計算機アーキテクチャ 129－2
（1998. 5. 15）

メディア処理を指向した組み込み用プロセッサのアーキテクチャ

木村通秀† 安里彰†
須賀敦浩† 久保沢元†

我々はメディア指向の組み込み用プロセッサ開発を行なっている．
本プロセッサは2way スーパスカラ型プロセッサで，汎用のマイクロプロセッサにメディア処理を指向した命令セットを追加することにより，メディア処理向けの組み込み用プロセッサとして設計されている．
また，本プロセッサは，浮動小数点SIMD 演算及びメディア処理用の16 ビットSIMD 演算等により，高い並列度での演算の実行が可能である．また，高速な画像データ処理向けに，ピクセルデータ処理指向のロードストア専用命令を備える．
本論文では，本プロセッサの基本アーキテクチャとメディア処理向けに特化したいくつかの特徴的な命令について述べる．

Architecture of an Embedded Processor for Media Processing

MICHIHIDE KIMURA,† AKIRA ASATO,† ATSUHIRO SUGA†
and HAJIME KUBOSAWA†

A media-processing oriented processor which we are developing is described. The processor is structured by adding some carefully chosen media processing oriented instructions to a generic 2 way superscalar processor.
The SIMD float-point and SIMD 16-bit integer execution instructions realize our processor to execute with the advantages of the highly parallel computations. Our processor also equips pixel load/store instructions for fast pixel data processing.
This paper describes the basic architecture of this processor and some unique instructions of it.

1. はじめに

我々はメディア処理を指向した組み込み用プロセッサの開発を行なっている．[1),2)] 本プロセッサの設計に当たっては，汎用のマイクロプロセッサのコアにメディア命令をうまく融合させ，SIMD☆命令と16 ビット整数演算による並列度の向上で性能を向上し，汎用のCPU でありながら，専用LSI に匹敵する性能をもつことを目標とした．

本プロセッサは浮動小数点SIMD 演算命令及びメディア処理用の16 ビットSIMD 演算等により，高い演算の並列実行が可能である．また，高速な画像データ処理向けに，ピクセルデータ処理指向のロードストア専用命令を備えている．

本プロセッサは，主に演算器からなるEunit，命令発行及びシーケンス制御を行なうIunit，命令及びデータキャッシュを制御するSunit，および，外部インタフェースを担当するBunit から構成される．

本論文では，まず，2章で本プロセッサのブロック図を示し，命令セットの概要と，命令発行方式およびパイプライン動作を説明する．3章では，高速性に寄与している2 種類のSIMD 命令の動作を説明する．4章では，16 ビット整数演算命令について述べる．まず，16 ビット整数の定義や演算の概要について述べ，次にその応用について述べる．5章では画像データ処理に向けた，ピクセルデータの高速な入出力に関与するピクセルロードストア命令について述べる．6章では，可変長符号に関する処理方式と，それをサポートするビット処理命令について述べる．

2. 基本アーキテクチャ

2.1 ブロック図

図1は本プロセッサのブロック図である．ブロック図内の各リソースについて表1にまとめた．また，各演算器のそれぞれで実行可能なオペレーションを表2にまと

† (株)富士通研究所
FUJITSU LABORATORIES LTD.
☆ SIMD: Single Instruction Multiple Data-stream

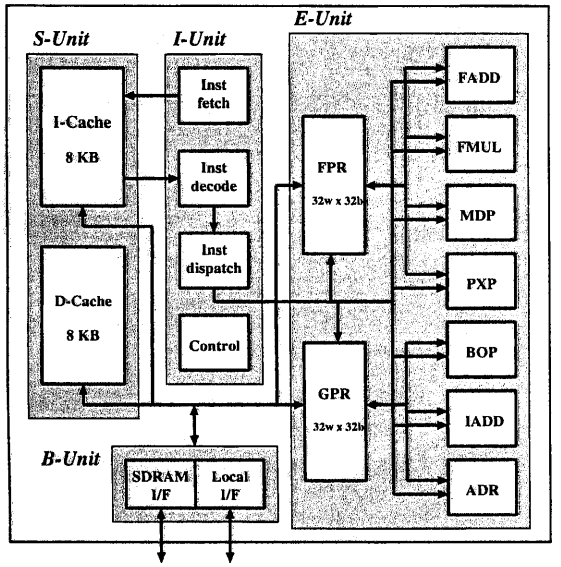

図1 本チップのブロック図
Fig. 1 Block diagram of the processor

めた.

FADD及びFMULのブロックには，SIMD演算用に各々2組の演算器が組み込まれており，SIMD命令を実行可能である．また，FMULには，除算命令及び整数乗算命令の機能があり，汎用系命令の一部はFMULを使って実行される．

2.2 命令セット

本プロセッサの命令セットは，大きく分けて，汎用系命令と呼ぶ汎用のマイクロプロセッサの命令群と，メディア命令と呼ぶ拡張された命令群に分かれる．

汎用系命令は，主に32bit整数を扱う整数系命令と，浮動小数点を扱う浮動小数点系命令，および制御系命令その他がある．

メディア系命令は，並列度の高いSIMD命令や，ピクセルデータ，行列演算，可変長ビットデータ等を扱うための専用命令等から構成されている．データ型も，浮動小数点型，32bit整数型，16bit整数型，可変長ビットデータ等を扱う命令群がある．

2.3 命令実行方式及びパイプライン

本プロセッサは2way Superscaler型プロセッサである．また，メディア命令にはSIMD型命令を実装し，並列度を上げている．

命令は，Iunitの命令フェッチ制御部でアドレスを生成して命令キャッシュから読み出し，メモリアドレスにアラインされた2命令を1単位として，命令バッファに書き込まれる．

命令の発行順序に関しては，In-Orderに発行され，後続命令が先行命令を追い越して発行されることはない．

表1 本プロセッサ内部のリソース
Table 1 Resourses of the processor

リソースの種類	個数・サイズ
命令キャッシュ (I-Cache)	8KB
データキャッシュ (D-Cache)	8KB
SDRAM インタフェース (SDRAM I/F)	32bit
LocalBus インタフェース (Local I/F)	32bit
浮動小数点レジスタ (FPR)	32bit x 32word
整数レジスタ (GPR)	32bit x 32word
浮動小数点 ALU パイプライン (FADD)	1個 (Main/Sub)
浮動小数点乗算パイプライン (FMUL)	1個 (Main/Sub)
メディア演算用パイプライン (MDP)	1個 (Main/Sub)
16ビット積和演算用 ACC レジスタ (MDP)	32bit x 4個
ピクセル演算用パイプライン (PXP)	1個 (Main/Sub)
ビットオペレーション用パイプライン (BOP)	1個
整数パイプライン (IADD)	1個
32bit 加算器 (ADR)	1個

表2 各演算器のオペレーション
Table 2 Operation of each block

FADD	乗除を除く32bit整数及び浮動小数点演算
FMUL	32bit整数及び浮動小数点乗除算 浮動小数点平方根及び逆数
MDP	16bit整数演算 シフト命令
PXP	16bit整数演算の一部 FPRへのロードストア
BOP	ビットオペレーション ビットオペレーション用ロードストア
IADD	乗除を除く32bit整数演算 整数乗除算時の結果の格納
ADR	乗除を除く32bit整数演算 GPRへのロードストア

表3 各パイプラインの実行段数
Table 3 Number of execute stages

命令のカテゴリ	実行段数
通常命令	4段
浮動小数点数の積和計算 (fmadd)	6段
浮動小数点数の逆数 (finv) 浮動小数点数の平方根の逆数 (fisqrt)	15段
浮動小数点数の除算 (fdiv) 浮動小数点数の平方根 (fsqrt)	19段
32bit整数除算命令 (idiv)	21段

各カテゴリ毎の命令のパイプライン実行段数について，表3にまとめた．

ここでは，4段より多いパイプライン段数を持つ命令をマルチサイクル命令と呼ぶ．レイテンシは，マルチサイクル命令を除き，整数演算が1，浮動小数点演算が2，ロードが3である．マルチサイクル命令については，実行段数-1のレイテンシを持つ．ここで，例えばレイテンシが1であるとは，すぐ後の命令が，待たずにその命令の結果を受け取れるという意味である．

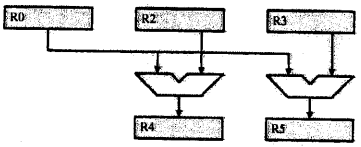

図2 1-2 型 SIMD 命令の概念図
Fig. 2 The model of 1-2 SIMD instruction

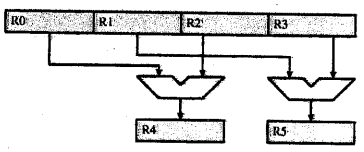

図3 2-2 型 SIMD 命令の概念図
Fig. 3 The model of 2-2 SIMD instruction

3. SIMD 命令

メディア系演算命令は 2 組の演算パイプラインを同時に使用して 32 ビット演算を行なうことができる．これを SIMD 命令と呼んでいる．SIMD 命令には 1-2 型と 2-2 型の 2 通りのタイプがある．

ただし，オペランドの指定はオペランドが異なる場合でも 1 個のレジスタ番号のみで行ない，レジスタ番号の下 1 ビットが異なる 2 個のレジスタが用いられる．

1-2 型は 2 個の演算のうち第一オペランドは双方に共通で，第二オペランドのみが異なる演算である．計算モデルを図 2に示す．これは，R0 が共通のオペランドとなって，それぞれ R2 及び R3 と演算を行ない，結果を R4 及び R5 にそれぞれ格納する命令である．

一方，2-2 型はどちらのオペランドも異なる演算である．計算モデルを図 3に示す．これは，R0 と R2 及び R1 と R3 でそれぞれ同じ演算を行ない，結果を R4 及び R5 にそれぞれ格納する命令である．

4. 16 ビット整数演算

4.1 概　　要

本章では高速なメディア演算を行なうための 16 ビット整数の定義と，その演算処理方式について述べる．

本プロセッサで使われている 16 ビット整数は，Packed Integer(以下 PI) と呼ばれる，16 ビット整数が 2 つ組になって 32 ビットのデータを形成している．ほぼ全ての 16 ビット整数演算は，この PI 同士の演算を行なう．さらに SIMD 命令を使うと，4 並列の演算が 1 命令で実行出来る．

16 ビット整数演算には，制御レジスタの値により，符号付き整数モードと符号なし整数モードがある．また，乗算については，固定小数点演算が可能で，小数点位置を 3 種類選べるようになっている．

4.2 演算の種類

ここでは，16bit 整数演算のいくつかのカテゴリについて，代表的な命令を挙げて動作を説明する．

本章で紹介する演算は，符号モードによる飽和演算が行なわれる．すなわち，オーバーフローを起こした場合は，その符号モードでの最大値が，アンダーフローを起こした場合は，その符号モードでの最小値がそれぞれ取られる．

4.2.1 加 減 算

PI 同士の加算，減算が定義されている．

例として，16 ビット整数 2-2SIMD 加算命令の例をあげる．

mpaddp R4,R0,R2 は，以下の動作を行なう．

$$R4[0:15] = R0[0:15] + R2[0:15]$$
$$R4[16:31] = R0[16:31] + R2[16:31]$$
$$R5[0:15] = R1[0:15] + R3[0:15]$$
$$R5[16:31] = R1[16:31] + R3[16:31]$$

4.2.2 乗　　算

例として，16 ビット整数 2-2SIMD 乗算命令の例をあげる．

mpmulp R4,R0,R2 は，以下の動作を行なう．

$$R4[0:15] = R0[0:15] \times R2[0:15]$$
$$R4[16:31] = R0[16:31] \times R2[16:31]$$
$$R5[0:15] = R1[0:15] \times R3[0:15]$$
$$R5[16:31] = R1[16:31] \times R3[16:31]$$

ここでの演算は，固定小数点モードによる乗算が行なわれ，小数点位置は，モードにより High/Middle/Low の 3 通りが定義されている．固定小数点の位置は，High が符号ビットを除く最上位ビットの左，Middle が中央のビット，Low が最下位ビットの右にあることを示す．乗算の結果が 32 ビットであることから，このモードに応じた丸めが行なわれて，PI データとして代入される．

4.2.3 積 和 演 算

積和演算は，乗算で得られた結果を ACC☆を使って加算する命令群である．行列の計算方式に対応した，水平加算と垂直加算の 2 つの計算方式がある．ここでは，特徴的な垂直加算の例を紹介する．

例として，16 ビット整数 2-2SIMD 垂直積和演算命令の例をあげる．

mpmava R4,R0,R2 は，以下の動作を行なう．

$$R4[0:15] = R0[0:15] \times R2[0:15] + ACC0$$
$$ACC0 = R0[0:15] \times R2[0:15] + ACC0$$
$$R4[16:31] = R0[16:31] \times R2[16:31] + ACC1$$
$$ACC1 = R0[16:31] \times R2[16:31] + ACC1$$
$$R5[0:15] = R1[0:15] \times R3[0:15] + ACC2$$
$$ACC2 = R1[0:15] \times R3[0:15] + ACC2$$
$$R5[16:31] = R1[16:31] \times R3[16:31] + ACC3$$
$$ACC3 = R1[16:31] \times R3[16:31] + ACC3$$

☆ ACC: Accumulator

```
short result[], vec[], matrix[][];

for (i=0; i<4; i++) {
  result[i] = 0;
  for(j=0;j<4;j++) {
    result[i] += vec[j] * matrix[j][i];
  }
}
```

図4 4×4行列演算の例(C言語)

Fig. 4 A C program of calculating matrix data

```
muldpd   R0,x;              /* R0<=x:x */
muld64u  R2,matrix++;       /* a,b,c,d element load */
mqmulp   R4,R0,R2; /* R4<=xa:xb, R5<=xc:xd (also ACC) */
muldpd   R0,y;              /* R0<=y:y */
muld64u  R2,matrix++;       /* e,f,g,h element load */
mqmava   R4,R0,R2; /* R4<=xa+ye:xb+yf,R5<=xc+yg:xd+yh *
muldpd   R0,x;              /* R0<z:z */
muld64u  R2,matrix++;       /* i,j,k,l element load */
mqmava   R4,R0,R2; /* R4<=xa+ye+zi:xb+yf+zj, R5<=... */
muldpd   R0,w;              /* R0<=w:w */
muld64u  R2,matrix++;       /* m,n,o,p element load */
mqmava   R4,R0,R2;          /* R4,R5 <= final result */
must64   R4,result;         /* store the result */
```

図5 4×4行列演算の例(メディア命令)

Fig. 5 An assembler program using media instructions

ここでの演算は，第4.2.2節と同様の固定小数点モードによる乗算が行なわれる．

4.2.4 差分絶対値

PI同士の差分の絶対値を求める演算が定義されている．

例として，16ビット整数2-2SIMD差分絶対値命令の例をあげる．

mpdiffp R4,R0,R2は，以下の動作を行なう．

$$R4[0:15] = abs(R0[0:15] - R2[0:15])$$
$$R4[16:31] = abs(R0[16:31] - R2[16:31])$$
$$R5[0:15] = abs(R1[0:15] - R3[0:15])$$
$$R5[16:31] = abs(R1[16:31] - R3[16:31])$$

4.3 命令の実際の動作

ここでは，4次のベクトルと4×4行列との積を求める計算を例題に，前述のメディア命令の有効性を述べる．

ここで取り上げた例は，以下の行列計算を行なうプログラムである．

$$\begin{bmatrix} x & y & z & w \end{bmatrix} \times \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix}$$

$$= \begin{bmatrix} xa+ye+zi+wm \\ xb+yf+zj+wn \\ xc+yg+zk+wo \\ xd+yh+zl+wp \end{bmatrix}^t$$

これをCで書くと，図4となる．メディア命令で書くと，図5のように簡単に記述することができる．(図5のみで，全ての演算を終了している．)

```
ldhu    R0,x;       /* R0<=x */
ldhu    R4,a;       /* a element load */
imul    R8,R0,R4;   /* R8<=xa */
ldhu    R1,y;       /* R1<=y */
ldhu    R5,e;       /* e element load */
imul    R9,R1,R5;
add     R8,R8,R9;   /* R8<=xa+ye */
ldhu    R2,z;       /* R2<=z */
ldhu    R6,i;       /* i element load */
imul    R9,R2,R6;
add     R8,R8,R9;   /* R8<=xa+ye+zi */
ldhu    R3,w;       /* R3<=w */
ldhu    R7,m;       /* m element load */
imul    R9,R3,R7;
add     R8,R8,R9;   /* R8 <= final result *,
cmp     R8,Max_value_reg;
bc      gt,exceed_max;
cmp     R8,Min_value_reg;
bc      lt,exceed_min;
store:
sth     R8,result;  /* store the result */
b       next_step
exceed_max:
mv      R8,Max_value_reg;
b       store;
exceed_min:
mv      R8,Min_value_reg;
b       store;
next_step:
```

図6 4×4行列演算の例(汎用系命令のみ)

Fig. 6 An assembler program using native instructions only

全て汎用系命令を使った場合は，図6となるが，これは1要素分のデータしか処理出来ない．

5. 画像データ処理のサポート

5.1 概　　要

メディア処理における画像データは8ビットの符号なし整数値をとるのが一般的であるが，画像データに対して演算を行なう場合，この形式のままでは誤差が大きくなる．そのため，16ビット符号付き整数などのビット幅の広いデータに変換して計算し，最終的な値を再び8ビット符号なし整数にクランプする処理がよく行なわれる．しかしながら，この処理のプログラムを一般的な命令セットを用いて記述するのは，特に最終ステップの型変換の処理が繁雑になる．本プロセッサではこの処理を効率良く実行するため，符号モードおよびピクセルロード/ストア命令を実装した．

5.2 ピクセルロードストアの仕様

- 符号モード
  レジスタデータとメモリデータが独立に符号モード(signed/unsigned)を持つことができる．signedをS，unsignedをUと略記する．
- ピクセルロード命令
  メモリ上の8ビットデータを16ビットに変換してレジスタに格納する．変換方式は両者の符号モードによって規定される．
- ピクセルストア命令

レジスタモード	メモリモード	レジスタ条件	メモリ格納結果	
S	S	R[0]=0	R[1:8]= all 0	M[0]=0 M[1:7]= R[9:15]
S	S	R[0]=0	R[1:8]= not all 0	M[0]=0 M[1:7]= all 1
S	S	R[0]=1	R[1:8]= all 1	M[0]=1 M[1:7]= R[9:15]
S	S	R[0]=1	R[1:8]= not all 1	M[0]=1 M[1:7]= all 0
S	U	R[0]=0	R[1:7]= all 0	M[0:7]= R[8:15]
S	U	R[0]=0	R[1:7]= not all 0	M[0:7]= all 1
S	U	R[0]=1		M[0:7]= all 0
U	S		R[0:8]= all 0	M[0]=0 M[1:7]= R[9:15]
U	S		R[0:8]= not all 0	M[0]=0 M[1:7]= all 1
U	U		R[0:7]= all 0	M[0:7]= R[8:15]
U	U		R[0:7]= not all 0	M[0:7]= all 1

図7 ピクセルストアのデータ変換
Fig. 7 Data transfer with pixel store

```
/* Pixcel: unsigned 8bit */
Pixcel source[], result[];
short diff[];

for (i=0; i<SIZE; i++) {
  result[i] = (pixel)((short)source[i] + diff[i]);
}
```

図8 ピクセル演算の例 (C 言語)
Fig. 8 A C program of calculating pixel data

```
loop:
  mpldpixu  R0, source++;  /* load pixel data (4 PI data load) */
  muld64u   R2, diff++;    /* load diff data (4 short data load) */
  mpaddp    R4, R0, R2;    /* load diff data (4 short data load) */
  mpstpixu  R4, result++;  /* store pixel data (4 PI data store) */
  bc        cond--, loop;  /* loop until counter is over */
```

図9 ピクセル演算の例 (メディア命令)
Fig. 9 An assembler program using media instructions

レジスタ上の 16 ビットデータを 8 ビットに変換してメモリに格納する．変換方式は両者の符号モードによって規定される．

例えば，16 ビットレジスタデータ reg[0:15] から 8 ビットメモリデータ mem[0:7] への変換は，符号モードによって図 7のように変換される．

5.3 ピクセルロードストアの使用例

この中で最も頻繁に用いられるのは，レジスタ S，メモリ U のケースだと思われる．このモードに設定すると，図 8の画素値計算プログラムは図 9のように簡単に記述することができる．(図 9の 1 回のループで 4 個の画素データを処理している．)

全て汎用系命令を使った場合は，図 10となるが，これは 1 回のループで 1 個の画素データしか処理できない．

```
loop:
  ldbu  R0, source++;  /* load pixcel data (1 pixel data load) */
  ldhu  R1, diff++;    /* load diff data (1 short data load) */
  add   R2, R0, R1;    /* calclate (1 pixel data) */
  bc    lt,neg;        /* check minus */
  cmpi  R2, 255;
  bc    gt,sat;        /* check overflow */
store:
  stb   R2, result++;  /* store pixel data(1 pixel data store) */
  bc    cond--, loop;  /* loop until counter is over */
  b     end;
neg:
  zero_set  R2;        /* set zero if minus */
  b     store;
sat:
  255_set  R2;         /* set maximum if overflow */
  b     store;
end:
```

図10 ピクセル演算の例 (汎用系命令のみ)
Fig. 10 An assembler program using native instructions only

6. 可変長符号処理のサポート

6.1 概　　要

メディア処理の分野では，画像や音声データのような大量のデータを記憶装置に格納したり他の装置に転送したりする際には，何らかのデータ圧縮を行ない，資源の有効活用を図っている．データ圧縮の結果，メディアデータは可変長符号 (以下 VLC) となるのが一般的である．代表的な VLC に，ハフマン符号がある．

本プロセッサでは，これらの VLC を効率良く処理する仕組みを取り入れた．

表4 VLC サポート用特殊レジスタ
Table 4 Special register for VLC support

BOPR0(Bit OPeration Register 0)	32 ビット
VLC を格納するバッファレジスタの前半部	
BOPR1(Bit OPeration Register 1)	32 ビット
VLC を格納するバッファレジスタの後半部	
BBR(Bit Base Register)	32 ビット
VLC をロードストアするメモリアドレスを格納する	
BOFR(Bit OFfset Register)	7 ビット
バッファの空き領域のビット数を表す	
BSOR(Bit Shit Out Register)	32 ビット
バッファからシフトアウトされたビット列を格納する	

6.2 VLC サポート

本プロセッサでは VLC 処理のサポートのため，表 4及び表 5に示すレジスタ群と命令群を用意した．

これらを用いれば，1 個の VLC のデコードが終了して次の VLC をセットアップする処理を以下の 3 命令で記述することができる．

(1) mubs64 r
　r にはあらかじめ VLC のビット長をセットする
(2) mubld
(3) muls32c r'
　r' にはあらかじめ 32 をセットする

レジスタ上のデータの動きは以下のようになる．

状態 0. プログラム実行前
　BOPR0,BOPR1 をつなげた 64bit の領域に，n ビットの VLC が格納されている．先頭は BOPR0[0] にある．空き領域のサイズ，すなわ

表5 VLC サポート用専用命令
Table 5 Instructions for VLC support

mubld BOFR ≧ 32 のとき，BBR の値をアドレスとするメモリから 4byte のデータを BOPR1 にロードし，BBR の値を 4 だけ増やす． BOFR < 32 のときは何も行なわない．
mubs32c rb BOFR ≧ 32 のとき，BOPR1 の内容を $(BOFR-32)$ ビットだけ左シフトし，BOPR0 の LSB 側の $(BOFR-32)$ ビットをシフトアウトされたビットで置き換え，BOFR の値をオペランドの汎用レジスタ rb の値だけ減らす． BOFR < 32 のときは何も行なわない．
mubs64 rb BOPR0 と BOPR1 の内容を連続した 64 ビットデータとして，オペランドの汎用レジスタ rb の値だけ左シフトし，BOFR の値を rb の値だけ増やす． また，シフトアウトされたビットパターンを BSOR の LSB 側に詰めて書き込み，それ以外の $(32-(rb))$ ビットの領域には'0' を書き込む．

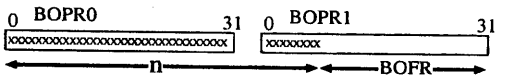

図11 ビットオペレーション初期状態
Fig. 11 An initial state of bit operation

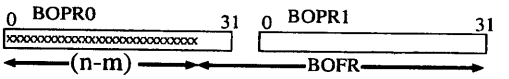

図12 ビットオペレーション状態 1
Fig. 12 Bit operation phase 1

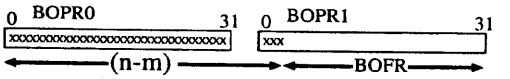

図13 ビットオペレーション状態 2
Fig. 13 Bit operation phase 2

ち $(64-n)$ の値は BOFR が保持している．また，次にロードすべきデータアドレスは BBR が指している．(図 11)

状態 1.1 mubs64 実行後 (1)
BOPR1 のデータがなくなる場合．(図 12)
m はオペランドで指定されたシフト量．

状態 1.2 mubs64 実行後 (2)
BOPR1 にデータが残る場合．(図 13)
m はオペランドで指定されたシフト量．

状態 2.1 mubld 実行後 (1)
BBR は自動的に +4 され，次のデータを指す．(図 14)

状態 2.2 mubs64 実行後 (2)
$(BOFR < 32)$ なので何もしない．(図 13)

状態 3.1 mubs32c 実行後 (1)

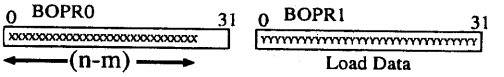

図14 オペレーション状態 3
Fig. 14 Bit operation phase 3

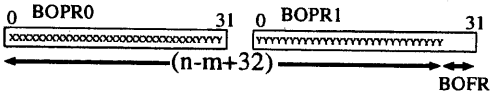

図15 ビットオペレーション状態 4
Fig. 15 Bit operation phase 4

オペランドでは 32 を指定する．(図 15)

状態 3.2 mubs64 実行後 (2)
$(BOFR < 32)$ なので何もしない．(図 13)

状態 3.1，状態 3.2 ともに初期状態 (状態 0.) と意味的に同じ状態になっている．従来の命令セットでは，0. → 1.1 → 2.1 → 3.1 のパスと 0. → 1.2 → 2.2 → 3.2 のパスは条件分岐によってプログラム上では別のパスにならざるを得ないが，本プロセッサでは同一の命令列 (しかも僅か 3 命令) で実現可能である．

この例はデコード時のものであるが，エンコードの場合は，別に用意されている専用命令を用いれば，同様に効率の良いプログラムが記述できる．

7. おわりに

本プロセッサの基本アーキテクチャと特徴について述べた．本プロセッサは，汎用のプロセッサコアにメディア処理に特化した命令セットを拡張することで，コンパクトで高性能な組み込み用途のメディア処理を行なうようデザインされている．

また，我々は本プロセッサを組み込み用マイクロプロセッサに関するノウハウを蓄積するための評価用プラットフォームとして位置付けており，現在もその観点からの評価を継続中である．

謝辞 日頃ご指導頂く，(株) 富士通研究所システム LSI 研究所安藤主管研究員，及び，同第二開発プロジェクト部高橋主任研究員に感謝いたします．

参考文献

1) 久保沢ほか: "A 1.2W, 2.1GOPS/720MFLOPS Embedded Superscalar Microprocessor for Multiedia Application", 1998 IEEE International Solid-State Circuits Conference (ISSCC 98), San Francisco, Vol.41, pp.290-295.
2) 須賀ほか: "A 1.2W, 2.1GOPS/720MFLOPS Embedded Superscalar Microprocessor for Multiedia Application", 電子情報通信学会 集積回路研究会 (ICD), 会津, 1998.4.