前回:
CPUについて
各種の演算と, それを実現する電子回路

今回:
プログラムカウンタ
サブルーチン
MIPSの構成

キーワード:
プログラムカウンタ, サブルーチン, スタック,
MIPS, アドレス空間, レジスタ構成

第5回 サブルーチン, MIPSプロセッサ

3.5.3 ALUの実現
命令ごとに各種の演算をどう切り替えるか?

論理和回路, 論理積回路, 加算器回路などをいつも動作させておき,
命令が要求する結果だけを取り出している
(ハードウェアの並列性を利用)

3.5.4 フリップフロップ, ラッチ
1ビットのメモリを実現

ふたつの「状態」をもつ - 順序回路という
  一方、AND, OR などは入力だけで出力が決まる
  状態がない - 組み合わせ回路という

メモリを実現する方法はこれ以外にもいろいろある

3.5.4.1 RSフリップフロップ
二つのnorゲートをたすきがけに接続
状態を1にする入力(Set)と0にする入力(Reset)を別に持つ

3.5.4.2 ラッチ (正式にはDラッチ)
クロック入力にしたがって
データ入力信号の状態を
  すどおしで出力 (クロック=1)
  記憶する (クロック 1→0)

3.5.4.3 Dフリップフロップ
クロック入力の立ち下がりのタイミングで
データ入力を記憶
  すどおし状態はない

すべて「記憶するタイミング」を指定するために
クロック信号が必要

3.6 プログラムカウンタ(3.3.2)
CPU内の記憶装置
  一種のレジスタ
次に実行する命令の番地を覚えている
  命令をメモリのその番地から読み出して実行

命令実行のあと, 次の命令の番地をさすよう変更
  +4 する (1命令が4バイトの場合)
  これで命令を順番に実行できる

プログラムカウンタの値を別の番地に変更すると…
  つぎは変更後の番地の命令を実行する
  つまり, 命令の実行位置を変更できる

ジャンプ命令とか分岐(ブランチ)命令という

高級言語の if, while, ... などに対応

3.7 サブルーチン (手続き、関数)

プログラム言語の概念
  ひとつの処理をする命令列をまとめておく
それを「呼び出す」(図5.14)
  同じ処理を何度も記述する必要なし
    コードサイズを小さくできる
  多くのプログラムで再利用可能
    ライブラリ
  （ソフトウェア的には理解を容易にするため）
高級言語では手続き, 関数という

呼出し側(caller) と 呼ばれ側(callee)

どうやって実現するか
  まず「戻り番地」を保存して
  サブルーチンの先頭へジャンプする
  サブルーチンの末尾で, 保存しておいた「戻り番地」に帰る

「戻り番地」をメモリのどこに保存するのがよいか
  メモリの固定番地?
    他のサブルーチンを呼ぶと上書きしてしまう
  サブルーチンごとに決まった番地?
    再帰呼出しの場合, 上書きしてしまう
  呼出しごとに別の番地に保存する必要
    スタックを利用する

スタック
  データ構造のひとつ
    push操作とpop操作
    後入れ先だし (Last-in First-out)
  配列(メモリ)とポインタ(スタックポインタ)で実現 (図5.16)

「戻り番地」はスタックにpushしておけばOK
  サブルーチンの末尾でpopして帰る
  入れ子になった呼び出し関係でも正しく動作する

「戻り番地」以外にも, 作業用の変数も同様に保存できる
  手続きのローカル変数はスタック領域にとるのが普通
  stack frame, activation frame

4. MIPS
CPUの一例

4.1 歴史
1981 Stanford大学 John L. Hennessy が設計に着手
1984 MIPS Computer Systems社 設立
1985 R2000
1986 R3000
現在は組込み用に使われている
NINTENDO64
プレイステーション, プレイステーション2, PSP

4.2 構成
機械命令の視点からみると, 内部はどういう構成になっているか
「命令セットアーキテクチャ」
  (どのような電子回路になっているか, とは異なる)

4.2.1 アドレス空間
バイトアドレス
アドレスは32ビット
  0番地 から 2^{32}-1 番地まで
アラインメント制約あり
ビッグエンディアン

4.2.2 レジスタ構成 (図2)
  レジスタ: CPUの中にある高速のメモリ

4.2.2.1 PC (プログラムカウンタ)
  次に実行する命令の番地を保持する

  4の倍数番地 - 下位2ビットはつねに0
  (命令が32ビット固定長, それとアラインメント制約)

2018-05-09

4.2.2.2 汎用レジスタ
  32個 ($0-$31)
  使い道が決められている - 機能名 (表1)
    命令上 - 命令によっては使うレジスタが決まっている
    慣習 - OSや言語によって使い道が決まっている

  スタック
    $sp ($29)が番地を保持
    普通のメモリアクセス命令でロード(読出し)/ストア(書込み)
      (専用の命令 - push/pop - はない)
    若いアドレスに向かって成長

  表1の「保存?」 - 後述
    サブルーチン呼出しの前後で値が保たれるか
    逆にいうと, 呼ばれ側で退避/回復が必要か

4.2.2.3 HI, LOレジスタ
  乗算命令の結果(積 64ビット)
    32bit * 32bit -> 64bit
  除算命令の結果
    商と剰余(それぞれ32bit)が同時に求まる

4.3 MIPSの機械命令
長さはすべて4バイト(32ビット)

二進(16進)表記では人間にわかりにくい
  記号的な表記方法 - アセンブリ言語という
    基本的に機械命令と一対一対応

  機械語への変換はアセンブラというプログラムで

4.4 アセンブリ言語での記法
具体的な命令の紹介に入る前に

4.4.1 プログラムはファイルに記録
  xxxx.s (.sという拡張子)

4.4.2 命令の記述
一行に一つの命令を書く
三つのフィールドからなる

```
  loop:   lw    $4, 0($5)
  ラベル  命令  オペランド
```

4.4.2.1 ラベル
その命令の置かれる番地に名前をつける
英数字,_,.,$ の列 (先頭は数字はダメ)

4.4.2.2 命令
記号名 - ニーモニック (mnemonic)
命令の種別をあらわす
  add, sub, ...

4.4.2.3 オペランド
演算の対象などを指定する

-レジスタ名は $0 - $31 と指定
  $t0 のような機能名（表1）もOK
-定数 (即値 immediate という)
-メモリ番地の指定(ラベル)

4.4.3 コメント
から行末まで

(行頭から始めてはいけない)

4.4.4 プログラム例

プリント 図3