前回:
計算機のなかでのデータの表現のしかた
自然数, 整数, 実数

今回:
ノイマン型コンピュータの構成要素を説明する.

キーワード:
メモリ, バイト, ワード, アラインメント, エンディアン
CPU, バス, レジスタ, 命令, プログラムカウンタ
---
3 ノイマン型コンピュータ

現在ふつうに使われているコンピュータの方式

まず構造と機能を説明する

プリント図1.2

3.1 メモリ
  主記憶(main memory)ともいう
  プログラムを構成する命令とデータを格納

バイト, ワード
  8ビットをバイト(Byte)   C言語の整数型では : char
  16ビットを半語 (half word)   C言語: short
  32ビットを語, ワード(Word)   C言語: int
  64ビットを倍語 (double word) C言語: long long int
  (8ビット以外の名称は計算機によってことなる)

3.1.1 アドレスの付け方 - アドレス付与規則
  メモリには識別のためにアドレス(番地)が付いている

  0番地からはじまり、いくつまで?
  アドレスのビット数がそれを決める
    最大限使えるメモリの量を制限する
    CPU, OSで定められている
  例: アドレス32ビットなら 2^32 ~ 40億(4GB)

3.1.1.1 バイトアドレス
    バイト単位にアドレスがつく
    <-> ビットアドレス, ワードアドレス

3.1.1.2 整列化(アラインメント)制約
  バイトを超える長さのデータは, キリのいい番地から始まる必要があることがある
  半語境界 偶数番地から始めないといけない制約
  語境界   4の倍数番地から
  倍語境界 8の倍数番地から
  (計算機の種類によって、制約がないものもある)

3.1.1.3 アドレス付け(エンディアン)規約
  バイトを超える長さを持つデータをどの順番にメモリに格納するか
    バイトオーダーともいう
  (例) 32ビットの語は4バイトからなる
    最上位バイト..最下位バイト
    それらの4バイトをどの順にメモリに格納するか

  プロセッサの種類によって二種類の方式
  (Big Endian) 最上位バイトを最初のアドレスに格納
    (図4.14, 4.17, 4.18)
    SPARC, MC68000, IBMメインフレーム - プロセッサの種類
  (Little Endian) 最下位バイトを最初のアドレスに格納
    (図4.19, 4.20, 4.21)
    インテルプロセッサ

  MIPSは両方に対応 - この授業でとりあげるプロセッサ
    この授業ではBig Endianで説明

3.1.2 プログラムを使ってメモリの中を調べる (実験)

ふつうはメモリの内容を計算機の外から見ることはできない
  -> プログラムを動かして調べてみよう

3.1.2.1 エンディアンを調べる

メモリの内容をアドレス順にバイト単位で出力する

番地の指定には C 言語のポインタを利用する
  ポインタ型の値は、じつは番地である
  (こういうことができるのが C言語の強み)
  (「低レベルの処理」という)
  (ざっくり「システム記述」ともいう)

```
#include <stdio.h>
#include <stdlib.h>

// 指定した番地から, 指定したバイト数だけ
// バイトごとにメモリの内容を16進2桁で出力
void dump_memory(unsigned char addr[], int nbyte){
  int i;
  for(i = 0; i<nbyte; i++){
    printf("%02x ", addr[i]);
  }
  printf("\n");
}

int main(int ac, char **av){
  short s = 0x1234;    // 16ビット(半語)の整数データ
  int i = 0x12345678;  // 32ビット(語)の整数データ
  long long int ll = 0x123456789abcdef0LL;
                       // 64ビット(倍語)の整数データ

  dump_memory((unsigned char *)&s, 2);
  dump_memory((unsigned char *)&i, 4);
  dump_memory((unsigned char *)&ll, 8);

  exit(0);
}
```

```
[CPU: Intel Core i3] 普通のPC
34 12
78 56 34 12
f0 de bc 9a 78 56 34 12

[CPU: SPARC] sun.edu.cc.uec.ac.jp
  - 昔の情報基盤センターのサーバ
12 34
12 34 56 78
12 34 56 78 9a bc de f0
```

3.1.2.2 アラインメント制約の有無を調べる

メモリの「キリのよくない」番地から長い値(2バイト以上)を
読み出してみる。読めるか?

```
#include <stdio.h>
#include <stdlib.h>

// この配列の先頭番地は、じつは倍語境界に合っている
unsigned char bytes[] = {
  0x12,
  0x34,
  0x56,
  0x78,
  0x9a,
  0xbc,
  0xde,
```

```
  0xf0,
};

int main(int ac, char **av){
  int *ip;
  int i;

  ip = (int *)&bytes[0];
  i = *ip;
  printf("%08x\n", i);

  ip = (int *)&bytes[1];
  i = *ip;
  printf("%08x\n", i);

  exit(0);
}
```

```
==================================
[CPU: Intel Core i3] 普通のPC
78563412
9a785634
  アラインメント制約なし

[CPU: SPARC] sun.edu.cc.uec.ac.jp
12345678
バスエラー (coreを出力しました)
  アラインメント制約のため, エラーになった
```

3.1.2.3 値の二進表現
二進表記に変換する方法をそのまま実装 (前回 2.1.4)

```
==================================
// 整数値nの下位8ビットを二進数で出力する
void dump_byte(unsigned int n){
  int buf[8];
  int i;
  // 下位から順に二進数にして配列に格納
  for(i=0; i<8; i++){
    buf[i] = n % 2;
    n = n / 2;
  }
  // 変換結果を上位から順に出力
  for(i=7; i>=0; i--){
    if(buf[i] == 0){
      printf("0");
    } else {
      printf("1");
    }
  }
}
==================================
```

3.1.2.4 浮動小数点数の表現 (第二回説明分)
上で説明した二進表現出力を使う

```
==================================
#include <stdio.h>
#include <stdlib.h>

// 指定した番地から, 指定したバイト数だけ
// バイトごとにメモリの内容を2進8桁で出力
void dump_bytes(unsigned char cp[], int nbyte){
  int i;
  for(i=0; i<nbyte; i++){
    dump_byte(cp[nbyte-i-1]); // little endian -> 逆順
    // dump_byte(cp[i]);      // big endian -> 番地順
    printf(" ");
  }
  printf("\n");
}
```

```
int main(int ac, char **av){
  float f;  // 単精度浮動小数点型 (32 bit)
  double d; // 倍精度浮動小数点型 (64 bit)

  // av[1]: プログラム実行の第一引数の文字列
  // atof: 引数で指定した値を倍精度浮動小数点数に変換
  f = atof(av[1]); // 代入の際に単精度に変換される
  d = atof(av[1]); // 倍精度そのまま

  dump_bytes((unsigned char *)&f, sizeof f);
  dump_bytes((unsigned char *)&d, sizeof d);

  exit(0);
}
==================================
./a.out 1.0
00111111 10000000 00000000 00000000
00111111 11110000 00000000 00000000 00000000 00000000 00000000 00000000

./a.out -1.0
10111111 10000000 00000000 00000000
10111111 11110000 00000000 00000000 00000000 00000000 00000000 00000000

./a.out 6.0
01000000 11000000 00000000 00000000
01000000 00011000 00000000 00000000 00000000 00000000 00000000 00000000

1.00000001 はどうなる?

3.2 入出力部
外界との情報交換
  人間相手
  他のコンピュータ

2次記憶, 外部記憶, 補助記憶  (<-> 主記憶)
  磁気ディスク(ハードディスク)
    電源を切っても記憶内容が消えない
      ファイルを保持
    他にも重要な使い道が - 授業後半で

3.3 CPU(central processing unit;中央処理装置)
プログラムの指定に従って演算などを実行する

3.3.1 演算部
演算ユニット (ALU)
  算術演算, 論理演算
レジスタ群
  高速メモリ
    高速とは: 読出し/書込みに要する時間が短い
  計算結果の一時格納用などに使う
  主記憶に比べて容量は少ない (32個とか)

3.3.2 制御部
  メモリから読み出した機械語命令を実行
  機械語命令 ((machine) instruction)
    CPUが直接実行できる
    0/1のならび
  プログラムカウンタ
    次に実行する機械語命令のアドレスを保持
      レジスタの一種とも考えられる
  命令レジスタ
    読み出した機械語命令を一時的に格納する
  命令デコーダ
    機械語命令を解読し, 各部を制御する

3.3.3 バス bus
CPU, メモリ, 入出力部をつなぐ信号線
```

  内部バス - CPU内部
  外部バス - CPUとメモリ,入出力部
複数の信号線で並列にやりとり
  バス幅 - 16, 32, ...など

アドレスバス
  メモリのアドレスをCPUから伝達するバス
  (入出力部にも機器を区別するアドレスがついている)

データバス
  CPUとメモリや入出力部のあいだでデータをやりとりするバス
  双方向にデータを転送

コントロールバス
  制御対象の選択 (メモリ or 入出力部, など)
  操作の指定 (読み出し or 書き込み, など)

3.3.4 メモリアクセス
メモリの読み出しの手順
(1) CPUがアドレスバスに対象アドレスを送出
(2) CPUがコントロールバスでリード信号を送出
(3) CPUがコントロールバスでメモリリクエスト信号を送出
    ...ここでメモリはデータを取り出し...
(4) メモリは対応するデータをデータバスに送出
(5) CPUはそのデータを読み取る

メモリへの書き込みの手順
(1) CPUがアドレスバスに対象アドレスを送出
(2) CPUがデータバスに書き込むデータを送出
(3) CPUがコントロールバスでライト信号を送出
(4) CPUがコントロールバスでメモリリクエスト信号を送出
(5) メモリはデータバスにあるデータを受け取って書き込み
    ...ここでメモリはデータを書き込む...

3.3.5 プログラム実行
四つのステップで実行
(1) 命令フェッチ instruction fetch
  メモリから命令を読み出す -> 命令レジスタに格納
(2) デコード decode
  命令レジスタの内容を解読
(3) 処理
  必要なデータを取り出し, 演算ユニットで計算
(4) 格納
  計算結果を格納する

3.4 ノイマン型計算機の定義
上記で説明した構成を持つ計算機

(1) プログラム可変内蔵方式
  プログラムをメモリに格納
  プログラムを書き換えることも可能
(2) 逐次制御方式
  命令はひとつずつ順に実行
  データもひとつずつ処理
(3) 一つのプロセッサ、一つのメモリ
(4) 線形アドレス空間

フォンノイマンボトルネック
  CPUとメモリの間の通信が多すぎて制約に
    命令の読出し
    データの読出しと書込み