システム設計演習（前期分）
第10回

秋田純一

http://j.mp/akita-class

akita@ifdl.jp (@akita11)

今日やること

- ☑コンピュータの「ソフト」と「ハード」をつなげる
  - ☑「ソフト」＝プログラム書き
  - ☑「ハード」＝回路設計
  - ☑原理的にはつながっているはず
  - ☑学問体系としては、ほとんど独立
  - ☑両者をつなげようとしてみる
  - ☑（次回から実習で実際に設計して動かしてみる）

似た問題：化学～生物

- ☑cf: 化学～生物学～医学の学問体系
  - ☑脳・知能
  - ☑生物（多細胞生物）
  - ☑細胞
  - ☑タンパク質・DNA
  - ☑分子・原子
- ☑化学と生物学をつなごうとする試み：
  - ☑分子生物学、生物物理学、・・・
  - ☑まだ成功はしていない

「化学ー生物」と「電気ー計算機」

- ☑学問体系が断絶した世界で発生した問題
  - ☑例：ガン細胞
    - ☑分子レベルからの発生メカニズムは完全には未解明
    - ☑対処療法：外科手術、化学療法など
  - ☑例：ガン化したトランジスタ・・・？
    - ☑コンピュータ＝決定論的システム＝構成要素の完全動作が前提
    - ☑微細化の進展により、量子効果による動作の不確実性が増加
    - ☑現状では、製造技術や設計技術で、なんとかおさえこんでいる
    - ☑・・・いつまでも可能なのか？
      - ☑ ハード屋の言い分：ソフトウエアでなんとかしてくれ（fault-tolerantなど）
      - ☑ ソフト屋の言い分：ハードウエアがしっかりしてくれ
  - ☑例：組込みシステム
    - ☑トレイ開閉ボタンを押してから45秒後にトレイが開くBDレコーダ（実話）
      - ☑ ソフト屋の言い分：CPUがもっと速くなってくれ（ソフトの実行ステップは見えない）
      - ☑ ハード屋の言い分：ソフトウエアをもっと効率化してくれ

先行事例

- ☑「CPUの創りかた」
  - ☑渡波郁著, 毎日コミュニケーションズ (ISBN: 4839909865)
  - ☑いわゆる「萌え本」の走り？
  - ☑論理回路IC(74シリーズ)でCPUを設計して動作させる、という内容

（復習確認）コンピュータの「動作」

- ☑「プログラムを実行」が本質
- ☑プログラム(program)=「手順書」
  - ☑「命令」の並び
- ☑「命令」: 変数代入、演算、分岐、・・・

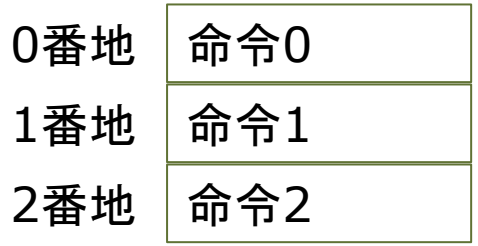
プログラムの例（C言語）

```
int a, b, p, i;
a = 3;
b = 4;
p = 0;
for (i = 0; i < b; i++)
  p += a;
```

- ☑（例）積を求めるプログラム
  - ☑p(積) = a × b
  - ☑「乗算＝加算の繰り返し」を素直にコーディング
    （p = a×4 = a + a + a + a)

「機械語」で書いてみる：定義

- ☑変数（レジスタ）：r0, r1（4ビット）
- ☑値（即値）：imm（4ビット）
  - ・・・プログラム中に記述する
- ☑命令は、メモリの中に入っている
  - ☑メモリ内の位置はアドレスで指定
- ☑フラグ(flag)：内部状態を表す
  - ☑Z (Zero-flag) : 演算結果が0ならば true(‘1’)になる
- ☑プログラムカウンタ(PC)：実行しているアドレス



命令セット(1):代入命令

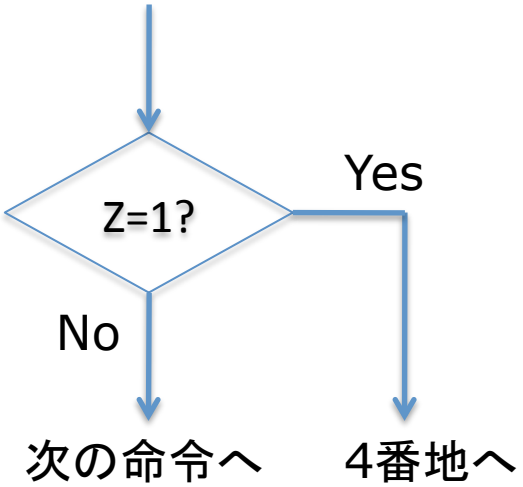
- ☑代入命令
  - ☑`mov imm, r0` ・・・値immをレジスタr0に代入
  - ☑`mov imm, r1` ・・・値immをレジスタr1に代入
  - ☑例:`mov 3, r0` ・・・r0に3が代入される

命令セット(2):演算命令

- 演算命令
  - `add rs, imm, rd` ・・・ rs + imm -> rd
    - rs: source(元)レジスタ
    - rd: destination(先)レジスタ
  - 例: `add r0, 4, r0` ・・・ r0 + 4 -> r0 / r0 += 4;
  - 例: `add r1,15,r1` ・・・ r1 + 15 -> r1 / r1 += 15;
    - ただしr1は4ビットなので結果として(r1-1)が求まる
      (桁あふれした分は無視する＝ないことにする)
    - 15(10進) = 1111(2進4bit) = -1 (+1(0001)の2の補数)
    - 例 : 4(0100) + 15(1111) = 19 (10011) -> 3(0011):4bitで

命令セット(3):分岐命令

- ☑分岐命令
  - ☑`jmp` 分岐先アドレス(無条件分岐)
  - ☑`jz` 分岐先アドレス(=jump if Z is true)
    (Zフラグ='1'/直前の演算結果が0なら分岐)
  - ☑例: `jmp 2` ・・・2番地からの命令を実行する
  - ☑例: `jz 4` ・・・Zフラグ='1'ならば4番地へ分岐



プログラムの例

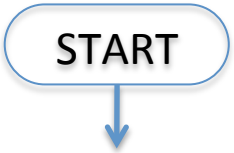
```
0: mov 0, r0
1: mov 4, r1
2: add r0, 3, r0
3: add r1, 15, r1
4: jz 6
5: jmp 2
6: jmp 6
```

命令実行の過程

機械語のプログラムを作ってみる

☑「15÷3」を求めるプログラムを書いてみる

フローチャート



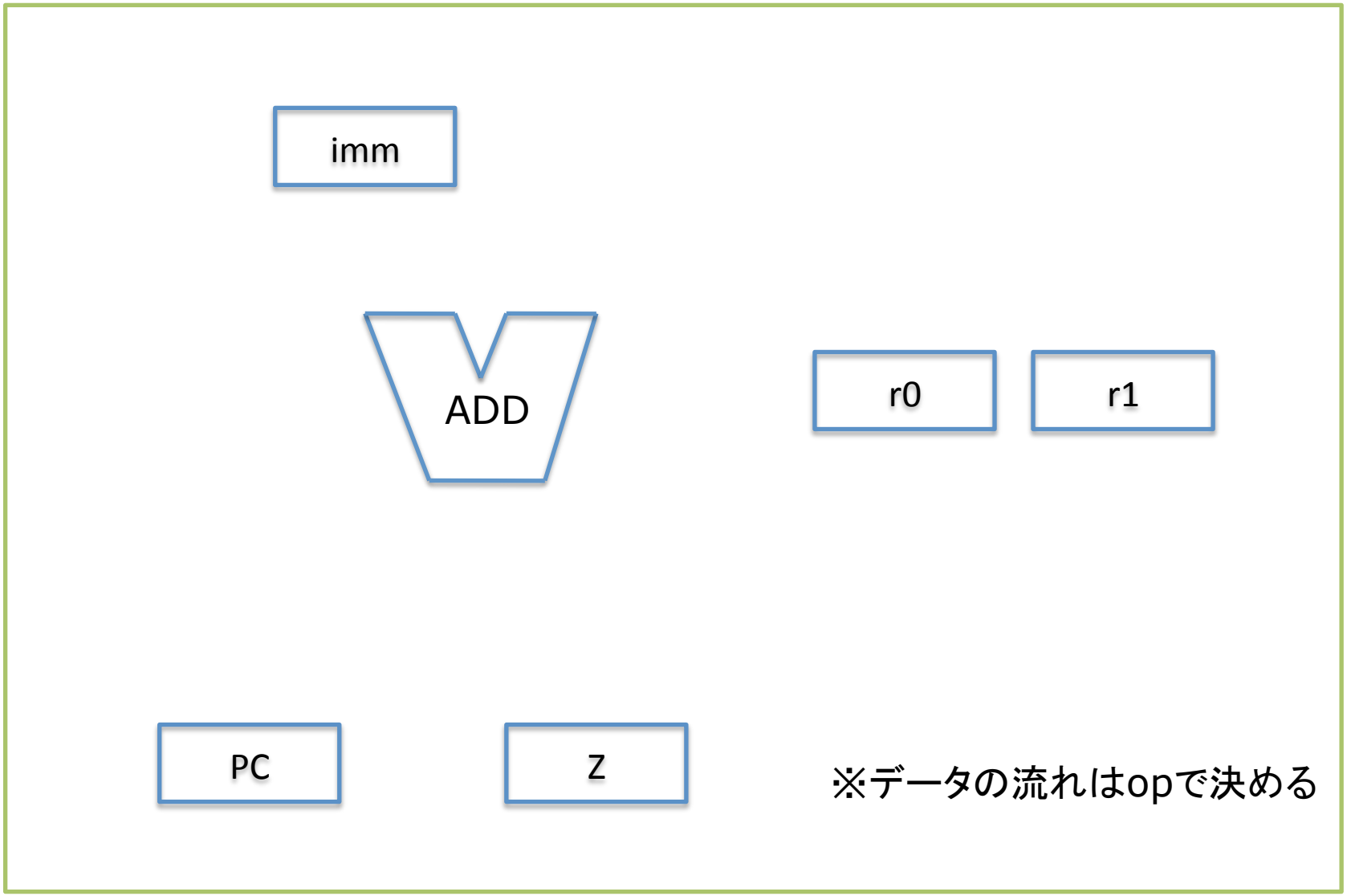
プログラム

0:
1:
2:
3:
4:
5:
6:
7:
8:
9:

命令を実行するCPUを設計する

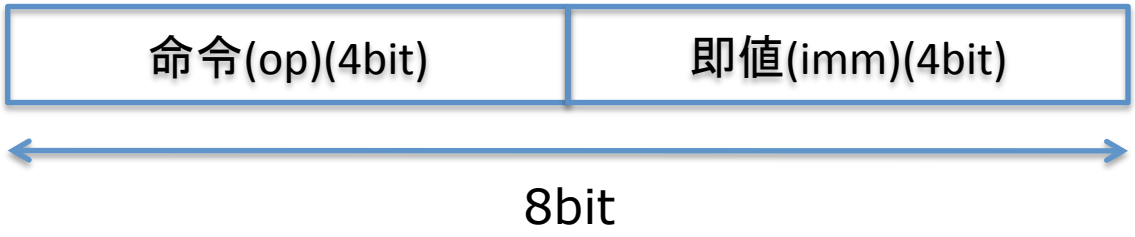
- ☑命令の種類
  - ☑代入命令:「値→レジスタ」の形式
  - ☑演算命令:「レジスタ＋値→レジスタ」の形式
  - ☑分岐命令:
    「値→プログラムカウンタ(PC)」と言い換えられる
- ☑アーキテクチャの概要を設計する
  - ☑データの流れ:値(＋レジスタ)→レジスタ or PC

全体アーキテクチャ



命令セットの設計

- ☑命令に含まれる情報
  - ☑命令：8種類(3bitで足りる）
  - ☑すべての命令に「即値 (imm)」がある(4bit)
  - ☑→きりのいいところで、「あわせて8bit」
    （将来的に命令は16種類まで増やせる）



命令セットの定義

命令コード(op)	命令の記述(ニーモニック)	実行される内容
0000	`mov imm, r0`	imm -> r0
0001	`mov imm, r1`	imm -> r1
0010	`add r0, imm, r0`	r0 + imm -> r0
0011	`add r0, imm, r1`	r0 + imm -> r1
0100	`add r1, imm, r0`	r1 + imm -> r0
0101	`add r1, imm, r1`	r1 + imm -> r1
0110	`jmp imm`	imm -> pc
0111	`jz imm`	imm -> pc, if Z='1'

全体の動作の設計

1. PCが示すアドレスのメモリから
   命令(8bit)を読み出す（”フェッチ”）
2. 命令(8bit)をop(4bit)とimm(4bit)に分ける
3. opに応じて、r0やr1に代入するべき値resを
   求める
4. resをr0やr1やPCに代入
   （必要に応じてZフラグをチェックする）
5. 分岐命令でなければ、PCを+1
   （次の命令の実行の準備）

実行例(1):代入命令

- ☑`mov 4, r1` (0001 0100) の場合
  - ☑1. “0001 0100”を読み出し
  - ☑2. op=“0001” , imm=“0100”
  - ☑3. res = imm = “0100” ※代入命令なので演算なし
  - ☑4. res -> r1
  - ☑5. PCを+1

実行例(2):演算命令

- ☑`add r0, 3, r0`
  - ☑1.
  - ☑2.
  - ☑3.
  - ☑4.
  - ☑5.

メモリの機能

- ☑今回は読み出しだけ
  （メモリに演算結果などを書き込む命令がないので）
- ☑入出力：
  - ☑addr：アドレス（入力）
  - ☑data：データ（出力）
- ☑機能：
  - ☑addrが示すメモリの内容をdataに出力
  - ☑配列変数の読み出しみたいなもの
    （data[addr]）

次回は・・・

- ☑このような動作をするCPUをVHDLで記述して、FPGAボードで動かしてみる
  - ☑cpu.vhd：CPU本体
  - ☑mem.vhd：メモリ
  - ☑cpu_top.vhd: 全体構成(SW, LED, 7segなど)
    - ☑クロックはスイッチで(sw_clk.vhd)
    - ☑7セグメントLEDも使用(seg7.vhd)