機能設計仕様書

1029338238　神事倫紀

執筆日:2023 年 5 月 12 日

1　全体をどのようなコンポーネントに分割したか

まず、現時点で完成している simple/B の全体図が下の図 1 である。まず、

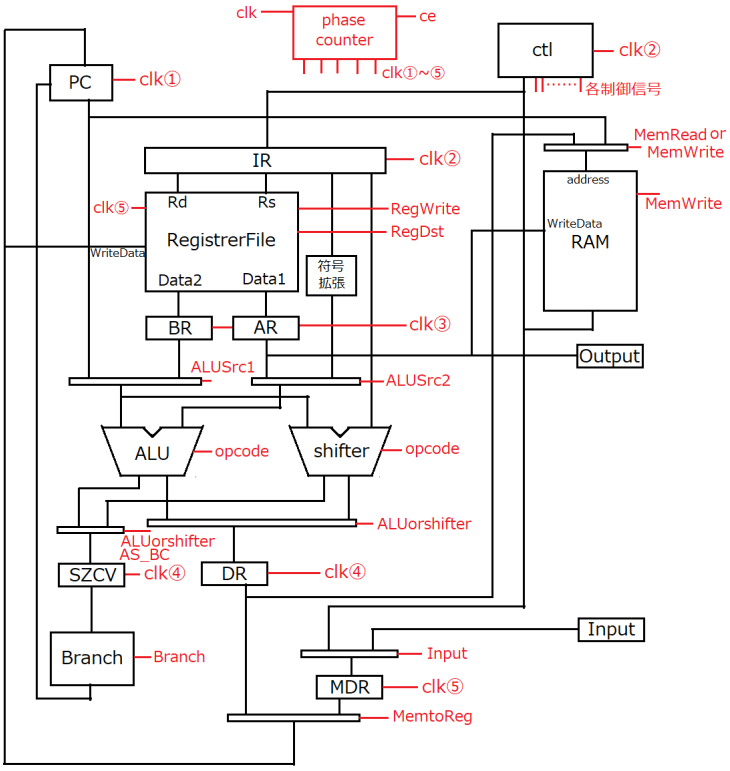

図 1: simple/B の全体図

命令を記憶しておくメモリや命令の結果を記憶しておく主記憶として RAM が用意されており、現在の命令の番号を記憶している PC、回路全体でどの操作を行うかを制御する ctl、命令に使う値や命令の結果をいったん記憶しておく汎用レジスタの RegisterFile、算術論理計算を行う ALU とシフトを行う Shifter、その結果に関して、条件分岐に用いる cond を記憶しておく SZCV、16 ビットの数を記憶しておく IR、AR、BR、DR、MDR、分岐命令の時に分岐するか否かを判断する Branch、また今のフェーズをカウントする phase counter、というように上図 1 のように各コンポーネントに分割した。外部入力から読み込んでくるときに導入課題の 3 と同様にチャタリングが起こってしまうのでそれの除去を行うモジュール RemoveChat も用意した。拡張していくにあたり、そのほかのコンポーネントが必要になる可能性や現状のものを書き換える必要ある可能性があるがそれらについては必要になるたびに都度追加し、最終レポートの時点で加筆修正するつもりである。

以下に自分が担当した部分のモジュールについて詳細を述べていく。

2 制御部

2.1 外部仕様

制御部では、まず今行うべき命令の内容を入力として読み込み、今の命令を正しく回路内で処理するために必要な制御信号をすべて出すのが役割である。具体的に、制御部が出力するものを下に示す。

- RegWrite 信号: レジスタファイルに書き込みが行われる命令の時にそれをレジスタファイルに知らせる信号。
- MemWrite 信号: 主記憶に書き込みが行われる命令の時にそれを主記憶に知らせる信号。
- MemRead 信号: 主記憶からのデータの読み込みが行われるときにそれを主記憶に知らせる信号。
- MemtoReg 信号: レジスタにデータとして渡すのが ALU や Shifter の結果かメモリや外部入力から得られたものかを制御する信号。
- ALU_Src1 信号:ALU への第 1 引数がレジスタファイルから読みだしたものか PC の値に 1 を足したものかを制御する信号。
- ALU_Src2 信号:ALU への第 2 引数がレジスタファイルから読みだしたものか d の値を符号拡張したものかを制御する信号。
- Output 信号: 外部への出力が行われるか否かを知らせる信号。
- Input 信号: 外部からの入力が行われているか否かを知らせる信号。
- ALUorShifter 信号:ALU と Shifter の結果のうちのどちらをレジスターに読み込むかを制御する信号。
- Halt 信号: 停止命令が来たときにそれを知らせる信号。
- BranchCond 信号: 分岐命令の時にそれを知らせる信号。ただし全体を組み合わせるときに不要であったものなので最終レポートでは削除される見込みである。
- AS_BC 信号:ALU もしくは Shifter が分岐命令の条件コードの下となる計算を行い、条件コード部分を書き換えるべき時にそれを知らせる信号。

以上が制御部から出力される 1 ビットの信号である。
続いて 2 ビット以上の信号を示す。

- opcode:ALU や Shifter で今どの計算をするべきかを制御するコード。4 ビット。
- RegDst: 書き込むレジスタファイルの番地。3 ビット。
- Branch: 今の命令においてどの条件分岐が行われるかを判断するためのコード。3 ビット。

以上が制御部が出力するものである。

2.2 内部仕様

以上のような外部仕様をみたす制御部の内部仕様について、制御部のソースコードを示しつつ説明する。

```
1  module ctl(
2          input clk,rst_n,
3          input [15:0] inst,
4          output MemRead,MemWrite,RegWrite,ALUSrc1,ALUSrc2,
               MemtoReg,Output,Input,ALUorShifter,Halt,BranchCond
               ,AS_BC,
5          output [3:0] opcode,
6          output [2:0] RegDst,
7          output [2:0] Branch);
8          wire [1:0] twobit;
9          wire [3:0] opcode_reg;
10         wire [15:0] inst_reg;
11         wire [2:0] brch_reg;
12         reg MemRead_wire,MemWrite_wire,RegWrite_wire,
               ALUSrc1_wire,ALUSrc2_wire,MemtoReg_wire,
               Output_wire,Input_wire,ALUorShifter_wire,Halt_wire
               ,BranchCond_wire,AS_BC_wire;
13         reg [3:0] opcode_wire;
14         reg [2:0] brch_wire;
15         reg [2:0] reg_dst_wire;
16         assign inst_reg = inst;
17         assign twobit = inst[15:14];
18         assign opcode_reg = inst[7:4];
19         assign brch_reg = inst[13:11];
20         always @ (posedge clk ) begin
21                 if(!rst_n) begin
22                         Halt_wire <= 1'b0;
23                 end else begin
24                         if (( twobit == 2'b11 && opcode_reg
                               != 4'b0111 && opcode_reg != 4'
                               b1101 && opcode_reg != 4'b1110 &&
```

```
                opcode_reg != 4'b1111 &&
               opcode_reg != 4'b0101) || (twobit
                == 2'b00 ) || (twobit == 2'b10
               && brch_reg == 3'b000)) begin
25                 RegWrite_wire <= 1'b1;
26             end else begin
27                 RegWrite_wire <= 1'b0;
28             end
29             if(twobit == 2'b01 ) begin
30                 MemWrite_wire <= 1'b1;
31             end else begin
32                 MemWrite_wire <= 1'b0;
33             end
34             if (twobit == 2'b00 ) begin
35                 MemRead_wire <= 1'b1;
36             end else begin
37                 MemRead_wire <= 1'b0;
38             end
39             if ((twobit == 2'b11 && opcode_reg ==
                4'b1100) || (twobit == 2'b00))
               begin//1101 ->1100 modified
40                 MemtoReg_wire <= 1'b1;
41             end else begin
42                 MemtoReg_wire <= 1'b0;
43             end
44             if ( twobit == 2'b10 && brch_reg !=
               3'b000) begin //3'b00 ->3'b000
               modified
45                 ALUSrc1_wire <= 1'b1;
46             end else begin
47                 ALUSrc1_wire <= 1'b0;
48             end
49             if(twobit ==2'b11 && (opcode_reg ==
               4'b0000 ||opcode_reg == 4'b0001
               ||opcode_reg == 4'b0010 ||
               opcode_reg == 4'b0011 ||
               opcode_reg == 4'b0100 ||
               opcode_reg == 4'b0101 ||
               opcode_reg == 4'b0110)) begin
50                 ALUSrc2_wire <= 1'b0;
51             end else begin
52                 ALUSrc2_wire <= 1'b1;
53             end
54             if(twobit == 2'b11 && opcode_reg ==
               4'b1101) begin
55                 Output_wire <= 1'b1;
```

```
56                      end else begin
57                          Output_wire <= 1'b0;
58                      end
59                      if(twobit == 2'b11 && opcode_reg ==
                            4'b1100) begin
60                          Input_wire <= 1'b1;
61                      end else begin
62                          Input_wire <= 1'b0;
63                      end
64                      if( twobit == 2'b11 )begin // &&
                            opcode_2reg != 4'b0111 &&
                            opcode_reg != 4'b1101 &&
                            opcode_reg != 4'b1110 &&
                            opcode_reg != 4'b1111) || (twobit
                             == 2'b11) && (opcode_reg == 4'
                            b1100 || opcode_reg == 4'b1101)
                            -> x
65                          opcode_wire <= opcode_reg;
66                      end else if(twobit == 2'b10 &&
                            brch_reg == 3'b000) begin
67                          opcode_wire <= 4'b0110;
68                      end else begin
69                          opcode_wire <= 4'b0000;
70                      end
71                      if(twobit == 2'b10 && brch_reg == 3'
                            b111) begin
72                          brch_wire <= inst[10:8];
73                          BranchCond_wire <=1'b1;
74                      end else if(twobit == 2'b10 &&
                            brch_reg == 3'b100) begin
75                          brch_wire <= brch_reg;
76                          BranchCond_wire <=1'b1;
77                      end else begin
78                          brch_wire <= 3'b111;
79                          BranchCond_wire <=1'b0;
80                      end
81                      if(twobit == 2'b00 ) begin
82                          reg_dst_wire <= inst[13:11];
83                      end else begin
84                          reg_dst_wire <= inst[10:8];
85                      end
86                      if(twobit ==2'b11 && (opcode_reg ==
                            4'b1000 ||opcode_reg == 4'b1001
                            ||opcode_reg == 4'b1010 ||
                            opcode_reg == 4'b1011 )) begin
87                          ALUorShifter_wire <= 1'b1;
```

```
88                          end else begin
89                                  ALUorShifter_wire <= 1'b0;
90                          end
91                          if(twobit == 2'b11 && opcode_reg ==
                                4'b1111) begin
92                                  Halt_wire <= 1'b1;
93                          end else begin
94                                  Halt_wire <= 1'b0;
95                          end
96                          if(twobit == 2'b11 && opcode_reg !=
                                4'b0111 && opcode_reg != 4'b1101
                                && opcode_reg != 4'b1110 &&
                                opcode_reg != 4'b1111 &&
                                opcode_reg != 4'b1100) begin
97                                  AS_BC_wire <= 1'b1;
98                          end else begin
99                                  AS_BC_wire <= 1'b0;
100                         end
101                 end
102         end
103         assign MemRead = MemRead_wire;
104         assign MemWrite = MemWrite_wire;
105         assign RegWrite = RegWrite_wire;
106         assign ALUSrc1 = ALUSrc1_wire;
107         assign ALUSrc2 = ALUSrc2_wire;
108         assign MemtoReg = MemtoReg_wire;
109         assign Output = Output_wire;
110         assign Input = Input_wire;
111         assign ALUorShifter = ALUorShifter_wire;
112         assign Halt = Halt_wire;
113         assign opcode = opcode_wire;
114         assign Branch = brch_wire;
115         assign RegDst = reg_dst_wire;
116         assign BranchCond = BranchCond_wire;
117         assign AS_BC = AS_BC_wire;
118 endmodule
```

制御部についてはクロックが来るたびに動く順序回路となっており、入力として命令の値 (16 ビット)、クロック信号、リセット信号を入力として受け取り、先ほど示した各信号を出力するようになっている。各信号について内部に記憶しておくレジスタを用意し、それを出力に割り当てるという形をとっているがその部分についてわけて表記せず、信号名そのままとして扱う。まず、16 ビットの入力のうち上位 2 ビット、5 ビット目から 8 ビット目までの 4 ビット、12 ビット目から 14 ビット目までの 3 ビットをそれぞれ内部のレジスタに記憶する。そのうえで、クロックが来るたびに条件分岐を用いて各

信号の値を変更していくといった形を用いている。1 ビットの各信号についてどのような仕様になっているかを以下に示す。

- RegWrite 信号: レジスタファイルに書き込みが行われる時 (ALU、 Shifter が使われる時、ロード命令の時、即値ロード命令の時、IN 命令の時) に 1、それ以外の以外の時には 0 となっている。
- MemWrite 信号: 主記憶に書き込みが行われる、ストア命令の時に 1、それ以外の時には 0 となっている。
- MemRead 信号: 主記憶からのデータの読み込みが行われる、ロード命令の時に 1、それ以外の時には 0 となっている。
- MemtoReg 信号: レジスタにデータとして渡すのがメモリや外部入力から得られたものとなるのは、ロード命令の時と IN 命令の時なので、その時に 1、それ以外の時には 0 となっている。0 の時は、ALU や Shifter の結果が選ばれている。
- ALU_Src1 信号:ALU への第 1 引数が PC の値に 1 を足したものとなるのは、条件分岐命令の時なのでその時に 1、それ以外の時は 0 となっており、0 の時はレジスタファイルから読みだした値が採用されている。
- ALU_Src2 信号:ALU への第 2 引数がレジスタファイルから読みだしたものとなるのは、ALU を用いて計算が行われる時なので、その時に 1、それ以外の時には 0 となっている。0 の時は d の値を符号拡張したものが採用されている。
- Output 信号: 外部への出力が行われるのは、OUT 命令の時のみなのでその時に 1、それ以外の時には 0 となっている。
- Input 信号: 外部からの入力が行われるのは、IN 命令の時なので、その時に 1、それ以外の時には 0 となっている。
- ALUorShifter 信号:Shifter を用いて計算が行われているの時に 1、それ以外の時には 0 となっている。0 の時は ALU の結果がレジスタに読み込まれるようになっている。
- Halt 信号: 停止命令が来た時に 1、それ以外の時には 0 となっている。
- BranchCond 信号: 分岐命令のの時に 1、それ以外の時には 0 となっている。。
- AS_BC 信号: 算術論理演算、移動演算、比較演算、シフト演算の時に 1、それ以外の時には 0 となっている。

続いて、2 ビット以上の出力について述べる。

- opcode: 命令の上位 2 ビットが 11 の時 (ALU や Shifter を用いる演算関連の条件コード) は、命令の 5 ビット目から 8 ビット目までの 4 ビット、即値ロード命令の時は 0110(移動演算の条件コード)、それ以外の時は 0000(算術加算の条件コード) となっている。
- RegDst: ロード命令の時は、命令の 12 ビット目から 14 ビット目までの 3 ビット、それ以外の時は命令の 9 ビット目から 11 ビット目までの 3 ビットとなっている。
- Branch: 条件分岐命令の時は、命令の 9 ビット目から 11 ビット目までの 3 ビット、無条件分岐命令の時は 100、それ以外の時は 111 を割り当てている。

以上が制御部の仕様である。

3 レジスタファイル

3.1 外部仕様

レジスタファイルには、16 ビットの 8 本の汎用レジスタが用意されており、2 つの読み出し番地の入力に対してその番地にある値を出力し、書き込み信号と書き込み番地、書き込む内容も信号として受け取り、書き込み番地に書き込む内容を書き込むという仕様である。

3.2 内部仕様

レジスタファイルの内部仕様について、レジスタファイルのソースコードを示しながら説明する。

```
1  module RegisterFile(
2          input [2:0] Read1,Read2,WriteReg,
3          input [15:0] WriteData,
4          input clk,RegWrite,
5          output [15:0] Data1,Data2,
6          output [15:0] reg_1,reg_2,reg_3,reg_4,reg_5,reg_6,
               reg_7,reg_0);
7          reg [15:0] RegFile [7:0];
8          assign Data1 = RegFile [Read1];
9          assign Data2 = RegFile [Read2];
10         assign reg_0 = RegFile[0];
11         assign reg_1 = RegFile[1];
12         assign reg_2 = RegFile[2];
13         assign reg_3 = RegFile[3];
14         assign reg_4 = RegFile[4];
```

```
15          assign reg_5 = RegFile[5];
16          assign reg_6 = RegFile[6];
17          assign reg_7 = RegFile[7];
18          always @ (WriteData) begin
19                  if(RegWrite==1'b1) begin
20                          case (WriteReg)
21                                  0:RegFile [0] <= WriteData;
22                                  1:RegFile [1] <= WriteData;
23                                  2:RegFile [2] <= WriteData;
24                                  3:RegFile [3] <= WriteData;
25                                  4:RegFile [4] <= WriteData;
26                                  5:RegFile [5] <= WriteData;
27                                  6:RegFile [6] <= WriteData;
28                                  7:RegFile [7] <= WriteData;
29                          endcase
30                  end
31          end
32  endmodule
```

上記がレジスタファイルのソースコードである。クロック信号、リセット信号、読み出し番地 2 つ、書き込み信号、書き込み番地、書き込む内容が入力として与えられ、読み出した内容 2 つが出力されている。バグの検証用に現在レジスタファイルに格納されている値も出力されるようになっているが、ここは最終レポートまでにすべてのバグの検証が終わったら消す予定なので気にしなくてよい。

内部の動き方としては、まず組み合わせ回路部分として読みだす番地にある値をそれぞれの出力に割り当てている。また、順序回路部分としてはクロック信号が来たときに、書き込み信号が 1 だったら書き込み番地として入力された番地に入力されたデータを書き込む。

以上のような仕様となっている。

4 レジスタ

4.1 外部仕様

16 ビットの数を記憶しておく IR、AR、BR、DR、MDR はすべてこのレジスタをインスタンス化したものである。このレジスタは、クロックが立ち上がるたびに入力された 16 ビットのデータを記憶し、記憶されている 16 ビットのデータを出力するという挙動を示す。

4.2 内部仕様

では、レジスタの内部仕様を以下のソースコードを用いながら示す。

```
1     module register(
2        input [15:0] WriteData,
3        input clk,rst_n,
4        output reg [15:0] DataOut);
5        always @ (posedge clk) begin
6                if(!rst_n) begin
7                        DataOut <= 16'b0000000000000000;
8                end else begin
9                        DataOut <= WriteData;
10               end
11       end
12 endmodule
```

現在はパイプライン化はしていないので、内部に記憶しておく必要もなく、クロック信号、リセット信号、データを入力として受け取り、クロックが立ち上がるときに出力にデータを割り当て、リセット信号が来たときには出力を 0 とするというものである。
以上がレジスタの内部仕様である。

5　チャタリング除去

5.1　外部仕様

ボタン入力に対して、そのままだとチャタリングしてしまうのでその除去を行うのが役割である。入力として、ボタンの値をクロック信号を受け取り、チャタリングの除去がなされたボタン入力の値を返すのが外部仕様である。

5.2　内部仕様

チャタリング除去の内部仕様について、チャタリング除去のソースコードを示しながら説明する。ただし、どうしてこれでチャタリングの除去ができるかという理論的な部分に関しては導入課題のレポートの課題 3 の部分で説明したことと全く同じなので今回は割愛する。

```
1    module RemoveChattering (
2        input clk, botton, rst_n,
3        output reg signal);
4        wire clk_10Hz;
5        reg botton_reg;
6        reg [7:0] remove_chat;
7        wire rst_n1,rst_n2,rst_n3;
8        divider b2(.clk(clk),.hz(30'd10),.rst_n(rst_n),.
             outclk(clk_10Hz));
```

```
 9          assign rst_n1 = rst_n;
10          assign rst_n2 = rst_n;
11          assign rst_n3 = rst_n;
12          always @ (posedge clk_10Hz or negedge rst_n2) begin
13                  if(!rst_n2) begin
14                          botton_reg <= 1'b0;
15                  end else begin
16                          botton_reg <= !botton;
17                  end
18          end
19          always @(posedge botton_reg or negedge rst_n3) begin
20                  if(!rst_n3) begin
21                          signal <= 1'b0;
22                  end else begin
23                          signal <= signal +1;
24                  end
25          end
26  endmodule
```

まず、クロック信号、ボタンの値、リセット信号を入力として受け取り、divider module を用いてクロック信号を 10Hz にしたのちに、そのクロック信号が立ち上がったときに内部記憶の値をボタン入力の否定を取ったものとする。次に、その内部記憶をクロックとして扱って、それが立ち上がったときに出力の値に 1 を足す。リセット信号が来たときには、出力、内部記憶ともに 0 にするというものである。

6 Branch

6.1 外部仕様

Banch は、制御部から出力される、今がどんな条件分岐命令であるかを判断する条件コードの値と、ALU や Shifter の演算結果から得られる cond の値を入力として受け取り、その値をもとに条件分岐するか否かを出力するものである。

6.2 内部仕様

Branch の内部仕様について、Branch のソースコードを以下に示しながら説明する。

```
1      module branch(
2          input [3:0] cond,
3          input [2:0] brch,
```

```
4           output brch_sig);
5           wire s,z,c,v;
6           assign s = cond[3];
7           assign z = cond[2];
8           assign c = cond[1];
9           assign v = cond[0];
10          assign brch_sig = (((brch == 3'b100))||((brch == 3'
                b000 && z == 1'b1 ))||((brch == 3'b001 && s ^ v
                 == 1'b1 )) || (brch == 3'b010 && (z == 1'b1 ||
                 s ^ v == 1'b1)) || (( brch == 3'b011 && z ==
                1'b0))) ? 1'b1:
11                  1'b0;
12  endmodule
```

まず、入力として 3 ビットの条件分岐コード、4 ビットの cond が与えられる。そして、cond の 4 ビット目に s、3 ビット目に z、2 ビット目に c、1 ビット目に v と名前を付ける。命令が BI の時、命令が BE で z が 1 の時、命令が BLT で s と v の排他的論理和が 1 の時、命令が BLE で z が 1 もしくは s と v の排他的論理和が 1 の時、命令が BNE で z が 0 の時、以上の条件に当てはまるとき条件分岐するので出力に 1 を割り当て、それ以外の時には 0 を割り当てている。