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

2019 年 2 月 5 日

柴田 裕一郎（shibata@cis.nagasaki-u.ac.jp）

1　バイトアクセス命令追加に伴う変更点

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してきました．しかし，文字列処理や画像処理などのプログラムを実行させるためには，バイト（8 ビット）単位でアクセスする命令もぜひ欲しいところです．今回の RISC プロセッサの設計ではビッグエンディアンを採用しており，図 1 に示すように奇数番地は 16 ビットのデータの下位 8 ビットを，偶数番地は上位 8 ビットに対応します．今回は，これらの 1 バイト（8 ビット）のデータを個別に読み出したり，書き込んだりすることもできるように変更してみましょう．

具体的には表 1 に示す LBU（Load Byte Unsigned）命令と SBU（Store Byte Unsigned）命令を追加することにします．LBU が 1 バイトのデータをロードする命令，SBU が 1 バイトのデータをストアする命令です．ニーモニックに「Unsigned」がついているのは，1 バイト（8 ビット）のデータを符号なしデータとみなし，16 ビットのレジスタに格納する際に符号拡張しないことを示しています．

8 番地と 9 番地へのアクセスを例として，LBU の動作を図 2 に，SBU の動作を図 3 に示します．これらの命令を実現するためには，プロセッサとメモリとのインタフェースを若干変更する必要があります．現在の設計では，奇数番地でアクセスが行われると，その番地から 1 を減じた偶数番地に 1 ワード（16 ビット）でアクセスするようになっています．LBU 命令でメモリからデータを読み出す場合には，16 ビットをまるまる読み出してプロセッサの内部で上位か下位の 8 ビット分を選んで取り込めばよいのですが，SBU 命令で書き込む場合には上位 8 ビットに書き込むのか下位 8 ビットに書き込むのかをメモリに指定してやる必要がありま

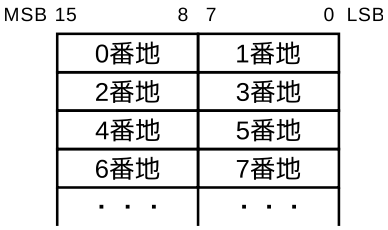

図 1　本実験で設計している RISC マシンのバイトオーダ

表 1　バイトアクセス命令

命令コード	ニーモニック	意味	動作
`00000dddsss10011`	LBU *d*, (*s*)	Load Byte Unsigned	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*)	Store Byte Unsigned	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

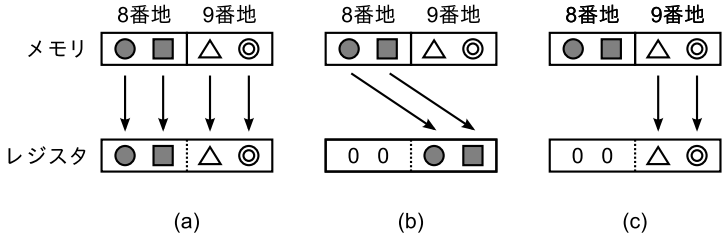

図 2　LD 命令と LBU 命令の動作（8 番地と 9 番地の例）．(a) 8 番地から LD 命令でワードデータを読み出し．(b) 8 番地から LBU 命令でバイトデータを読み出し．(c) 9 番地から LBU 命令でバイトデータを読み出し．

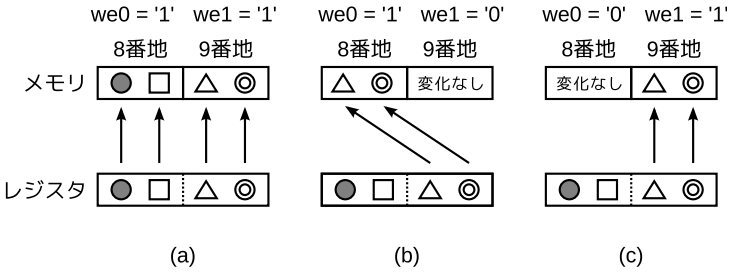

図 3　ST 命令と SBU 命令の動作（8 番地と 9 番地の例）．(a) 8 番地に ST 命令でワードデータを書き込み．(b) 8 番地に SBU 命令でバイトデータを書き込み．(c) 9 番地に SBU 命令でバイトデータを書き込み．

ソースコード 1　risc16ba のモジュールインタフェース記述

```
1  module risc16ba
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
```

す．そこで，書き込み制御信号 we（write enable）を we0 と we1 に分離してやり，we0 がアクティブな場合には偶数番地（上位 8 ビット）に，we1 がアクティブな場合には奇数番地（下位 8 ビット）に書き込むことにします．1 ワード（16 ビット）のデータをストアする ST 命令を実行する際には，we0 と we1 を同時にアクティブにすることによって 16 ビットのデータを一度に書き込みます．変更後のモジュール（以降，モジュール名を risc16ba とします）のインタフェース記述はソースコード 1 のようになります．

2　プロジェクト課題: 画像処理プロセッサの開発

2.1　ストーリー

あなたはあるハードウェア設計会社に勤務するアーキテクトで，高速画像処理システム開発のプロジェクトに参加しています．このシステムは入力画像のインパルスノイズを除去する処理がボトルネックになることが予想されることから，この処理を高速に実行できる専用プロセッサを開発することになりました．開発の deadline は 2019 年 2 月 28 日（木) です．

2.2　処理内容と仕様

入力画像データにメディアンフィルタをかけるとインパルスノイズの影響を低減することができます．

2.2.1　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とします．各ピクセルは 1 バイトのデータで表され，0x00（黒）から 0xff（白）までの値をとります．システムの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，図 4 に示す順番に左上隅から右下隅までのピクセルの値が並んでいるものとします（ラスタスキャン)．

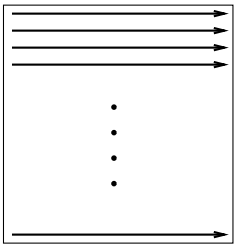

図 4　ラスタスキャン

2.2.2　メディアンフィルタ

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，今回のフィルタ処理は，

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

で表されます．ただし，画像の縁の画素（i や j が 0 か 127 の値をとるとき）については、$P'(i, j) = P(i, j)$ とします．別の言い方をすれば「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」ことで新しいピクセル値を計算するということになります．参考までにメディアンフィルタによるノイズ除去処理の例を以下に示します．図 5 が元画像，図 6 がフィルタ処理後の画像です．

2.2.3　メモリマップ

今回のシステムでは，実験で設計した RISC と同様にメモリ空間を 64 KB とします（アドレスバスは 16 ビット)．画像データは $128 \times 128 \times 1$ B = 16 KB となるので図 7 のように主記憶の領域を分けます．

図 5　原画像

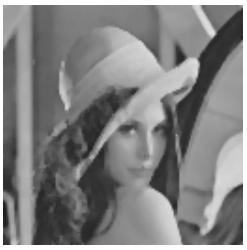

図 6　結果画像

原画像のデータは，あらかじめ 0x8000 番地から 0xBFFF 番地までに 16 KB の領域に置かれるものとします．フィルタ処理を行うプログラムは 0x0000 番地から 0x0x7FFF 番地までの 32KB の空間に置いてください．処理後の結果の画像データは 0xC000 番地から 0xFFFF 番地までの 16 KB の空間に書き込むことにします．

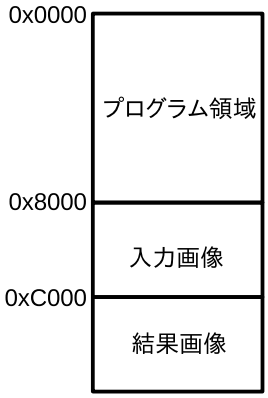

図 7　メモリマップ

2.2.4　プロセッサインタフェースの仕様

プロセッサの部分はどんな設計をしても構いませんが，メモリとのインタフェースだけは決まっているものとし，実験で設計した 16 ビット RISC（risc16ba）と同じとします．プロセッサのモジュール名は「risc16ba」とします．トップモジュールの記述は配布されたものを使用し，変更してはいけないものとします．プロセッサの命令セットは自由に変更して構いません．ただし，任意の画像に対して正しく処理が行えなければなりません．

2.2.5　デザインキット

まず，開発に必要なデザインキット（risc16ba.tar.gz）をダウンロードして展開してください．

```
% tar xvzf risc16ba.tar.gz
```

デザインキットには以下のファイルが含まれています．

- 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（図 5）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）

2.3 開発の流れ

（1）risc16ba.sv を作ります．サンプルプログラムの実行には LBU，SBU 命令の実装が必要です．

（2）sim_risc16ba.sv の内容を確認して下さい．このシミュレーションモジュールでは，シミュレーション開始前に imfiles/set_image.mem の画像データをメモリに張り付けます．また，シミュレーション終了時に実行結果を sim_risc16ba.dump ファイルにダンプさせます．必要に応じて，シミュレーションの終了条件（sim_risc16ba.sv の 76 行目の停止させるプログラムカウンタの値）や，ログファイルに表示させる信号線名などを変更してください．

（3）シミュレーションを実行します．sim_risc16ba.dump が生成されます．シミュレーションログは sim_risc16ba.log に保存されます．波形データは sim_risc16ba.wlf に保存されます．設計によってはかなり時間がかかります．

```
% make sim
```

（4）シミュレーションにエラーが無ければ結果を画像ファイルに変換するとともに，リファレンスデータと比較します．

```
% make check
```

エラーが無ければ以下のように diff コマンドの結果は何も表示されません

```
...
./check_sim_risc16ba sim_risc16ba.dump sim_risc16ba.pgm
diff sim_risc16ba.dump imfiles/nega.dump
```

リファレンスデータとの不一致が見つかると，

```
...
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
```

のように不一致部が表示されます．

なお，メディアンフィルタプログラムの結果をチェックする場合には，Makefile の 21 行目を

```
REF_DUMP = median.dump
```

のように変更してから上記手順を実行します．

（5）結果の画像は sim_risc16ba.pgm ファイルに保存されています．確認のため画像を画面に表示します．画像上で「q」を押すと終了します．

```
% display sim_risc16b.pgm &
```

（6）論理合成・配置配線は make コマンドで行います．

```
% make
```

2.4 提出方法

PDF のレポートと電子的な設計データの両方を提出する必要があります．どちらも締め切りは 2019 年 2 月 28 日（木）の 17:00 です．

2.4.1 レポート

以下の事項をまとめてレポートとし，PDF ファイルを LACS に提出してください．

- 設計したプロセッサの構成図（オリジナルの図を書くこと！）
- 命令の一覧
- 実験で作った最終的なプロセッサからどのような変更を行ったか（少なくとも何かを変更しなければならないこととします）
- プログラム（mem ファイル）の説明
- 設計にあたって工夫した点・アピールポイント
- 設計したプロセッサの性能評価
- 考察
- 感想

2.4.2 設計データ

設計した risc16ba.sv と sim_risc16ba.mem を添付したメールを shibata-report@cis.nagasaki-u.ac.jp 宛に送信してください．その際，subject には学籍番号を記入して下さい．なお，このメールアドレスは課題提出専用のもので，提出期限を過ぎると自動的に使用不能になります．

2.5 評価について

本プロジェクトは設計コンテスト方式をとり，フィルタ処理にかかる時間の短さを競います．処理時間は『処理に必要なクロック数』×『最大遅延』とします（最大遅延は最大動作周波数の逆数）．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ものとします．もちろん論理合成・配置配線できないものは不可とします．ぜひ頑張ってください．

プロジェクトの評価は，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します．なお，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ですが採点しません．

それでは幸運を祈ります！