情報工学実験 IV
第 8 回： バイトアクセス命令の追加とプロジェクト課題

柴田裕一郎・松尾堅太郎・元島晃伸

shibata@cis.nagasaki-u.ac.jp
情報工学コース

2019 年 2 月 5 日

RISC マシンを完成させる

- これまでメモリアクセスはワード（今回は 16 ビット）単位で行う命令だけを実装
- 文字列処理や画像処理などのプログラムなどでは，バイト（8 ビット）単位でアクセスする命令も欲しい

$\Rightarrow$ バイトアクセス命令の追加

RISC16 のバイトオーダ

- 今回はビッグエンディアンを採用している
  - 奇数番地はワードデータの下位 8 ビット
  - 偶数番地はワードデータの上位 8 ビット

バイトアクセス命令

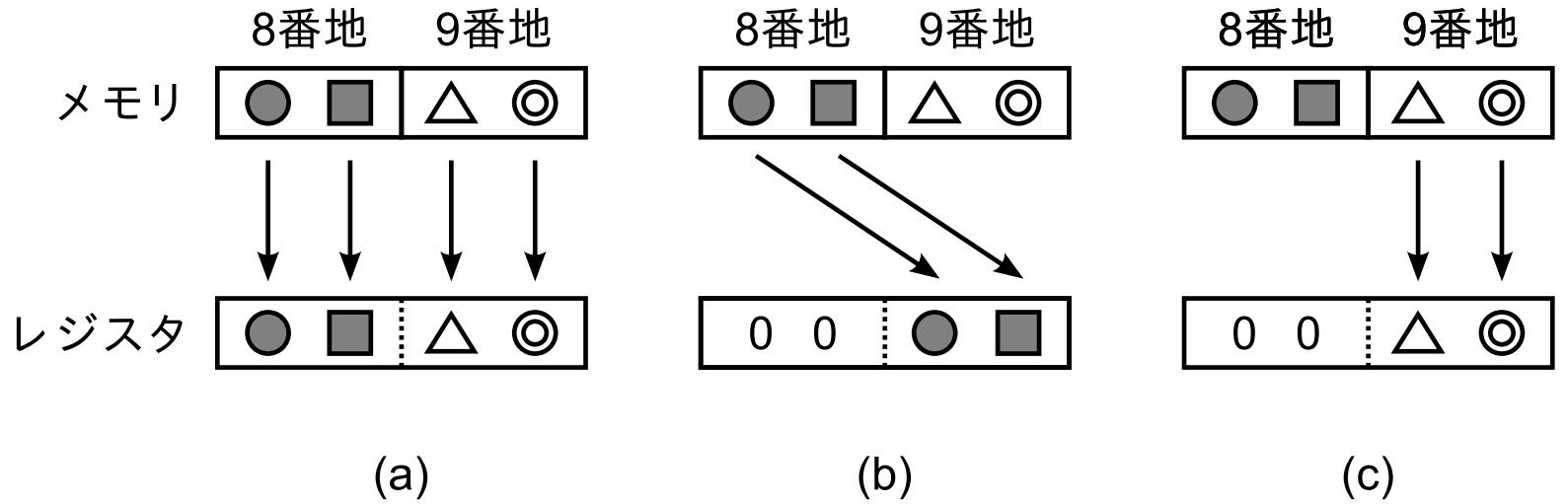
- 2 命令を追加
  - LBU（Load Byte Unsigned）命令
  - SBU（Store Byte Unsigned）命令
  - “Unsigned” はバイトデータを符号拡張しないことを意味

命令コード	ニーモニック	動作
00000dddsss10011	LBU d, (s)	レジスタ s で示す番地にバイトアクセスして d に格納（上位 8 ビットは ‘0’ づめ）
00000dddsss10010	SBU d, (s)	レジスタ d の下位 8 ビットを s で示す番地にバイトアクセスして格納

書き込み制御信号の分離

- メモリへの書き込み制御出力 we（write enable）を分離
  - we0: 偶数番地（上位 8 ビット）に書き込み
  - we1: 奇数番地（下位 8 ビット）に書き込み
- ワード（16 ビット）単位で書き込む ST 命令では両方を同時に 1 にする

ロードの動作（8 番地・9 番地の例）

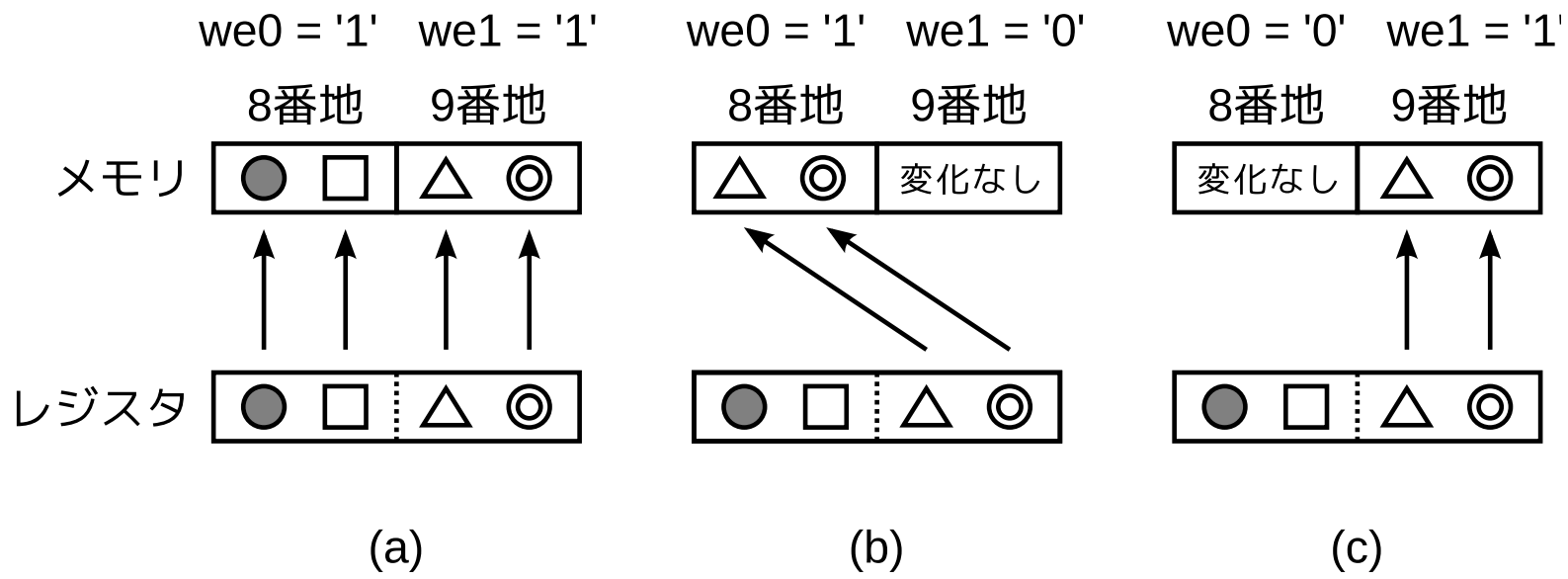


(a) 8 番地から LD 命令でワードデータ読み出し

(b) 8 番地から LBU 命令でバイトデータ読み出し（上位 8 ビットは 0 詰め）

(c) 9 番地から LBU 命令でバイトデータ読み出し（上位 8 ビットは 0 詰め）

ストアの動作（8 番地・9 番地の例）



(a) 8 番地に ST 命令でワードデータを書き込み（we0，we1 ともに 1）

(b) 8 番地に SBU 命令でバイトデータを書き込み（we0 だけ 1）

(c) 9 番地に SBU 命令でバイトデータを書き込み（we1 だけ 1）

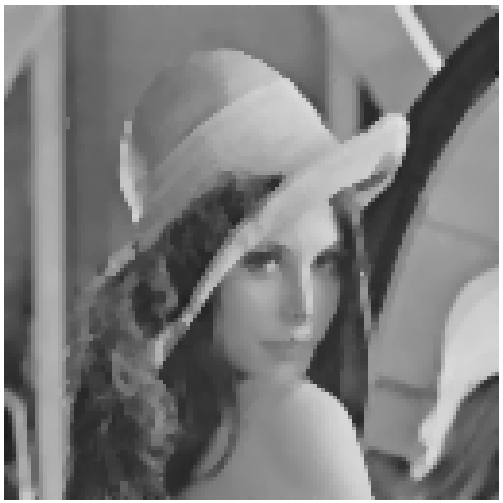
⇒ we0 と we1 を別々に制御する必要がある

risc16 モジュールのインタフェース記述

```
1  module risc16
2    (
3     input wire           clk,
4     input wire           rst,
5     input wire [15:0]    din,
6     output logic [15:0] dout,
7     output logic [15:0] addr,
8     output logic         oe,
9     output logic         we0, we1
10    );
11 ...
12 endmodule
```

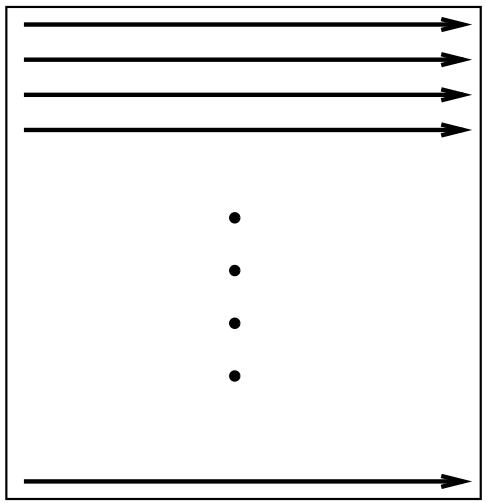
課題内容

- 入力画像に対するメディアンフィルタを用いたノイズ除去処理をなるべく高速に実行できるプロセッサとプログラムを実装する



入力画像のフォーマット

- 入力画像は 128 × 128 ピクセルの 256 階調のグレースケール
- クセルは 1 バイトのデータで表され，0x00（黒）から 0xff（白）までの値をとる
- メモリ上ではアドレスの小さい方から大きい方に向かって，左上隅から右下隅までのピクセルの値が並んでいる



メディアンフィルタのアルゴリズム

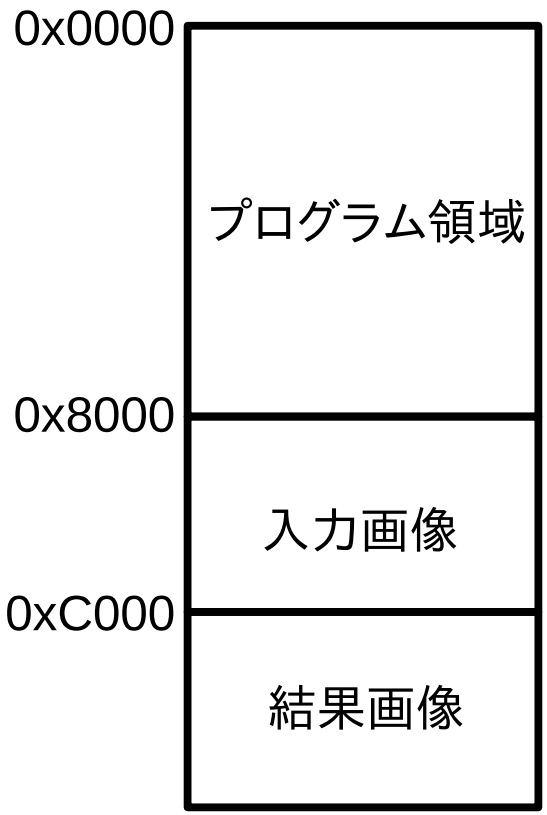
- 原画像の i 行 j 列のピクセルの値を $P(i,j)$ とし，処理を行った後の新しいピクセルの値を $P'(i,j)$ とすると，

$$P'(i,j) = \underset{\substack{i-1 \leq n \leq i+1 \\ j-1 \leq m \leq j+1}}{\text{median}} P(n,m)$$

- ただし i や j が画像の縁の画素の場合は $P'(i,j) = P(i,j)$
- 「元のピクセルおよび隣接する 8 ピクセル（合計 9 ピクセル）の輝度の中央値（メディアン）を求める」

メモリマップ

- メモリ空間は 64 KB（アドレスバスは 16 ビット）
- 原画像は 0x8000 番地から 0xBFFF 番地までに置かれる
- 処理プログラムは 0x0000 番地から 0x0x7FFF 番地までに置く
- 結果画像は 0xC000 番地から 0xFFFF 番地までに書き込む



プロセッサインタフェースの仕様

- プロセッサのインタフェースは risc16ba と同じ
- インタフェースは変更してはいけない
- プロセッサ内部はどんな設計をしてもよい
  - ただし FPGA に載らないのは NG
  - サンプルと異なる入力画像に対しても正しく動作すること

配布ファイル（risc16ba.tar.gz）

- sim_risc16ba.mem: 画像のネガを作成するサンプルプログラム
- sim_risc16ba.sv: シミュレーションモジュール
- imfiles/set_image.mem: サンプル画像データのメモリ初期化ファイル（sim_risc16ba.sv で読み込まれる）
- risc16ba_top.sv: 合成用トップモジュール
- risc16ba_top.tcl: 合成用スクリプト
- risc16ba_top.sdc: 合成用遅延制約記述
- Makefile: Makefile
- check_sim_risc16ba.c: シミュレーション結果のメモリダンプを PGM ファイルに変換するプログラム
- imfiles/nega.dump: 画像のネガを作成するサンプルプログラムの結果（リファレンスデータ）
- imfiles/median.dump: メディアンフィルタの結果（リファレンスデータ）

開発の流れ

- risc16ba.sv を作る（サンプルの実行には LBU，SBU が必要）
- 必要に応じて sim_risc16.sv を修正
- make sim する
  - 結果画像のダンプファイル sim_risc16ba.dump が生成される
- make check する
  - 結果のデータを画像ファイルに変換
  - リファレンスデータと比較
- make する
  - 動作周波数などを確認

make check の確認方法

- エラーが無ければ diff コマンドの結果は何も表示されない

```
% make check
...
diff sim_risc16ba.dump imfiles/nega.dump
%
```

- リファレンスデータとの不一致があればそれが表示される

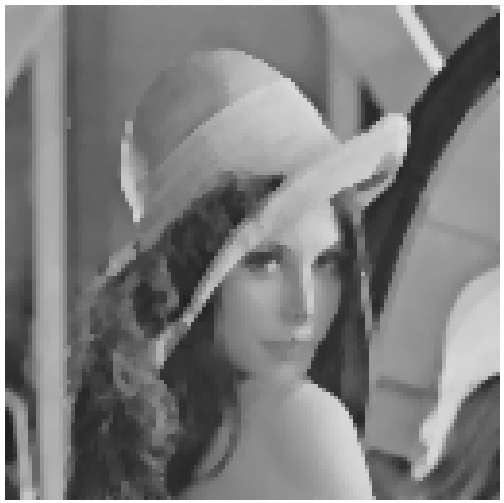
```
% make check
...
diff sim_risc16ba.dump imfiles/nega.dump
4c4
< 92 95 8f 8a 8b 85 81 80
---
> 92 97 8f 8a 8b 85 81 80
make: *** [check] エラー 1
```

結果画像の確認

- 結果の画像は sim_risc16ba.pgm ファイルに保存されている

```
% display sim_risc16ba.pgm &
```

- 画像上で「q」を押すと終了する



レポート提出

- PDF のレポートと電子的な設計データの両方を提出する
- どちらも 2019 年 2 月 28 日（木）の 17:00 まで
- PDF のレポートは LACS に提出
- 設計データは `risc16ba.sv` と `sim_risc16ba.mem` を添付したメールを `shibata-report@cis.nagasaki-u.ac.jp` 宛に送信
- subject には学籍番号を記入のこと

評価について

- 設計コンテスト方式でフィルタ処理にかかる時間の短さを競う
- 処理時間は『処理に必要なクロック数』×『最大遅延』
- ハードウェア量は FPGA の容量が許す限りどんなに増大しても構わない
- 論理合成・配置は緯線できないものは不可
- 評価は，処理の正しさ，実行時間の速さ，設計のオリジナリティ，考察などを総合的に判断
- 他の人のものを（一部でも）コピーしたと見なせるものは当然ながら採点しない