前回:
コンピュータ開発の歴史
コンピュータ性能評価
CPUの一般論
アドレシングモード

今回:
CPUでの命令の実行
パイプライン

キーワード:
ステージ、クロック、パイプライン、レイテンシ、スループット

第10回 CPUでの命令の実行

ここから(授業の最後まで)
  命令を実行する仕組み
  命令を高速で実行する仕組み

6. 命令実行の手順

高速化の前に, 命令を実行する手順を理解する

授業第3回(3.3.5 プログラム実行)で概略を説明した
  四つのステップ

たとえば:
  レジスタどうしの加算命令
    add $10, $8, $9
  レジスタと即値の加算命令
    add $10, $8, 1

6.1 MIPSの命令実行のステップ(ステージ)
(図6.10)

6.1.1 図に出現する要素
mux マルチプレクサ(multiplexor)
  複数の入力から出力をひとつ選択する回路
  左から二本の線が入力され, 右に一本出て行く
    どちらを選択するかはこの図にない制御線で決まる

ALU 演算回路
  左から二つの入力線(の組; 32bit)が入り, 右に結果が出て行く

  加算専用のもあるが, 一般の場合, 演算の種類は
  この図には書いてない制御線で決める

レジスタ
  CPUの中のレジスタ

  読み出しレジスタ番号1 で指定したレジスタの値が
    読み出しデータ1 へ出て行く
  2, についても同様 (1, 2 は同時に読み出せる)

  書き込みレジスタ番号に指定したレジスタに
    書き込みデータの値が書き込まれる
    書き込み指示（クロック）の立ち上がりで

命令メモリ, データメモリ
  メインメモリのこと(と思って下さい)

  左から読み出しアドレスを入れ
  右にそのアドレスの内容が出て行く

  書き込みの場合(データメモリのみ)
  左からアドレスとデータを指定して
  書き込み指示（クロック）を立ち上げると
  その番地に書き込まれる

読み出しと書き込みのどちらを行うかは
この図には書いてない制御線で決める

PC プログラムカウンタ
一種のレジスタ
現在の値が右に出て行き,
次の値が左から入る
こちらも書き込みタイミングはクロックの立ち上がり

6.1.3 MIPSの機械命令の形式 (授業 4.10)
R形式

```
    6      5     5     5     5     6
  ......|.....|.....|.....|.....|......
    op    rs    rt    rd   shamt funct
```

```
  add $10, $9, $8
```

I形式

```
    6      5     5          16
  ......|.....|.....|................
    op    rs    rt        immediate
```

```
  lw  $10, 4($8)
  sw  $10, 4($8)
  (add $10, $9, 8 などもそう)
  beq $10, $11, label  (label は実際には PC+4 との変位)
```

もうひとつ J形式 があるがここでは無視

6.1.2 命令実行のステージ

五つのステージ(段階)

(1)命令フェッチ IF
メモリから命令をフェッチ
PCからアドレスが出て命令メモリに入り
命令メモリから命令が読み出される

PCは普通(分岐命令以外)は +4 して更新する

(2)命令デコード ID
命令デコード
命令のopフィールドなどから(図にはない)制御線に信号を出す
レジスタ読み出し
命令の rs, rt フィールドからレジスタ番号を取り出し
レジスタの値を読み出す
このふたつは並行にできる

(3)ALU操作 EX
ALUを使って計算
入力はレジスタから, あるいは即値
演算 (下のALU)
たとえばadd, sub命令
アドレス計算
メモリアクセス命令の実効アドレス (下のALU)
分岐命令の判定(下のALU)と分岐先 (上のALU)

(4)メモリアクセス MA
データメモリにあるオペランドにアクセス
読み出し or 書き込み

(5)レジスタ書き込み WB
結果をレジスタに書き込む
ALUの演算結果 または メモリの読み出しデータ

6.1.3 信号の伝搬

論理回路での処理には有限の時間がかかる
  入力信号が定まったあと, 出力信号がそれにあわせて
  定まるまでの時間

  メモリやレジスタの読み出しについても同様

書き込みはクロックでタイミングをとる
  レジスタ, メモリ, PCはすべて
  クロックの立ち上がりでデータを記憶

単一クロックサイクルの実行 とは
  IFステージで、PCからのアドレス出力で実行が始まり
  右方向にじわじわ進んでいき,
  最終的に書き込みデータ(と書き込みアドレス)が定まる.
    場所: PC, レジスタ, メモリ

  そこでクロックが立ち上がると, 更新

  「じわじわ進む」のが間に合うようにクロック周期を設定
    速くしすぎるわけにはいかない

6.1.4 単一クロックサイクルでの実行時間

クロックサイクル
  クロックの立ち上がりから次の立ち上がりまで(1周期)

命令実行の全ステップ(ステージ)を1クロックサイクルで完了
すべてのステージで信号の伝搬が完了している必要
つまり
  クロックサイクル = 1命令実行時間 >= ステージ実行時間の和

クロックを高速にしすぎると, 次のクロック立ち上がりの時点で
書き込まれるべきデータがまだ届いていないことになる

2018-06-20

6.2 パイプラインのアイデア
クロックサイクルを短くして
  (命令全体でなく)1ステージが1クロックサイクルで完了するように

実行をステージごとに重ねる
  ステージ数(ここでは5) パイプラインの段数, 深さともいう
  図7.5 → 図7.6

ステージ相互には使う設備(資源)が重ならないことが必要
  そこは次のクロックサイクルで次の命令が使う

レイテンシ latency
  個々の命令の実行時間(開始から終了まで)
スループット throughput
  単位時間あたりの実行命令数

レイテンシを短縮するのではなくスループットを増加させる

パイプライン処理における命令の実行時間(理想的)
  非パイプライン処理での実行時間 / ステージ数
現実にはステージによって処理時間が異なる
  最長の時間のステージにあわせる(図7.6での2ns)

6.3 パイプラインのオーバヘッド
ステージの間に記憶回路が必要 - ラッチ
  前のステージの結果を保持しておくため
時間が余計にかかるようになる

ラッチ遅延
  ステージ間をつなぐ記憶回路の時間
クロックスキュー
  各ラッチへのクロックの到達時間の差があると問題

6.4 MIPSの命令セットの設計
パイプライン実行を意識した設計

6.4.1 命令長が同じ
デコードしなくても次の命令の開始位置がわかる

6.4.2 命令形式が少数 (R, I, J形式)
どの命令でもソースレジスタの指定位置はおなじ
(2)IDステージでデコードとレジスタ読出しが並行にできる

6.4.3 メモリオペランドはロードストアのみ
演算命令の対象ではない
実効アドレスの計算を(3)EXステージで実行可能

6.4.4 オペランドはメモリ中で整列化 (アラインメント)
語データは4の倍数番地から
データ転送はメモリアクセス一回のみで可能
（メモリとのやりとりが32ビット単位であるなら）