機能設計仕様書

1029338238　神事倫紀

執筆日:2023 年 6 月 8 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、

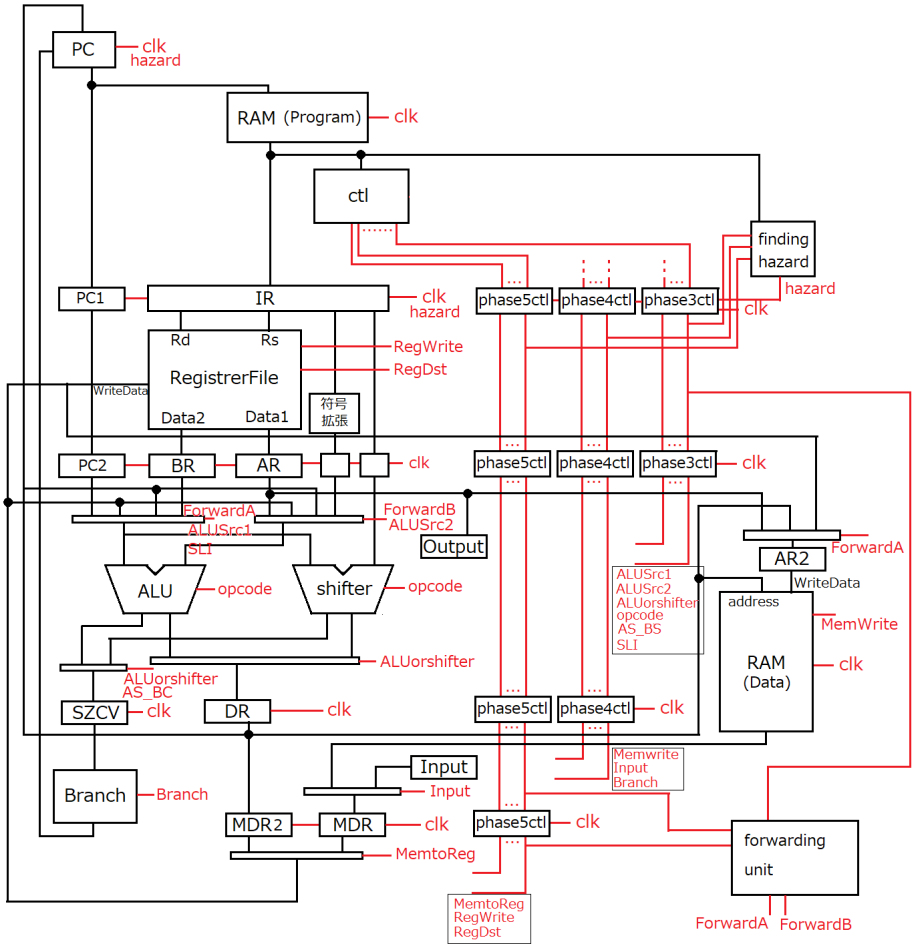

図 1: simple/B の全体図

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。ほかにも、フォワーディングのためのフォワーディングユニット、ハザード検出のためのハザード検出ユニット、拡張機能として内部にあるデータを可視化できるように出力する display、各フェーズにおいてどのような制御信号が出ていたかを記録しておく各フェーズのパイプラインレジスタを中間レポートから追加してある。
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。

- 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。
- SLI 信号:SLI 命令という特殊な命令が来たときにのみそれを察知しうまく処理するために滅入れ

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。ただし、下のコードについてはレポートに書くにあたってインデントや 1 行に書く内容等を調整しているのでそのままコピーアンドペーストして動くことは保障しない。

```
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,AS_BC,SLI,
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_wire;
10         wire [15:0] inst_wire;
11         wire [2:0] brch_wire;
12         assign inst_wire = inst;
13         assign twobit = inst[15:14];
14         assign opcode_wire = inst[7:4];
15         assign brch_wire = inst[13:11];
16
17
18         assign RegWrite = (( twobit == 2'b11
19                     && opcode_wire != 4'b0111
```

```
20                  && opcode_wire != 4'b1101
21                  && opcode_wire != 4'b1110
22                  && opcode_wire != 4'b1111
23                  && opcode_wire != 4'b0101)
24                  || (twobit == 2'b00 ) ||
25                  (twobit == 2'b10 &&
26                  (brch_wire == 3'b000
27                  || brch_wire == 3'b001
28                  ||brch_wire == 3'b010
29                  ||brch_wire == 3'b101))) ? 1'b1:
30                  1'b0;
31          assign MemWrite = (twobit == 2'b01 ) ? 1'b1:
32                      1'b0;
33          assign MemRead = (twobit == 2'b00 )? 1'b1:
34                      1'b0;
35          assign MemtoReg = ((twobit == 2'b11
36                      && opcode_wire == 4'b1100)
37                      || (twobit == 2'b00)) ? 1'b1:
38                      1'b0;
39          assign ALUSrc1 = ( twobit == 2'b10
40                      && brch_wire != 3'b000
41                      && brch_wire != 3'b001
42                      && brch_wire != 3'b010
43                      && brch_wire != 3'b011
44                      && brch_wire != 3'b101 ) ?1'b1:
45                      1'b0;
46          assign ALUSrc2 = (twobit ==2'b11
47                  && (opcode_wire == 4'b0000
48                  ||opcode_wire == 4'b0001
49                  ||opcode_wire == 4'b0010
50                  ||opcode_wire == 4'b0011
51                  ||opcode_wire == 4'b0100
52                  ||opcode_wire == 4'b0101
53                  ||opcode_wire == 4'b0110)) ? 1'b0:
54                  1'b1;
55          assign Output = (twobit == 2'b11
56                  && opcode_wire == 4'b1101) ? 1'b1:
57                  1'b0;
58          assign Input = (twobit == 2'b11
59                  && opcode_wire == 4'b1100) ? 1'b1:
60                  1'b0;
61          assign opcode = ( twobit == 2'b11 ) ? opcode_wire:
62                  (twobit == 2'b10
63                  && brch_wire == 3'b000)? 4'b0110:
64                  (twobit == 2'b10
65                  && brch_wire == 3'b010)? 4'b0001:
```

```
66                      (twobit == 2'b10
67                      && brch_wire == 3'b011)? 4'b0101:
68                      (twobit == 2'b10
69                      && brch_wire == 3'b101)? 4'b1000:
70                      4'b0000;
71          assign Branch = (twobit == 2'b10
72                      && brch_wire == 3'b111) ? inst[10:8]:
73                      (twobit == 2'b10
74                      && brch_wire == 3'b100) ? brch_wire:
75                      3'b111;
76          assign RegDst = (twobit == 2'b00 ) ? inst[13:11]:
77                      inst[10:8];
78          assign ALUorShifter = ((twobit ==2'b11
79                          && (opcode_wire == 4'b1000
80                          ||opcode_wire == 4'b1001
81                          ||opcode_wire == 4'b1010
82                          ||opcode_wire == 4'b1011 ))
83                          ||( twobit == 2'b10
84                          && brch_wire == 3'b101)) ? 1'b1:
85                          1'b0;
86          assign Halt = (twobit == 2'b11 && opcode_wire == 4'
                b1111) ? 1'b1:
87                          1'b0;
88          assign AS_BC = ((twobit == 2'b11
89                      && opcode_wire != 4'b0111
90                      && opcode_wire != 4'b1101
91                      && opcode_wire != 4'b1110
92                      && opcode_wire != 4'b1111
93                      && opcode_wire != 4'b1100)
94                      || (twobit == 2'b10
95                      && brch_wire == 3'b011)) ? 1'b1:
96                      1'b0;
97          assign SLI = ( twobit == 2'b10
98                      && brch_wire == 3'b101) ? 1'b1:
99                      1'b0;
100
101 endmodule
```

制御部については中間レポートの時点では順序回路としていたが、パイプライン化するにあたって組み合わせ回路に変更した。その理由としては、各命令や値を収納するレジスタもクロックで動くので同じクロックを用いて制御する命令を調整するのではタイミング制約的に厳しくなってしまうということがあげられる。これは、クロックの反転を使うといったことでも解決できそうではあるが、そもそも順序回路ではなくすれば、タイミング制約を悩む必要もないのでできるだけ単純化するために組み合わせ回路とした。制御

部の挙動としては、入力として命令の値 (16 ビット) 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。中間レポート時点では内部に値を記憶しておくレジスタを用意していたが、組み合わせ回路となったのでその必要もなく出力に直接値を割り当てている。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 ビットをそれぞれワイヤで分ける。そのうえで、そのワイヤの各値をもと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、ADDI,SUBI,CMPI,SLI 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。
- 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。
- SLI 信号:SLI 命令が来たときに 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)、SUBI 命令の時は 0001(算術減算の条件コード)、CMPI 命令の時は 0101(比較演算の条件コード)、SLI 命令の時は 1000(左論理シフトの条件コード)、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。

2.3 単体での性能評価

制御部の単体での性能評価について述べる。

- LUT 数：34(¡1%)
- 遅延時間：一番後ろの図 2〜4 を参照。

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,rst_n,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 @ (posedge clk or negedge rst_n) begin
19                 if(!rst_n) begin
20                         RegFile [0] <= 16'd0;
21                         RegFile [1] <= 16'd0;
22                         RegFile [2] <= 16'd0;
23                         RegFile [3] <= 16'd0;
24                         RegFile [4] <= 16'd0;
25                         RegFile [5] <= 16'd0;
26                         RegFile [6] <= 16'd0;
27                         RegFile [7] <= 16'd0;
28                 end else begin
29                         if(RegWrite==1'b1) begin
30                                 RegFile [WriteReg] <=
                                       WriteData;
31                         end else begin
32                                 RegFile [0] <= RegFile [0];
33                                 RegFile [1] <= RegFile [1];
34                                 RegFile [2] <= RegFile [2];
35                                 RegFile [3] <= RegFile [3];
36                                 RegFile [4] <= RegFile [4];
37                                 RegFile [5] <= RegFile [5];
38                                 RegFile [6] <= RegFile [6];
39                                 RegFile [7] <= RegFile [7];
40                         end
```

```
41                      end
42              end
43  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つもりだったが、よく考えたら拡張機能の範囲かもしれないと思ったので残してある。この値をどう使うかについての詳しい話は display モジュールの部分に任せる。

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、リセット信号が来たときにはレジスタファイルの中身をすべて 0 にリセットし、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。

3.3 単体での性能評価

レジスタファイルの単体での性能評価について述べる。LUT 数:185¡1% 遅延時間：遅延時間については下の図 5〜7 を参照のこと

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
```

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。

```
module RemoveChattering (
        input clk, botton, rst_n,
        output reg signal);
        reg botton_reg1;
        reg botton_reg2;
        reg [50:0] count;
        always @(posedge clk or negedge rst_n) begin
                        if (!rst_n)
                                count <= 26'd0;
                        else if (count == 26'd2_000_000)
                                count <= 26'd0;
                        else
                                count <= count + 26'd1;
        end
        always @(posedge clk or negedge rst_n) begin
                if (!rst_n) begin
                        botton_reg1 <= 1'b0;
                        botton_reg2 <= 1'b0;
                        signal <= 1'b0;
                end else if (count == 26'd1_000_000) begin
                        botton_reg1 <= botton_reg2;
                        botton_reg2 <= !botton;
                        if (botton_reg1 == 1'b0 &&
                            botton_reg2 == 1'b1) begin
                                signal <= !signal;
                        end
                end else begin
                        botton_reg1 <= botton_reg1;
                        botton_reg2 <= botton_reg2;
                        signal <= signal;
                end
        end
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、内部のカウンタ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))
11                        ||((brch == 3'b000
12                        && z == 1'b1 ))
13                        ||((brch == 3'b001
14                        && s ^ v == 1'b1 ))
15                        ||(brch == 3'b010
16                        && (z == 1'b1 || s ^ v == 1'b1))
17                        || (( brch == 3'b011
18                        && z == 1'b0))) ? 1'b1:
19                        1'b0;
20 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。

7　フォワーディングユニット

7.1　外部仕様

フォワーディングユニットは、フォワーディングするべき状況の時にその判断をしてどのデータをフォワーディングするかについての信号を出すことが仕事である。フォワーディングする先は、レジスタ A とBの二つが存在するので、それらの両方にどのデータをフォワーディングするべきかを出力する。

7.2　内部仕様

フォワーディングユニットの内部仕様について、フォワーディングユニットのソースコードを示しながら説明する。

```
1     module forwardingunit (
2         input EX_MEM_RegWrite,MEM_WB_RegWrite,
3         input [2:0] EX_MEM_RegDst,MEM_WB_RegDst,
              ID_EX_RegisterRa,ID_EX_RegisterRb,
4         output [1:0] ForwardA,ForwardB);
5         assign ForwardA = ((EX_MEM_RegWrite == 1'b1)
6                           &&(EX_MEM_RegDst == ID_EX_RegisterRa))
                                ? 2'b01:
7                           ((MEM_WB_RegWrite == 1'b1)
8                           &&(MEM_WB_RegDst == ID_EX_RegisterRa))
                                ? 2'b10:
9                           2'b00;
10
11        assign ForwardB = ((EX_MEM_RegWrite == 1'b1)
12                          &&(EX_MEM_RegDst == ID_EX_RegisterRb))
                                ? 2'b01:
13                          ((MEM_WB_RegWrite == 1'b1)
14                          &&(MEM_WB_RegDst == ID_EX_RegisterRb))
                                ? 2'b10:
15                          2'b00;
16 endmodule
```

入力として EX_MEM ステージにあるレジスタ書き込み信号、MEM_WB ステージにあるレジスタ書き込み信号、EX_MEM ステージのレジスタ書き込み番地、MEM_WB ステージのレジスタ書き込み番地、ID_EX ステージの計算に用いられているレジスタ番号 2 つを入力として受け取り、レジスタ A のほうでどのデータをフォワーディングするべきかを知らせる信号 ForwardA とレジスタ B のほうでどのデータをフォワーディングするべきかを知らせる信号 ForwardB を出力する。まず、EX_MEM ステージからフォワーディングを行わないといけないのは、EX_MEM ステージでの結果がレジスタに

書き込まれ、かつ EX_MEM ステージでの結果を書き込むレジスタの値を次の演算で使う時なので、その条件の時に 2 ビットの 01 という信号を出力する。同様に、MEM_WB ステージからのフォワーディングを行わなければならないときは、MEM_WB ステージでの結果がレジスタに書き込まれ、かつ MEM_WB ステージでの結果を書き込むレジスタの値を次の演算で使う時なので、その条件の時に 2 ビットの 10 という信号を出力する。それ以外の時については 2 ビットの 00 という信号を出力している。以上がフォワーディングユニットの仕様である。

8 ハザード検出ユニット

8.1 外部仕様

パイプライン化を行う時にフォワーディングだけでは間に合わず、データハザードが起こってしまうことがある。それは、ロード命令でメモリからフェッチしてきた値を直後の命令で使って何らかの演算を行う時と、Input 命令で外部入力から入力として得たものを直後の命令で何らかの演算の引数として用いるときである。この時は、直後の命令をいったんストールさせ、間に一つ nop 命令を挟まなければならない。そのハザードを検出して信号を出すのがハザード検出ユニットの役割である。

8.2 内部仕様

ハザード検出ユニットの内部仕様について、ハザード検出ユニットのソースコードを示しながら説明する。

```
1     module finding_hazard (
2         input ID_EX_MemRead,ID_EX_Input,
3         input [2:0] ID_EX_RegisterRa,IF_ID_RegisterRa,
              IF_ID_RegisterRb,
4         output hazard_ctl
5         );
6         assign hazard_ctl = (((ID_EX_Input == 1'b1)
7                               ||( ID_EX_MemRead == 1'b1))
8                               && (( ID_EX_RegisterRa ==
                                    IF_ID_RegisterRa)
9                               ||(ID_EX_RegisterRa ==
                                    IF_ID_RegisterRb))) ? 1'b0:
10                              1'b1;
11
12 endmodule
```

前述のとおり、データハザードが起こってしまうのはロード命令の直後の命令がそのロード先のデータを使う時と Input 命令で外部入力から入力として得たものを直後の命令で何らかの演算の引数として用いるときである。その条件を判定するために以下のように設定した。

入力として、ID_EX ステージのメモリ読み込み信号と ID_EX ステージの Input 信号、ID_EX ステージの書き込み先のレジスタの値、IF_ID ステージの演算に用いる 2 つのレジスタの番号を受け取り、1 ビットの信号を出力する。データハザードが起こってしまう時は、ID_EX ステージのメモリ読み込み信号と ID_EX ステージの Input 信号のどちらかが 1 であり、かつ ID_EX ステージの書き込み先のレジスタが、IF_ID ステージにおいての読み出される 2 つのレジスタのいずれかと等しいときに出力が 1 になり、それ以外の時に 0 となっている。この信号をパイプラインレジスタ等に入れ、レジスタ書き込み信号やメモリ書き込み信号等、その命令が実行されることによって変更されてしまうものをフラッシュしてストールを実現する。以上がハザード検出ユニットの内部仕様である。

9 ディスプレイ

9.1 外部仕様

このモジュールは拡張機能であるが、MU500-7SEG 上の 4 桁× 16 個の LED とそのうえ部分にある LED も光らせるためのモジュールである。16 ビットの 2 進数を 4 桁の 16 進数として表現したものを 16 個光らせることが目的なので、このモジュールの外部仕様としては 16 個の 16 桁の 2 進数を入力として受け取り、それを 16 個の 4 桁の 16 進数に変換して光らせる、というものである。

9.2 内部仕様

```
1 module display(
2         input clk,rst_n,
3         input [15:0] reg_0,reg_1,reg_2,reg_3,reg_4,reg_5,
              reg_6,reg_7,reg_8,reg_9,reg_10,reg_11,reg_12,
              reg_13,reg_14,reg_15,
4         input [7:0] ctl,
5         output reg [7:0] disp_1,disp_2,disp_3,disp_4,disp_5,
              disp_6,disp_7,disp_8,
6         output [8:0] sl_out);
7         wire [15:0] wire_reg0,wire_reg1,wire_reg2,wire_reg3,
              wire_reg4,wire_reg5,wire_reg6,wire_reg7,wire_reg8,
```

```
        wire_reg9,wire_reg10,wire_reg11,wire_reg12,
        wire_reg13,wire_reg14,wire_reg15;
8       wire [7:0] disp_reg0_1,disp_reg0_2,disp_reg0_3,
        disp_reg0_4,disp_reg1_1,disp_reg1_2,disp_reg1_3,
        disp_reg1_4,disp_reg2_1,disp_reg2_2,disp_reg2_3,
        disp_reg2_4,disp_reg3_1,disp_reg3_2,disp_reg3_3,
        disp_reg3_4,disp_reg4_1,disp_reg4_2,disp_reg4_3,
        disp_reg4_4,disp_reg5_1,disp_reg5_2,disp_reg5_3,
        disp_reg5_4,disp_reg6_1,disp_reg6_2,disp_reg6_3,
        disp_reg6_4,disp_reg7_1,disp_reg7_2,disp_reg7_3,
        disp_reg7_4,disp_reg8_1,disp_reg8_2,disp_reg8_3,
        disp_reg8_4,disp_reg9_1,disp_reg9_2,disp_reg9_3,
        disp_reg9_4,disp_reg10_1,disp_reg10_2,disp_reg10_3
        ,disp_reg10_4,disp_reg11_1,disp_reg11_2,
        disp_reg11_3,disp_reg11_4,disp_reg12_1,
        disp_reg12_2,disp_reg12_3,disp_reg12_4,
        disp_reg13_1,disp_reg13_2,disp_reg13_3,
        disp_reg13_4,disp_reg14_1,disp_reg14_2,
        disp_reg14_3,disp_reg14_4,disp_reg15_1,
        disp_reg15_2,disp_reg15_3,disp_reg15_4;
9       wire sl_clk_wire,sl_rst_wire;
10      reg [5:0] t;
11      reg [8:0] sel;
12      assign wire_reg0 = reg_0;
13      assign wire_reg1 = reg_1;
14      assign wire_reg2 = reg_2;
15      assign wire_reg3 = reg_3;
16      assign wire_reg4 = reg_4;
17      assign wire_reg5 = reg_5;
18      assign wire_reg6 = reg_6;
19      assign wire_reg7 = reg_7;
20      assign wire_reg8 = reg_8;
21      assign wire_reg9 = reg_9;
22      assign wire_reg10 = reg_10;
23      assign wire_reg11 = reg_11;
24      assign wire_reg12 = reg_12;
25      assign wire_reg13 = reg_13;
26      assign wire_reg14 = reg_14;
27      assign wire_reg15 = reg_15;
28      assign sl_clk_wire = clk;
29      assign sl_rst_wire = rst_n;
30      number reg0(.data_sig(wire_reg0), .disp_out1(
        disp_reg0_1), .disp_out2(disp_reg0_2), .disp_out3
        (disp_reg0_3), .disp_out4(disp_reg0_4));
31      number reg1(.data_sig(wire_reg1), .disp_out1(
        disp_reg1_1), .disp_out2(disp_reg1_2), .disp_out3
```

```
        (disp_reg1_3), .disp_out4(disp_reg1_4));
32      number reg2(.data_sig(wire_reg2), .disp_out1(
        disp_reg2_1), .disp_out2(disp_reg2_2), .disp_out3
        (disp_reg2_3), .disp_out4(disp_reg2_4));
33      number reg3(.data_sig(wire_reg3), .disp_out1(
        disp_reg3_1), .disp_out2(disp_reg3_2), .disp_out3
        (disp_reg3_3), .disp_out4(disp_reg3_4));
34      number reg4(.data_sig(wire_reg4), .disp_out1(
        disp_reg4_1), .disp_out2(disp_reg4_2), .disp_out3
        (disp_reg4_3), .disp_out4(disp_reg4_4));
35      number reg5(.data_sig(wire_reg5), .disp_out1(
        disp_reg5_1), .disp_out2(disp_reg5_2), .disp_out3
        (disp_reg5_3), .disp_out4(disp_reg5_4));
36      number reg6(.data_sig(wire_reg6), .disp_out1(
        disp_reg6_1), .disp_out2(disp_reg6_2), .disp_out3
        (disp_reg6_3), .disp_out4(disp_reg6_4));
37      number reg7(.data_sig(wire_reg7), .disp_out1(
        disp_reg7_1), .disp_out2(disp_reg7_2), .disp_out3
        (disp_reg7_3), .disp_out4(disp_reg7_4));
38      number reg8(.data_sig(wire_reg8), .disp_out1(
        disp_reg8_1), .disp_out2(disp_reg8_2), .disp_out3
        (disp_reg8_3), .disp_out4(disp_reg8_4));
39      number reg9(.data_sig(wire_reg9), .disp_out1(
        disp_reg9_1), .disp_out2(disp_reg9_2), .disp_out3
        (disp_reg9_3), .disp_out4(disp_reg9_4));
40      number reg10(.data_sig(wire_reg10), .disp_out1(
        disp_reg10_1), .disp_out2(disp_reg10_2), .
        disp_out3(disp_reg10_3), .disp_out4(disp_reg10_4)
        );
41      number reg11(.data_sig(wire_reg11), .disp_out1(
        disp_reg11_1), .disp_out2(disp_reg11_2), .
        disp_out3(disp_reg11_3), .disp_out4(disp_reg11_4)
        );
42      number reg12(.data_sig(wire_reg12), .disp_out1(
        disp_reg12_1), .disp_out2(disp_reg12_2), .
        disp_out3(disp_reg12_3), .disp_out4(disp_reg12_4)
        );
43      number reg13(.data_sig(wire_reg13), .disp_out1(
        disp_reg13_1), .disp_out2(disp_reg13_2), .
        disp_out3(disp_reg13_3), .disp_out4(disp_reg13_4)
        );
44      number reg14(.data_sig(wire_reg14), .disp_out1(
        disp_reg14_1), .disp_out2(disp_reg14_2), .
        disp_out3(disp_reg14_3), .disp_out4(disp_reg14_4)
        );
```

```
45          number reg15(.data_sig(wire_reg15), .disp_out1(
                disp_reg15_1), .disp_out2(disp_reg15_2), .
                disp_out3(disp_reg15_3), .disp_out4(disp_reg15_4)
                );
46          assign sl_out = sel;
47          reg [25:0] count;
48          always @(posedge clk or negedge rst_n) begin
49                  if (!rst_n)
50                          count <= 26'h0;
51                  else if (count == (26'd20_000_000 / 300))
52                          count <= 26'h0;
53                  else
54                          count <= count + 26'h1;
55          end
56          always @(posedge clk or negedge rst_n) begin
57                  if (!rst_n)
58                          t <= 4'd0000;
59                  else if (count == (26'd10_000_000 / 300))
                        begin
60                          t <= (t + 1) % 36;
61                          sel[7] <= (t == 2) ? 1:
62                                  (t == 4) ? 0:
63                                  sel[7];
64                          sel[6] <= (t == 6) ? 1:
65                                  (t == 8) ? 0:
66                                  sel[6];
67                          sel[5] <= (t == 10) ? 1:
68                                  (t == 12) ? 0:
69                                  sel[5];
70                          sel[4] <= (t == 14) ? 1:
71                                  (t == 16) ? 0:
72                                  sel[4];
73                          sel[3] <= (t == 18) ? 1:
74                                  (t == 20) ? 0:
75                                  sel[3];
76                          sel[2] <= (t == 22) ? 1:
77                                  (t == 24) ? 0:
78                                  sel[2];
79                          sel[1] <= (t == 26) ? 1:
80                                  (t == 28) ? 0:
81                                  sel[1];
82                          sel[0] <= (t == 30) ? 1:
83                                  (t == 32) ? 0:
84                                  sel[0];
85                          sel[8] <= (t == 34) ? 1:
86                                  (t == 0) ? 0:
```

```
87                  sel[8];
88                  disp_1 <=
89                  (t == 1)? disp_reg0_4:
90                  (t == 5)? disp_reg2_4:
91                  (t == 9)? disp_reg4_4:
92                  (t == 13)? disp_reg6_4:
93                  (t == 17)? disp_reg8_4:
94                  (t == 21)? disp_reg10_4:
95                  (t == 25)? disp_reg12_4:
96                  (t == 29)? disp_reg14_4:
97                  (t == 34)? ctl:
98                  disp_1;
99                  disp_2 <=
100                 (t == 1)? disp_reg0_3:
101                 (t == 5)? disp_reg2_3:
102                 (t == 9)? disp_reg4_3:
103                 (t == 13)? disp_reg6_3:
104                 (t == 17)? disp_reg8_3:
105                 (t == 21)? disp_reg10_3:
106                 (t == 25)? disp_reg12_3:
107                 (t == 29)? disp_reg14_3:
108                 disp_2;
109                 disp_3 <=
110                 (t == 1)? disp_reg0_2:
111                 (t == 5)? disp_reg2_2:
112                 (t == 9)? disp_reg4_2:
113                 (t == 13)? disp_reg6_2:
114                 (t == 17)? disp_reg8_2:
115                 (t == 21)? disp_reg10_2:
116                 (t == 25)? disp_reg12_2:
117                 (t == 29)? disp_reg14_2:
118                 disp_3;
119                 disp_4 <=
120                 (t == 1)? disp_reg0_1:
121                 (t == 5)? disp_reg2_1:
122                 (t == 9)? disp_reg4_1:
123                 (t == 13)? disp_reg6_1:
124                 (t == 17)? disp_reg8_1:
125                 (t == 21)? disp_reg10_1:
126                 (t == 25)? disp_reg12_1:
127                 (t == 29)? disp_reg14_1:
128                 disp_4;
129                 disp_5 <=
130                 (t == 1)? disp_reg1_4:
131                 (t == 5)? disp_reg3_4:
132                 (t == 9)? disp_reg5_4:
```

```
                        (t == 13)? disp_reg7_4:
                        (t == 17)? disp_reg9_4:
                        (t == 21)? disp_reg11_4:
                        (t == 25)? disp_reg13_4:
                        (t == 29)? disp_reg15_4:
                        disp_5;
                        disp_6 <=
                        (t == 1)? disp_reg1_3:
                        (t == 5)? disp_reg3_3:
                        (t == 9)? disp_reg5_3:
                        (t == 13)? disp_reg7_3:
                        (t == 17)? disp_reg9_3:
                        (t == 21)? disp_reg11_3:
                        (t == 25)? disp_reg13_3:
                        (t == 29)? disp_reg15_3:
                        disp_6;
                        disp_7 <=
                        (t == 1)? disp_reg1_2:
                        (t == 5)? disp_reg3_2:
                        (t == 9)? disp_reg5_2:
                        (t == 13)? disp_reg7_2:
                        (t == 17)? disp_reg9_2:
                        (t == 21)? disp_reg11_2:
                        (t == 25)? disp_reg13_2:
                        (t == 29)? disp_reg15_2:
                        disp_7;
                        disp_8 <=
                        (t == 1)? disp_reg1_1:
                        (t == 5)? disp_reg3_1:
                        (t == 9)? disp_reg5_1:
                        (t == 13)? disp_reg7_1:
                        (t == 17)? disp_reg9_1:
                        (t == 21)? disp_reg11_1:
                        (t == 25)? disp_reg13_1:
                        (t == 29)? disp_reg15_1:
                        disp_8;
                end else begin
                        t <= t;
                        sel <= sel;
                        disp_1 <= disp_1;
                        disp_2 <= disp_2;
                        disp_3 <= disp_3;
                        disp_4 <= disp_4;
                        disp_5 <= disp_5;
                        disp_6 <= disp_6;
                        disp_7 <= disp_7;
```

```
179                         disp_8 <= disp_8;
180                 end
181         end
182 endmodule
183 module SEVENSEG_LED (
184         input [3:0] a,
185         output [7:0] output_signal);
186         assign output_signal = (a == 4'b0000) ? 8'b1111_1100:
187                                (a == 4'b0001)? 8'b0110_0000:
188                                (a == 4'b0010)? 8'b1101_1010:
189                                (a == 4'b0011)? 8'b1111_0010:
190                                (a == 4'b0100)? 8'b0110_0110:
191                                (a == 4'b0101)? 8'b1011_0110:
192                                (a == 4'b0110)? 8'b1011_1110:
193                                (a == 4'b0111)? 8'b1110_0000:
194                                (a == 4'b1000)? 8'b1111_1110:
195                                (a == 4'b1001)? 8'b1111_0110:
196                                (a == 4'b1010)? 8'b1110_1110:
197                                (a == 4'b1011)? 8'b0011_1110:
198                                (a == 4'b1100)? 8'b0001_1010:
199                                (a == 4'b1101)? 8'b0111_1010:
200                                (a == 4'b1110)? 8'b1001_1110:
201                                8'b1000_1110;
202
203 endmodule
204 module number( ビットの値を入力として四ケタずつの値（１６進数）
        を返す//16
205         input [15:0] data_sig,
206         output [7:0] disp_out1,disp_out2,disp_out3,disp_out4)
                ;
207         wire n_wire_clk,n_wire_rst;
208         wire [7:0] disp_wire1,disp_wire2,disp_wire3,
                disp_wire4;
209         wire [3:0] data_wire1,data_wire2,data_wire3,
                data_wire4;
210         assign data_wire1 = data_sig [3:0]; //___0
211         assign data_wire2 = data_sig [7:4]; //__0_
212         assign data_wire3 = data_sig [11:8]; //_0__
213         assign data_wire4 = data_sig [15:12]; //0___
214         SEVENSEG_LED l1(.a(data_wire1), .output_signal(
                disp_wire1));
215         SEVENSEG_LED l2(.a(data_wire2), .output_signal(
                disp_wire2));
216         SEVENSEG_LED l3(.a(data_wire3), .output_signal(
                disp_wire3));
```

```
217         SEVENSEG_LED l4(.a(data_wire4), .output_signal(
                disp_wire4));
218         assign disp_out1 = disp_wire1;
219         assign disp_out2 = disp_wire2;
220         assign disp_out3 = disp_wire3;
221         assign disp_out4 = disp_wire4;
222
223 endmodule
```

まず、この display モジュールの内部では導入課題で用意したものとほとんど同じモジュールがいくつか導入されている。SEVENSEG_LED モジュールは組み合わせ回路であり、4 桁の 2 進数を入力として受け取りそれが 16 進数 1 桁に直すとどのように LED 上に表示されるかを返すモジュールである。続いて、number モジュールはこちらも組み合わせ回路であり、16 桁の 2 進数を入力として受け取り、まずそれぞれを 4 桁の 2 進数 4 つに分ける。そのうえで先ほど用意した SEVENSEG_LED を用いてその 4 桁の 2 進数を 1 桁の 16 進数にするとどのように表示されるべきかを 4 桁分計算しその結果を出力する、というものである。その number モジュールを用いて動作するのが display モジュールである。まず、入力として 16 桁の 2 進数 16 個 (MU500-7SEG 上の 4 桁× 16 個の LED を光らせるためのもの) と 8 桁の 2 進数 1 つ (上の LED を光らせるためのもの) を受け取る。まず 16 個の 2 進数についてそれぞれを先ほどの number モジュールに入れることによって 4 桁の 16 進数の表示の仕方に変える。ここで、MU500-7SEG 上の 4 桁× 16 個の LED は 16 個同時に光らせることはできず、同時に 8 桁までしか光らせることが出来ない。これには上の LED も含まれているので、導入課題と同様にダイナミック点灯をしなくてはならない。これと同時に、今どの LED を光らせるかを選ぶセレクタの値と LED に表示させたい値を同時に更新するとうまく表示されない (セレクタの値が 1 になっている間に値を更新するとその更新が反映されるという仕様なので) ので、異なるタイミングで変えなければならない。そのためにまず、クロックが立ち上がるたびに 1 足されていくカウンタを内部で用意しそのカウンタの値を 36 で割ったときの余りで場合分けし、基本的にその値が偶数の時はセレクタの値を更新して奇数の時は出力に割り当てる数字を変えるという風にしてある。このような操作をした結果 8 つの 8 桁の 2 進数 (8 桁分の 1 桁の 16 進数の光らせ方) を出力する。また、20MHz 等の速いクロックをそのまま値の更新部分のクロックとして扱ってしまうと早すぎてうまく動かなくなってしまうので適当な遅延を入れている。細かい更新の仕方は省略する。以上が display モジュールの内部仕様である。

10　考察と感想

10.1　考察

まず、全体についての考察だが、周波数の限界について述べようと思う。ソート速度コンテストに提出するために様々なソートのアルゴリズムを用意したが、アルゴリズムによって動く最大周波数に違いがあった。一番単純なバブルソートは 120MHz まで動いたが、より複雑な基数ソートやクイックソートなどはもっと低い周波数でしか動かなかった。動かないというのは、きちんと動作しないという意味であり、こちら側が想定していない番地に書き込みが行われたり、ソートのループを抜けずに回り続けてしまうといった挙動を示すことである。このようなことが起こってしまう原因として僕が考えるのは分岐命令関連のところが周波数を上げると間に合わなくなってしまうのではないかと考える。バブルソートでは、分岐命令が少なく単純なものとなっているが基数ソートやクイックソートにおいては分岐命令の数も多く複雑になってしまっている。この時の条件コードの計算を直前の比較演算命令で行っていたり、分岐命令の直後に (不成立の場合に通る) もう一つの分岐命令があったりといった形になっているので、その部分での判定が周波数を上げることによって計算が間に合っていないのではないかと思う。そのために、条件分岐のループを抜けることが出来ずに停止しないといったことが起こってしまうのではないかと考える。

次に私が担当したモジュールについての考察だが、制御部を組み合わせ回路にしたことの利点と欠点、実際にどちらのほうが速く動きそうかについて述べようと思う。先述の通り、自分たちはパイプライン化するときに制御部もほかの部分と同じクロックで動くように設計すると制御コードの送信が間に合わないのではないか、順序回路にするより組み合わせ回路にしたほうがタイミング制約が楽なのではないかと考えたので組み合わせ回路にした。しかし、それは本当に正しかったのだろうか。うまくタイミング制約をしてやれば、順序回路のほうがいいのではないか。実際に友人の班を見ると制御部が順序回路である班はあったと思う。しかし、やはり制御部が組み合わせ回路であることによってそのほかのコンポーネントとの連携が楽になるというメリットは回路を組み立てる上では速度よりも大切なものであると思うので、組み合わせ回路にして正解であったと考える。

10.2　感想

はじめは何をどうしていいかわからなかったが協力しながら組み立てていくのは楽しかった。しかし、1 つ 1 つの部品を作っていくときにその段階でしっかりとテストを行わずに一気に全体を組み立ててからテストを行おうとすると、バグが発生した時にどこがおかしいのかを見つけるのがとても大変

だったので、部品が必ず正しい挙動を示すということが保障されていることがどれだけ大切かを学んだ。同時に、ソートのためのアセンブラを書いて、そのデバッグを行っているときに、そのバグがアセンブラの問題なのかハードウェアの問題なのかわからないという状態が生じた時に、普段我々がコードを書くときにハードウェアではバグが起こらないと保証されていることがいかに大切かを学んだ。

	Input Port	Output Port	RR	RF	FR	FF
1	inst[4]	ALUSrc2	8.795			9.279
2	inst[4]	AS_BC		8.197	8.671	
3	inst[4]	Halt	7.965			8.208
4	inst[4]	Input		8.082	8.570	
5	inst[4]	MemtoReg		10.324	10.601	
6	inst[4]	Output	7.935			8.175
7	inst[4]	RegWrite		8.463	8.946	
8	inst[4]	opcode[0]	8.600			8.925
9	inst[5]	ALUSrc2	8.953			9.436
10	inst[5]	AS_BC		8.371	8.830	
11	inst[5]	Halt	8.104			8.389
12	inst[5]	Input		8.332	8.808	
13	inst[5]	MemtoReg		10.574	10.839	
14	inst[5]	Output		8.016	8.486	
15	inst[5]	RegWrite		8.711	9.181	
16	inst[5]	opcode[1]	8.366			8.632
17	inst[6]	ALUSrc2	8.853			9.362
18	inst[6]	ALUorShifter		7.905	8.356	
19	inst[6]	AS_BC		8.256	8.709	
20	inst[6]	Halt	8.112			8.412
21	inst[6]	Input	8.419			8.658
22	inst[6]	MemtoReg	10.450			10.900
23	inst[6]	Output	8.098			8.349
24	inst[6]	RegWrite		8.522	8.985	
25	inst[6]	opcode[2]	7.994			8.243
26	inst[7]	ALUSrc2	9.022			9.549
27	inst[7]	ALUorShifter	8.566			8.835
28	inst[7]	AS_BC		8.451	8.905	
29	inst[7]	Halt	8.409			8.688
30	inst[7]	Input	8.655			8.911
31	inst[7]	MemtoReg	10.686			11.153
32	inst[7]	Output	8.381			8.654
33	inst[7]	RegWrite		8.717	9.181	
34	inst[7]	opcode[3]	8.275			8.529
35	inst[8]	Branch[0]	7.852			8.143
36	inst[8]	RegDst[0]	7.869			8.161
37	inst[9]	Branch[1]	7.881			8.198
38	inst[9]	RegDst[1]	8.025			8.375
39	inst[10]	Branch[2]	7.777			8.067
40	inst[10]	RegDst[2]	7.711			7.985
41	inst[11]	ALUorShifter	8.926			9.227
42	inst[11]	AS_BC	8.967			9.270
43	inst[11]	Branch[0]	8.378	8.356	8.752	8.767
44	inst[11]	Branch[1]	8.346	7.981	8.427	8.741

図 2: 制御部遅延 1

	Input Port	Output Port	RR	RF	FR	FF
45	inst[11]	Branch[2]		8.370	8.872	
46	inst[11]	RegDst[0]	7.909			8.245
47	inst[11]	RegWrite	9.247	9.134	9.623	9.547
48	inst[11]	SLI	7.662			8.002
49	inst[11]	opcode[1]		8.676	9.228	
50	inst[11]	opcode[2]	8.327	8.304	8.706	8.722
51	inst[11]	opcode[3]	8.614			8.872
52	inst[12]	ALUorShifter		8.934	9.433	
53	inst[12]	AS_BC	9.090			9.357
54	inst[12]	Branch[0]	8.503	8.482	8.890	8.860
55	inst[12]	Branch[1]	8.468	8.107	8.565	8.826
56	inst[12]	Branch[2]		8.492	8.957	
57	inst[12]	RegDst[1]	7.804			8.100
58	inst[12]	RegWrite		9.255	9.755	
59	inst[12]	SLI		7.709	8.169	
60	inst[12]	opcode[0]	8.287			8.613
61	inst[12]	opcode[1]		8.806	9.365	
62	inst[12]	opcode[2]	8.450	8.430	8.838	8.809
63	inst[12]	opcode[3]		8.579	9.121	
64	inst[13]	ALUorShifter	9.387			9.708
65	inst[13]	AS_BC		9.337	9.830	
66	inst[13]	Branch[0]	8.752	8.903	9.305	9.139
67	inst[13]	Branch[1]	8.766	8.528	8.980	9.176
68	inst[13]	Branch[2]		8.790	9.307	
69	inst[13]	RegDst[2]	8.159			8.480
70	inst[13]	RegWrite		9.668	10.160	
71	inst[13]	SLI	8.123			8.483
72	inst[13]	opcode[0]		8.579	9.022	
73	inst[13]	opcode[1]		9.227	9.779	
74	inst[13]	opcode[2]		8.844	9.244	
75	inst[13]	opcode[3]	9.075			9.353
76	inst[14]	ALUSrc1		7.432	7.868	
77	inst[14]	ALUSrc2		8.285	8.541	
78	inst[14]	ALUorShifter	8.683	10.140	10.595	9.000
79	inst[14]	AS_BC	7.989	10.116	10.612	8.272
80	inst[14]	Branch[0]	9.742	9.574	9.960	10.102
81	inst[14]	Branch[1]	9.417	9.611	9.974	9.727
82	inst[14]	Branch[2]	9.742			9.998
83	inst[14]	Halt	8.526			8.853
84	inst[14]	Input	8.772			9.076
85	inst[14]	MemRead		8.527	8.840	
86	inst[14]	MemWrite	8.569			8.892
87	inst[14]	MemtoReg	10.803	9.851	10.171	11.318
88	inst[14]	Output	8.498			8.819

図 3: 制御部遅延 2

88	inst[14]	Output	8.498			8.819
89	inst[14]	RegDst[0]	8.336	8.263	8.702	8.668
90	inst[14]	RegDst[1]	8.288	8.204	8.651	8.604
91	inst[14]	RegDst[2]	8.260	8.171	8.627	8.577
92	inst[14]	RegWrite	8.264	8.146	8.619	8.536
93	inst[14]	SLI		8.915	9.331	
94	inst[14]	opcode[0]	8.766	8.712	9.129	9.113
95	inst[14]	opcode[1]	8.226	10.021	10.557	8.487
96	inst[14]	opcode[2]	8.271	9.646	10.030	8.602
97	inst[14]	opcode[3]	7.976	9.785	10.283	8.247
98	inst[15]	ALUSrc1	7.634			7.957
99	inst[15]	ALUSrc2		8.521	8.754	
...	inst[15]	ALUorShifter	10.361			10.665
...	inst[15]	AS_BC	10.378			10.641
...	inst[15]	Branch[0]	9.726	9.868	10.267	10.099
...	inst[15]	Branch[1]	9.740	9.493	9.942	10.136
...	inst[15]	Branch[2]		9.764	10.267	
...	inst[15]	Halt	8.722			9.052
...	inst[15]	Input	8.968			9.275
...	inst[15]	MemRead		8.433	8.739	
...	inst[15]	MemWrite		8.745	9.183	
...	inst[15]	MemtoReg	10.999	9.714	10.049	11.517
...	inst[15]	Output	8.694			9.018
...	inst[15]	RegDst[0]	8.500	8.530	8.969	8.802
...	inst[15]	RegDst[1]	8.493	8.386	8.824	8.833
...	inst[15]	RegDst[2]	8.424	8.438	8.892	8.709
...	inst[15]	RegWrite	8.470	8.438	8.906	8.747
...	inst[15]	SLI	9.097			9.440
...	inst[15]	opcode[0]	7.939			8.258
...	inst[15]	opcode[1]	10.323			10.546
...	inst[15]	opcode[2]	9.796			10.171
...	inst[15]	opcode[3]	10.049	8.079	8.565	10.310

図 4: 制御部遅延 3

Input Port	Output Port	RR	RF	FR	FF
Read1[0]	Data1[0]	11.182	11.096	11.585	11.490
Read1[0]	Data1[1]	11.289	11.206	11.735	11.547
Read1[0]	Data1[2]	13.521	13.561	13.925	13.965
Read1[0]	Data1[3]	11.320	11.271	11.767	11.608
Read1[0]	Data1[4]	12.480	12.371	12.934	12.844
Read1[0]	Data1[5]	11.672	11.736	12.118	12.065
Read1[0]	Data1[6]	13.341	13.297	13.788	13.652
Read1[0]	Data1[7]	13.473	13.365	13.918	13.724
Read1[0]	Data1[8]	13.147	13.116	13.537	13.497
Read1[0]	Data1[9]	12.440	12.301	12.826	12.692
Read1[0]	Data1[10]	14.550	14.407	14.940	14.788
Read1[0]	Data1[11]	13.555	13.407	13.910	13.877
Read1[0]	Data1[12]	12.186	12.121	12.577	12.503
Read1[0]	Data1[13]	13.314	13.283	13.704	13.664
Read1[0]	Data1[14]	12.101	12.100	12.491	12.572
Read1[0]	Data1[15]	12.286	12.136	12.675	12.516
Read1[1]	Data1[0]	11.572	11.560	12.036	11.890
Read1[1]	Data1[1]	11.090	10.950	11.495	11.346
Read1[1]	Data1[2]	13.946	13.949	14.362	14.402
Read1[1]	Data1[3]	11.118	11.012	11.523	11.408
Read1[1]	Data1[4]	12.276	12.186	12.672	12.582
Read1[1]	Data1[5]	11.479	11.486	11.884	11.882
Read1[1]	Data1[6]	13.141	13.040	13.546	13.436
Read1[1]	Data1[7]	13.284	13.119	13.689	13.515
Read1[1]	Data1[8]	13.548	13.570	13.980	13.902
Read1[1]	Data1[9]	12.891	12.738	13.267	13.133
Read1[1]	Data1[10]	14.953	14.860	15.383	15.193
Read1[1]	Data1[11]	13.671	13.522	14.022	13.988
Read1[1]	Data1[12]	12.585	12.573	13.018	12.906
Read1[1]	Data1[13]	13.855	13.881	14.259	14.171
Read1[1]	Data1[14]	12.214	12.212	12.598	12.679
Read1[1]	Data1[15]	12.686	12.589	13.119	12.943
Read1[2]	Data1[0]	9.027	8.901	9.402	9.315
Read1[2]	Data1[1]	8.664	8.557	9.050	8.993
Read1[2]	Data1[2]	10.532	10.525	10.908	10.940
Read1[2]	Data1[3]	8.998	8.931	9.438	9.271
Read1[2]	Data1[4]	10.474	10.349	10.886	10.752
Read1[2]	Data1[5]	9.867	9.829	10.279	10.232
Read1[2]	Data1[6]	10.550	10.430	10.965	10.882
Read1[2]	Data1[7]	10.654	10.566	11.128	10.940
Read1[2]	Data1[8]	9.907	9.866	10.346	10.296
Read1[2]	Data1[9]	10.015	9.869	10.454	10.299
Read1[2]	Data1[10]	10.581	10.412	11.019	10.841
Read1[2]	Data1[11]	10.362	10.236	10.801	10.666

図 5: レジスタファイル遅延 1

Input Port	Output Port	RR	RF	FR	FF
Read1[2]	Data1[12]	9.560	9.476	9.997	9.904
Read1[2]	Data1[13]	10.413	10.326	10.800	10.824
Read1[2]	Data1[14]	10.074	9.983	10.462	10.482
Read1[2]	Data1[15]	8.551	8.447	8.903	8.790
Read2[0]	Data2[0]	11.819	11.705	12.256	12.175
Read2[0]	Data2[1]	13.939	14.044	14.330	14.459
Read2[0]	Data2[2]	11.652	11.545	12.066	11.955
Read2[0]	Data2[3]	12.560	12.488	12.997	12.958
Read2[0]	Data2[4]	13.047	12.851	13.460	13.264
Read2[0]	Data2[5]	14.667	14.461	15.080	14.888
Read2[0]	Data2[6]	14.437	14.412	14.850	14.816
Read2[0]	Data2[7]	12.367	12.155	12.780	12.586
Read2[0]	Data2[8]	12.159	12.071	12.557	12.460
Read2[0]	Data2[9]	12.002	11.982	12.400	12.371
Read2[0]	Data2[10]	14.492	14.299	14.890	14.692
Read2[0]	Data2[11]	12.403	12.361	12.802	12.751
Read2[0]	Data2[12]	12.852	12.781	13.250	13.170
Read2[0]	Data2[13]	13.525	13.410	13.923	13.799
Read2[0]	Data2[14]	12.072	11.993	12.470	12.382
Read2[0]	Data2[15]	11.891	11.784	12.289	12.173
Read2[1]	Data2[0]	11.737	11.631	12.135	12.056
Read2[1]	Data2[1]	13.860	13.943	14.250	14.324
Read2[1]	Data2[2]	12.128	12.026	12.513	12.402
Read2[1]	Data2[3]	12.195	12.109	12.585	12.490
Read2[1]	Data2[4]	13.375	13.192	13.778	13.628
Read2[1]	Data2[5]	14.995	14.803	15.398	15.239
Read2[1]	Data2[6]	14.765	14.754	15.167	15.189
Read2[1]	Data2[7]	12.695	12.493	13.097	12.928
Read2[1]	Data2[8]	12.091	12.017	12.551	12.510
Read2[1]	Data2[9]	11.934	11.928	12.415	12.421
Read2[1]	Data2[10]	14.424	14.245	14.885	14.739
Read2[1]	Data2[11]	12.335	12.307	12.810	12.801
Read2[1]	Data2[12]	12.784	12.727	13.248	13.220
Read2[1]	Data2[13]	13.457	13.356	13.919	13.850
Read2[1]	Data2[14]	12.004	11.939	12.465	12.433
Read2[1]	Data2[15]	11.823	11.730	12.292	12.223
Read2[2]	Data2[0]	9.217	9.102	9.614	9.526
Read2[2]	Data2[1]	11.379	11.445	11.772	11.829
Read2[2]	Data2[2]	8.924	8.794	9.317	9.178
Read2[2]	Data2[3]	9.531	9.442	9.924	9.826
Read2[2]	Data2[4]	10.384	10.218	10.763	10.588
Read2[2]	Data2[5]	11.967	11.772	12.346	12.142
Read2[2]	Data2[6]	11.629	11.628	12.042	12.032
Read2[2]	Data2[7]	8.828	8.668	9.177	9.008

図 6: レジスタファイル遅延 2

Read2[2]	Data2[8]	9.179	9.073	9.552	9.437
Read2[2]	Data2[9]	9.085	9.031	9.497	9.434
Read2[2]	Data2[10]	10.540	10.389	10.951	10.791
Read2[2]	Data2[11]	10.256	10.113	10.624	10.597
Read2[2]	Data2[12]	10.077	9.995	10.490	10.399
Read2[2]	Data2[13]	10.158	10.086	10.541	10.506
Read2[2]	Data2[14]	9.731	9.629	10.114	10.049
Read2[2]	Data2[15]	8.716	8.598	9.069	8.942

図 7: レジスタファイル遅延 3