C.機能設計仕様書
グループ4　竹田原俊介　1029324054

方式設計仕様書のブロック図は下図1、図2です。

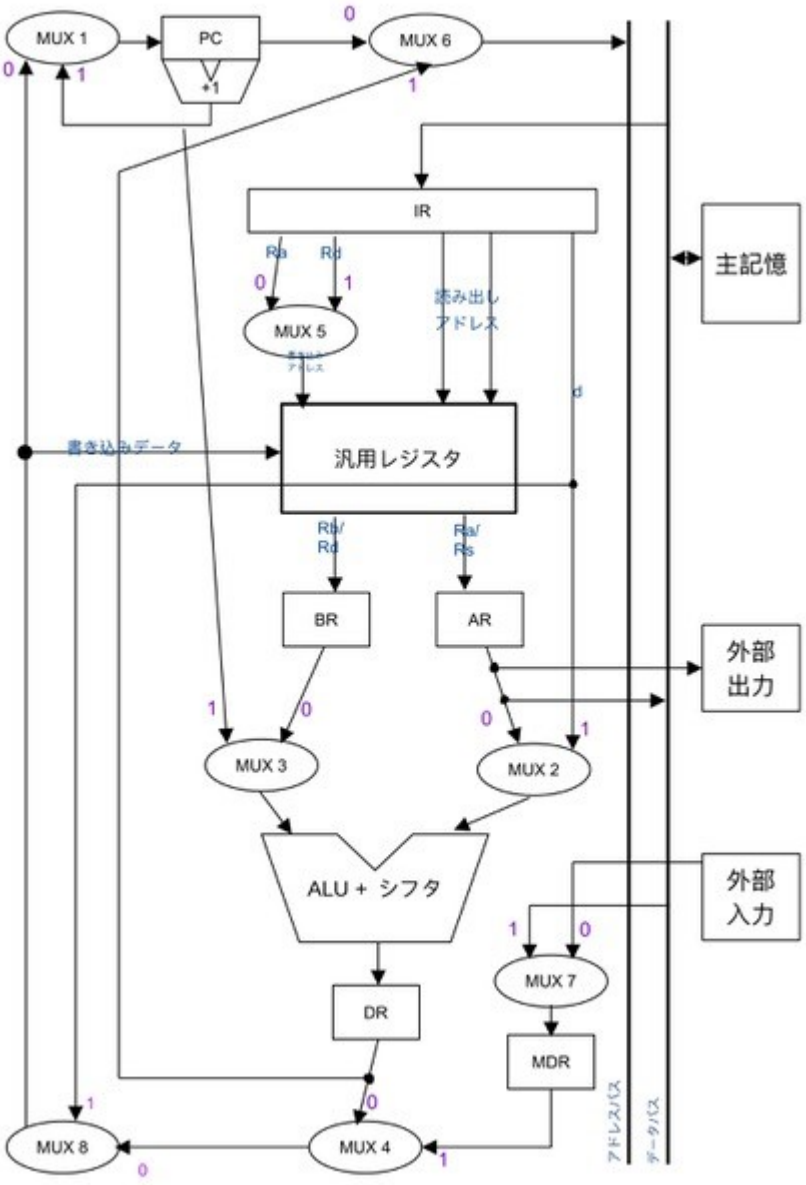

図1

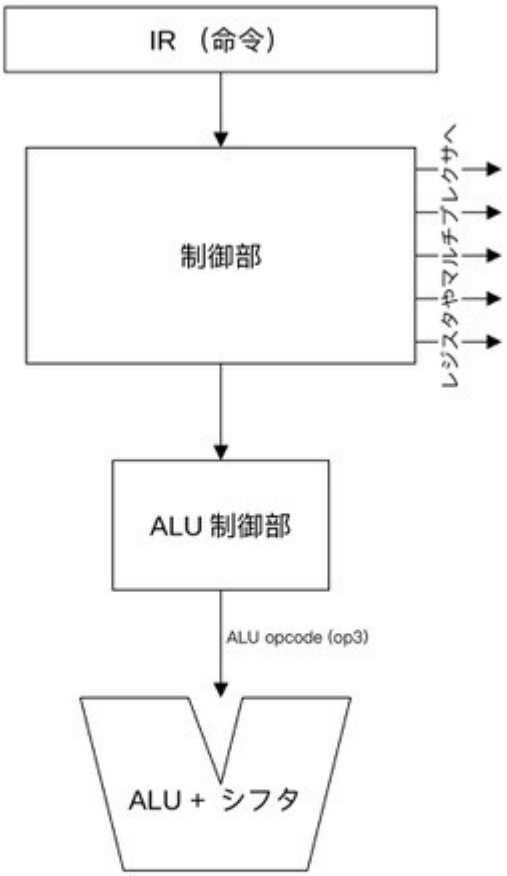

図2

分担状況：

- 主記憶：T
- MUX：T
- ALU：T
- PC：T

- 汎用レジスタ：S
- レジスタ：S
- 制御部：S
- ALU制御部

ブロック	設計 / 実装
Program counter	竹田原
レジスタ	竹田原・Chung
汎用レジスタ	Chung
ALU	竹田原・Chung
マルチプレクサ	竹田原
符号拡張	Chung
制御部	Chung・竹田原
ALU制御部	Chung
トップレベル	竹田原・Chung
主記憶	竹田原

図3

自分が担当したコンポーネントは、図3のように、プログラムカウンタと、ALUと、マルチプレクサと、トップレベルです。主記憶は、コードを書いているわけではないので、省略します。

1 プログラムカウンタ

まず、プログラムカウンタの外部仕様は、入力がpc_e,rst,j_flag,j_addrで、出力がpc_outです。pc_e,rst,j_flagが1ビットで、j_addr,pc_outは12ビットです。

プログラムカウンタを出力する役割を担っています。

内部仕様としては、pc_eは、クロックの役割をしており、rstはリセットの役割で、これは、まだ実装できていませんが、今後実装する予定です。j_flagは、無条件分岐命令や条件分岐命令が起こる際に、プログラムカウンタをj_addrに変える役割をしています。

pc_outはプログラムカウンタを表しており、命令フェッチをするため、主記憶に入力されていきます。コードは、次の通りです。このように、クロック信号が立ち上がるたびに、pc_outに1が足され、j_flagが1の場合は、プログラムカウンタをj_addrに変えます。このようにして、プログラムカウンタは構成されます。

```
module program_counter(pc_e, rst,j_flag,j_addr, pc_in, pc_out);
        input           pc_e;
        input           rst;
        input j_flag;
        input[11:0]j_addr;
```

```
output  [11:0] pc_out;

reg     [11:0] pc_out=12'b000000000000;

always @(posedge pc_e ) begin
        if(j_flag==1)begin
                pc_out<=j_addr;
        end
        else begin
        pc_out <= pc_out+12'b000000000001;//pc_inwo16'b1nikaeta
        end
end


endmodule
```

2 ALU

まず、ALUの外部仕様は、入力がopcode、d、alu_in_a、alu_in_bで出力がalu_outalu_in_a、alu_in_bです。opcode,dが4ビットでalu_in_a,alu_in_b,alu_outが16ビット、S、Z、C、Vが1ビットです。opcodeが演算の種類であり、dはシフト演算の際に使う値であり、出力のalu_outは、alu_in_a、alu_in_bの演算結果を表しています。出力のS、Z、C，Vは条件コードを表しています。

ALUの内部仕様は、

まず、SLRのシフト演算は、簡単には、表すことができなかったので、function文を用いて、先に定義しています。コードは下図4です。51行目から62行目で、opcode（演算の種類）によって、どのような計算をするかということを表しています。64行目から92行目で、S、Z、C、Vはopcodeによって、どのような値を取るかということを表しています。

Text Editor - C:/Users/admin/Desktop/simple/simple - simple - [alu.v]*

File Edit View Project Processing Tools Window Help

```
module alu( opcode,d, alu_in_a, alu_in_b, alu_out, S,Z,C,V);

    input [3:0] opcode;
    input [15:0] alu_in_a;
    input [15:0] alu_in_b;
    input [3:0] d;
    output [15:0] alu_out;
    output S;
    output Z;

    output C;
    output V;

    wire [16:0] SUM;
    wire [16:0] SUB;
    wire [15:0] AND;
    wire [15:0] OR;
    wire [15:0] XOR;

    function [15:0]SLR;
        input [15:0] alu_in_a;

        input [3:0] D; // d 4 bit takes values 0 to 16
        begin
            case(D)
                4'b0000:SLR=alu_in_a[15:0];
                4'b0001:SLR={alu_in_a[14:0], alu_in_a[15:15]};
                4'b0010:SLR={alu_in_a[13:0], alu_in_a[15:14]};
                4'b0011:SLR={alu_in_a[12:0], alu_in_a[15:13]};
                4'b0100:SLR={alu_in_a[11:0], alu_in_a[15:12]};
                4'b0101:SLR={alu_in_a[10:0], alu_in_a[15:11]};
                4'b0110:SLR={alu_in_a[9:0], alu_in_a[15:10]};
                4'b0111:SLR={alu_in_a[8:0], alu_in_a[15:9]};
                4'b1000:SLR={alu_in_a[7:0], alu_in_a[15:8]};
                4'b1001:SLR={alu_in_a[6:0], alu_in_a[15:7]};
                4'b1010:SLR={alu_in_a[5:0], alu_in_a[15:6]};
                4'b1011:SLR={alu_in_a[4:0], alu_in_a[15:5]};
                4'b1100:SLR={alu_in_a[3:0], alu_in_a[15:4]};
                4'b1101:SLR={alu_in_a[2:0], alu_in_a[15:3]};
                4'b1110:SLR={alu_in_a[1:0], alu_in_a[15:2]};
                4'b1111:SLR={alu_in_a[0:0], alu_in_a[15:1]};
            endcase
        end
    endfunction

    wire shift;
    wire [16:0] alu_intermediate;

    assign ADD={1'b0, alu_in_a }+{1'b0, alu_in_b};
    assign shift=8*d[3]+4*d[2]+2*d[1]+d[0];
    assign alu_intermediate=(opcode==4'b0000) ? alu_in_a +alu_in_b
                            (opcode==4'b0001) ? alu_in_a-alu_in_b:
                            (opcode==4'b0010) ? alu_in_a & alu_in_b:
                            (opcode==4'b0011) ? alu_in_a | alu_in_b:
                            (opcode==4'b0100) ? alu_in_a ^ alu_in_b:
                            (opcode==4'b0110) ? alu_in_b:
                            (opcode==4'b0111) ? 16'b0000000000000000:
                            (opcode==4'b1000) ? alu_in_a<<d:
                            (opcode==4'b1001) ? SLR(alu_in_a,d):
```

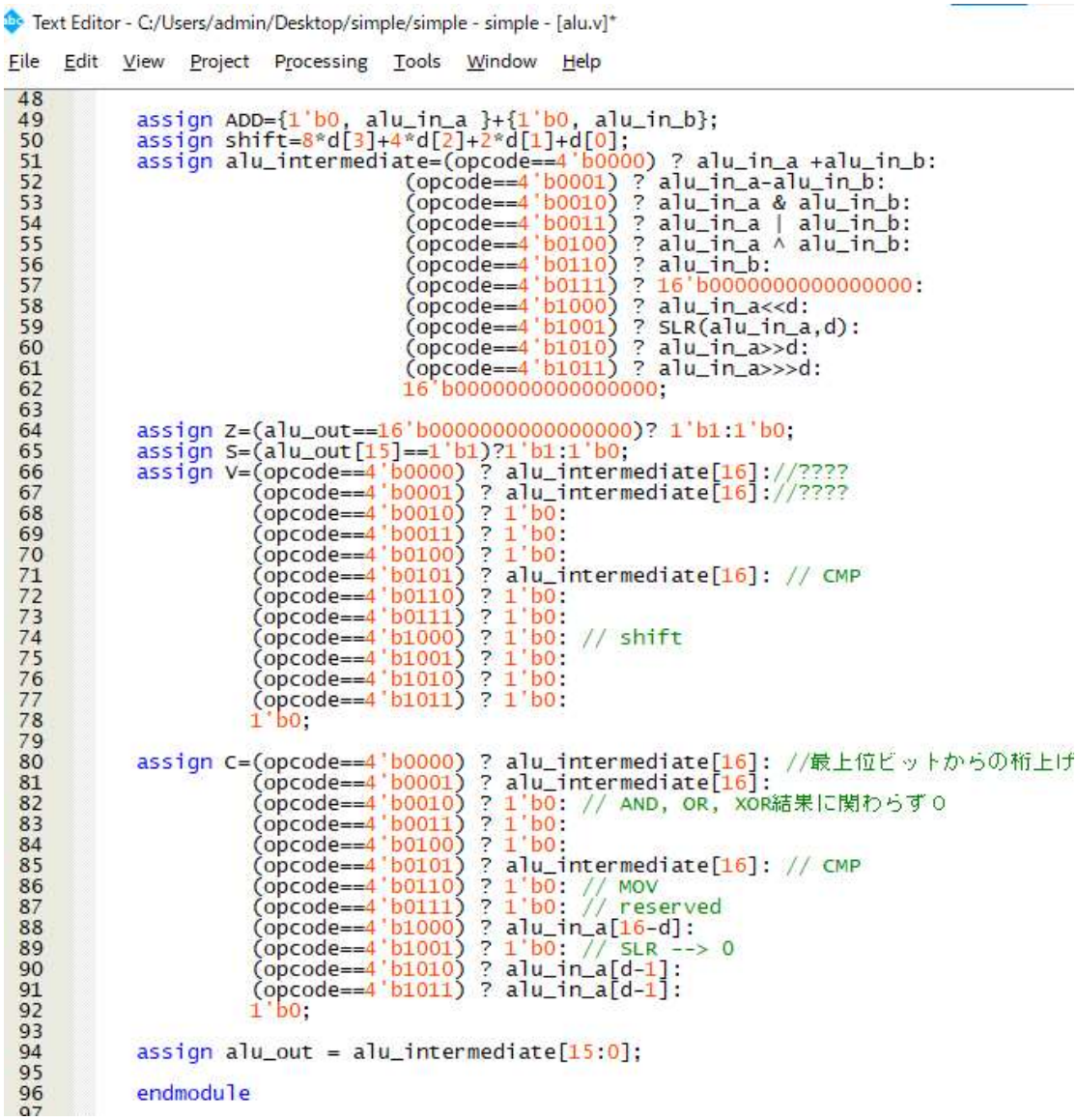

図4

3 マルチプレクサ

外部仕様として、まず、入力がmux_s,mux_in_a,mux_in_bで出力がmux_outです。mux_sは1ビットで、mux_in_a,mux_in_b,mux_outが16ビットです。mux_sが0の時は、mux_in_aを、1の時は、mux_in_bを出力するようになります。内部仕様としては、7行目の条件演算子で、mux_in_aとmux_in_bのどちらを出力するかを決めます。コードは下図5です。

```
1  module multiplexer_16(mux_s, mux_in_a, mux_in_b, mux_out);
2     input        mux_s;
3     input [15:0] mux_in_a;
4     input [15:0] mux_in_b;
5     output   [15:0] mux_out;
6
7     assign mux_out = (mux_s == 1'b0) ? mux_in_a : mux_in_b;
8
9  endmodule
```

図5

4 トップレベル

トップレベルの入力はclk,rst,inで、出力は、outです。
トップレベルのコードは下図6のようになります。clkはクロック、rstはリセット、inは外部入力を、outは外部出力を、表しています。clk、rstは1ビット、in、outは16ビットです。このトップレベルでフェーズを管理しています。フェーズ5になると、新しい命令を制御部に渡し、それを用いて、他のコンポーネントの信号が入力されていきます。

```
module simple(clk,rst,in,out);
	input clk;
	input rst;
	input[15:0]in;
	output[15:0]out;


	wire ar_e,br_e,dr_e,mr_e,ir_e,S,Z,C,V,
	mem_e,mem_w,m1_s,m2_s,m3_s,m4_s,m5_s,m6_s,reg_write,reg_read;
	wire [3:0] ALU_Cnt; //alu opcode
	wire[5:0] instruction_six;
	wire [15:0] ar; //AR content
	wire[15:0] br; //BR content
	wire[15:0] dr; //DR content
	wire[15:0] mdr; //MDR content
	wire[15:0] ir; //ir content
	wire[15:0] pc; //
	wire[15:0] pc_inc; //pc+1
	wire[15:0] m1;
	wire[15:0] m2;
	wire[15:0] m3;
	wire[15:0] m4;
	wire[15:0] m5;
	wire[15:0] m6;
	wire[15:0] mem_out1; //meireifech
	wire[15:0] mem_out2; //roadmeirei P4
	wire [15:0] address;
	wire [15:0] alu_out;
	reg[15:0] MEI;
	reg pc_e;
	wire[15:0]out;

	output reg[2:0]phase=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: Phase1, 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
control controls(.clk(clk),.rst(rst),.S(S),.Z(Z),.C(C),
.V(V),.meirei(MEI),.ar_e(ar_e)
,.br_e(br_e),.dr_e(dr_e),.mr_e(mdr_e),.ir_e(ir_e),.reg_e(reg_e)
,.mem_e(mem_e)
,.mem_w(mem_w) ,.m1_s(m1_s),.m2_s(m2_s),.m3_s(m3_s),.m4_s(m4_s)
,.m5_s(m5_s),.m6_s(m6_s),.alu_opcode(alu_instruction));


register_16 IR(.reg_e(clk), .reg_write_en(ir_e), .reg_in(MEI)
, .reg_out(ir));

register_16 AR(.reg_e(clk), .reg_write_en(ar_e), .reg_in(m2)
, .reg_out(ar));

register_16 BR(.reg_e(clk), .reg_write_en(br_e), .reg_in(m3)
, .reg_out(br));

register_16 DR(.reg_e(clk), .reg_write_en(dr_e), .reg_in(alu_out)
, .reg_out(dr));

register_16 MDR(.reg_e(clk),.reg_write_en(mdr_e),.reg_in(m7)
,.reg_out(mdr));
```

```
register_general(.clk(clk),.rst(rst),
.reg_write_en(reg_e)
,.reg_write_dest(m5),.reg_write_data(m4),.reg_read_addr_1(MEI[13:11])
,.reg_read_data_1(re0),.reg_read_addr_2(MEI[10:8]),.reg_read_data_2
(re1));

alu_control_unit aluconu(.alu_control_unit_e(aluc_e)
,.instruction_six(alu_instruction),.ALU_Cnt(ALU_Cnt));

alu alu_0( .opcode(ALU_Cnt),.d(ir[3:0])
,. alu_in_a(ar), .alu_in_b(br), .alu_out(alu_out), .S(S),.Z(Z)
,.C(C),.V(V));

ram memory_0(.clk_2(clk_2),.mem_e(mem_e),.mem_w(mem_w),
.mem_address(m6),.mem_in(dr),.mem_out(mem_out));

program_counter pc_0(.pc_e(pc_e),.rst(rst),.pc_in(m2),.pc_out(pc),
.pc_inc_out(pc_inc));

sign_extension siex(.d(ir[7:0]),.result(exd));

multiplexer_16 m1_0(.mux_s(m1_s),.mux_in_a(m4),.mux_in_b(pc_inc)
,.mux_out(m1));

multiplexer_16 m2_0(.mux_s(m2_s),.mux_in_a(re0),.mux_in_b(exd)
,.mux_out(m2));

multiplexer_16 m3_0(.mux_s(m3_s),.mux_in_a(re1),.mux_in_b(pc_inc)
,.mux_out(m3));

multiplexer_16 m4_0(.mux_s(m4_s),.mux_in_a(dr),.mux_in_b(mdr)
,.mux_out(m4));

multiplexer_16 m5_0(.mux_s(m5_s),.mux_in_a(MEI[13:11]),.mux_in_b(MEI[10:8])
,.mux_out(m5));

multiplexer_16 m6_0(.mux_s(m6_s),.mux_in_a(pc),.mux_in_b(dr)
,.mux_out(m6));

multiplexer_16 m7_0(.mux_s(m7_s),.mux_in_a(mem_out),.mux_in_b(in)
,.mux_out(m7));


assign out=ar;

endmodule
```

図6