提出日：2023 年 6 月 9 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

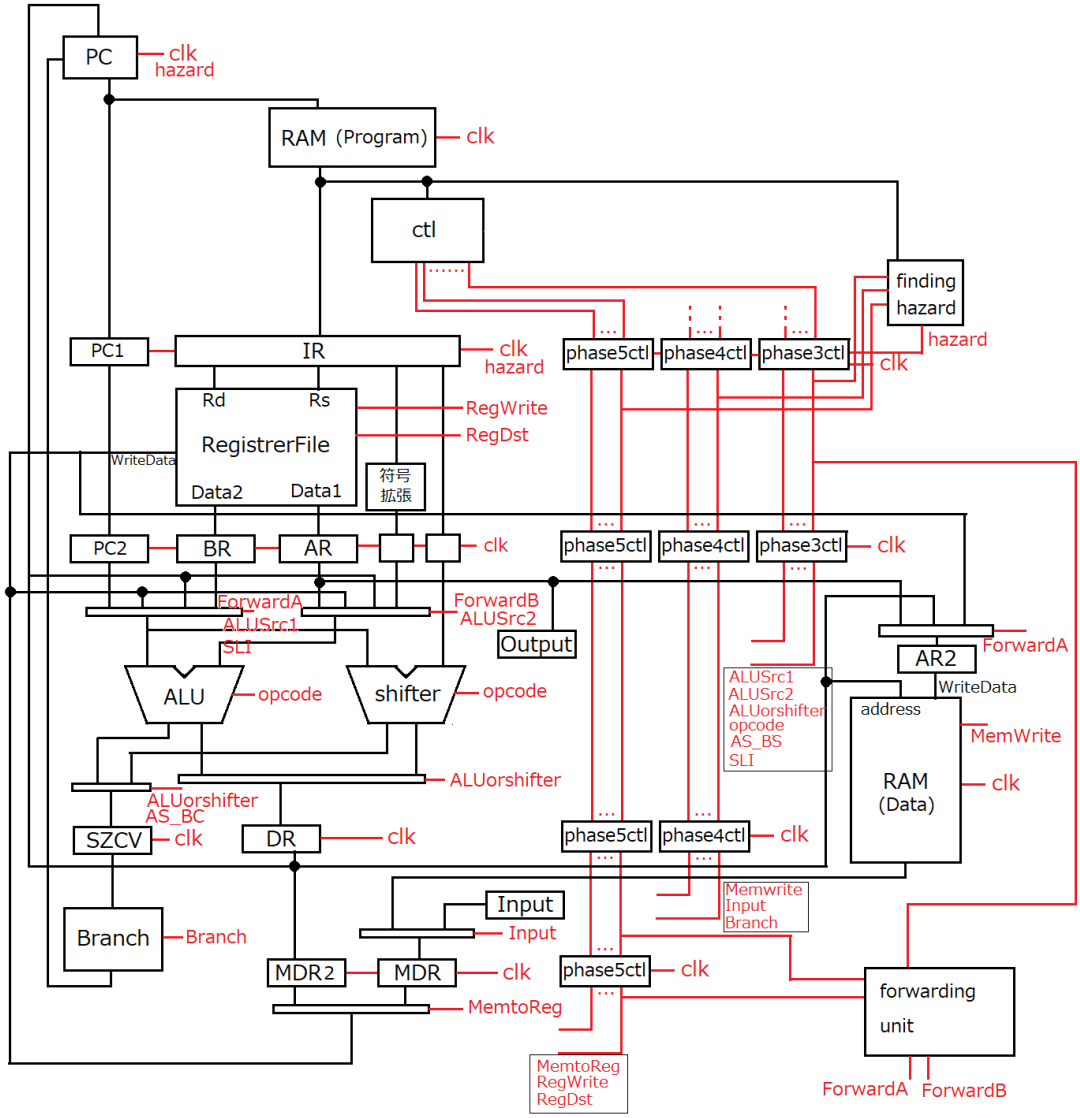

図 1　5P-SIMPLE のブロック図

図 1 に設計したプロセッサ（以下、5P-SIMPLE(Five-stage pipeline SIxteen-bit MicroProcessor for Laboratory Experiment)とよぶ）のブロック図を示す。

コンポーネント	説明
PC	PC の値を計算し、メモリのアドレス入力に入力する。内部で+1 した値をあらかじめ計算しておき、分岐命令が来たときは分岐先のアドレスと命令と分岐するかどうかの信号を受けとり、次のアドレスを出力する。
ctl	命令を受け取り、各制御信号を出力する。
各種データ用レジスタ (IR,AR,AR2,BR,PC1,PC2, SZCV,DR,MDR,MDR2)	フェーズごとにデータを受け取り、保持し、出力する。
各種制御信号用レジスタ (phase3ctl,phase4ctl, phase5ctl)	フェーズごとに制御信号を受け取り、保持し、出力する。 phase *n* ctl はフェーズ n で使われる信号を保持する。
RegisterFile	16bit 8 本のレジスタから構成されている。レジスタの読み出しと書き込みをする。
ALU	四則演算、論理演算、移動演算、比較演算を計算する。
Shifter	シフト演算を計算する。
Branch	条件コードと branch 命令の種類を示す信号受け取り、分岐するかどうかの信号を出力する。
RAM(Program)	命令用の主記憶メモリ。命令をここから読み出す。
RAM(Data)	データ用の主記憶メモリ。データの読み出しと書き込みをする。
finding_hazard	ハザードを検出し、hazard 信号を出す。
forwarding_unit	フォワーディングが必要な場合を検知し、forward 信号を出す。
pll	40MHz のクロックを入力として、100MHz のクロックを出力する。

表 1　各コンポーネントの説明

表 1 に各コンポーネントの説明を示す。

このうち設計を担当したのは ALU,shifter,pll,各種制御信号用レジスタおよび全体の設計であり、中間レポートからの追加分は pll,各種制御信号用レジスタである。

次項からこれらの機能設計仕様を示す。

2. 機能設計仕様(追加分)

2.1　pll

2.1.1 外部仕様

次の外部仕様を満たす pll を論理設計した。

- 入力信号
  - inclk0: マスタークロック
  - areset:リセット信号
- 出力信号
  - c0:出力クロック
- 機能仕様
  - inclk0 で入力されたクロックを指定された周波数のクロックに変換する。

2.1.2 内部仕様

```
// synopsys translate_off
`timescale 1 ps / 1 ps
// synopsys translate_on
module pll (
    areset,
    inclk0,
    c0);

    input     areset;
    input     inclk0;
    output    c0;
`ifndef ALTERA_RESERVED_QIS
// synopsys translate_off
`endif
    tri0      areset;
`ifndef ALTERA_RESERVED_QIS
// synopsys translate_on
`endif

    wire [4:0] sub_wire0;
    wire [0:0] sub_wire4 = 1'h0;
```

```
    wire [0:0] sub_wire1 = sub_wire0[0:0];
    wire  c0 = sub_wire1;
    wire  sub_wire2 = inclk0;
    wire [1:0] sub_wire3 = {sub_wire4, sub_wire2};

    altpll  altpll_component (
                .areset (areset),
                .inclk (sub_wire3),
                .clk (sub_wire0),
                .activeclock (),
                .clkbad (),
                .clkena ({6{1'b1}}),
                .clkloss (),
                .clkswitch (1'b0),
                .configupdate (1'b0),
                .enable0 (),
                .enable1 (),
                .extclk (),
                .extclkena ({4{1'b1}}),
                .fbin (1'b1),
                .fbmimicbidir (),
                .fbout (),
                .fref (),
                .icdrclk (),
                .locked (),
                .pfdena (1'b1),
                .phasecounterselect ({4{1'b1}}),
                .phasedone (),
                .phasestep (1'b1),
                .phaseupdown (1'b1),
                .pllena (1'b1),
                .scanaclr (1'b0),
                .scanclk (1'b0),
                .scanclkena (1'b1),
                .scandata (1'b0),
                .scandataout (),
```

```
                .scandone (),
                .scanread (1'b0),
                .scanwrite (1'b0),
                .sclkout0 (),
                .sclkout1 (),
                .vcooverrange (),
                .vcounderrange ());
    defparam
        altpll_component.bandwidth_type = "AUTO",
        altpll_component.clk0_divide_by = 40,
        altpll_component.clk0_duty_cycle = 50,
        altpll_component.clk0_multiply_by = 100,
        altpll_component.clk0_phase_shift = "0",
        altpll_component.compensate_clock = "CLK0",
        altpll_component.inclk0_input_frequency = 25000,
        altpll_component.intended_device_family = "Cyclone IV E",
        altpll_component.lpm_hint = "CBX_MODULE_PREFIX=pll",
        altpll_component.lpm_type = "altpll",
        altpll_component.operation_mode = "NORMAL",
        altpll_component.pll_type = "AUTO",
        altpll_component.port_activeclock = "PORT_UNUSED",
        altpll_component.port_areset = "PORT_USED",
        altpll_component.port_clkbad0 = "PORT_UNUSED",
        altpll_component.port_clkbad1 = "PORT_UNUSED",
        altpll_component.port_clkloss = "PORT_UNUSED",
        altpll_component.port_clkswitch = "PORT_UNUSED",
        altpll_component.port_configupdate = "PORT_UNUSED",
        altpll_component.port_fbin = "PORT_UNUSED",
        altpll_component.port_inclk0 = "PORT_USED",
        altpll_component.port_inclk1 = "PORT_UNUSED",
        altpll_component.port_locked = "PORT_UNUSED",
        altpll_component.port_pfdena = "PORT_UNUSED",
        altpll_component.port_phasecounterselect = "PORT_UNUSED",
        altpll_component.port_phasedone = "PORT_UNUSED",
        altpll_component.port_phasestep = "PORT_UNUSED",
        altpll_component.port_phaseupdown = "PORT_UNUSED",
```

```
        altpll_component.port_pllena = "PORT_UNUSED",
        altpll_component.port_scanaclr = "PORT_UNUSED",
        altpll_component.port_scanclk = "PORT_UNUSED",
        altpll_component.port_scanclkena = "PORT_UNUSED",
        altpll_component.port_scandata = "PORT_UNUSED",
        altpll_component.port_scandataout = "PORT_UNUSED",
        altpll_component.port_scandone = "PORT_UNUSED",
        altpll_component.port_scanread = "PORT_UNUSED",
        altpll_component.port_scanwrite = "PORT_UNUSED",
        altpll_component.port_clk0 = "PORT_USED",
        altpll_component.port_clk1 = "PORT_UNUSED",
        altpll_component.port_clk2 = "PORT_UNUSED",
        altpll_component.port_clk3 = "PORT_UNUSED",
        altpll_component.port_clk4 = "PORT_UNUSED",
        altpll_component.port_clk5 = "PORT_UNUSED",
        altpll_component.port_clkena0 = "PORT_UNUSED",
        altpll_component.port_clkena1 = "PORT_UNUSED",
        altpll_component.port_clkena2 = "PORT_UNUSED",
        altpll_component.port_clkena3 = "PORT_UNUSED",
        altpll_component.port_clkena4 = "PORT_UNUSED",
        altpll_component.port_clkena5 = "PORT_UNUSED",
        altpll_component.port_extclk0 = "PORT_UNUSED",
        altpll_component.port_extclk1 = "PORT_UNUSED",
        altpll_component.port_extclk2 = "PORT_UNUSED",
        altpll_component.port_extclk3 = "PORT_UNUSED",
        altpll_component.width_clock = 5;

endmodule
```

図 2　pll の HDL コード

図 2 に実装した pll の HDL コードを示す。この pll は Quartus Prime のマクロ機能を使って実装したので、コードの詳細な説明は省略するが、

```
       altpll_component.clk0_divide_by = 40,
        altpll_component.clk0_duty_cycle = 50,
        altpll_component.clk0_multiply_by = 100,
```

の部分で、出力するクロックの周波数と Dirty 比を指定する。例えばこの場合は出力されるクロックは Dirty 比 50%、$40 \times \frac{100}{40} = 100$MHzのクロックが出力される。

2.2 各種制御信号用レジスタ(phase3ctl,phase4ctl,phase5ctl)

2.2.1 外部仕様

次の外部仕様を満たす制御信号用レジスタを論理設計した。

- 入力信号
  - 各種制御信号: phase *n* ctl はフェーズ n で必要になる制御信号を保持し、出力する。

コンポーネント	保持する制御信号
phase3ctl	ALUSrc1,ALUSrc2,ALUorshifter,AS_BC,MemRead,SLI,Ra,Rb,opcode
phase4ctl	MemWrite,Input,Branch
phase5ctl	MemtoReg,RegWrite,RegDst

    各制御信号の意味は ctl の機能設計仕様書を参照。
  - clk:入力クロック
  - rst_n:リセット信号
- 出力信号
  - 各種制御信号
- 機能仕様
  - クロックが立ち上がるたびに入力される制御信号を保持し、出力する。

2.2.2 内部仕様

```
module phase3ctl(
    input ALUSrc1in,ALUSrc2in,ALUorshifterin,AS_BCin,MemReadin,SLIin,
    input [2:0] Rain,Rbin,
    input [3:0] opcodein,
    input clk,rst_n,
    output reg
ALUSrc1out,ALUSrc2out,ALUorshifterout,AS_BCout,MemReadout,SLIout,
    output reg [2:0] Raout,Rbout,
    output reg [3:0] opcodeout);
    always @ (posedge clk) begin
        if(!rst_n) begin
            ALUSrc1out <= 1'b0;
            ALUSrc2out <= 1'b0;
            ALUorshifterout <= 1'b0;
            AS_BCout <= 1'b0;
            opcodeout <= 4'b0000;
            MemReadout <= 1'b0;
```

```
            SLIout <= 1'b0;
            Raout <= 3'b000;
            Rbout <= 3'b000;
        end else begin
            ALUSrc1out <= ALUSrc1in;
            ALUSrc2out <= ALUSrc2in;
            ALUorshifterout <= ALUorshifterin;
            AS_BCout <= AS_BCin;
            opcodeout <= opcodein;
            MemReadout <= MemReadin;
            SLIout <= SLIin;
            Raout <= Rain;
            Rbout <= Rbin;
        end
    end
endmodule
```

図 3　phase3ctl の HDL コード

```
module phase4ctl(
    input MemWritein, Inputin,
    input [2:0] Branchin,
    input clk,rst_n,
    output reg MemWriteout, Inputout,
    output reg [2:0] Branchout);
    always @ (posedge clk) begin
        if(!rst_n) begin
            MemWriteout <= 1'b0;
            Inputout <= 1'b0;
            Branchout <= 3'b000;
        end else begin
            MemWriteout <= MemWritein;
            Inputout <= Inputin;
            Branchout <= Branchin;
        end
    end
endmodule
```

図 4　phase4ctl の HDL コード

```
module phase5ctl(
    input MemtoRegin,RegWritein,
    input [2:0] RegDstin,
    input clk,rst_n,
    output reg MemtoRegout, RegWriteout,
    output reg [2:0] RegDstout);
    always @ (posedge clk) begin
        if(!rst_n) begin
            MemtoRegout <= 1'b0;
            RegWriteout <= 1'b0;
            RegDstout <= 3'b000;
        end else begin
            MemtoRegout <= MemtoRegin;
            RegWriteout <= RegWritein;
            RegDstout <= RegDstin;
        end
    end
endmodule
```

図 5　phase5ctl の HDL コード

図 3、図 4、図 5 にそれぞれ設計した phase3ctl,phase4ctl,phase5ctl の HDL コードを示す。設計は単純で、always 文を使い、クロックがたちあがるごとに出力レジスタに制御信号を入力している。

3. 性能評価

3.1 ALU

Analysis & Synthesis Status	Successful - Thu Jun 08 14:47:31 2023
Quartus Prime Version	20.1.0 Build 711 06/05/2020 SJ Lite Edition
Revision Name	ALU
Top-level Entity Name	ALU
Family	Cyclone IV E
Total logic elements	95
Total registers	0
Total pins	56
Total virtual pins	0
Total memory bits	0
Embedded Multiplier 9-bit elements	0
Total PLLs	0

図 6　ALU の Analysis & Synthesis 結果

図 6 に ALU の Analysis & Synthesis 結果を示す。Total logic elements は 95 となった。

1	A[0]	Out[0]	8.562	8.526	8.653	8.538
2	A[0]	Out[1]	8.278	8.102	8.615	8.448
3	A[0]	Out[2]	8.804	8.618	8.908	8.742
4	A[0]	Out[3]	8.760	8.690	9.055	8.985
5	A[0]	Out[4]	8.834	8.646	8.920	8.746
6	A[0]	Out[5]	8.895	8.703	9.234	9.051
7	A[0]	Out[6]	9.208	9.098	9.349	9.239
8	A[0]	Out[7]	8.580	8.465	8.875	8.760
9	A[0]	Out[8]	8.383	8.257	8.469	8.398
10	A[0]	Out[9]	9.803	9.587	10.098	9.883
11	A[0]	Out[10]	8.979	8.803	9.120	8.943
12	A[0]	Out[11]	11.053	11.089	11.348	11.384

図 7　ALU の Propagation Delay の内容(一部)

図 7 に Propagation Delay の内容(一部)を示す。遅延時間は 5ns~15ns となった。

3.2 shifter

Flow Status	Successful - Thu Jun 08 15:19:42 2023
Quartus Prime Version	20.1.0 Build 711 06/05/2020 SJ Lite Edition
Revision Name	shifter
Top-level Entity Name	shifter
Family	Cyclone IV E
Device	EP4CE30F23I7
Timing Models	Final
Total logic elements	207 / 28,848 (< 1 %)
Total registers	0
Total pins	44 / 329 (13 %)
Total virtual pins	0
Total memory bits	0 / 608,256 (0 %)
Embedded Multiplier 9-bit elements	0 / 132 (0 %)
Total PLLs	0 / 4 (0 %)

図 8　shifter の Analysis & Synthesis 結果

図 8 に shifter の Analysis & Synthesis 結果を示す。Total logic elements は 207 となった。

	Input Port	Output Port	RR	RF	FR	FF
1	d[1]	Out[14]	17.428	17.341	17.824	17.799
2	A[8]	Out[14]	17.376	17.389	17.853	17.766
3	A[7]	Out[14]	17.339	17.352	17.783	17.696
4	A[2]	Out[14]	17.297	17.310	17.808	17.721
5	d[0]	Out[14]	17.235	17.148	17.569	17.582
6	A[10]	Out[14]	17.190	17.203	17.582	17.495
7	A[0]	Out[14]	17.150	17.163	17.644	17.557
8	A[9]	Out[14]	17.056	17.069	17.495	17.408
9	A[1]	Out[14]	16.956	16.969	17.476	17.389

図 9　shifter の Propagation Delay の内容(一部)

図 9 に Propagation Delay の内容(一部)を示す。遅延時間は 9ns~18ns となった。

3.3 制御信号用レジスタ(phase3ctl,phase4ctl,phase5ctl)

Flow Status	Successful - Thu Jun 08 15:39:58 2023
Quartus Prime Version	20.1.0 Build 711 06/05/2020 SJ Lite Edition
Revision Name	phase3ctl
Top-level Entity Name	phase3ctl
Family	Cyclone IV E
Device	EP4CE30F23I7
Timing Models	Final
Total logic elements	16 / 28,848 (< 1 %)
Total registers	16
Total pins	34 / 329 (10 %)
Total virtual pins	0
Total memory bits	0 / 608,256 (0 %)
Embedded Multiplier 9-bit elements	0 / 132 (0 %)
Total PLLs	0 / 4 (0 %)

Flow Status	Successful - Thu Jun 08 15:54:27 2023
Quartus Prime Version	20.1.0 Build 711 06/05/2020 SJ Lite Edition
Revision Name	phase3ctl
Top-level Entity Name	phase4ctl
Family	Cyclone IV E
Device	EP4CE30F23I7
Timing Models	Final
Total logic elements	10 / 28,848 (< 1 %)
Total registers	5
Total pins	12 / 329 (4 %)
Total virtual pins	0
Total memory bits	0 / 608,256 (0 %)
Embedded Multiplier 9-bit elements	0 / 132 (0 %)
Total PLLs	0 / 4 (0 %)

Flow Status	Successful - Thu Jun 08 15:58:10 2023
Quartus Prime Version	20.1.0 Build 711 06/05/2020 SJ Lite Edition
Revision Name	phase3ctl
Top-level Entity Name	phase5ctl
Family	Cyclone IV E
Device	EP4CE30F23I7
Timing Models	Final
Total logic elements	10 / 28,848 (< 1 %)
Total registers	5
Total pins	12 / 329 (4 %)
Total virtual pins	0
Total memory bits	0 / 608,256 (0 %)
Embedded Multiplier 9-bit elements	0 / 132 (0 %)
Total PLLs	0 / 4 (0 %)

図 10　制御信号用レジスタの Analysis & Synthesis 結果(phase3ctl,phase4ctl,phase5ctl)

図 10 に制御信号用レジスタの Analysis & Synthesis 結果を示す。Total logic elements はそれぞれ 16,10,10 となった。

	Fmax	Restricted Fmax	Clock Name	Note
1	102.91 MHz	102.91 MHz	clk	

	Fmax	Restricted Fmax	Clock Name	Note
1	144.05 MHz	144.05 MHz	clk	

	Fmax	Restricted Fmax	Clock Name	Note
1	144.05 MHz	144.05 MHz	clk	

図 11　制御信号用レジスタの Fmax Summary の内容(phase3ctl,phase4ctl,phase5ctl)

図 11 に Fmax Summary の内容を示す。最大周波数はそれぞれ 102.91MHz,144.05MHz,144.05MHz となった。

4. 考察

4.1 設計全体について

担当コンポーネントの設計に加え、各コンポーネントを組み合わせてプロセッサ全体の構成も担当した。
苦労したことの一つは、全体を構成するにあたって何の制御信号が必要かを考える過程である。最初から全体の構成を明確に決めて設計したわけではなく、設計する中で必要になると気づいた制御信号を制御部の設計を担当したチームメンバーに要請していくという形で全体の構成をした。
また、どのコンポーネントを組み合わせ回路にし、どのコンポーネントを順序回路にするのかについても様々な考察を行ったうえで設計をした。例えば、中間レポート時点では制御部は順序回路であったが、パイプライン化をするうえで 1 フェーズ内で制御信号の計算を終わらせる必要があり、組み合わせ回路に仕様変更をした。
主記憶メモリに加え、レジスタファイルに与えるクロックも逆クロックにしている。その理由は、逆クロックにしないと書き込むべきデータが届く前にレジスタを書き換えてしまい、正しい値がレジスタに保存されないという問題が発生したからである。

設計するにあたって、デバッグが容易になるようなモジュールをまず実装した。display モジュールは MU500-7SEG のすべての 7 セグメント(4 ケタずつ)に指定した 16 ビット 16 進数の数を表示させることができる。これによりすべてのレジスタの値を常時確認でき、加えて ALU の出力などを確認することができる。応用プログラムのデバッグに大いに役立った。

マルチサイクルプロセッサは各コンポーネントを単純につなげるだけなので比較的容易に実装が完了したが、パイプラインプロセッサを設計する際には様々な問題が生じた。まず、どの場合にハザードが起こり、どの場合にフォワーディングが必要になるか、そしてそれらをどこで検出するのかを考えるのにとても苦労した。今回設計したプロセッサは制御信号の計算をフェーズ 1 で、ハザード検出をフェーズ 1 とフェーズ 2 にわたって行っているので、ハザードでストールすべき命令をフェッチした瞬間にハザードを検出することが可能となっている。その代わり、遅延が大きいメモリへアクセスした直後に制御信号を計算し、出力しなければならないので、フェーズ 1 の実行に時間がかかっている。そのため、最大周波数は 100MHz にとどまったと考えられる。
制御部をフェーズ 2 に移すことも実装途中で考え、その方向性で実装を進めていたが、最後までバグが取れずに断念した。
また、分岐命令で不成立分岐予測をし、分岐が成立した場合に今実行中の命令をフラッシ

ュしなければならないが、どの信号をフラッシュし、どの命令をフラッシュしないかを考えることに苦労した。結論としては、レジスタとメモリを書き換えることを示す信号(RegWrite, MemWrite)と、分岐命令であることを示す信号(Branch)、条件コードを書き換える信号(AS_BC)をフラッシュするように実装した。こうすることで不成立分岐予測した命令はなかったことに出来る。

成立分岐予測にすることも考え、実装自体は比較的容易であったものの、最大周波数が大きく下がってしまったので(20MHz)、あえなく不成立分岐予測を採用した。最大周波数が大きく下がってしまった理由として、成立分岐予測の場合、分岐命令をフェッチすると分岐先のアドレスを計算して次のフェーズで PC の値を書き換えるが、フェーズ 1 でそれを行うため、分岐命令かどうかを判別した後に分岐先のアドレスを計算するようにするとさらにフェーズ 1 の計算に時間がかかってしまうためであると考えられる。

以上のことから、最大周波数をある程度担保しつつ成立分岐予測を実装するには
① 制御信号の計算をフェーズ 2 で行うようにする。
② 分岐先のアドレス計算をフェーズ 1 で優先して行うようにする。
とよいと考えられる。

4.2 設計を担当したコンポーネントについて

設計を担当したのは ALU,shifter,pll,制御信号用レジスタである。

4.2.1 ALU について

二回生の後期の実験で一度実装しているので比較的容易に実装が完了した。図 12 に実装した ALU の HDL コードを示す。最初は順序回路として実装していたが、動作速度を確保するために途中でできるだけシンプルな組み合わせ回路に仕様変更した。マルチプレクサ構文を用いて opcode 毎の演算を指定している。

```
module ALU(
       input [3:0] ALUctl,
       input [15:0] A,B,
       output [15:0] Out,
       output [3:0] Outcond
       );
       wire [16:0] C;
       wire [3:0] cond;
       assign C[16:0] = (ALUctl == 4'b0000) ? A + B:
                        (ALUctl == 4'b0001) ? A - B:
                        (ALUctl == 4'b0010) ? {1'b0,A & B}:
                        (ALUctl == 4'b0011) ? {1'b0,A | B}:
                        (ALUctl == 4'b0100) ? {1'b0,A ^ B}:
                        (ALUctl == 4'b0101) ? A - B:
                        {1'b0,B};
       assign cond[3] = C[15];
       assign cond[2] = (C[15:0]==16'b0) ? 1'b1:
                         1'b0;
       assign cond[1] = C[16];
       assign cond[0] = (ALUctl == 4'b0000) ? (A[15] ~^ B[15]) & (A[15] ^ C[15]):
                        (ALUctl == 4'b0001) ? (A[15] ^ B[15]) & (A[15] ^ C[15]):
                        (ALUctl == 4'b0010) ? 1'b0:
                        (ALUctl == 4'b0011) ? 1'b0:
                        (ALUctl == 4'b0100) ? 1'b0:
                        (ALUctl == 4'b0101) ? (A[15] ^ B[15]) & (A[15] ^ C[15]):
                       1'b0;
       assign Outcond = cond;
       assign Out = C[15:0];
endmodule
```

図 12　ALU の HDL コード

組み合わせ回路にしたことで遅延が大きく減少した。

4.2.2 shifter について

図 13 に実装した shifter の HDL コードを示す。基本は ALU と相違ないが、Verilog の場合インデックスに変数を指定することができないので、シフト桁数に対してマルチプレクサ構文を使って循環シフトや条件コードを計算しなければならないことに苦労した。こちらも最初は順序回路として実装していたが、のちに組み合わせ回路に仕様変更した。

```
module shifter (
       input signed [15:0] A,
       input [3:0] opcode,
       input [3:0] d,
       output [15:0] Out,
       output [3:0] Outcond
       );
       wire [15:0] C;
       wire [15:0] D;
       wire [31:0] E;
       wire [3:0] cond;
       wire [0:0] cond1;
       assign D[15:0] = (opcode == 4'b1000) ? A << d:
                        (opcode == 4'b1001) ? C[15:0]:
                        (opcode == 4'b1010) ? A >> d:
                        A >>> d;
       assign E = A << d;
       assign C = (d == 4'b0000) ? E[15:0]:
                  (d == 4'b0001) ? {E[15:1],E[16]}:
                  (d == 4'b0010) ? {E[15:2],E[17:16]}:
                  (d == 4'b0011) ? {E[15:3],E[18:16]}:
                  (d == 4'b0100) ? {E[15:4],E[19:16]}:
                  (d == 4'b0101) ? {E[15:5],E[20:16]}:
                  (d == 4'b0110) ? {E[15:6],E[21:16]}:
                  (d == 4'b0111) ? {E[15:7],E[22:16]}:
                  (d == 4'b1000) ? {E[15:8],E[23:16]}:
                  (d == 4'b1001) ? {E[15:9],E[24:16]}:
                  (d == 4'b1010) ? {E[15:10],E[25:16]}:
                  (d == 4'b1011) ? {E[15:11],E[26:16]}:
                  (d == 4'b1100) ? {E[15:12],E[27:16]}:
                  (d == 4'b1101) ? {E[15:13],E[28:16]}:
                  (d == 4'b1110) ? {E[15:14],E[29:16]}:
                  {E[15],E[30:16]};
       assign cond1 = (d == 4'b0001) ? A[0]:
                      (d == 4'b0010) ? A[1]:
                      (d == 4'b0011) ? A[2]:
                      (d == 4'b0100) ? A[3]:
                      (d == 4'b0101) ? A[4]:
                      (d == 4'b0110) ? A[5]:
                      (d == 4'b0111) ? A[6]:
                      (d == 4'b1000) ? A[7]:
                      (d == 4'b1001) ? A[8]:
                      (d == 4'b1010) ? A[9]:
                      (d == 4'b1011) ? A[10]:
                      (d == 4'b1100) ? A[11]:
                      (d == 4'b1101) ? A[12]:
                      (d == 4'b1110) ? A[13]:
                      A[14];
       assign cond[0] = 1'b0;
       assign cond[1] = (d==0||opcode ==4'd9) ? 1'b0:
                        (opcode == 4'd8) ? E[16]:
                        cond1;
       assign cond[2] = (D[15:0]==16'b0) ? 1'b1:
                         1'b0;
       assign cond[3] = D[15];
       assign Out = D;
       assign Outcond = cond;
endmodule
```

図 13　shifter の HDL コード

4.2.3 制御信号用レジスタについて

　フェーズごとに信号を分けたことでフォワーディングユニットやハザード検出ユニットを組み合わせるときに実装が容易になった。

5. 感想

この実験は自分にとって大いに有意義なものとなった。基本的なプロセッサの仕組みを深く理解することができたのはもちろん、デバッグの方法(デバッグ用モジュールを実装する)や実装の手順(トップダウンかボトムアップか)についても効率的な方法を学ぶことができた。それに加え、同じ目標物に対し複数人で作業を分担して行い、完成させるのは新鮮な経験であった。

この経験を研究室での研究などに生かしていきたい。