目次

★2　CPU と機械語

★2.1　メモリと番地の指定

前回の授業では，演算に用いるデータは既にレジスタに保持されているとして CPU の大まかなしくみを説明した．実際には，データはメモリに保存されており，そこからレジスタに転送する必要がある（そして演算の後には必要に応じてメモリに書き戻す）．したがって，メモリとのデータのやりとりも，ALU を用いた演算と同様に CPU の大事な仕事である．

メモリには**番地（アドレス）**が割り振られており，CPU がメモリとデータのやりとりをする際にはこの番地を用いてメモリ中のデータの格納場所を指定する．多くのコンピュータでは，番地の割り振り方は次のいずれかである．

ワードアドレッシング 1 語 (☆ 1) 単位で番地をつける．1 語 1 番地
バイトアドレッシング 1 バイト (☆ 2) 単位で番地をつける

☆ 1) **語** (**word**): CPU が命令・データを扱う際の単位．現代の PC 用 CPU では 1 語は 32bit か 64bit であることがほとんど．ひと昔前の PC には 1 語 8bit や 16bit の CPU が用いられていた (現代でも組み込み機器によく用いられる)．

☆ 2) 1 バイトは普通 8bit．1byte または 1B とも書く．

Q1. 1 語単位で番地をつけるあるコンピュータに，256 語分のメモリが搭載されているとする．このメモリの番地を 0 からはじまる連続した整数で表すとすると，番地の最大は 10 進数でいくつか．また，それは 2 進数でいくつか．
Q2. 1 語 32bit で 512 語分 (1 語 1 番地とする) のメモリが搭載されたコンピュータを考える．メモリの容量は何 B か，またそれは何 kB か (☆ 3)．

☆ 3) $1\text{kB} = 2^{10}\text{B} = 1024\text{B}$ とする．

★2.2　プログラム内蔵方式

現代のほとんど全ての計算機の CPU は，プログラム内蔵方式をとっている．

プログラム内蔵方式: CPU に対する命令（プログラム）をメモリに保持し，それに基づいて計算機を動作させる方式．

例えば ENIAC(☆ 4) の場合，動作を変えるためには真空管やリレーをつなぐ配線を組み替えなければならなかったので，プログラム内蔵方式ではなかった．

この方式の中でも特に，プログラムとデータを区別なしにメモリに記憶する方式を**ノイマン型アーキテクチャ**という (☆ 5) と言う．プログラム用のメモリとデータ用のメモリを別に用意するハーバードアーキテクチャというものもある．

☆ 4) ENIAC: 世界初のコンピュータ (汎用電子計算機) としてよく紹介される．1946 年完成．より厳密な「コンピュータ」の定義からすると，世界初とは言えないという意見も多い．

☆ 5) ノイマン: 数学者 von Neumann のこと．彼がプログラム内蔵方式について記述した論文を最初に記したため，ノイマン型という呼び名が定着した．彼が発明したわけではない．

★ 2.3 計算機 HOGE とその機械語

CPU を動作させる機械語プログラムがどんなものであるかわかりやすいように，次のような仕様の架空の CPU「HOGE」とその機械語「HOGE 用機械語」を考えてみよう（これらは実際のものよりはるかに簡略化されています）.

★ 2.3.1 計算機 HOGE

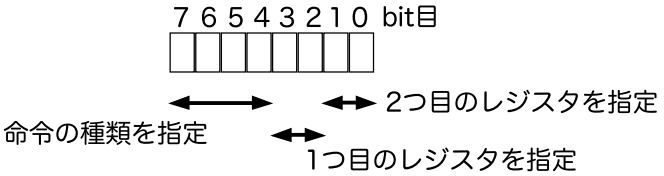
- プログラム内蔵方式かつノイマン型アーキテクチャである
- 命令もデータも 8bit を基本単位とする（1 語 8bit）
- 1 語を保持できるレジスタが 4 つある（それぞれ a,b,c,d という名前とする）
- 8bit の符号付き整数の加減算ができる ALU を 1 つ備えている
- 符号付き整数には 2 の補数表現を用いる
- 最大 256 語を記憶できるメモリが接続されており，1 語 1 番地で 0_{10} から 255_{10} までの番地がついている

★ 2.3.2 HOGE 用の機械語

計算機 HOGE は 1 語 8bit であるから，機械語プログラムもその単位で作るのが自然だろう．上記の仕様から次のことがわかるので，これを手がかりに機械語の構成を考えてみよう.

- レジスタ数は 4 だから，そのうちの 1 つを指定するには 2bit あればよい
- メモリの番地を指定するには 8bit 必要

加算や減算の命令 2 つのレジスタの値を用いて演算を行い，結果を 1 つ目のレジスタに書き込む (☆ 6)．このような命令の場合，例えば 8bit を次のように使って表現したらよい.



例えば，以下のように決めたとすると，00011011 という 8bit の 2 進数は「1 つ目のレジスタを c，2 つ目を d として，加算を実行せよ（c+d の結果を c に書き込む）」という命令と解釈することができる (☆ 7).

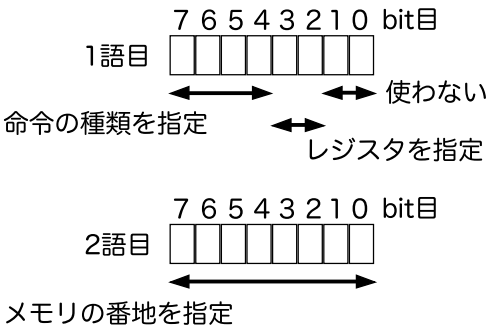
- 命令の種類: 0001 は加算，0010 は減算，など
- レジスタの指定: 00 はレジスタ a，01 は b，10 は c，11 は d

Q3. この例の場合，レジスタ a,b,c,d の内容が 10 進数でそれぞれ 1, 0, 5, 99 のときに 00101000 という命令を実行すると，それぞれのレジスタの内容はどう変化するか.

☆ 6) 書き込み先も任意に指定できるように機械語を設計してもよい．その場合はレジスタを 3 つ指定することになる.

☆ 7) 命令の構成要素のうち，命令の種類を指定する部分を**オペコード** (operation code, 略して opcode) といい，演算に用いる値やその格納場所を指定する部分を**オペランド** (operand) という．左の例では，上位 4bit（7bit 目から 4bit 目まで）がオペコード，下位 4bit がオペランド.

データ転送の命令 レジスタからメモリへ，またはその逆にデータを転送する命令．レジスタを 1 つとメモリの番地を 1 つ指定する．レジスタの指定は上述の場合と同様に考えればよい．一方，番地の指定には 8bit 必要なので，命令を 2 語で構成することになる (☆ 8)．命令の種類の指定は，加減算の命令と区別できるように，例えば 0011 がメモリからレジスタへの転送，0100 がレジスタからメモリへの転送，などとすればよい．



☆ 8) このようにした場合，プロセッサが命令を解釈する際にそれが 1 語のものか 2 語のものかを判断する必要がある (例えば，データ転送命令（の 1 語目）を見たら次の語を続けて読んでそれを番地指定とみなす，など)．現実には，命令長が十分 (32bit とか) あったり，または別種の工夫のために，1 語で番地指定までできる機械語もある．

Q4. 00110000, 00001010 というビットの並びを上記の説明にしたがって HOGE 用の機械語命令と解釈すると，どんな命令になっているか．

次のような命令を考えてみよう．

1. 72 番地の内容をレジスタ a へ転送（読み込む）
2. 73 番地の内容をレジスタ b へ転送
3. レジスタ a と b の値を加算（結果はレジスタ a へ）
4. 74 番地の内容をレジスタ c へ転送
5. レジスタ a の値から c の値を減算（結果はレジスタ a へ）
6. レジスタ a の内容を 72 番地へ転送（書き出す）

Q5. メモリの 72,73,74 番地にそれぞれ 3,7,-7 という整数値が格納されていたとする．このとき，上の命令を実行するとどんな計算が行われるか説明せよ．

上の命令を HOGE 用の機械語に翻訳してみると，次のようになる (☆ 9)．

```
00110000
01001000
00110100
01001000
00010001
00111000
01001010
00100010
01000000
01001000
```

☆ 9) この例では，プログラムの終わりを示すものが何もない．このような場合，CPU はどこがプログラムの終わりかわからないので，最後の命令を実行した後も命令を実行しようとし続ける．その際には，メモリ中でこのプログラムの後にあるもの（他のプログラムの一部かもしれないし何らかのデータかもしれない）が何だろうと無理矢理命令だと解釈して実行し続けるので，おかしなことになる．そういうことにならないように，実際にはプログラムの終わりを意味する命令を追加しておく必要がある．

Q6. 上記の機械語には誤り（1. から 6. と合わない所）がある．それを見つけて訂正しなさい．

★ 2.4 命令セットアーキテクチャ

機械語プログラムを作成しようとする場合，プログラマは，そのプログラムが動作する計算機にはどんな命令が用意されているのか，レジスタはどんな構成なのか（種類や数など），などを知っていなければならない．このような，ソフトウェア作成者の側が知っておく必要のある計算機の構成（アーキテクチャ）のことを**命令セットアーキテクチャ** (☆ 10) という．同一の命令セットアーキテクチャをもつ計算機ならば共通の機械語プログラムを動作させることができる．

その計算機の CPU がどんな部品で作られているか（真空管か電子回路か人間かなど），クロック周波数はいくつなのか，といったことは，機械語プログラムを正しく動作させられるかどうかと独立なので，命令セットアーキテクチャには含まれない．

☆ 10) 命令セットアーキテクチャ: Instruction Set Architecture

● 命令セットアーキテクチャの例（この他にもたくさんある）

- x86 アーキテクチャ (IA-32): Intel が 8086 プロセッサ (1978 年) 以降，Pentium4, Intel Core, Intel Core 2 等現在にいたるまで採用している命令セットアーキテクチャ．何度か拡張されているが，下位互換性を維持している（8086 用の機械語プログラムが Intel Core i7 でも動作するということ）．AMD など Intel 以外のメーカもこのアーキテクチャをサポートした互換プロセッサを生産している．
- PowerPC: IBM と Motorola の共同開発．IBM のサーバや Apple の PowerMac シリーズ，ゲーム機（Wii, PlayStation 3, Xbox 360），組み込み機器などに採用されていた．Apple は 2006 年に，Mac の CPU を PowerPC から x86 に切り替えた．
- SPARC アーキテクチャ: SUN Microsystems の開発したプロセッサ SPARC の命令セットアーキテクチャ．同社のワークステーションに採用．

★ 2.5 機械語とアセンブリ言語

人間が機械語プログラムを書いたり入力したりデバッグしたりするのは大変なので，機械語の命令をほぼ 1 つずつ英数字を使った記号の並びに置き換えた，**アセンブリ言語** (☆ 11) というものが考えだされた．

☆ 11) アセンブリ言語: assembly language

```
00110000          ← 機械語プログラムの例
01001000          それをアセンブリ言語に翻訳したもの →
00110100
01001000
00010001
00111000
01001010
00100010
01000000
01001000
```

```
LD    ra, X
LD    rb, X
ADDA  ra, rb
LD    rc, Z
SUBA  ra, rc
ST    ra, X
```

C 言語などの**高級言語** (☆ 12) のプログラムは，**コンパイラ** (☆ 13) を用いて**コンパイル** (☆ 14) することで機械語プログラムに翻訳される (☆ 15)（Java のような例外もあるが）．命令セットアーキテクチャが異なれば機械語も異なるが，コンパイラはターゲットとするマシンのアーキテクチャにあわせて翻訳してくれるので，ソースコードを書くプログラマは普段その違いを気にすることはない．

一方，アセンブリ言語のプログラムは，**アセンブラ** (☆ 16) を用いて**アセンブル** (☆ 17) することで機械語プログラムに翻訳される．アセンブリ言語の命令はプロセッサの機械語命令とほぼ 1 対 1 で対応しており，アーキテクチャが異なればアセンブリ言語も異なる．

☆ 12) 機械語・アセンブリ言語との対比で，ほぼ全てのプログラミング言語は「高級」に分類される．

☆ 13) コンパイラ: compiler

☆ 14) コンパイル: compile

☆ 15) 正確には，コンパイラがソースをアセンブリ言語に翻訳し，それをアセンブラが機械語に翻訳する

☆ 16) アセンブラ: assembler

☆ 17) アセンブル: assemble（集める，組み立てる）