```
module EX

//input
        input clock,
        input reset,
        input [15:0] PC,
        input [2:0] WBaddress,
        input [5:0] control,          // 5 ALUSrc , 4 Branch
        input [3:0] ALUcontrol,
        input [15:0] immidiate,
        input [15:0] Rs_Ra,Rd_Rb,
//output
        output reg [15:0] IMnextPC_,
        output reg branch_,
        output reg [2:0] WBaddress_,
        output reg [3:0] control_,
        output reg [15:0] result_,
        output reg [15:0] Ra_,
        output reg [3:0] ConditionCode_   // S,Z,C,V

////////////////////////////////////////////////////////////////////

// 機能
        計算機能を提供するフェーズ EX を提供する。
        出力は clock ごとに同期される。

////////////////////////////////////////////////////////////////////

//input 詳細
        clock           クロック信号
        reset           リセット信号。 clock 立ち上がり時0でレジスタを0に更新
        PC              16 bit (次の)プログラムカウンタの値
        WBaddress       通常： 3 bit 書き込み用アドレス
                        Branch 命令時： ブランチ命令の 3bit 判定種類コード cond
        control         6bit 制御コード。 EX で使用するのは以下の2つ
                                第 5bit ALUSrc      ALU 第2入力の選択
                                                    (0: Rs_Ra , 1: immidiate)
                                第 4bit Branch      Branch 命令のフラグ
                                                    1の際出力 branch_が有効化
        ALUcontrol      4bitALU 制御信号。機能対応は SIMPLE 仕様書参照
        immidiate       16bit 即値。 IMnextPC の計算と第2入力候補として使用
        Rs_Ra           ALU への 16bit 第2入力候補。 ロード、ストアの書き込み用アドレス。
```

Rd_Rb　　ALU への 16bit 第1入力

//output 詳細

IMnextPC_　　16bit の即値を加算した次の PC 候補。 branch_有効時次の PC として使用。

branch_　　1bit の次の PC の選択制御信号。 1 の際 IMnextPC_を使用。
内部的には第4制御信号 *Branch* 有効時に、ブランチ命令の判定種類と SZCV フラグの対応によって決定。

WBaddress_　　3bit 書き込み用アドレス

control_　　次フェーズ以降で使用する 4bit 制御コード

result_　　16bit の ALU 演算結果

Ra_　　ロード、ストアの書き込み用アドレス

ConditionCode_　　4bit SZCV フラグ。 現在のところ外部での使用用途はない。

//
// コンパイル結果

Flow Status Successful - Sun May 7 18:10:56 2017
Quartus II 32-bit Version 13.0.1 Build 232 06/12/2013 SP 1 SJ Full Version
Revision Name EX
Top-level Entity Name EX
Family Cyclone IV E
Device EP4CE30F23I7
Timing Models Final
Total logic elements 438 / 28,848 (2 %)
Total combinational functions 438 / 28,848 (2 %)
Dedicated logic registers 43 / 28,848 (< 1 %)
Total registers 43
Total pins 139 / 329 (42 %)
Total virtual pins 0
Total memory bits 0 / 608,256 (0 %)
Embedded Multiplier 9-bit elements 0 / 132 (0 %)
Total PLLs 0 / 4 (0 %)

//
Flow Status Successful - Mon Apr 24 15:30:14 2017
Quartus II 32-bit Version 13.0.1 Build 232 06/12/2013 SP 1 SJ Full Version
Revision Name EX
Top-level Entity Name EX

Family Cyclone IV E
Device EP4CE30F23I7
Timing Models Final
Total logic elements 424 / 28,848 (1 %)
Total combinational functions 401 / 28,848 (1 %)
Dedicated logic registers 60 / 28,848 (< 1 %)
Total registers 60
Total pins 138 / 329 (42 %)
Total virtual pins 0
Total memory bits 0 / 608,256 (0 %)
Embedded Multiplier 9-bit elements 0 / 132 (0 %)
Total PLLs 0 / 4 (0 %)

Fmax (10ns) 651.47 MHz
restricted Fmax 250.0 MHz

```
////////////////////////////////////////////////

module EX(
//input
        input clock, reset,
        input [15:0] PC,
        input [2:0] WBaddress,
        input [5:0] control,        // 5 ALUSrc , 4 Branch
        input [3:0] ALUcontrol,
        input [15:0] immidiate,
        input [15:0] Rs_Ra,Rd_Rb,
//output
        output [15:0] IMnextPC_,
        output branch_,
        output reg [2:0] WBaddress_,
        output reg [3:0] control_,
        output reg [15:0] result_,
        output reg [15:0] Ra_,
        output reg [3:0] ConditionCode_ // S,Z,C,V
        );

        parameter S = 3,
                                Z = 2,
                                C = 1,
                                V = 0,
```

```
                                ALUSrc = 5,
                                Branch = 4;

//unclocked moving
assign IMnextPC_ = PC + immidiate;  //immidiate jump
assign branch_ = branch;

// clock moving

always @(posedge clock)
begin
        if(reset == 1)begin
                WBaddress_ <= WBaddress;                        //WDaddress
                control_ <= control[3:0];
                result_ <= result;
                Ra_ <= Rs_Ra;
                ConditionCode_[S] <= result[15];
                ConditionCode_[Z] <= (result == 16'b0);
                ConditionCode_[C] <= codeC;
                ConditionCode_[V] <= codeV;
        end else begin
                WBaddress_ <= 3'b000;
                control_ <= 4'b0000;
                result_ <= 16'b0000_0000_0000_0000;
                Ra_ <= 16'b0000_0000_0000_0000;
                ConditionCode_ <= 4'b0000;
        end
end



//branch dicidion

        wire branch = control[Branch]
                                        &&              branch_evaluate(WBaddress,

        ConditionCode_[S],

        ConditionCode_[Z],

        ConditionCode_[V]);
```

```
        function branch_evaluate(
                input [2:0] cond,
                input nowS,
                input nowZ,
                input nowV);
         begin
                case(cond)
                        3'b000: branch_evaluate = nowZ;
        //BE
                        3'b001: branch_evaluate = nowS ^ nowV;                  //BLT
                        3'b010: branch_evaluate = nowZ||(nowS^nowV);    //BLE
                        3'b011: branch_evaluate = !nowZ;
        //BNE
                        3'b100: branch_evaluate = 1;                                                    //B
                        3'b101: branch_evaluate = 0;
                        3'b110: branch_evaluate = 0;
                        3'b111: branch_evaluate = 0;
                endcase
         end
        endfunction


//ALU calculation

        wire [15:0] valueA,valueB;
        assign valueA = Rd_Rb;
        assign valueB = (control[ALUSrc] == 0)? Rs_Ra : immidiate;
        wire [15:0] result;
        reg codeC=0, codeV=0;

        assign result = ALU(ALUcontrol,valueA,valueB);


        //ALU
        function [15:0] ALU(
                input [3:0] ALUctl,
                input [15:0] A,B
        );
         begin
                reg [16:0] ans;
                case(ALUctl)
                        // +
```

```
                4'b0000 : begin ans = {1'b0,A} + {1'b0,B};
                                                codeC = ans[16];
                                                codeV = (A[15]&B[15]&~ans[15])
                                                                                |
(~A[15]&~B[15])&ans[15];
                                                ALU = ans[15:0]; end
                // -
                4'b0001 : begin ans = {1'b0,A} - {1'b1,B};
                                                codeC = ans[16];
                                                codeV = (A[15]&~B[15]&~ans[15])
                                                                                |
(~A[15]&B[15]&ans[15]);
                                                ALU = ans[15:0]; end
                // &
                4'b0010 : begin ALU = A & B;
                                                codeC = 0;
                                                codeV = 0; end
                // |
                4'b0011 : begin ALU = A | B;
                                                codeC = 0;
                                                codeV = 0; end
                // ^
                4'b0100 : begin ALU = A ^ B;
                                                codeC = 0;
                                                codeV = 0; end
                // -(compare)
                4'b0101 : begin ans = {1'b0,A} - {1'b1,B};
                                                codeC = ans[16];
                                                codeV = (A[15]&~B[15]&~ans[15])
                                                                                |
(~A[15]&B[15]&ans[15]);
                                                ALU = ans[15:0]; end
                // through
                4'b0110 : begin ALU = B;
                                                codeC = 0;
                                                codeV = 0;
                                        end
                // ###
                4'b0111 : ALU = 0;
                // shift_left_logical
                4'b1000 : begin ans = shift_left_logical(A,B);
                                                codeC = ans[16];
```

```
                                        codeV = 0;
                                        ALU = ans[15:0];
                                end
                // shift_left_rotate
                4'b1001 : begin ans = shift_left_rotate(A,B);
                                        codeC = ans[16];
                                        codeV = 0;
                                        ALU = ans[15:0];
                                end
                // shift_right_logical
                4'b1010 : begin ans = shift_right_logical(A,B);
                                        codeC = ans[16];
                                        codeV = 0;
                                        ALU = ans[15:0];
                                end
                // shift_right_arithmetic
                4'b1011 : begin ans = shift_right_arithmetic(A,B);
                                        codeC = ans[16];
                                        codeV = 0;
                                        ALU = ans[15:0];
                                end
                // through(input)
                4'b1100 : begin ALU = B;
                                        codeC = 0;
                                        codeV = 0;
                                end
                // through(output)
                4'b1101 : begin ALU = B;
                                        codeC = 0;
                                        codeV = 0;
                                end
                // ##
                4'b1110 : ALU = 0;
                // through(halt)
                4'b1111 : begin ALU = B;
                                        codeC = 0;
                                        codeV = 0;
                                end
        endcase
 end
endfunction
```

```
//shifter
	//left logical (16...C)
	function [16:0] shift_left_logical(
		input [15:0] num,
		input [3:0] shift
	);
	begin
		reg [16:0] n[0:2];
		n[2] = (shift[3]==1)? ({num[8],num[7:0],{8{1'b0}}}) : ({1'b0,num});
	//8bit
		n[1] = (shift[2]==1)? {n[2][12],n[2][11:0],{4{1'b0}}} : n[2];		//4bit
		n[0] = (shift[1]==1)? {n[1][14],n[1][13:0],{2{1'b0}}} : n[1];		//2bit
		shift_left_logical
			= (shift[0]==1)? {n[0][15],n[0][14:0],1'b0} : n[0];  //1bit
	end
	endfunction

	//left rotate (16...C =0)
	function [16:0] shift_left_rotate(
		input [15:0] num,
		input [3:0] shift
	);
	begin
		reg [15:0] n[0:2];
		n[2] = (shift[3]==1)? {num[7:0],num[15:8]}   : num;			//8bit
		n[1] = (shift[2]==1)? {n[2][11:0],n[2][15:12]} : n[2];		//4bit
		n[0] = (shift[1]==1)? {n[1][13:0],n[1][15:14]} : n[1];		//2bit
		shift_left_rotate
			= (shift[0]==1)? {1'b0,n[0][14:0],n[0][15]} : {1'b0,n[0]};  //1bit
	end
	endfunction

	//right logical (16... C)
	function [16:0] shift_right_logical(
		input [15:0] num,
		input [3:0] shift
	);
	begin
		reg [16:0] n[0:2];
		n[2] = (shift[3]==1)? {num[7],{8{1'b0}},num[15:8]} : {1'b0,num};
```

```
        //8bit
                n[1] = (shift[2]==1)? {n[2][3],{4{1'b0}},n[2][15:4]} : n[2];  //4bit
                n[0] = (shift[1]==1)? {n[1][1],{2{1'b0}},n[1][15:2]} : n[1];  //2bit
                shift_right_logical
                        = (shift[0]==1)? {n[0][0],{1{1'b0}},n[0][15:1]} : n[0];       //1bit
        end
        endfunction

        //right arithmetic (16... C)
        function [16:0] shift_right_arithmetic(
                input [15:0] num,
                input [3:0] shift
        );
        begin
                reg [16:0] n[0:2];
                n[2] = (shift[3]==1)? {num[7],{8{num[15]}},num[15:8]} : {1'b0,num};
        //8bit
                n[1] = (shift[2]==1)? {n[2][3],{4{num[15]}},n[2][15:4]} : n[2];     //4bit
                n[0] = (shift[1]==1)? {n[1][1],{2{num[15]}},n[1][15:2]} : n[1];     //2bit
                shift_right_arithmetic
                        = (shift[0]==1)? {n[0][0],{1{num[15]}},n[0][15:1]} : n[0];
        //1bit
        end
        endfunction


endmodule
```