提出日：2023 年 5 月 12 日

2023 年度　3 回生前期学生実験 HW

機能設計仕様書

学籍番号：1029337123

入学年度：2021

氏名：加藤利梓

1. 全体の構成

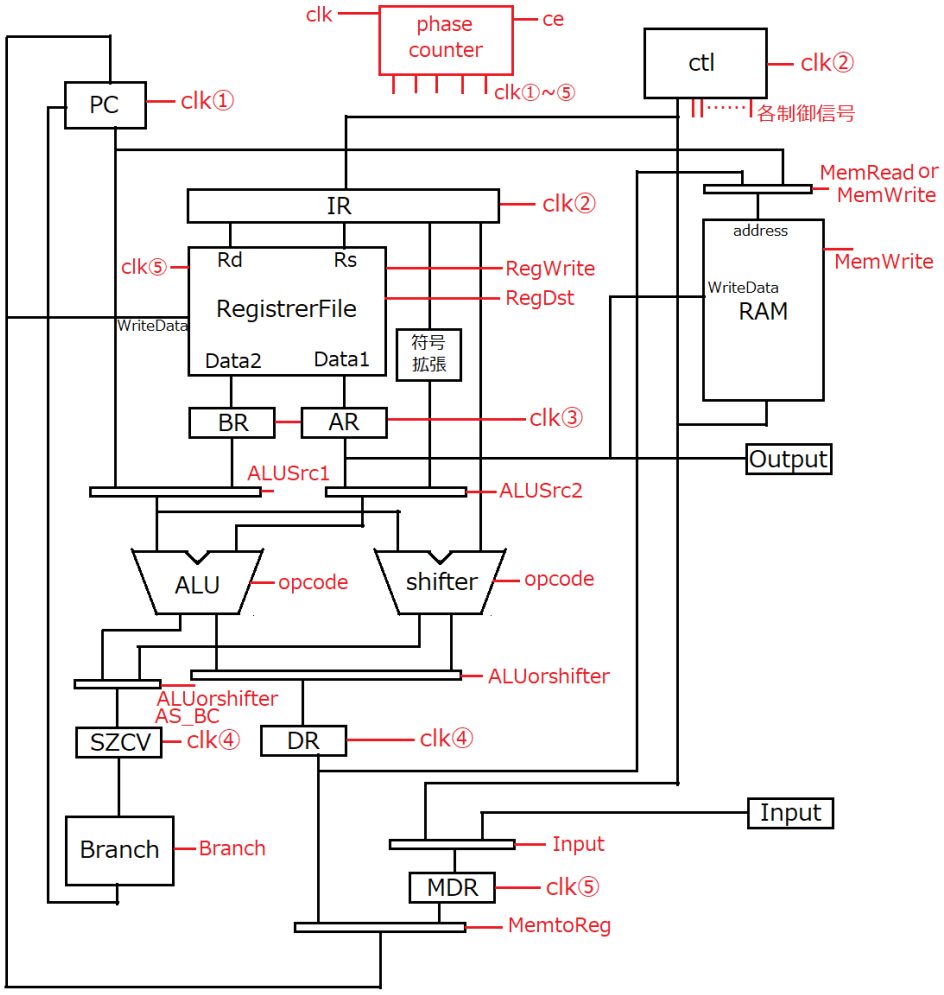

図１　設計するアーキテクチャのブロック図

図１に設計するプロセッサのブロック図を示す。

PC,phasecounter,ctl,IR,RegisterFile,RAM,BR,AR,ALU,shifter,SZCV,DR,Branch,MDR があり、このうち担当したのは ALU,shifter,phasecounter である。

2. ALU

2.1　外部仕様

次の外部仕様を満たす 16 ビットの ALU を論理設計した。

- 入力信号
  - A[15..0]: 16 ビットの入力値 A
  - B[15..0]: 16 ビットの入力値 B
  - ALUctl[3..0]: 4 ビットの操作コード
- 出力信号
  - Out[15..0]: 16 ビットの演算結果
  - Outcond[3:0]: 4 ビットの条件コード
    - S: 演算結果の符号ビットを表すフラグ 負ならば 1
    - Z: 演算結果がゼロであるかを表すフラグ ゼロならば 1
    - C: 演算結果の桁上げを表すフラグ 桁上げがあれば 1
      但し論理演算および移動演算の場合は結果に関わらず 0 となる
    - V: 演算結果が符号付き 16 ビットで表せる範囲を超えた場合は 1
- 機能仕様
  - 操作コードに応じて、表 1 に示す各種の算術論理演算を行う。
  - 演算結果に応じて，条件コードの各フラグを出力する.

操作コード	機能	補足説明
0000	Out = A + B	算術可算（ADD）
0001	Out = A - B	算術減算（SUB）
0010	Out = A & B	論理積（AND）
0011	Out = A | B	論理和（OR）
0100	Out = A ^ B	排他的論理和（XOR）
0101	Z = 1 if !(B - A)	比較演算（CMP）
0110	Out = A	移動演算（MOV）

表 1　ALU の機能仕様

表 1 に今回作成した ALU の機能仕様を示す。

2.2　内部仕様

```
module ALU(
      input [3:0] ALUctl,
      input [15:0] A,B,
      output [15:0] Out,
      output [3:0] Outcond
      );
      reg [16:0] C;
      reg [3:0] cond;
      always @(ALUctl,A,B) begin
         case (ALUctl)
            0: begin
                  C <= A + B;
               if(A[15]==B[15] && A[15]!=C[15])
                  cond[0] <= 1'b1;
               else
                  cond[0] <= 1'b0;
               end
            1: begin
                  C <= A - B;
               if(A[15]!=B[15] && A[15]!=C[15])
                  cond[0] <= 1'b1;
               else
                  cond[0] <= 1'b0;
               end
            2: begin
                  C[15:0] <= A & B;
                  cond[0] <= 1'b0;
               end
            3: begin
                  C[15:0] <= A | B;
                  cond[0] <= 1'b0;
               end
            4: begin
                  C[15:0] <= A ^ B;
                  cond[0] <= 1'b0;
               end
            5: begin
                  C <= A - B;
               if(A[15]!=B[15] && A[15]!=C[15])
                  cond[0] <= 1'b1;
               else
                  cond[0] <= 1'b0;
               end
            6: begin
                  C[15:0] <= B;
                  C[16] <= 1'b0;
               end
         endcase

         if(C==16'b0)
            cond[2] <= 1'b1;
         else
            cond[2] <= 1'b0;
         cond[3] <= C[15];
         cond[1] <= C[16];
      end
      assign Outcond = cond;
      assign Out = C[15:0];
endmodule
```

図 2　ALU の HDL コード

図 2 に実装した ALU の HDL コードを示す。Always 文を使い、入力信号が切り替わった時に演算をし直す仕様になっている。Always 文の中では、case 文で ALUctl の値に応じた演算を行っている。

条件コードの演算は以下のように行った。(表 2)

条件コード	演算方法
cond[3](S)	符号ビットの値
cond[2](Z)	演算結果が 0 なら 1、それ以外は 0
cond[1](C)	演算結果を 17bit として演算し、最上位ビットの値
cond[0](V)	ADD：二つの被演算数の符号が同じで、かつ演算結果の符号がそれらと異なるとき、1 SUB、CMP：二つの被演算数の符号が異なり、かつ被減算数の符号と演算結果が異なるとき、1 それ以外：常に 0

表 2　条件コードの演算方法

3. Shifter

3.1　外部仕様

次の外部仕様を満たす 16 ビットの shifter を論理設計した。

- 入力信号
  - A[15..0]: 16 ビットの入力値 A
  - d[3..0]:4 ビットの入力値 d
  - opcode[3..0]: 4 ビットの操作コード
- 出力信号
  - Out[15..0]: 16 ビットの演算結果
  - Outcond[3:0]: 4 ビットの条件コード
    - S: 演算結果の符号ビットを表すフラグ 負ならば 1
    - Z: 演算結果がゼロであるかを表すフラグ ゼロならば 1
    - C: 最後にシフトアウトされたビットの値
      但しシフト桁数が 0 のときおよび SLR の場合は結果に関わらず 0 となる
    - V: 常に 0
- 機能仕様
  - 操作コードに応じて、表 1 に示す各種の算術論理演算を行う。
  - 演算結果に応じて，条件コードの各フラグを出力する.

操作コード	機能	補足説明
1000	SLL	左論理シフト、空いた部分に 0 を入れる
1001	SLR	左循環シフト、空いた部分にシフトアウトされたビット列を入れる
1010	SRL	右論理シフト、空いた部分に 0 を入れる
1011	SRA	右算術シフト、空いた部分に符号ビットを入れる

表 3　shifter の機能仕様

表 3 に今回作成した shifter の機能仕様を示す。

3.2　内部仕様

```
module shifter (
      input signed [15:0] A,
      input [3:0] opcode,
      input [3:0] d,
      output [15:0] Out,
      output [3:0] Outcond
      );
      reg [31:0] C;
      reg [15:0] D;
      reg [3:0] cond;
      always @(A,d,opcode) begin
         case (opcode)
            8: begin
                  D <= A << d;
                  C <= A << d;
               end
            9: begin
                  C <= A << d;
               case (d)
                  1: C[0] <= C[16];
                  2: C[1:0] <= C[18:16];
                  3: C[2:0] <= C[19:16];
                  4: C[3:0] <= C[20:16];
                  5: C[4:0] <= C[21:16];
                  6: C[5:0] <= C[22:16];
                  7: C[6:0] <= C[23:16];
                  8: C[7:0] <= C[24:16];
                  9: C[8:0] <= C[25:16];
                  10: C[9:0] <= C[26:16];
                  11: C[10:0] <= C[27:16];
                  12: C[11:0] <= C[28:16];
                  13: C[12:0] <= C[29:16];
                  14: C[13:0] <= C[30:16];
                  15: C[14:0] <= C[31:16];
               endcase
                  D <= C[15:0];
               end
            10: D <= A >> d;
            11: D <= A >>> d;
         endcase
         if (d==0 || opcode ==4'd9)
            cond[1] <= 1'b0;
         else if (opcode == 4'd8)
            cond[1] <= C[16];
         else begin
            case (d)
               1: cond[1] <= A[0];
               2: cond[1] <= A[1];
               3: cond[1] <= A[2];
               4: cond[1] <= A[3];
               5: cond[1] <= A[4];
               6: cond[1] <= A[5];
               7: cond[1] <= A[6];
               8: cond[1] <= A[7];
               9: cond[1] <= A[8];
               10: cond[1] <= A[9];
               11: cond[1] <= A[10];
               12: cond[1] <= A[11];
               13: cond[1] <= A[12];
               14: cond[1] <= A[13];
               15: cond[1] <= A[14];
            endcase
         end
         if(D==16'b0)
            cond[2] <= 1'b1;
         else
            cond[2] <= 1'b0;
         cond[0] <= 1'b0;
         cond[3] <= D[15];
      end
      assign Out = D;
      assign Outcond = cond;
endmodule
```

図 3　shifter の HDL コード

図 3 に実装した shifter の HDL コードを示す。Always 文を使い、入力信号が切り替わった時に演算をし直す仕様になっている。Always 文の中では、case 文で opcode の値に応じた演算を行っている。SLL、SRL、SRA の場合は HDL に備わっている演算子を使えば実装は簡単だったが、SLR の場合は d の値に応じてシフトアウトされた値を空いた部分に代入しなおしている。

条件コードの演算は以下のように行った。(表 4)

条件コード	演算方法
cond[3](S)	符号ビットの値
cond[2](Z)	演算結果が 0 なら 1、それ以外は 0
cond[1](C)	シフト桁数が 0 もしくは SLR の場合は 0、 それ以外の場合は最後にシフトアウトされた値、 SRL、SRA の場合は d の値に応じて case 分を使って演算結果を返した。
cond[0](V)	常に 0

表 4　条件コードの演算方法

4. Phasecounter

4.1　外部仕様

次の外部仕様を満たす 16 ビットの phasecounter を論理設計した。

- 入力信号
  - clk: マスタークロック
  - rst_n: リセット信号
  - ce: カウントを進めるか否かの信号
- 出力信号
  - p[4:0]: フェイズを表す 5bit の信号
    p[4],p[3],p[2],p[1],p[0]がそれぞれフェイズ 1,2,3,4,5 に対応している。
- 機能仕様
  - 図 4 のように入力クロックが立ち下がるごとにフェイズを一つ進め、それを出力する。

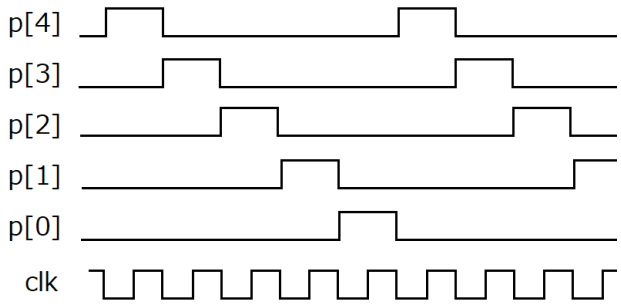

図 4　phasecounter の動作

4.2　内部仕様

```
module phasecounter(
          input clk,rst_n,ce,
          output reg [4:0] p
          );
          always @(posedge !clk or negedge rst_n) begin
             if (!rst_n) begin
                p <= 5'b10000;
             end else begin
                if (ce==1'b1) begin
                   if (p==5'b00001)
                      p <= 5'b10000;
                   else
                      p <= p >> 1;
                end
             end
          end
endmodule
```

図 5　phasecounter の HDL コード

図 5 に実装した phasecounter の HDL コードを示す。alwalys 分を使い、ce が 1 の時に限り、クロックが立ち下がるごとにフェイズ信号を一つ進めている。