目次

1　実験の目的

ハードウェア記述言語（HDL）を用いて論理回路を設計する手法の収得

2　実験器具及びツール

Quartus (Quartus Prime 20.1) Lite Edition

3　実験内容

3.1　実習 1

図 1 のように 4 つのレジスタと入力 DIN をマルチプレクサに接続し、選択 S で指定したレジスタまたは DIN の値を MUX から出力させる。そして入力制御信号が 1 になっているレジスタに値を出力するような回路を VerilogHDL で記述した。

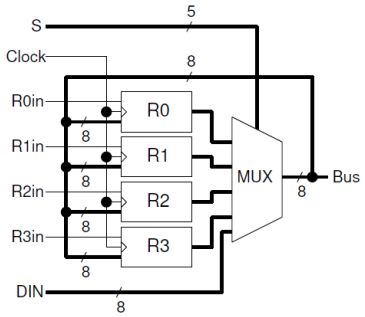

図 1　マルチプレクサからのレジスタへの入力

3.2　実習 2

ALU モジュールとAレジスタおよびGレジスタを VerilogHDL で記述し実習 1 で作成した回路を追加した。

3.3　実習 3

表 1 に基づいて制御ユニットの動作を VerilogHDL で記述し実習 2 の回路に追加した。

4　実験結果

4.1　実習 1

VerilogHDL でコード 4 のように記述した。コード 1 はコード 4 から一部抜粋したものである。まず、Reg0 から Reg3 のモジュール Reg を作成した。Mux は入力部の選択信号を S として宣言をした。S に応じて Bus

に出力する値をレジスタから選択させた。

ソースコード 1　実習 1 の一部抜粋

```
1  module Reg(D,Clk,En,Res,Q);
2          input [7:0]D;
3          input Clk,En,Res;
4          output [7:0]Q;
5          reg [7:0]Q1;
6
7          always @(posedge Clk)begin
8                  if(Res == 0)
9                          Q1 <= 0;
10                 else
11                         if(En == 1)
12                                 Q1 <= D;
13         end
14         assign Q = Q1;
15 endmodule
16
17 module MUX(Clk,R0,R1,R2,R3,Din,S,Bus);
18         input Clk;
19         input [7:0]R0,R1,R2,R3,Din;
20         input [4:0]S;
21         output [7:0]Bus;
22         reg [7:0]Bus1;
23
24         always @(posedge Clk)begin
25                 if(S[0] == 1)
26                         Bus1 <= R0;
27                 else if(S[1] == 1)
28                         Bus1 <= R1;
29                 else if(S[2] == 1)
30                         Bus1 <= R2;
31                 else if(S[3] == 1)
32                         Bus1 <= R3;
33                 else if(S[4] == 1)
34                         Bus1 <= Din;
35         end
36         assign Bus = Bus1;
37 endmodule
```

実際、機能シミュレータの結果は図 2 のようになった。

Din から読み取った値を Reg0 に格納し、その値を Reg1 にコピーをしてみる。R1 の値は R0 の変更に続いて Din の値 11111111 となっている事がわかり予想と同じ結果になった。次に処理過程をみてみる

まず、clk の立ち上がり 6ns のとき、選択信号 S=10000 で S[4]=1 だから Din が選択された。そして AUX にて Din の値がレジスタに伝搬される。このとき、レジスタの En が 1 のものは R0 ただ一つである。そのため、RegR0 に Din の値すなわち 11111111 が格納される。

次に S が変化したとき、S は 00001 となり R 0 の値が各レジスタに伝搬ｓる。このとき En が 1 のものは R1

だけであり、RegR1 に RegR0 の値すなわち 11111111 が格納されていることが確かめられる。

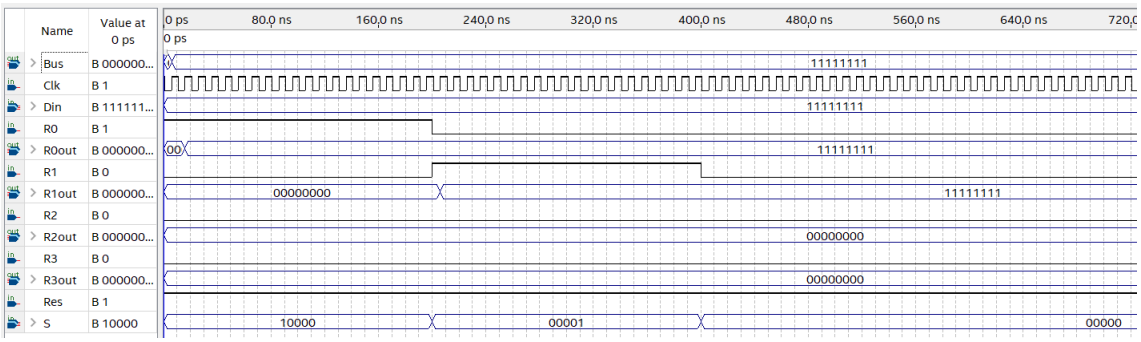

図 2　実習 1 における MUX の結果

4.2　実習 2

ALU モジュールとAレジスタおよびGレジスタを VerilogHDL で記述し実習 1 で作成した回路を追加した。

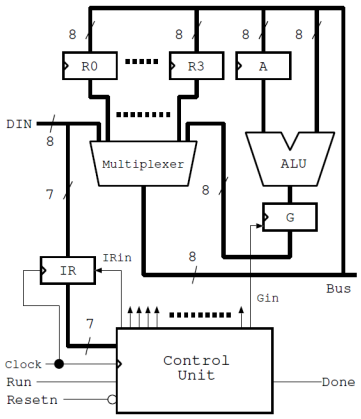

図 3　プロセッサのブロック図

VerilogHDL でコード 5 のように記述した。コード 2 はコード 5 から一部抜粋したものである。ALU モジュールでは減算を足し算と同じようにふるまうよう mode が 1 のときは 2 の補数をとり足しあわせた。また今回はキャリーについては考えていないが、zenkasan モジュールも作成した。

ソースコード 2　実習 2 の一部抜粋

```
1  module Zenkasan(A,B,Cin,S,Cout);
2          input A, B, Cin;
3          output S,Cout;
4
5          assign S = (A^B)^Cin;
6          assign Cout = (A&B)|(B&Cin)|(Cin&A);
7  endmodule
8
```

```
9
10  module ALU(Mode,A,B,Y);
11          input Mode;
12          input [7:0]A,B;
13          output [7:0]Y;
14          wire [7:0]A2,C;
15
16          function [7:0]returnA;
17                  input mode;
18                  input [7:0]Ain;
19
20                  if(mode == 1)
21                          returnA = (Ain^8'b11111111)+1'b1;
22                  else
23                          returnA = Ain;
24          endfunction
25
26          assign A2 = returnA(Mode,A);
27
28          Zenkasan zenkasan0(A2[0],B[0],0,Y[0],C[0]);
29          Zenkasan zenkasan1(A2[1],B[1],C[0],Y[1],C[1]);
30          Zenkasan zenkasan2(A2[2],B[2],C[1],Y[2],C[2]);
31          Zenkasan zenkasan3(A2[3],B[3],C[2],Y[3],C[3]);
32          Zenkasan zenkasan4(A2[4],B[4],C[3],Y[4],C[4]);
33          Zenkasan zenkasan5(A2[5],B[5],C[4],Y[5],C[5]);
34          Zenkasan zenkasan6(A2[6],B[6],C[5],Y[6],C[6]);
35          Zenkasan zenkasan7(A2[7],B[7],C[6],Y[7],C[7]);
36  endmodule
```

実際、機能シミュレータの結果は図4のようになった。
0ps から 10ps の間を①,10ps から 20ps を②,20ps から 30ps を③,30ps から 40ps を④とした。
①のとき、制御信号は Din をさしているので、Multiplexer には Din の値が入る。実際、Din の値は 10000 であり、Mulplexer の出力 Bus も 10000 となっていて Din の値が選択された事が分かる。また、ALU は非同期回路なので、Bus の値がそのまま ALU に入力され出力される。今回は、Mode が 0 で A の値も 0 なので、出力は Bus の値となる。実際、ALU の出力 Aluout は 10000 となっている。このことから①の状態は意図した通り動いていることが確認される。
②のとき、RegA の Enable(EnA) を 1 にした。すると RegA には Bus の値が入力され ALU に値を出力する。実際、RegA の出力 Aout は 0 から Bus の値 10000 に変化したことが見て分かる。そして ALU で Bu の値と RegA の値が加減されて出力される。実際、ALU の出力結果 Aluout では 100000 となっている。これは Bus の値 10000 と Aout の値 10000 の足し合わせとなっている。このことから②の状態は意図した通り動いていることが確認される。
③のとき、RegG の Enable(EnR) を 1 にした。すると、RegG には ALU の結果が入力される。実際、RegG の出力結果 Gout では 0 から 100000 に変化していることがわかる。これは ALU の結果 Aluout100000 と一致している。また、Multiplexer では Din が選択されているため、Multiplexer には Gout が選択されていないことも分かる。このことから③の状態は意図した通り動いていることが確認される。

④のとき、S[4] を 0 とし S[5] を 1 とした。これはそれぞれ Din と RegG の制御信号にあたる。よって Multiplexer では RegG の値が選択され Bus に出力されると予期される。実際、Bus の値は 10000 から 100000 に変更されたことがわかる。この値は Din の値 (10000) と RegG の値 (100000) と一致していて確かに Din から RegG が選択されたことが分かる。

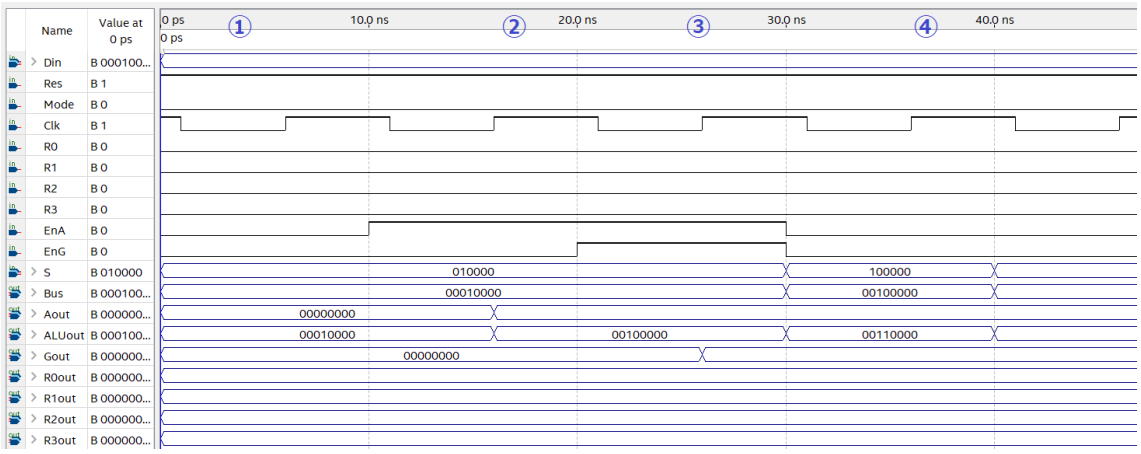

図 4　実習 2 における結果

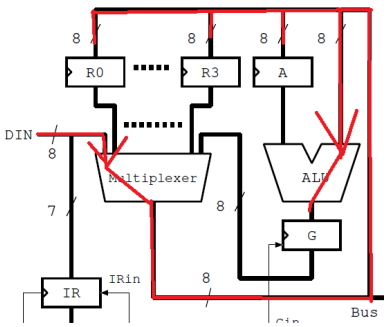

図 5　実習 2 における①の結果

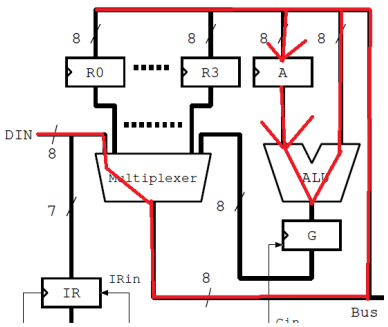

図 6　実習 2 における②の結果

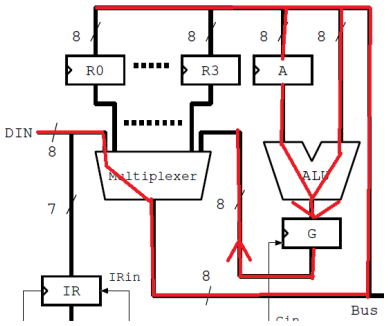

図 7　実習 2 における③の結果

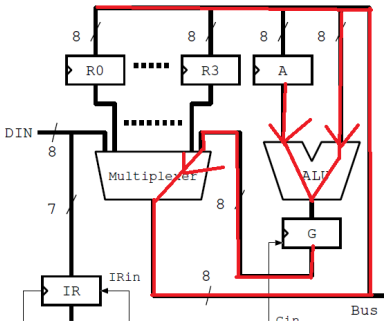

図 8　実習 2 における④の結果

次に Mode を 1 にして減算ができるか確認を行った。まず,Din の値を 01 とし、RegA の Enable(EnA) を 1 とした。そして、EnA を 0 とした後、Din を 11 とした。これは RegA には 01 が保持され、Bus から 11 が出力されるので、$11-1=10(=3-1=2)$ となる事が予期される. 実際 RegA は 0 から 01 が保持され、Alu の出力 Aluout は 10 になった事が確認できる。

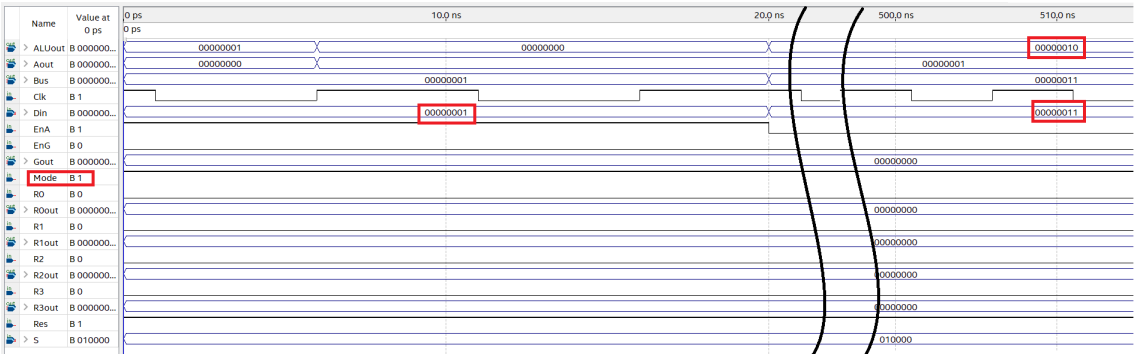

図 9　実習 2 における結果

4.3　実習 3

表 1 に基づいて制御ユニットの動作を VerilogHDL で記述し実習 2 の回路に追加した。

表 1　各タイミングステップで出力する制御信号

命令	T_0	T_1	T_2	T_3
mv	IR_{in}	RY_{out}, RX_{in}, $Done$	-	-
mvi		DIN_{out}, RX_{in}, $Done$	-	-
add		RX_{out}, A_{in}	RY_{out}, G_{in}	G_{out}, RX_{in}, $Done$
sub		RX_{out}, A_{in}	RY_{out}, G_{in}, $Mode$	G_{out}, RX_{in}, $Done$

VerilogHDL でコード 6 のように記述した。コード 3 はコード 6 から一部抜粋したものである。まず、状態遷移の状態、オペランドの値、オペコードの値をそれぞれパラメータ化を行った。次に状態遷移においては順序回路を用いた。続いて、オペコードフェチとオペランドフェチをＩＲから入力された命令から読み取る。そして、Decode や制御信号生成に関しては組み合わせ回路を用いた。Decode ではまず、タイミングステップで分岐させた。そして、どの命令であるかを確認し、各命令の各タイミングステップに適したレジスタの Enable を 1 に行う処理をした。制御生成の部分も同様に、タイミングステップで分岐させ、命令を確認し、制御信号を生成した。

ソースコード 3　実習 3 の一部抜粋

```
1
2  //set parameter
3  parameter T0 = 2'b00,
```

```
                                T1 = 2'b01,
                                T2 = 2'b10,
                                T3 = 2'b11;

    parameter OPNONE = 6'b000000,
                                OPG = 6'b000001,
                                OPD = 6'b000010,
                                OPR3 = 6'b000100,

    parameter OPMV = 3'b000,
                                OPMVI = 3'b001,
                                OPADD = 3'b010,
                                OPSUB = 3'b011;

always @(posedge Clk)

                            begin
                                    case(cur_st)
                                            T0: cur_st <= T1;
                                            T1: cur_st <= T2;
                                            T2: cur_st <= T3;
                                            T3: cur_st <= T3;
                                            default: cur_st <= T0;
                                    endcase

            end

  //opeland fetch and opecode fetch
    assign num_Ry = Ir[1:0];
    assign num_Rx = Ir[3:2];
    assign num_Ir = Ir[6:4];


    //Decode of instruct
    assign Rxin = Decoder(cur_st,num_Ir,num_Rx);

    //Generate signal
    assign Rxout = Generater(cur_st,num_Ir,num_Rx,num_Ry);

  /*
    Decode of signal function_________________
    */
    function [6:0]Decoder;

            begin
                    case(cur_st)
                            T1: decoder2 = T1Decode(num_Ir,num_Rx);
                            T2: decoder2 = T2Decode(num_Ir,num_Rx);
                            T3: decoder2 = T3Decode(num_Ir,num_Rx);
```

```
53                                  default: decoder2 = OPNONE;
54                              endcase
55                      end
56
57                      Decoder = {decoder2,decoder1};
58          endfunction
59
60
61      function [5:0] T1Decode;
62
63          if(num_ir1 == OPMV) //mv
64          begin
65              case(num_rx1)
66                  2'b00: Decode = OPR0;
67                  2'b01: Decode = OPR1;
68                  2'b10: Decode = OPR2;
69                  2'b11: Decode = OPR3;
70              endcase
71
72          end
73          else if(num_ir1 == OPMVI) //mvi
74          begin
75              case(num_rx1)
76                  2'b00: Decode = OPR0;
77                  2'b01: Decode = OPR1;
78                  2'b10: Decode = OPR2;
79                  2'b11: Decode = OPR3;
80              endcase
81          end
82
83          else if(num_ir1 == OPADD) // add
84              Decode = 6'b000010;
85          else if(num_ir1 == OPSUB) //sub
86              Decode = 6'b000010;
87
88          T1Decode = Decode;
89
90      endfunction
```

表1のように、サンプルプログラムでテストを行った。object code が 10H 05H 04H 21H 30H なので Din に 001_0000 000_0101 000_0100 010_0001 011_0000 を入力した。このときの結果は図10のようになった。まず、サンプルプログラムでは mvi R0 #05 を行っているので、R0 に即値 5 が格納されることが予想される。実際に、R0 の値 R0out をみてやると、R0out の値が初めて変化したときの値は確かに 0101(2)=5 となっている。

次に、サンプルプログラムでは mv R1 R0 を行っているので、R1 に R0 の値である 5 が格納されることが予想される。実際に、R1 の値 R1out をみてやると、R1out の値が始めて変化したときの値は確かに 0101(2)=R0 の値になっていることがわかる。

次に、サンプルプログラムでは add R0 R1 を行っている。R0 の値 5 と R1 の値 5 が足しあわされ 10 が R0 に格納されることが予想される。実際に、次に変化したときの R0 の値は 1010(2)=10 となっていて R0 と R1 の足し合わせの結果と一致していることが分かる。
最後にサンプルプログラムでは sub R0 R0 を行っているので R0 の値から R0 自身で引いた結果 0 が R0 に格納されることが予想される。実際に、次に変化した R0out を読み取ってやると 0000 であり、R0 から R0 を引いた結果と一致していることがわかる。
さらに、タイムステップの状態を表すパラメータに注目すると、mv 命令,mvi 命令は 2 ステップ、add 命令,sub 命令は 4 ステップである。実際にタイムステップを表す state に注目すると、mv, mvi のときは state は 00 01 00 01 となっている。また、add, sub のとき state は　00 01 10 11 00 01 10 11 となっており、state の遷移数が各命令のタイムステップ数に一意している。また、state が 01 から 00 になるとき、11 から 00 になるとき命令が終了し Done が 1 に変化して、命令を終了できていることも確認できる。以上より各命令に対する結果の妥当性とタイムステップと Done の適切な動きをしていることから、制御ユニットを作成できたと言える。

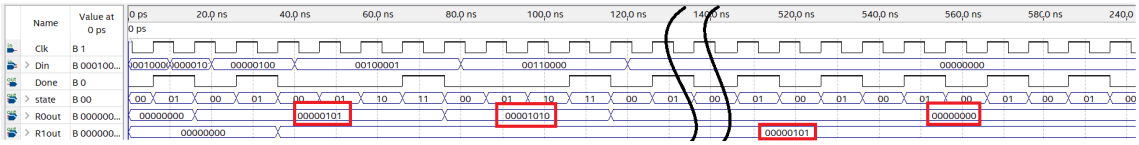

図 10　実習 3 における結果

5　演習

5.1　演習 1

外部メモリを用いることで人間が Din 入力へ情報を与える作業がなくなる。しかしこの回路構成にもまだ残されている課題があり、その課題と解決策を考えた。課題は次実行すべき命令が指定できていないところで、プログラミングカウンタを用いて解決ができるのはと考えた。
実際実習 3 で命令を実行したが、命令を実行するに当たって、命令コードの内容とタイミングが重要であった。この命令コードを毎回毎回手動で変更しなければならなかったが、外部ファイルを用いると一度登録した実行命令を呼び出せるようになる。しかしこれだけでは、どのタイミングでどの命令コードを呼び出すかが分からないため、次々に命令を実行できない課題がある。そのために、次に実行すべき命令が格納されているアドレスを保持しているプログラミングカウンタを用いる必要があると考えられる。作成方法としては、IR のようにレジスタを一つ作成する。そして、実行する命令は外部メモリに連続して保存されているという仮定のもと考えたとき、一つの命令が実行がされたときプログラミングカウンタを 1 インクリメントする。この命令が実行されたかどうかは今回作成した DONE が true になったときと考えればよさそうである。また、命令 MVI のように 2 ワード命令のときは 1 ではなく 2 インクリメントする必要がある。このようにして PC を作成できると考えた。

5.2　演習 2

関数を呼び出すコール命令、及びリターン命令を追加する場合プロセッサ内部及び周辺部をどのように拡張する必要があるか検討した。図 12 のように拡張し、スタックポインタを実装する必要があると考えた
コール命令、リターン命令が呼び出されたとき、今まで実行していた命令を一度中断し、新たな命令の実行後元の命令を再度実行する必要がある。そのため、レジスタのの値やプログラミングカウンタの値を退避させる必要必要が生じる。この退避させる領域をスタック領域と読んだとき、スタック領域の先頭番地を示すレジスタを用意しておけば、レジスタの情報を保持しながら新たな実行を命令したあと、再度もとの命令に戻ることができる。以上演習 1 におけるプログラミングカウンタ (PC) とスタックポインタ (SP) をプロセッサ内部に実装し、外部メモリとつなげたときのブロック図は図 11 のようになると考えられる。まず、IR の入力は外部メモリの命令コードの値が入る。そして、プログラミングカウンタには命令コードのアドレスと結び付けれられる。一方でスタックポインタはスタック領域のアドレスと結び付けられる。スタック領域では後だし先だしに基づき下からスタックを積んでいき、実行を上から行っていく。赤の矢印が下から上に向いているのはそのことを指している。
さらに、図 11 について考察したとき、プロセッサとアドレスのつなぐ線をまとめることができると考えた。今回は外部メモリのアドレス部と値部に分けて考え、アドレスバスとデータバスとしてつなげた。(図 12) 今回は最低限の線でプロセッサと外部メモリをつなげたが、今後拡張するとき線が複雑にならないようすることができると考えられる。

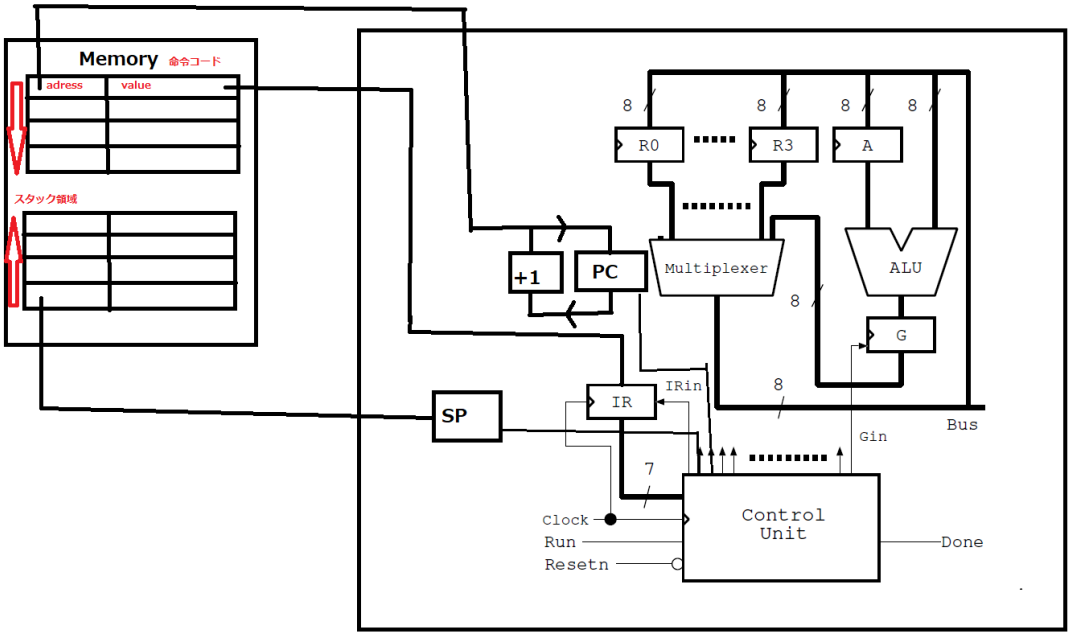

図 11　演習 2 におけるブロック図

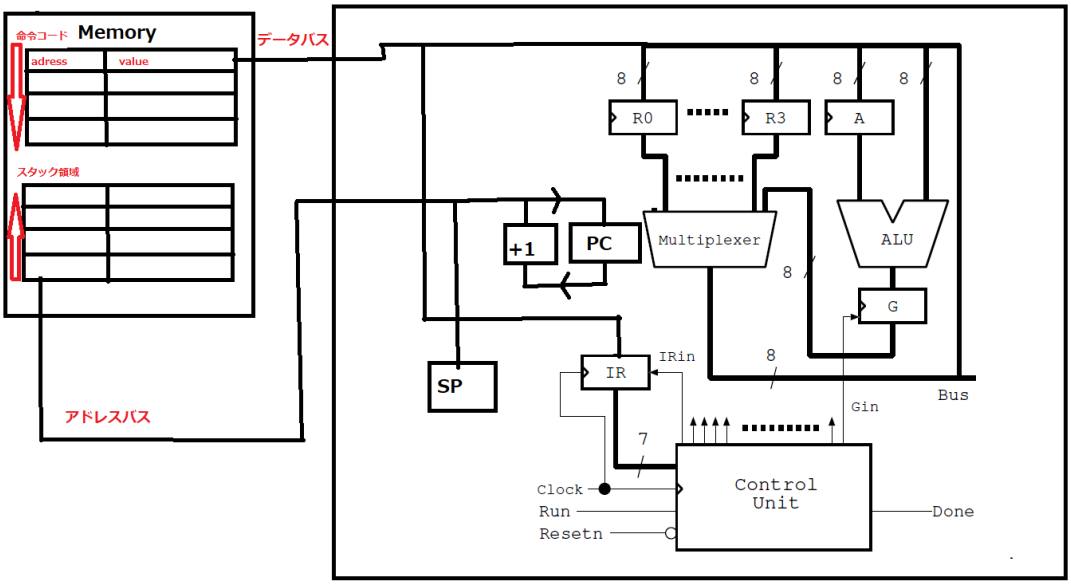

図 12　演習 2 におけるブロック図

6　考察

実習 1 では MUX(図 2) を作成したが、これはレポート 1 の実習 1 で 5to1 マルチプレクサ (図 13) の応用で作成することができた。図 13 と図 2 を比べたとき、I0 から I3 までは R0 から R3 までに対応していて、I4 が Din,S が制御信号、O が Bus に対応していたことが分かった。いきなり、MUX を作るとなると難しそうだが、単純なモデル 5to1 マルチプレクサで考えると簡単にできることが分かる。

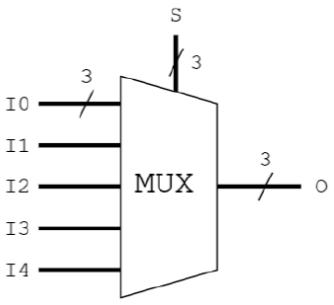

図 13　5to1 マルチプレクサ

実習 2 では実習 1 を基に ALU(図 3) を追加した。これはレポート 2 の実習 1 の加減算器 (図 14) の生成の応用で作成することができた。図 14 と図 3 を比べたとき、AddSub が ALU で RegA がレジスタ A で RegS がレジスタ G と対応していることが分かる。Clk やリセット、Mode による加減算の切り替えはそのまま流用することができた。実習 1 と同様にいきなり ALU を作るのは難しそうだが、AddSub だけで動くか試してみ

て、そのうえで必要な部分を流用することで効率よく ALU を作成できたと考えられる。

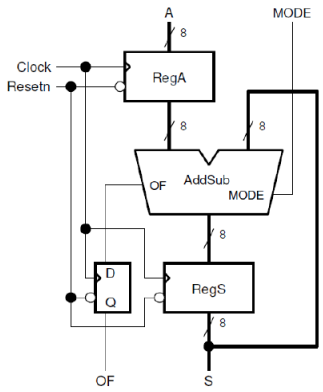

図 14　加減算器

実習３では各状態に応じた命令を解読し実行するプロセッサを作成したが、これはレポート１の実習３でしたパラメータを用いた状態遷移やこれまで行ってきた順序回路、組み合わせ回路、そしてこれは実習１や実習２で少しずつできるものを拡張して拡張してできた結果である。今回は学生実験ということで講義テキストに基づいて実験を行ったが、何事もいきなり大きいものを作ろうとはせず、細かな部分を先につくり、後から足し合わせることが大事と気付かされた。

7　ソースコード

ソースコード 4　zissyu1.v

```
1  module Zissyu1(R0,R1,R2,R3,Din,S,Clk,Res,R0out,R1out,R2out,R3out,Bus);
2          input R0,R1,R2,R3;
3          input [7:0]Din;
4          input [4:0]S;
5          input Clk,Res;
6          output [7:0]R0out,R1out,R2out,R3out,Bus;
7
8          Reg Reg0(Bus,Clk,R0,Res,R0out);
9          Reg Reg1(Bus,Clk,R1,Res,R1out);
10         Reg Reg2(Bus,Clk,R2,Res,R2out);
11         Reg Reg3(Bus,Clk,R3,Res,R3out);
12         MUX Mux(Clk,R0out,R1out,R2out,R3out,Din,S,Bus);
13
14 endmodule
15
16 module Reg(D,Clk,En,Res,Q);
17         input [7:0]D;
18         input Clk,En,Res;
19         output [7:0]Q;
20         reg [7:0]Q1;
```

```
21
22          always @(posedge Clk)begin
23                  if(Res == 0)
24                          Q1 <= 0;
25                  else
26                          if(En == 1)
27                                  Q1 <= D;
28          end
29          assign Q = Q1;
30  endmodule
31
32  module MUX(Clk,R0,R1,R2,R3,Din,S,Bus);
33          input Clk;
34          input [7:0]R0,R1,R2,R3,Din;
35          input [4:0]S;
36          output [7:0]Bus;
37          reg [7:0]Bus1;
38
39          always @(posedge Clk)begin
40                  if(S[0] == 1)
41                          Bus1 <= R0;
42                  else if(S[1] == 1)
43                          Bus1 <= R1;
44                  else if(S[2] == 1)
45                          Bus1 <= R2;
46                  else if(S[3] == 1)
47                          Bus1 <= R3;
48                  else if(S[4] == 1)
49                          Bus1 <= Din;
50          end
51          assign Bus = Bus1;
52  endmodule
```

ソースコード 5　zissyu2.v

```
1  module Zissyu2(R0,R1,R2,R3,EnA,EnG,Mode,Din,S,Clk,Res,R0out,R1out,R2out,R3out,Bus,Sin,Aout,
       Gout,ALUout);
2          input R0,R1,R2,R3,EnA,EnG,Mode;
3          input [7:0]Din;
4          input [5:0]S; //signal of Enable S[] = {R0,R1,R2,R3,Din,G}
5          input Clk,Res;
6          output [7:0]R0out,R1out,R2out,R3out,Bus,Sin,Aout,Gout,ALUout;
7
8          Reg Reg0(Bus,Clk,R0,Res,R0out);
9          Reg Reg1(Bus,Clk,R1,Res,R1out);
10         Reg Reg2(Bus,Clk,R2,Res,R2out);
11         Reg Reg3(Bus,Clk,R3,Res,R3out);
12         Reg RegA(Bus,Clk,EnA,Res,Aout);
13         ALU Alu(Mode,Aout,Bus,ALUout);
14         Reg RegG(ALUout,Clk,EnG,Res,Gout);
15         MUX Mux(Clk,S,R0out,R1out,R2out,R3out,Din,Gout,Bus);
16
17 endmodule
18
19 module Reg(D,Clk,En,Res,Q);
20         input [7:0]D;
21         input Clk,En,Res;
22         output [7:0]Q;
23         reg [7:0]Q1;
24
25         always @(posedge Clk)begin
26                 if(Res == 0)
27                         Q1 <= 0;
28                 else
29                         if(En == 1)
30                                 Q1 <= D;
31         end
32         assign Q = Q1;
33 endmodule
34
35 module MUX(Clk,S,R0,R1,R2,R3,Din,Gout,Bus);
36         input Clk;
37         input [5:0]S;
38         input [7:0]R0,R1,R2,R3,Din,Gout;
39         output [7:0]Bus;
40         reg [7:0]Bus1;
41
42         always @(posedge Clk)begin
43                 if(S[0] == 1)
44                         Bus1 <= R0;
45                 else if(S[1] == 1)
46                         Bus1 <= R1;
```

```
47                  else if(S[2] == 1)
48                          Bus1 <= R2;
49                  else if(S[3] == 1)
50                          Bus1 <= R3;
51                  else if(S[4] == 1)
52                          Bus1 <= Din;
53                  else if(S[5] == 1)
54                          Bus1 <= Gout;
55          end
56          assign Bus = Bus1;
57  endmodule
58
59  module Zenkasan(A,B,Cin,S,Cout);
60          input A, B, Cin;
61          output S,Cout;
62
63          assign S = (A^B)^Cin;
64          assign Cout = (A&B)|(B&Cin)|(Cin&A);
65  endmodule
66
67
68  module ALU(Mode,A,B,Y);
69          input Mode;
70          input [7:0]A,B;
71          output [7:0]Y;
72          wire [7:0]A2,C;
73
74          function [7:0]returnA;
75                  input mode;
76                  input [7:0]Ain;
77
78                  if(mode == 1)
79                          returnA = (Ain^8'b11111111)+1'b1;
80                  else
81                          returnA = Ain;
82          endfunction
83
84          assign A2 = returnA(Mode,A);
85
86          Zenkasan zenkasan0(A2[0],B[0],0,Y[0],C[0]);
87          Zenkasan zenkasan1(A2[1],B[1],C[0],Y[1],C[1]);
88          Zenkasan zenkasan2(A2[2],B[2],C[1],Y[2],C[2]);
89          Zenkasan zenkasan3(A2[3],B[3],C[2],Y[3],C[3]);
90          Zenkasan zenkasan4(A2[4],B[4],C[3],Y[4],C[4]);
91          Zenkasan zenkasan5(A2[5],B[5],C[4],Y[5],C[5]);
92          Zenkasan zenkasan6(A2[6],B[6],C[5],Y[6],C[6]);
93          Zenkasan zenkasan7(A2[7],B[7],C[6],Y[7],C[7]);
94  endmodule
```

ソースコード 6　zissyu3.v

```
1  module Zissyu3(Din,Clk,Res,Done,Mode,R0,R1,R2,R3,EnA,EnG,EnI,S,R0out,R1out,R2out,R3out,Bus,
       Aout,Gout,ALUout,IRout,state,opland1,opland2,opcode,Rxin,Rxout);
2          input [7:0]Din;
3          input Clk,Res;
4          output Done,Mode;// pulu or minasu
5          output R0,R1,R2,R3,EnA,EnG,EnI; // signal of in
6          output [5:0]S; //signal of out {R0,R1,R2,R3,Din,G}
7          output [7:0]R0out,R1out,R2out,R3out,Bus,Aout,Gout,ALUout;
8          output [6:0]IRout;
9          output [1:0]state,opland1,opland2;
10         output [2:0]opcode;
11         output [6:0]Rxin; //R0,R1,R2,R3,EnA,EnG,EnI
12         output [5:0] Rxout;
13
14         Reg Reg0(Bus,Clk,R0,Res,R0out);
15         Reg Reg1(Bus,Clk,R1,Res,R1out);
16         Reg Reg2(Bus,Clk,R2,Res,R2out);
17         Reg Reg3(Bus,Clk,R3,Res,R3out);
18         Reg RegA(Bus,Clk,EnA,Res,Aout);
19         ALU Alu(Mode,Aout,Bus,ALUout);
20         Reg RegG(ALUout,Clk,EnG,Res,Gout);
21         MUX Mux(Clk,S,R0out,R1out,R2out,R3out,Din,Gout,Bus);
22         IR Ir(Din[6:0],Clk,EnI,Res,IRout);
23         ContorolUnity(Clk,Res,IRout,R0,R1,R2,R3,EnA,EnG,EnI,S,Done,Mode,state,opland1,opland2,
               opcode,Rxin,Rxout);
24
25 endmodule
26
27 module Reg(D,Clk,En,Res,Q);
28         input [7:0]D;
29         input Clk,En,Res;
30         output [7:0]Q;
31         reg [7:0]Q1;
32
33         always @(posedge Clk)begin
34                 if(Res == 0)
35                         Q1 <= 0;
36                 else
37                         if(En == 1)
38                                 Q1 <= D;
39         end
40         assign Q = Q1;
41 endmodule
42
43 module IR(D,Clk,En,Res,Q);
44         input [6:0]D;
45         input Clk,En,Res;
```

```
46          output [6:0]Q;
47          reg [6:0]Q1;
48
49          always @(posedge Clk)begin
50                  if(Res == 0)
51                          Q1 <= 0;
52                  else
53                          if(En == 1)
54                                  Q1 <= D;
55          end
56          assign Q = Q1;
57  endmodule
58
59  module MUX(Clk,S,R0,R1,R2,R3,Din,Gout,Bus);
60          input Clk;
61          input [5:0]S;//signal of out {R0,R1,R2,R3,Din,G}
62          input [7:0]R0,R1,R2,R3,Din,Gout;
63          output [7:0]Bus;
64
65          function [7:0]returnBus;
66                  input [5:0]S;
67                  input [7:0]R0,R1,R2,R3,Din,Gout;
68
69                  if(S[0] == 1)
70                          returnBus = Gout;
71                  else if(S[1] == 1)
72                          returnBus = Din;
73                  else if(S[2] == 1)
74                          returnBus = R3;
75                  else if(S[3] == 1)
76                          returnBus = R2;
77                  else if(S[4] == 1)
78                          returnBus = R1;
79                  else if(S[5] == 1)
80                          returnBus = R0;
81          endfunction
82
83          assign Bus = returnBus(S,R0,R1,R2,R3,Din,Gout);
84  endmodule
85
86
87
88
89  module Zenkasan(A,B,Cin,S,Cout);
90          input A, B, Cin;
91          output S,Cout;
92
93          assign S = (A^B)^Cin;
94          assign Cout = (A&B)|(B&Cin)|(Cin&A);
```

```
95  endmodule
96
97
98  module ALU(Mode,A,B,Y);
99          input Mode;
100         input [7:0]A,B;
101         output [7:0]Y;
102         wire [7:0]A2,C;
103
104         function [7:0]returnA;
105                 input mode;
106                 input [7:0]Ain;
107
108                 if(mode == 1)
109                         returnA = (Ain^8'b11111111)+1'b1;
110                 else
111                         returnA = Ain;
112         endfunction
113
114         assign A2 = returnA(Mode,A);
115
116         Zenkasan zenkasan0(A2[0],B[0],0,Y[0],C[0]);
117         Zenkasan zenkasan1(A2[1],B[1],C[0],Y[1],C[1]);
118         Zenkasan zenkasan2(A2[2],B[2],C[1],Y[2],C[2]);
119         Zenkasan zenkasan3(A2[3],B[3],C[2],Y[3],C[3]);
120         Zenkasan zenkasan4(A2[4],B[4],C[3],Y[4],C[4]);
121         Zenkasan zenkasan5(A2[5],B[5],C[4],Y[5],C[5]);
122         Zenkasan zenkasan6(A2[6],B[6],C[5],Y[6],C[6]);
123         Zenkasan zenkasan7(A2[7],B[7],C[6],Y[7],C[7]);
124 endmodule
125
126
127
128
129
130 module ContorolUnity(Clk,Rst,Ir,R0,R1,R2,R3,EnA,EnG,EnI,S,Done,Mode,state,num_Ry,num_Rx,
        num_Ir,Rxin,Rxout);
131         input [6:0]Ir; //instruct register
132         input Clk,Rst;
133
134         output R0,R1,R2,R3,EnA,EnG,EnI; //enable of register
135         output [5:0]S; //R0,R1,R2,R3,Din,G, //signal of multiplexer
136         output Done;
137         output Mode;
138         output [1:0]state;
139         parameter T0 = 2'b00,
140                                 T1 = 2'b01,
141                                 T2 = 2'b10,
142                                 T3 = 2'b11;
```

```
143
144         parameter OPNONE = 6'b000000,
145                                 OPG = 6'b000001,
146                                 OPD = 6'b000010,
147                                 OPR3 = 6'b000100,
148                                 OPR2 = 6'b001000,
149                                 OPR1 = 6'b010000,
150                                 OPR0= 6'b100000;
151
152         parameter OPMV = 3'b000,
153                                 OPMVI = 3'b001,
154                                 OPADD = 3'b010,
155                                 OPSUB = 3'b011;
156
157         reg [1:0] cur_st; //curent state
158         reg [1:0] cur_code;
159         reg done;
160         output [1:0]num_Ry,num_Rx; //opeland
161         output[2:0]num_Ir; //opecode
162         output [6:0]Rxin; //R0,R1,R2,R3,EnA,EnG,EnI
163         output [5:0] Rxout;
164
165         always @(posedge Clk)
166                 begin
167                         if(!Rst)
168                                 begin
169                                         cur_st <= T0;
170                                         done = 0;
171                                 end
172                         else if(Done)
173                                 begin
174                                         cur_st <= T0;
175                                         done = 0;
176                                 end
177                         else
178                                 begin
179                                         case(cur_st)
180                                                 T0: cur_st <= T1;
181                                                 T1: cur_st <= T2;
182                                                 T2: cur_st <= T3;
183                                                 T3: cur_st <= T3;
184                                                 default: cur_st <= T0;
185                                         endcase
186                                         done = 1;
187                                 end
188                 end
189         //renew instruct register when T0
190   // Rxin[0] = EnIr(cur_st);
191
```

```
192         //opeland fetch and opecode fetch
193         assign num_Ry = Ir[1:0];
194         assign num_Rx = Ir[3:2];
195         assign num_Ir = Ir[6:4];
196 
197 
198         //Decode of instruct
199         assign Rxin = Decoder(cur_st,num_Ir,num_Rx);
200 // Rxin[6:1] = Decoder(cur_st,num_Ir,num_Rx);
201 // (cur_st == T1) ? T1Decode(num_Ir,num_Rx):(
202 // (cur_st == T2) ? T2Decode(num_Ir,num_Rx):(
203 // (cur_st == T3) ? T3Decode(num_Ir,num_Rx)));
204 
205         assign Mode = ModeChanger(num_Ir);
206 
207 
208         //Generate signal
209         assign Rxout = Generater(cur_st,num_Ir,num_Rx,num_Ry);
210 // (cur_st == T1) ? T1Generater(num_Ir,num_Rx,num_Ry):(
211 // (cur_st == T2) ? T2Generater(num_Ir,num_Ry):(
212 // (cur_st == T3) ? T3Generater(num_Ir,num_Ry)));
213 
214 
215         assign Done = DoneChanger(cur_st,num_Ir,done);
216 
217         assign R0 = Rxin[6];
218         assign R1 = Rxin[5];
219         assign R2 = Rxin[4];
220         assign R3 = Rxin[3];
221         assign EnA = Rxin[2];
222         assign EnG = Rxin[1];
223         assign EnI = Rxin[0];
224         assign S = Rxout;
225         assign state = cur_st;
226 
227 
228         /*
229         Decode of signal function_____________________________________________________
230         */
231 
232         function [6:0]Decoder;
233         input [1:0]cur_st;
234         input [2:0]num_Ir;
235         input [1:0]num_Rx;
236         reg decoder1;
237         reg [5:0]decoder2;
238 
239                 if(cur_st == T0)
240                         decoder1 = 1'b1;
```

```
241                 else
242                         decoder1 = 1'b0;
243
244                 begin
245                         case(cur_st)
246                                 T1: decoder2 = T1Decode(num_Ir,num_Rx);
247                                 T2: decoder2 = T2Decode(num_Ir,num_Rx);
248                                 T3: decoder2 = T3Decode(num_Ir,num_Rx);
249                                 default: decoder2 = OPNONE;
250                         endcase
251                 end
252
253                 Decoder = {decoder2,decoder1};
254         endfunction
255
256         function [5:0] T1Decode;
257                 input [2:0]num_ir1;
258                 input [1:0]num_rx1;
259                 reg [5:0]Decode;
260
261                 if(num_ir1 == OPMV) //mv
262                 begin
263                         case(num_rx1)
264                                 2'b00: Decode = OPR0;
265                                 2'b01: Decode = OPR1;
266                                 2'b10: Decode = OPR2;
267                                 2'b11: Decode = OPR3;
268                         endcase
269 //  Decode =
270 //  (num_rx1 == 2'b00) ? 6'b000100:(
271 //  (num_rx1 == 2'b01) ? 6'b001000:(
272 //  (num_rx1 == 2'b10) ? 6'b010000:(
273 //  (num_rx1 == 2'b11) ? 6'b100000)));
274                 end
275                 else if(num_ir1 == OPMVI) //mvi
276                 begin
277                         case(num_rx1)
278                                 2'b00: Decode = OPR0;
279                                 2'b01: Decode = OPR1;
280                                 2'b10: Decode = OPR2;
281                                 2'b11: Decode = OPR3;
282                         endcase
283                 end
284 //  Decode =
285 //  (num_rx1 == 2'b00) ? 6'b000100:(
286 //  (num_rx1 == 2'b01) ? 6'b001000:(
287 //  (num_rx1 == 2'b10) ? 6'b010000:(
288 //  (num_rx1 == 2'b11) ? 6'b100000)));
289                 else if(num_ir1 == OPADD) // add
```

```
290                         Decode = 6'b000010;
291                 else if(num_ir1 == OPSUB) //sub
292                         Decode = 6'b000010;
293
294                 T1Decode = Decode;
295
296         endfunction
297
298
299         function [5:0] T2Decode;
300                 input [2:0]num_ir2;
301                 input [1:0]num_rx2;
302                 reg [5:0] Decode;
303
304                 if(num_ir2 == OPADD) // add
305                         Decode = OPG;
306                 else if(num_ir2 == OPSUB) //sub
307                         Decode = OPG;
308
309                 T2Decode = Decode;
310
311         endfunction
312
313         function [5:0] T3Decode;
314                 input [2:0]num_ir3;
315                 input [1:0]num_rx3;
316                 reg [5:0]Decode;
317
318                 if(num_ir3 == OPADD) // add
319                 begin
320                         case(num_rx3)
321                                 2'b00: Decode = OPR0;
322                                 2'b01: Decode = OPR1;
323                                 2'b10: Decode = OPR2;
324                                 2'b11: Decode = OPR3;
325                         endcase
326                 end
327 //  Decode =
328 //  (num_rx3 == 2'b00) ? 6'b000100:(
329 //  (num_rx3 == 2'b01) ? 6'b001000:(
330 //  (num_rx3 == 2'b10) ? 6'b010000:(
331 //  (num_rx3 == 2'b11) ? 6'b100000)));
332                 else if(num_ir3 == OPSUB) //sub
333                 begin
334                         case(num_rx3)
335                                 2'b00: Decode = OPR0;
336                                 2'b01: Decode = OPR1;
337                                 2'b10: Decode = OPR2;
338                                 2'b11: Decode = OPR3;
```

```
339                     endcase
340             end
341 //  Decode =
342 //  (num_rx3 == 2'b00) ? 6'b000100:(
343 //  (num_rx3 == 2'b01) ? 6'b001000:(
344 //  (num_rx3 == 2'b10) ? 6'b010000:(
345 //  (num_rx3 == 2'b11) ? 6'b100000)));
346
347             T3Decode = Decode;
348     endfunction
349
350
351
352     /*
353     Genetater of signal function_____________________________________________________
354     */
355     function[5:0] Generater;
356             input [1:0]cur_st;
357             input [2:0]num_Ir;
358             input [1:0]num_Rx;
359             input [1:0]num_Ry;
360
361             begin
362                     case(cur_st)
363                             T1: Generater = T1Generater(num_Ir,num_Rx,num_Ry);
364                             T2: Generater = T2Generater(num_Ir,num_Ry);
365                             T3: Generater = T3Generater(num_Ir,num_Ry);
366                             default: Generater = OPNONE;
367                     endcase
368             end
369
370     endfunction
371
372     function [5:0] T1Generater;
373             input [2:0]num_ir;
374             input [1:0]num_rx;
375             input [1:0]num_ry;
376             reg [5:0]Generater;
377
378             if(num_ir == OPMV) //mv
379             begin
380                     case(num_ry)
381                             2'b00: Generater = OPR0;
382                             2'b01: Generater = OPR1;
383                             2'b10: Generater = OPR2;
384                             2'b11: Generater = OPR3;
385                     endcase
386             end
387 //  Generater =
```

```
388 //  (num_ry == 2'b00) ? 6'b000100:(
389 //  (num_ry == 2'b01) ? 6'b001000:(
390 //  (num_ry == 2'b10) ? 6'b010000:(
391 //  (num_ry == 2'b11) ? 6'b100000)));
392                 else if(num_ir == OPMVI) //mvi
393                         Generater = OPD;
394                 else if(num_ir == OPADD) // add
395                 begin
396                         case(num_rx)
397                                 2'b00: Generater = OPR0;
398                                 2'b01: Generater = OPR1;
399                                 2'b10: Generater = OPR2;
400                                 2'b11: Generater = OPR3;
401                         endcase
402                 end
403 //  Generater =
404 //  (num_rx == 2'b00) ? 6'b000100:(
405 //  (num_rx == 2'b01) ? 6'b001000:(
406 //  (num_rx == 2'b10) ? 6'b010000:(
407 //  (num_rx == 2'b11) ? 6'b100000)));
408                 else if(num_ir == OPSUB) //sub
409                 begin
410                         case(num_rx)
411                                 2'b00: Generater = OPR0;
412                                 2'b01: Generater = OPR1;
413                                 2'b10: Generater = OPR2;
414                                 2'b11: Generater = OPR3;
415                         endcase
416                 end
417 //  Generater =
418 //  (num_rx == 2'b00) ? OPR0:(
419 //  (num_rx == 2'b01) ? 6'b001000:(
420 //  (num_rx == 2'b10) ? 6'b010000:(
421 //  (num_rx == 2'b11) ? 6'b100000)));
422                 T1Generater = Generater;
423         endfunction
424
425
426         function [5:0] T2Generater;
427                 input [2:0]num_ir;
428                 input [1:0]num_ry;
429                 reg [5:0]Generater;
430
431                 if(num_ir == OPADD) // add
432                 begin
433                         case(num_ry)
434                                 2'b00: Generater = OPR0;
435                                 2'b01: Generater = OPR1;
436                                 2'b10: Generater = OPR2;
```

```
437                             2'b11: Generater = OPR3;
438                     endcase
439             end
440 //  Generater =
441 //  (num_ry == 2'b00) ? 6'b000100:(
442 //  (num_ry == 2'b01) ? 6'b001000:(
443 //  (num_ry == 2'b10) ? 6'b010000:(
444 //  (num_ry == 2'b11) ? 6'b100000)));
445             else if(num_ir == OPSUB) //sub
446             begin
447                     case(num_ry)
448                             2'b00: Generater = OPR0;
449                             2'b01: Generater = OPR1;
450                             2'b10: Generater = OPR2;
451                             2'b11: Generater = OPR3;
452                     endcase
453             end
454 //  Generater =
455 //  (num_ry == 2'b00) ? 6'b000100:(
456 //  (num_ry == 2'b01) ? 6'b001000:(
457 //  (num_ry == 2'b10) ? 6'b010000:(
458 //  (num_ry == 2'b11) ? 6'b100000)));
459
460             T2Generater = Generater;
461
462         endfunction
463
464
465         function [5:0] T3Generater;
466             input [2:0]num_ir;
467             input [1:0]num_ry;
468             reg [5:0]Generater;
469
470             if(num_ir == OPADD) // add
471                     Generater = OPG;
472             else if(num_ir == OPSUB) //sub
473                     Generater = OPG;
474
475             T3Generater = Generater;
476
477         endfunction
478
479
480         function ModeChanger;
481             input [2:0]num_ir;
482             reg Changer;
483
484             if(num_ir == OPADD) //add
485                     Changer = 1'b0;
```

```
486                 else if(num_ir == OPSUB) //sub
487                         Changer = 1'b1;
488                 else
489                         Changer = 1'b0;
490
491                 ModeChanger = Changer;
492         endfunction
493
494
495         function DoneChanger;
496                 input [1:0]cur_st;
497                 input [2:0]num_ir;
498                 input done;
499                 reg Changer;
500
501                 if(done == 0)
502                         Changer = 0;
503                 if(cur_st == T1)
504                         if(num_ir == OPMV)
505                                 Changer = 1;
506                         else if(num_ir == OPMVI)
507                                 Changer = 1;
508                         else
509                                 Changer = 0;
510                 else if(cur_st == T3)
511                         if(num_ir == OPADD)
512                                 Changer = 1;
513                         else if(num_ir == OPSUB)
514                                 Changer = 1;
515                         else
516                                 Changer = 0;
517                 else
518                         Changer = 0;
519
520                 DoneChanger = Changer;
521         endfunction
522
523         function EnIr;
524                 input [1:0]cur_st;
525                 reg enir;
526
527                 if(cur_st == T0)
528                         enir = 1'b1;
529                 else
530                         enir = 1'b0;
531
532                 EnIr = enir;
533         endfunction
534
```

```
535         // renew state of timestep
536
537
538
539 endmodule
```