの処理
  0.1111101011
  1.111101011       正規化
```

（3 桁の保護桁による計算）

GRS

```
  1.000001100
 −0.000010110101 ←
  0.111110101011
+              1     丸めの処理
  0.1111101011
  1.111101011       正規化
```

2．3　命令の構成

CISC(complex instruction set computer)

- 200以上の命令
- ねらい：命令の機能を高度化し、実行命令数を減らすことにより高性能化を実現する

RISC(reduced instruction set computer)

- 50～100命令
- ねらい：計算機の構造を単純化し、1命令のサイクル数とサイクル時間を削減して高性能化をはかる。
- 1980年にPattersonらが提案
- CPUのシングルチップ化を容易化

2.3.1 命令とオペランド

- オペランド：被演算子　オペレータ：演算子
  - A←B■C
  - A：演算結果の代入場所（デスティネーション）
  - ■：演算子
  - B，C：演算対象(ソース)
  - A、B、C：主記憶か中央処理装置内（アキュムレータ/レジスタ）

2.3.1 命令とオペランド

- アキュムレータ型計算機（計算機の原型）

A←B■C
A＝B＝アキュムレータ
C：主記憶内

a+b→c の場合
load b
add a
store c

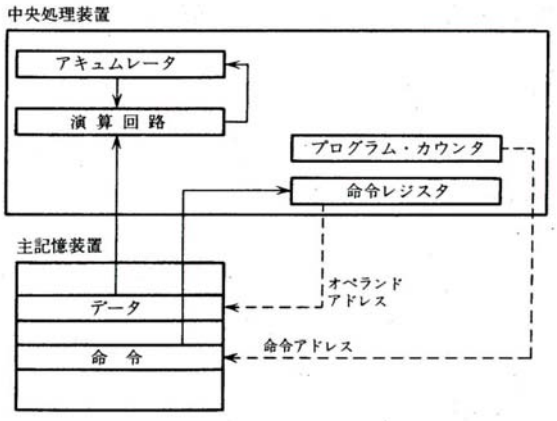


図 2.4 アキュムレータ型計算機

2.3.1 命令とオペランド

- スタック型計算機（現存しない）

A←B■C
スタックの先頭をB、C（ポップして利用）し、結果Aをスタックにプッシュ

a+b→c の場合
```
push  b
push  a
add
pop c
```

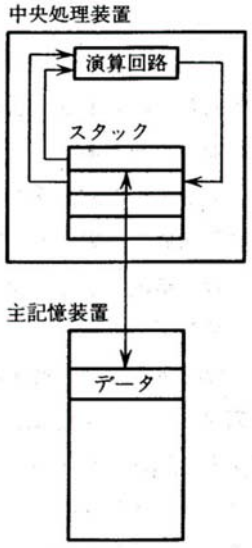


図 2.6　スタック型計算機

2.3.1 命令とオペランド

- レジスタ型計算機（現在の計算機の基本形）

A←B■C
A,B,C:レジスタや主記憶

a+b→c の場合
```
add   r1,r2,c/
add   r1,r2,r3/
add (r1),r2,(r3)/
```
など

LSIの進歩により、CPUの素子数を少なくする必要がなくなり、アキュムレータ型計算機やスタック型計算機は現状では使われていない。

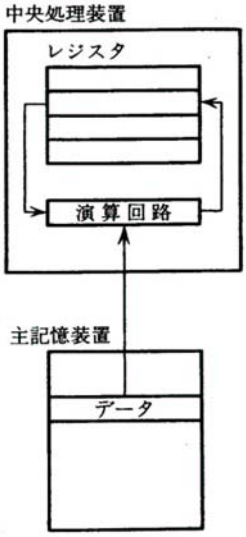


図 2.5　レジスタ型計算機

2.3.2　命令の種類

- 転送命令：レジスタ間、レジスタ・主記憶間、主記憶・主記憶間（主として可変長データ）
- 算術演算命令：
  - データ形式：10進/2進　正数/整数　浮動小数点（単精度、倍精度、4倍精度）
  - 演算：加減乗除
  - 条件コード：正、負、零、オーバフロー
- 論理演算：AND、OR　EOR（ビット毎）
- シフト命令：算術シフト（2倍/4倍など）、論理シフト
- 比較命令：条件コードの設定
- 分岐命令：条件分岐、無条件分岐、サブルーチンコール
- その他：入出力命令、システム制御

2.3.2　オペランドの指定

レジスタか主記憶かがあり、主記憶の場合は次のように分類できる。

- （場合１）主記憶の内容をオペランドで指定する場合
  - 直接アドレスと間接アドレス
  - 絶対アドレスと相対アドレス
  - インデックス修飾
- （**場合２**）命令に値そのものが含まれいる
  - 即値形式（命令に含まれている値そのものがオペランド）

主記憶アクセスの削減（レジスタ多用と命令コード長削減）による処理速度向上と主記憶量の削減。

2.3.2 オペランドの指定

- 直接アドレス
- 間接アドレス
  - レジスタ間接
  - メモリ間接

間接アドレスは、様々な局面で便利である。サブルーチンコールの引数受け渡しを考えた場合、その便利さは理解できる。

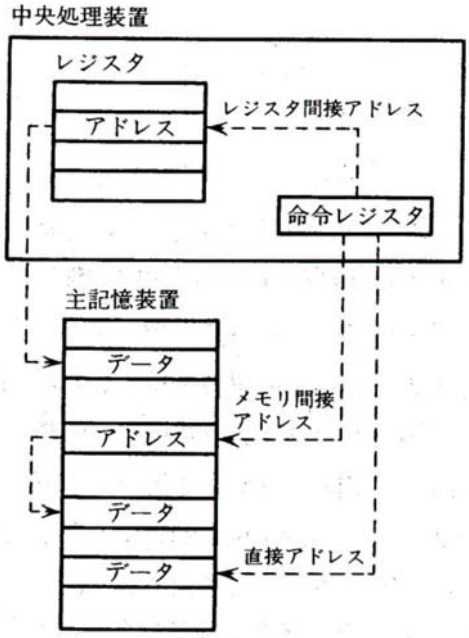


図 2.7 直接アドレスと間接アドレス

2.3.2 オペランドの指定

- 絶対アドレス
- 相対アドレス
  - レジスタ相対アドレス
  - PC相対アドレス

相対アドレスの必要性:
•リロケータブルプログラムの作成(この際のベースアドレスを格納するレジスタを**ベースレジスタ**と呼ぶ)
•メモリアドレス指定部(のメモリ量)の節約、
•大規模メモリの搭載(1語で表現できる空間以上)、

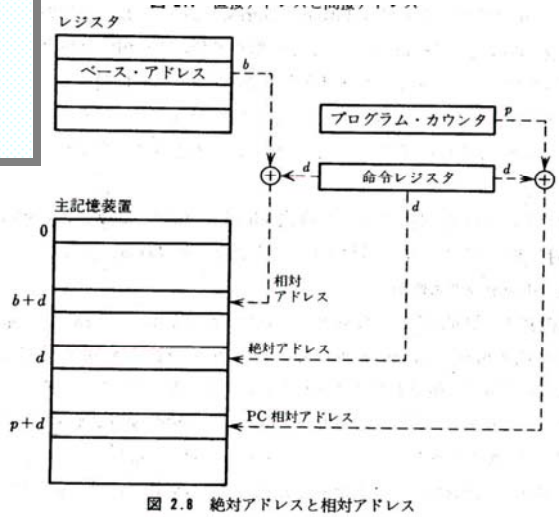
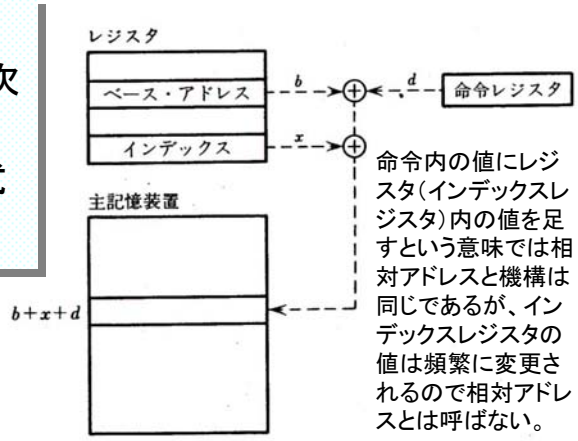


図 2.8 絶対アドレスと相対アドレス

2.3.2 オペランドの指定

- インデックス修飾
  - 配列データの順次参照
  - 他のアドレス形式と併用される



命令内の値にレジスタ(インデックスレジスタ)内の値を足すという意味では相対アドレスと機構は同じであるが、インデックスレジスタの値は頻繁に変更されるので相対アドレスとは呼ばない。

図 2.9 インデックス修飾（相対アドレスにインデックス修飾を行った場合）

配列内のデータ
A(i)を表現する場合
•ベースアドレス:データ領域の先頭アドレス
•命令レジスタ:データ領域でのAの先頭番地
•インデックス:配列の何番目か(i)

2.3.2 オペランドの指定

主記憶アドレスと語境界

- ワードアドレス／バイトアドレス （バイトアドレスが主流）

バイトアドレスの懸案事項1:語境界があるか

- System370、VAX11は語境界がないが、RISCは語境界がある。（RISCの方が新しいアーキテクチャであるのに、語境界があることに注意。）

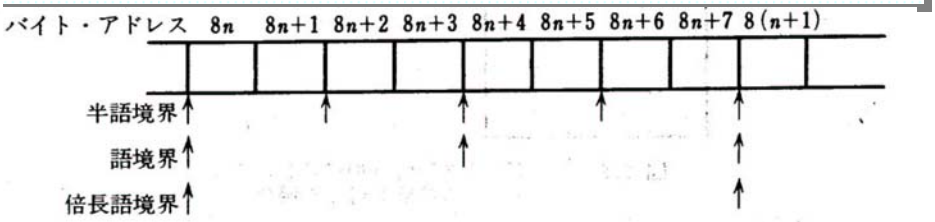


図 2.10 語境界

2.3.2 オペランドの指定

バイトアドレスの懸案事項2：ビッグ／リトルエンディアンのどちらか？（プログラムの移植に問題）

- System370はビッグ／リトルエンディアン、VAX11はビッグ／リトルエンディアン語境界がないが、RISCは両者に対応

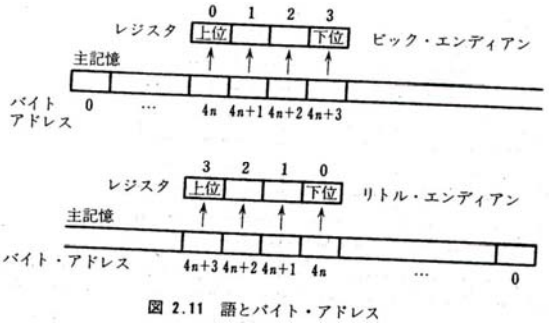


図 2.11 語とバイト・アドレス

2.3.4 ＩＢＭ　System370の命令体系

32ビットマシン。バイトアドレス。

- 命令長：16ビット、32ビット、48ビット。
- レジスタ指定は4ビット：汎用レジスタ16個と浮動小数点レジスタ16個の区別は命令コードで判断。インデックスレジスタとしてレジスタ0が指定された場合は特別な意味（インデックス修飾を行わない）をもつ場合がある。
- **ベースレジスタ**：主記憶はベースレジスタ(b1/b2)とベースレジスタからの偏移(displacement d1/d2)の和でアドレスを指定。
- オペランド数：オペランドの最後の数字がオペランドの種別を表現。2オペランドが主。
- 可変長データの処理：RS形式でのレジスタセーブ命令、文字や10進数処理のSS形式命令。

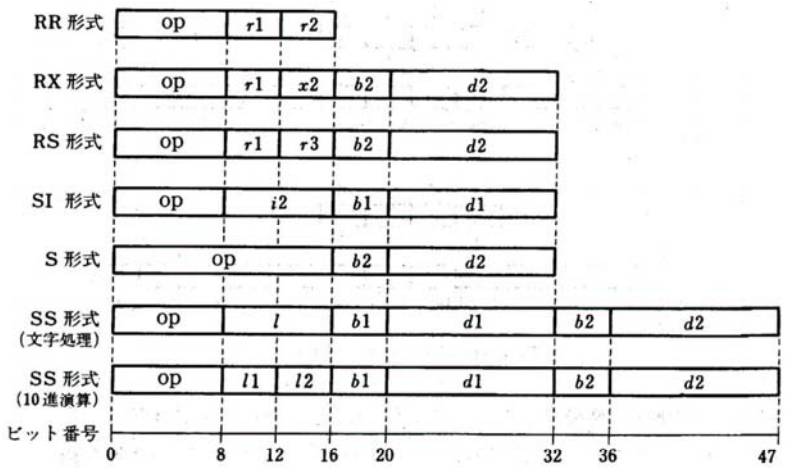


図 2.12 System 370 の命令語の形式

op ：操作符号
r ：汎用レジスタまたは浮動小数点レジスタを指定（オペランド）
x ：汎用レジスタを指定（インデックス・レジスタ）
b ：汎用レジスタを指定（ベース・レジスタ）
d ：ベース・アドレスからの偏位
i ：イミディエイト・オペランド
l ：可変長オペランドの長さを指定

2.3.4 DEC VAX11の命令体系

- 32ビットマシンであるが、16ビットを語(word)、32ビットを長語(long word)と呼んでいる。バイトマシン。
- レジスタ指定が8ビット：上位4ビットでモード（後述）を指定し、下位4ビットで16個のレジスタのどれかを指定
  - R0～R11：汎用レジスタ、R12,R13：手続き呼び出し用のスタックポインター 、R14：作業用スタックポインター、R15：プログラムカウンター
- 2進数で長さの異なる5つのデータ形式、浮動少数点で4形式など14種類のデータ形式。
- 命令数300
- オペランド数は0～3。一つのオペランドを1か2のオペランド指定子で指定。1オペランドに1～5バイト。
- プログラムカウンターは1バイト読むたびに加算されるのでR15を利用する場合は注意が必要。

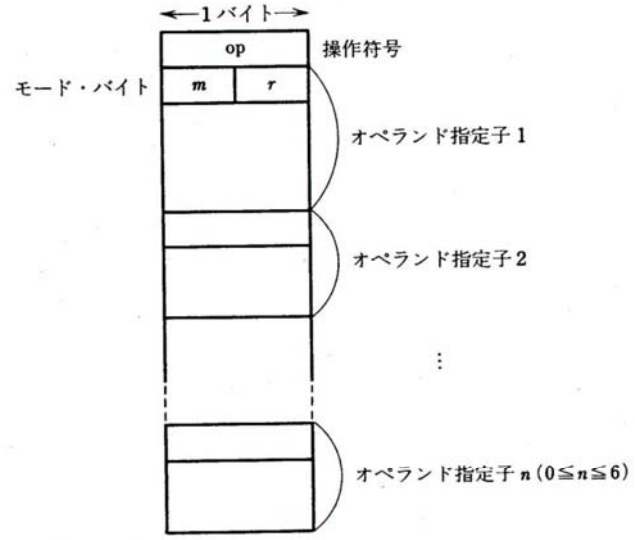


図 2.13 VAX 11の命令語の形式
命令語は可変長であり，最大6個のオペランドを指定する．
オペランド指定子の最初の1バイトはモード・バイトであり，
mでモード番号，rでレジスタ番号を指定する．
モード・バイトの後に，アドレスの偏位や即値オペランドが
続く．

2.3.4 DEC VAX11の命令体系（モード）

- 上位4ビットがモードmを指定し、下位4ビットがレジスタ名rを指定。
- モードにより1オペランドに必要となるバイト数が異なる。
- アセンブラ記法での記号
  - #x:xの値そのもの (x):xの内容 −:kをひく ＋:kを足す
  - @:間接アドレス
  - D(Rn):レジスタ相対、 A:アドレス（その番地を指すラベル）
  - S↑ #x:xの6ビット表現 B↑#x:xの1バイト表現 W↑#x:xの2バイト表現（語） L↑#x:xの4バイト表現（長語）
- 有効アドレス部での記号
  - EA:有効アドレス [x]:xの内容 Rn:rで指定されたレジスタ
  - k:オペランドの長さ（対象となるデータ長）で、1，2，4バイトがある。操作符号部opで指定されたオペレーション（演算）によって決まる。
  - A:4バイトの絶対アドレス

2.3.4　DEC VAX11の命令体系（モード）

	上位4ビット	アセンブラ記法	有効アドレス	オペランド指定子の長さ
1.	00xx	S↑#V	V	1byte
2.	0100	(4～13)[Ri]	EA=(4～13)+k[Ri]	1byte
3.	0101	Rn	EA=Rn	1byte
4.	0110	(Rn)	EA=[Rn]	1byte
5.	0111	-(Rn)	Rn←[Rn]-k　EA=[Rn]	1byte
6.	1000	(Rn)+	EA=[Rn] Rn←[Rn]+k	1byte
7.	1001	@(Rn)+	EA=[[Rn]] Rn ←[Rn]+4	1byte
8.	1010	B↑#D(Rn)	EA=D+[Rn]	2byte
9.	1100	W↑#D(Rn)	EA=D+[Rn]	3byte
10.	1110	L↑#D(Rn)	EA=D+[Rn]	5byte
11.	1011	@B↑#D(Rn)	EA=D+[Rn]	2byte
12.	1101	@ W↑#D(Rn)	EA=D+[Rn]	3byte
13.	1111	@ L↑#D(Rn)	EA=D+[Rn]	5byte

レジスタとしてプログラムカウンタを指定した場合（r=1111）

	上位4ビット	アセンブラ記法	有効アドレス	オペランド長
1.	00xx			
2.	0100			
3.	0101			
4.	0110			
5.	0111			
6.	1000	I↑#V	EA=[PC] PC←[PC]+k	1+k byte
7.	1001	@#A	EA=[[PC]] PC ←[PC]+4	5byte
8.	1010	B↑A	EA=D+[PC]=A	2byte
9.	1100	W↑A	EA=D+[PC]=A	3byte
10.	1110	L↑A	EA=D+[PC]=A	5byte
11.	1011	@B↑A	EA=D+[PC]=[A]	2byte
12.	1101	@ W↑A	EA=D+[PC]=[A]	3byte
13.	1111	@ L↑A	EA=D+[PC] =[A]	5byte

注：Dの値はアセンブラが計算する(D=A-[PC]

2.3.4　DEC VAX11の命令体系

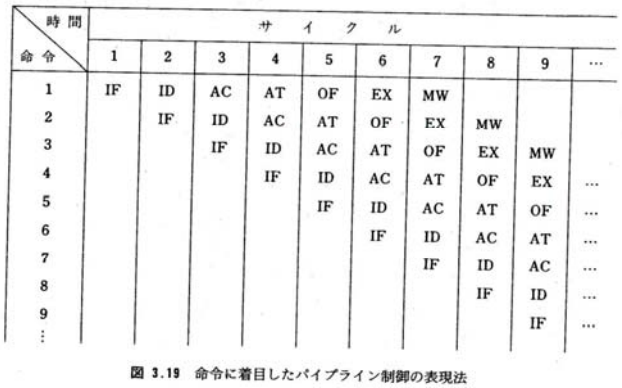
アセンブラでプログラミングをする人にとっては、

- 非常に便利（ソフトウェアの生産性が高い）
- 命令体系がシンプル
- 密度の高い（命令数の少ない）プログラムが書ける。
- スタックを有効に活用でき、スタックマシンともいえる。（計算機の完成された命令体系、マニアック？）

メモリの大容量化、低価格化により、アセンブラでプログラムを作る機会が減って、その優位性はなくなった。

2.3.4　RISC型計算機（MIPS）の命令体系

パイプライン制御を最大限利用するアーキテクチャ



時間 / 命令	サイクル 1	2	3	4	5	6	7	8	9	…
1	IF	ID	AC	AT	OF	EX	MW			
2		IF	ID	AC	AT	OF	EX	MW		
3			IF	ID	AC	AT	OF	EX	MW	
4				IF	ID	AC	AT	OF	EX	…
5					IF	ID	AC	AT	OF	…
6						IF	ID	AC	AT	…
7							IF	ID	AC	…
8								IF	ID	…
9									IF	…
⋮										

図 3.19　命令に着目したパイプライン制御の表現法

IF：命令読み出し　ID：命令解読　AC：アドレス計算　*AT：アドレス変換*
OF：オペランド読み出し　EX：演算実行　MW：メモリ書き込み

2.3.4　RISC型計算機（MIPS）の命令体系

パイプライン制御

- プロセッサの内部動作を機能ブロックの動作ステージに分割し，
- 各ステージが互いに独立に動作するように構成する．
- 各ブロックは入力を処理して次のステージのブロックへ結果を渡す．
- したがって，各ステージは並列に処理を実行し，命令がオーバーラップして実行されている．
- 一つの命令実行時間は長くても，出口のブロックをみると短時間に命令が次々に処理されてくる．
- これをパイプライン制御という．
- ただし、制御の乱れ（例えば条件分岐）が起こる

2.3.4　RISC型計算機（MIPS）の命令体系

- ねらい
- パイプライン制御ができるだけ乱れないようにする。
- 処理を単純化して1サイクルを短くする。
- メモリアクセスをできるだけ少なくする。

- 命令およびオペランドの指定方法を使用頻度の高いものに限定し命令の形式も可能な限り統一する
- 汎用レジスタの数を増やし、演算をレジスタ間で行えるようにする。
- 主記憶の参照はロード命令とストア命令に限定

2.3.4　RISC型計算機（MIPS）の命令体系

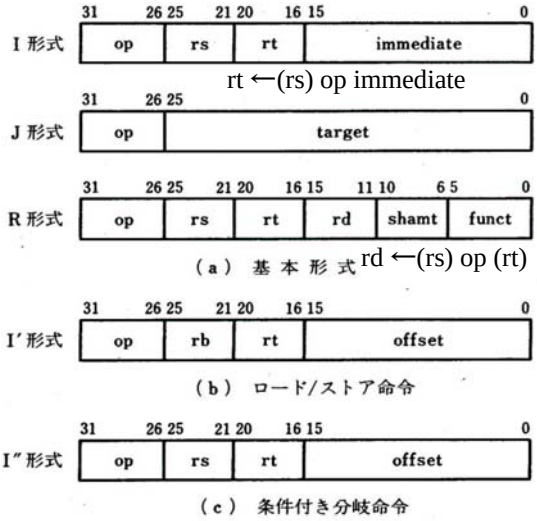


図 2.14　MIPS R2000/R3000 の命令語の形式

op: 操作符号
funct:操作符号の拡張部
rs:ソースレジスタ
rd:デスティネーションレジスタ
rt:ソースまたは
　デスティネーションレジスタ
rb:ベースレジスタ
target:飛び越しアドレス
offset:アドレス偏移
immediate:即値オペランド
shamt:シフト桁数

2.3.4　RISC型計算機（MIPS）の命令体系

- 1ワード4バイトでバイトアドレス
- 32個の汎用レジスタ、レジスタ指定に5ビット
- すべての命令は1語
- 命令数が少ない、op部が6ビット（ただし、R形式のfuncにより100以上の命令を指定可能にしている）

- 演算は3オペランド（I形式またはR形式）でレジスタか即値のみを指定
- インデックスレジスタはなくベースレジスタ相対のみ（プログラムでベースレジスタの値を変更する）
- 主記憶参照はロード命令とストア命令のみ（I’形式）
- 無条件分岐はJ形式(直接アドレス）
- 条件分岐はI’’形式で分岐先はPC相対、rsとrt部で判定条件を指定

2.3.4　CISCとRISC

CISC(complex instruction set computer)
- 命令数：200以上　命令長：可変
- プログラム：少ない命令で実現（実行命令数の削減で高速化）
- ハードウェア：複雑
- アセンブラプログラミングが容易

RISC(reduced instruction set computer)
- 命令数：50～100命令　命令長：1語に固定
- プログラム：命令数は増加（ 1サイクル時間の短縮で高速化）
- ハードウェア：単純（1チップLSI化が可能）
- アセンブラでプログラミングは不可能。高級言語利用が必須
- コンパイラ技術を駆使し、パイプラインの乱れを防ぐ必要がある。

現状では1チップCPUが主流、16ビット以下のプロセッサがCISCで32ビット以上がRISCがおおまかな図式であるが、LSIの集積度向上でCISCがもりかえしている。

[問12]CISCとRISCが高速化に対するアプローチの違いを述べよ。