計算機科学実験３A
機能設計仕様書

チーム４：竹田原俊介　Chung Mung Tim
Chung Mung Tim

2022-05-12

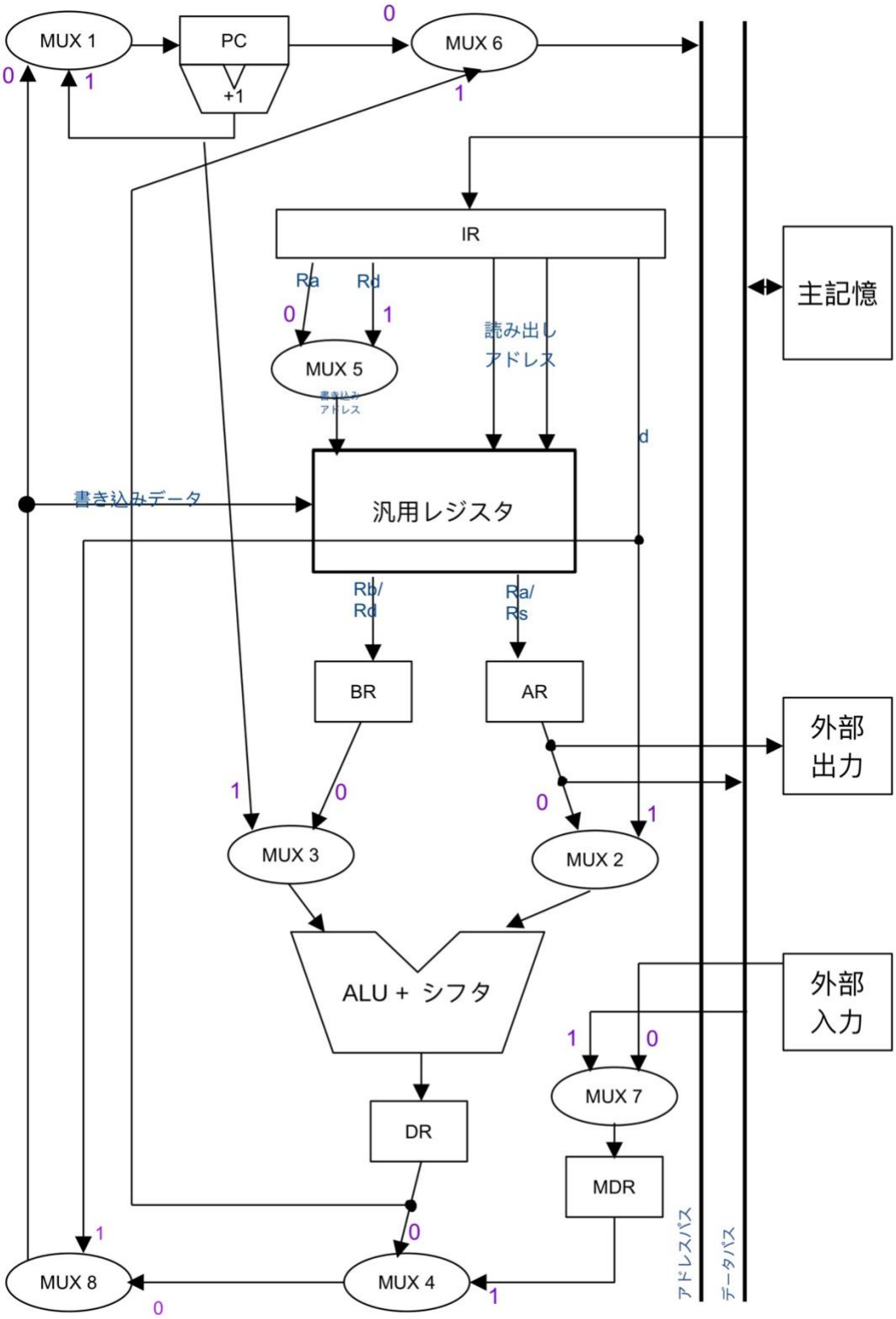
第１章　概要

図１は私たちが設計されたSIMPLEの全体のブロック図です．その中，私が担当するコンポーネントは以下である：

1. レジスタ（IR, AR, BR, DR, MDR）
2. 汎用レジスタ
3. 制御部（図３を参照する）
4. ALU制御部（図３を参照する）

以下のレポートでは，これらの部品の外部仕様や内部仕様，CPU全体の中で果たす役割について詳しく説明する．

図１：SIMPLEのブロック図



第２章　レジスタ

SIMPLEで，IR，AR，BR，DR，MDRの6つの16ビット幅レジスタを使用している。レジスタの機能は，入力された一つ値を記憶し，次にクロックが上がったに出力することである．それを実現するために，レジスタの外部仕様は以下である：
入力：

- クロック信号　reg_e
- イネーブラ信号 reg_write_en
- 16ビットデータ reg_in

出力：

- 16ビットデータ reg_out

レジスタのVerilog HDLコードは以下である：

```
module register_16(reg_e, reg_write_en, reg_in, reg_out);
	input			reg_e;
	input			reg_write_en;
	input		[15:0] reg_in;
	output	reg [15:0] reg_out;

	always @(posedge reg_e) begin
		if(reg_write_en) begin
			reg_out <= reg_in;
			end

endmodule
```

予期せぬ動作を防ぐため，デザインで使用される6つのレジスタはすべて同じクロックを共有することになる．ただし、各レジスタにはイネーブラ信号がある．入力は常にクロック信号に沿って更新されるが，イネーブラ信号が1になったときのみ出力が更新される．

第３章　汎用レジスタ

SIMPLEは８個の汎用レジスタが備えられ，r[0], r[1], …
r[7]と表記される．一つのレジスタは16ビット幅である．命令演算のロース／デスティネーションや主記憶のアドレス計算に以下の通り用いられる．

1. IR（Instruction
   register）にある命令の適切なフィールドを読み出しアドレスとして，最大二つの16ビットデータを読み出す．
2. 計算結果や保存したいデータを汎用レジスタ書き込む．１クロックごとに最大一つの16ビットデータが書き込める．

それを実現するために，汎用レジスタの外部仕様は以下である：
入力：

- クロック信号clk
- リセット信号rst
- 書き込みイネーブラ信号reg_write_en
- 書き込みデータデスティネーションreg_write_dest（8個あるので３ビット）
- 書き込み16ビットデータreg_write_data
- 読み出しデータアドレス１ reg_read_addr_1
- 読み出しデータアドレス２ reg_read_addr_2

出力：

- 読み出した16ビットデータ１ reg_read_data_1
- 読み出した16ビットデータ２ reg_read_data_2

汎用レジスタのVerilog HDLコードは以下である：

```
module register_general( //
    input clk,
    input rst,
    //  データを書き込む
    input     reg_write_en, // データをエントリーする信号
    input     [2:0] reg_write_dest,
    input     [15:0] reg_write_data,
    //  データを読み込む
    input     [2:0] reg_read_addr_1,
    output    [15:0] reg_read_data_1,
    //  データを読み込む
    input     [2:0] reg_read_addr_2,
```

```
    output    [15:0] reg_read_data_2

);
    reg         [15:0] reg_array [7:0]; // 8 registers

    always @ (posedge clk) begin    // 書き込むは順序回路
        if(rst) begin // 全てのデータを削除
            reg_array[0] <= 16'b0;
            reg_array[1] <= 16'b0;
            reg_array[2] <= 16'b0;
            reg_array[3] <= 16'b0;
            reg_array[4] <= 16'b0;
            reg_array[5] <= 16'b0;
            reg_array[6] <= 16'b0;
            reg_array[7] <= 16'b0;
        end else begin
            if(reg_write_en) begin
                reg_array[reg_write_dest] <= reg_write_data;
            end
        end
    end

    assign reg_read_data_1 = reg_array[reg_read_addr_1];    // 読み出すは組み合わせ回路
    assign reg_read_data_2 = reg_array[reg_read_addr_2];

endmodule
```

16ビットデータ8個の配列が宣言されている．書き込みの部分が順序回路になっているが，読み出しの部分が組み合わせ回路である．イネーブラ信号が1に設定されている場合のみ，16ビット入力データが配列内の正しいアドレスに書き込まれる．

　また，リセット信号が１になる時，配列の中の全てのデータが０になる．

図２は設計のブロック図の中汎用レジスタに関する部分である．マルチプレクサMUX５は，書き込みしたいアドレスを選択する．命令フォマットによって，書き込みアドレスを示すフィールドの場所が違うので，MUX５が必要である．（例：演算命令での書き込み場所は命令の第８～10ビットであるが，ロード命令のが第11～13ビットである．

図２：汎用レジスタの周り

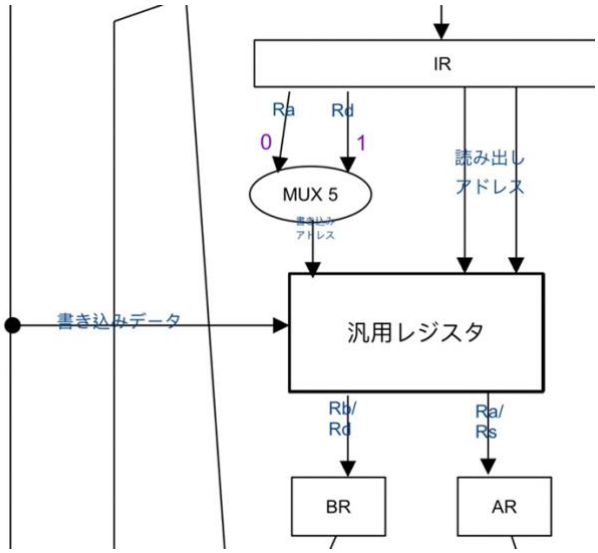
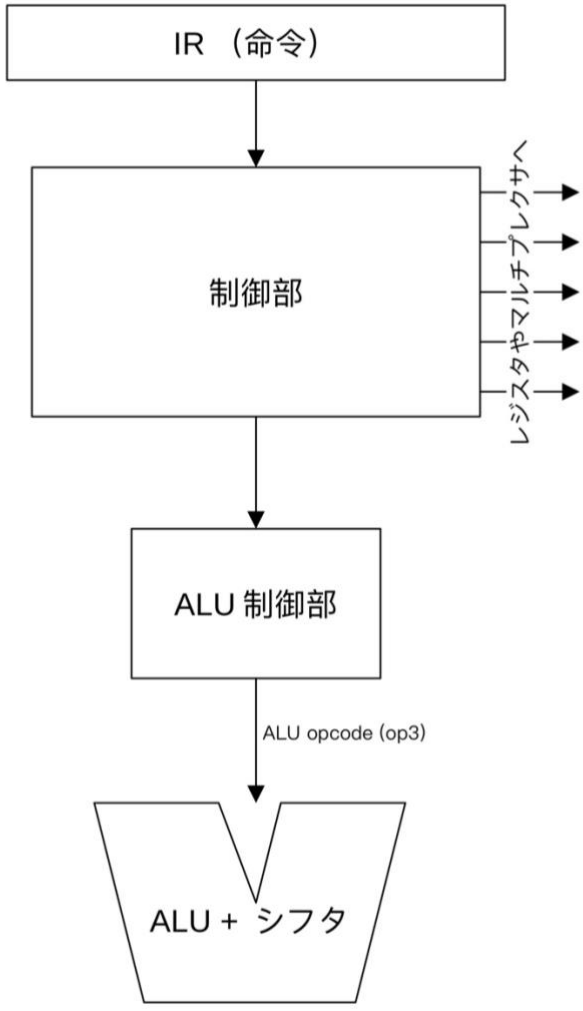


図３：SIMPLEの制御回路



第４章　制御部

制御部に必要な機能を列挙する．

1. 必要な部品に制御信号・イネブラー信号・セレクタ信号を送ること．
2. ５フェーズの逐次活性化すること．
3. プロセッサの起動・停止（exec），リセット(reset)を制御する．

５フェーズについて説明すると，SIMPLEプロセッサへの命令は，５つのフェーズを順番に活性化しながら実行する．その５つのフェーズは以下である：

1. Phase１（命令フェッチ）
   PC（Program Counter）が保持するアドレスの命令を主記憶からフェッチし，IR（Instruction Register）に格納すると共に，PC に１を加える．
2. Phase２（レジスタ読み出し）
   IR が保持する命令の，Ra /Rs フィールドと Rb /Rd フィールドで指定される汎用レジスタの値を読み出し，それぞれをレジスタ AR と BR に格納する．
3. Phase３（演算）
   ALU またはシフト回路により，命令が定める演算（アドレス渓山や単なるデータ移動を含む）を行い，その結果をレジスタ DR（Data Register）に格納する．また演算命令では条件コード S, Z, C, V をセットする．演算のソースには AR, BR の他，PC や命令の即値が用いられることがある．
4. Phase４（主記憶アクセス）
   ロード／ストア命令では，DR の値をアドレスとして主記憶をアクセスする．ロード命令では読み出し値をレジスタ MDR（Memory Data Register）に格納し，ストア命令では AR が保持する値を書き込む．また入出力命令では，入力スイッチの値の MDR への格納や，AR の値の 7SEG LED などへの表示を行う．
5. Phase５（レジスタ書き込み）
   汎用レジスタの書き込みを伴う命令では，DRまたはMDRの値を，命令のRa，Rb，またはRsフィールドで指定される汎用レジスタに書き込む．また分岐命令ではDRの値を分岐先アドレスとしてPCに書き込む．

また，execとresetの機能は以下である：

reset（リセット信号）

- FPGA ボード上のプッシュスイッチを用いて，スイッチを押すと１が，離すと０が供給されるようにする．reset が１になると，PC 等をクリアし，初期状態に移行する．

exec（起動／停止信号）

- FPGA ボード上のプッシュスイッチを用いて，スイッチを押すと 1 が，離すと 0 が供給されるようにする．
- SIMPLE が停止状態にある時に exec が 0 から 1 に変化すると，SIMPLE は命令の実行を開始する．
- SIMPLE が実行状態にある時に exec が 0 から 1 に変化すると，その時点で実行中の命令が完了してから停止する．

組み合わせ回路の制御部と順序回路の制御部の両方作っている．制御部の種類によって、必要なトップレベルモジュールも異なる．以下では、両種類の制御部とも紹介する．

4.1 組み合わせ回路の制御部

組み合わせ回路の制御部にはクロックが入れていない．この時点でパイプライン化はまだ考えていないので，制御部はフェーズ 1 から 5 までずっと同じ信号を出力してもよいである．つまり，この命令の実行が終了するまで，同じ信号を出力し続けるのである．

この実装では，フェーズ 0 から 5 まで（初期フェーズ0を含む）カウントしてフェーズを追跡するフェーズカウンタが必要である．それはトップレベルのモジュールが担当する．

また，制御部がフェーズを管理しないので，execのようなフェーズに関する情報が必要な信号は，トップレベルが管理することがやりやすい．なぜから，フェーズカウンターがトップレベルで実装されているからである．

トップレベル*simple.v*のVerilog HDLコードの一部を以下に示す：

```
reg[2:0]cnt=3'b000;
reg executing = 0; // 実行中・停止中を表す
reg stop_flag = 0; // if stop_flag == 1, then stop after this instruction


// 3'b000: 初期状態, 3'b001: Phase 1 , 3'b010: Phase 2, ...
always@(posedge clk)begin
    if(rst)begin
        phase <= 3'b000;
        executing <= 0;
```

```
        end else begin
            if (phase == 3'b000) begin // if Phase 0
                if ( (executing==0 & exec) || (executing & exec==0) ) begin
                    phase <= 3'b001;
                    executing <= 1;
                end else begin
                    phase <= 3'b000; //stay in 初期状態
                end
            end
            if (executing & exec) begin
                stop_flag <= 1;
            end
            pc_e <= 1'b0;
            phase <= phase + 3'b001;
            if(phase == 3'b101)begin // if Phase 5
                if(stop_flag) begin
                    phase <= 3'b000;
                    executing <= 0;
                end else begin
                phase <= 3'b000;
                pc_e <= 1'b1;
                MEI <= meirei;
                end
            end
        end
    end
```

ここで，

- cntはフェーズを表すregisterである．つまり，初期状態フェーズ 0 の時，cnt = 3’b000, フェーズ 1 の時，cnt = 3’b001，など．
- executingはプロセッサは今実行しているか，または停止しているかを表すフラフである．
- stop_flagというregisterは，現在実行中の時にexecが 0 から 1 になると，stop_flagは 1 になる．つまり，stop_flagは「この命令が終わると停止する」という意味を表すフラグである．
- phaseという変数は現在はフェーズ数を表す．

このコードは制御部のexecとreset機能を管理するトップレベルの部分である．ここでのポイントは制御部なので、これ以上の詳しい説明は割愛する．

組み合わせ回路の制御部*control_combination.v*のコードが長すぎるので，ここでは重要な部分のみ抜粋して説明する．

外部仕様として，この制御部の入出力は以下である：
入力

- リセット信号rst
- 起動・停止信号exec
- フェーズを表す 3 ビット信号phase
- 条件コードS, Z, C, V
- 16ビット命令instruction

出力

- ALU制御部への信号 aluc_e
- レジスタへのイネブラー信号 ar_e, br_e, dr_e, mdr_e, ir_e
- レジスタへのクロック信号 reg_e
- 汎用レジスタの書き込みイネブラー信号 genr_w
- 主記憶の読み出し信号 mem_e
- 主記憶の書き込み信号 mem_w
- プログラムカウンタのジャンプ信号jump
- マルチプレクサへのセレクタ信号 m2_s, m3_s, m4_s, m5_s, m6_s, m7_s, m8_s
- ALU制御部への 6 ビット操作コード alu_instruction

```
    input rst, exec; // exec is 0 by default
    input [2:0] phase;
    input S, Z, C, V;
    input [15:0] instruction;
    output reg    aluc_e,
                  ar_e,br_e,dr_e,mdr_e,ir_e,
                  reg_e,
                  genr_w,
                  //pc_e,
                  mem_e, mem_w,
                  jump,m2_s,m3_s,m4_s, m5_s, m6_s, m7_s, m8_s;
    output [5:0] alu_instruction; // ALU制御部へ
```

制御部本文の部分はalways
@(＊)を使い，自動的にセンシティブリストを作る．本文の部分は，大きく命令別に分かれ，命令によって各部品への制御信号やセレクタ信号を変わる．それを実現

するために，以下のコードのように，入力16ビット命令をデコードし，commandという５ビット変数を作成する．条件分岐もここでする．commandは全ての命令を表すことである．

```
// set the value of "command" depending on the instruction
        case(op)
            2'b11: command <= {1'b0,alu_op}; // ALU
            2'b00: command <= 5'b10000; //LD    r[Ra]=*(r[Rb]+sign_ext(d))
            2'b01: command <= 5'b10001; //ST    *(r[Rb]+sign_ext(d))=r[Ra]
            2'b10: begin
                case(r1)
                    3'b000: command <= 5'b10010; //LI r[Rb]=sign_ext(d)
                    3'b100: command <= 5'b10011; //B PC=PC+1+sign_ext(d)
                    3'b111: begin
                        case(r2)
                            3'b000: if(Z) command <= 5'b10100; //BE
                            3'b001: if (S^V) command <= 5'b10101; //BLT
                            3'b010: if (Z||(S^V)) command <= 5'b10110; //BLE
                            3'b011: if (!Z) command <= 5'b10111; //BNE
                        endcase
                    end
                endcase
            end
        endcase
```

まず，全ての制御信号を初期化をし，０にする．その後，現在のフェーズを表すphaseという変数は3’b000であれば，またはリセットされたら，初期化に戻す．そじゃなければ，命令commandによって，それぞれの部品への制御信号が変わる．

4.2 順序回路の制御部

次は順序回路の制御部を紹介していく．順序回路の制御部*control.v*のコードが長すぎるので，ここでは重要な部分のみ抜粋して説明する．この制御部はクロック同期に動き，クロックを入れている．

外部仕様として，この制御部の入出力は以下である：
入力

- クロック信号clk
- リセット信号rst

- 起動・停止信号exec
- 条件コードS, Z, C, V
- 16ビット命令instruction

出力

- ALU制御部への信号 aluc_e
- レジスタへのイネブラー信号 ar_e, br_e, dr_e, mdr_e, ir_e
- レジスタへのクロック信号 reg_e
- 汎用レジスタの書き込みイネブラー信号 genr_w
- 主記憶の読み出し信号 mem_e
- 主記憶の書き込み信号 mem_w
- プログラムカウンタのジャンプ信号jump
- マルチプレクサへのセレクタ信号 m2_s, m3_s, m4_s, m5_s, m6_s, m7_s, m8_s
- ALU制御部への６ビット操作コード alu_instruction

```
input clk, rst, exec; // exec is 0 by default
input S, Z, C, V;
input [15:0] instruction;
output reg    aluc_e,
                ar_e,br_e,dr_e,mdr_e,ir_e,
                reg_e,
                genr_w,
                pc_e,
                mem_e, mem_w,
                jump,m2_s,m3_s,m4_s, m5_s, m6_s, m7_s, m8_s;
output [5:0] alu_instruction; // ALU 制御部へ
```

この場合，フェーズカウンターの仕事も制御の中に入っている．つまり，フェーズの管理，exec，reset 機能の実行も制御部がすることである．

制御部の本文は大きく６フェーズに分けられている．それは，全ての制御信号が０である初期状態のフェーズ０と，第４章の最初に紹介された５つのフェーズである．クロックが上がるたびに，次のフェーズに移行する．フェーズ０から１までは，命令にかかわらず制御信号が同じのため，まとめて書いてある．その以降のフェーズでは命令 command によって，各部品へ送る制御信号が変わる．

また，reset と exec 命令は以下のように実装されている：

- phaseという３ビット変数を使って，現在にいるフェーズを表す．

- executingは現在実行しているかを表すフラフである．実行していたら１，停止していたら０．フェーズ５にいるとき，もしexecutingが１であれば，フェーズ０に戻り，次の命令を実行していく．
- stop_flagは「この命令が終わると停止する」という意味を表すフラグである．

リセットが１になったら，初期状態に戻す．

```
// reset
        if(rst) phase <= 3'b000; // if rst==0, then  初期状態に戻す
```

現在実行中かつ exec が１になったら，この命令が終わると SIMPLE を停止する．

```
// exec
        if(executing & exec) begin // if executing and exec==1
            stop_flag <= 1; // stop after this instruction
        end
```

初期状態にいる時，停止中かつ exec が１の時に，実行し初め，次のフェーズに移動する．実行中かつ exec が０の時にも実行する．そうではなければ，初期状態のままにする．

```
3'b000:begin //初期状態
                if( (executing==0 & exec) || (executing & exec==0)) begin
                    phase <= 3'b001; //次の phase に行く
                    executing <= 1;
                end else begin
                    phase <= 3'b000; //stay in  初期状態
                end
```

最後に，フェーズ５にいる時に，もしその時点で stop_flag が１であれば，実行を停止する．

```
if(stop_flag) begin
                    executing <= 0;
                    end
            end
```

第５章　ALU 制御部

図２を参照してください．ALU制御部はSIMPLEの制御回路の一部であり，制御部から６ビットの入力を受け取り，それをデコードし，適当な演算を選び，ALUへ４ビットの操作コードを出力する．なので，外部仕様は以下である：

入力：

- クロック信号 alu_control_unit_e
- ６ビットの入力命令 instruction_six

出力：

- ４ビットの操作コード ALU_Cnt;

ALU制御部のVerilog HDLコードは以下である：

```
module alu_control_unit(alu_control_unit_e, instruction_six, ALU_Cnt);
	input alu_control_unit_e;
	input [5:0] instruction_six; //制御部から
	output reg [3:0] ALU_Cnt;

	always @(posedge alu_control_unit_e) begin
		casex (instruction_six)
			6'b00xxxx: ALU_Cnt=4'b0000; // load --> add
			6'b01xxxx: ALU_Cnt=4'b0000; // store --> add

			6'b10111x: ALU_Cnt=4'b0000; // 条件分岐命令 --> add

			// ALU演算・入出力命令: 11で始まるやつ
			6'b110000: ALU_Cnt=4'0000;
			6'b110001: ALU_Cnt=4'0001;
			6'b110010: ALU_Cnt=4'0010;
			6'b110011: ALU_Cnt=4'0011;
			6'b110100: ALU_Cnt=4'0100;
			6'b110101: ALU_Cnt=4'0001; // CMP --> SUB
			6'b110110: ALU_Cnt=4'0110;
			6'b110111: ALU_Cnt=4'0111;
			6'b111000: ALU_Cnt=4'1000; // Shift
			6'b111001: ALU_Cnt=4'1001;
			6'b111010: ALU_Cnt=4'1010;
			6'b111011: ALU_Cnt=4'1011;
```

```
                6'b111100: ALU_Cnt=4'1100;
                6'b111101: ALU_Cnt=4'1101;
                6'b111110: ALU_Cnt=4'1110;
                6'b111111: ALU_Cnt=4'1111;

                default: ALU_Cnt=4'b0000;
            endcase
        end

endmodule
```

このコードでは，casex文を使い、すべての可能の入力とそれに対応する出力を網羅する．特に注目したいのは，CMP命令（比較命令）の時，出力の操作コードはSUB命令（減算）を表す0001である．これは，CMP命令におけるの演算は減算だからである．そして，ロード命令，ストア命令と条件分岐命令におけるALUの実際の演算は加算なので，出力はADD命令（加算）を表す0000である．その以外は全て６ビット入力の下位４ビットを出力だけである．