『新・標準プログラマーズライブラリ
RISC-V で学ぶコンピュータアーキテクチャ 完全入門』
演習問題と解答

第 1 章

Q1. 販売されているパーソナルコンピュータ（PC）で利用される 3 個の入力装置と 3 個の出力装置を挙げてください。

入力装置には、キーボード、マウス、マイク、スキャナー、ウェブカメラなどがあります。例えば、3 個の入力装置は、キーボード、マウス、マイクです。

出力装置には、ディスプレイ、プリンター、スピーカー、ヘッドフォン、プロジェクターなどがあります。例えば、3 個の出力装置は、ディスプレイ、プリンター、スピーカーです。

Q2. できるだけ高い動作周波数で動作して、5 万円以内の予算で購入できる PC 用のプロセッサを見つけてください。その最大の動作周波数はどのような値になるでしょうか。

2024 年 1 月の時点で、例えば、Intel Core i7-11700KF プロセッサは 5 万円以内の予算で購入できて、その最大の動作周波数は 5.0GHz です。

Q3. コンピュータの記憶装置として DDR5 という規格のメモリが利用されています。32GB の容量の DDR5 メモリと 64GB の容量の DDR5 メモリの価格を調べましょう。64GB のメモリの価格は、32GB のメモリの価格の 2 倍より高いでしょうか、低いでしょうか。

2024 年 1 月の時点で、例えば、A-Tech 64GB RAM は 64GB の DDR5 メモリで約 73,000 円です。一方、A-Tech 32GB RAM は 32GB の DDR5 メモリで約 38,000 円です。これらの比較では、64GB のメモリの価格は、32GB のメモリの価格の 2 倍より低くなっています。

Q4. コンピュータアーキテクチャは、建築あるいは建築様式を意味するアーキテクチャの前にコンピュータという単語を追加した専門用語です。同様に、アーキテクチャの前に単語を追加した 3 個の専門用語を見つけてください。

例えば、ソフトウェアアーキテクチャ、ネットワークアーキテクチャ、システムアーキテクチャといった専門用語が利用されています。

Q5. 販売されている携帯電話に搭載されているプロセッサが採用している命令セットアーキテクチャが何なのか調べてください。また、それは RISC なのか、それとも CISC なのかを調べてください。

例えば、携帯電話の iPhone 15 は A16 Bionic プロセッサを搭載しています。A16 Bionic プロセッサで採用されている命令セットアーキテクチャは、64 ビットの ARM です。これは RISC になります。

Q6. あるプログラム A をコンピュータ Y で実行したところ 30 秒を要しました。また、同じプログラム A をコンピュータ X で実行したところ 15 秒を要しました。この結果から、コンピュータ Y に対するコンピュータ X の性能向上率を求めてください。

コンピュータ Y に対するコンピュータ X の性能向上率は、30/15 = 2 になります。

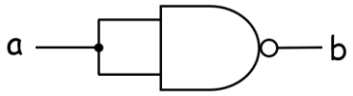
Q7. あるプログラム A をコンピュータ Y で実行したところ 30 秒を要しました。また、同じプログラム A をコンピュータ X で実行したところ 15 秒を要しました。別のプログラム B をコンピュータ Y で実行したところ 100 秒を要しました。また、同じプログラム B をコンピュータ X で実行したところ 10 秒を要しました。この結果から、プログラム A と B を続けて実行する場合に、コンピュータ Y に対するコンピュータ X の性能向上率を求めてください。

コンピュータ Y に対するコンピュータ X の性能向上率は、(30+100)/(15+10) = 130/25 = 5.2 になります。

第 2 章

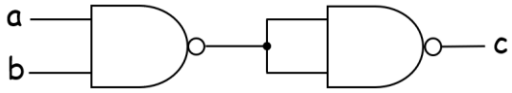
Q1. NAND ゲートのみを利用して NOT ゲートを実現し、その回路図を示してください。

回路図は次になります。



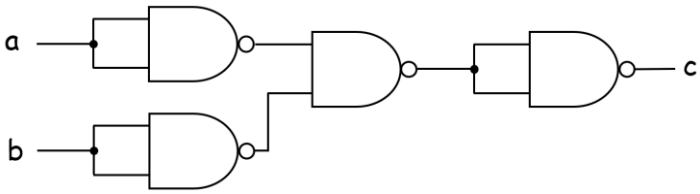
Q2. NAND ゲートのみを利用して AND ゲートを実現し、その回路図を示してください。

回路図は次になります。



Q3. NAND ゲートのみを利用して NOR ゲートを実現し、その回路図を示してください。ヒント：NOR ゲートの入力をそれぞれ反転させた時の真理値表を書いてみましょう。

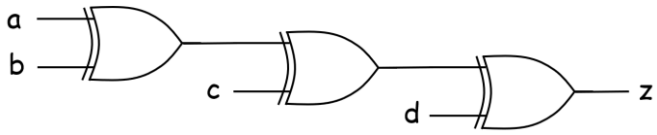
回路図は次になります。



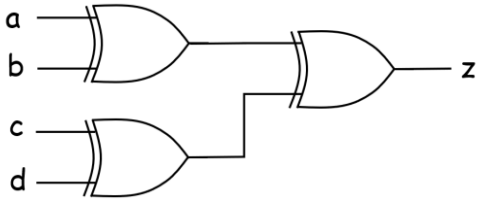
Q4. 3 個の 2 入力の XOR を利用して 4 入力の XOR 回路を実現し、その回路図を示してください。構成した 4 入力の XOR 回路の入力信号を a, b, c, d として、出力信号を z とします。入力

に 1 が含まれないとき、1 個だけ含まれるとき、2 個だけ含まれるとき、3 個だけ含まれるとき、4 個のすべてが 1 のとき、のそれぞれの場合における z の値を示してください。

例えば、回路図は次になります。



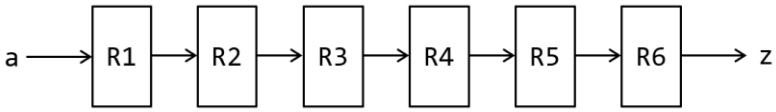
次の回路図でも正解です。



入力に 1 が含まれないときの z の値は 0 になります。入力に 1 が 1 個だけ含まれるときの z の値は 1 になります。入力に 1 が 2 個だけ含まれるときの z の値は 0 になります。入力に 1 が 3 個だけ含まれるときの z の値は 1 になります。入力の 4 個のすべてが 1 のときの z の値は 0 になります。

Q5. 入力信号を a、出力信号を z とする 6 ビットのシフトレジスタの回路図を示してください。利用する 6 個のレジスタには R1～R6 のラベルを付けてください。

回路図は次になります。

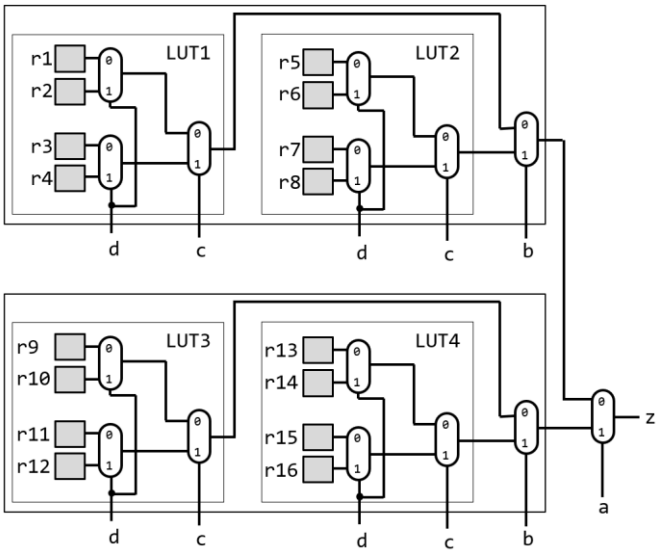


Q6. 図 2-25 の LUT を利用して、XOR ゲートを実現します。この時に、r1, r2, r3, r4 に格納すべき値を示してください。

r1, r2, r3, r4 に格納すべき値は、それぞれ 0, 1, 1, 0 です。

Q7. 図 2-29 の 3 入力の LUT を利用して、4 入力の LUT を実現する回路を示してください。入力信号を a, b, c, d として、出力信号を z とします。構成した 4 入力の LUT を利用して、4 入力の XOR 回路を実現します。この時に、すべての LUT の中のレジスタに格納すべき値を示してください。

回路図は次になります。



r1 から r16 に格納すべき値はそれぞれ、0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0 です。

第 3 章

Q1. 次のコードのシミュレーションの表示を示してください。

```
1  module m_top();
2    reg  [3:0] r_a=4'b1010, r_b=4'b1100;
3    wire [3:0] w_c = r_a ^ r_b;
4    initial #1 $display("%b %b %b", r_a, r_b, w_c);
5  endmodule
```

シミュレーションの出力は次の通りです。

```
1010 1100 0110
```

Q2. 次のコードのシミュレーションの表示を示してください。

```
1  module m_top();
2    reg  [3:0] r_a=4'b1010, r_b=4'b1100;
3    wire [3:0] w_c = ~r_a ^ r_b;
4    initial #1 $display("%b %b %b", r_a, r_b, w_c);
5  endmodule
```

シミュレーションの出力は次の通りです。

```
1010 1100 1001
```

Q3. 次のコードのシミュレーションの表示を示してください。

```
1  module m_top();
2    reg  [3:0] r_a=4'b1010, r_b=4'bzzxx;
3    wire [3:0] w_c = r_a ^ r_b;
4    initial #1 $display("%b %b %b", r_a, r_b, w_c);
5  endmodule
```

シミュレーションの出力は次の通りです。

```
1010 zzxx xxxx
```

Q4. 次のコードのシミュレーションの表示を示してください。

```
1  module m_top();
2    initial forever #100 $display("a");
3    initial #550 $finish;
4    initial #350 $display("b");
5    initial begin #260 $display("c"); #260 $display("d"); end
6  endmodule
```

シミュレーションの出力は次の通りです。

```
a
a
c
a
b
a
a
```

Q5. 次のコードのシミュレーションの表示を示してください。

```
1  module m_top();
2    initial forever #100 $write("a");
3    initial #550 $finish;
4    initial #350 $write("b");
5    initial begin #260 $write("c"); #260 $write("d\n"); end
6  endmodule
```

シミュレーションの出力は次の通りです。

```
aacabaad
```

Q6. 次のコードのシミュレーションの表示を示してください。

```
1  module m_top();
2    wire signed [7:0] w_a = -1, w_b = -2;
3    initial #1 $display("%d %b %d %b", w_a, w_a, w_b, w_b);
4  endmodule
```

シミュレーションの出力は次の通りです。

```
-1 11111111   -2 11111110
```

Q7. 次のコードのシミュレーションの表示を示してください。

```
1  module m_top();
2    reg [7:0] r_a = 0;
3    initial begin r_a = r_a + 1; r_a = r_a + 2; end
4    initial #1 $display("%d %b", r_a, r_a);
5  endmodule
```

シミュレーションの出力は次の通りです。

```
3 00000011
```

Q8. 次のコードのシミュレーションの表示を示してください。

```
1  module m_top();
2    reg r_clk = 0; initial #200 r_clk = 1;
3    reg [7:0] r_a = 0;
4    always@(posedge r_clk) r_a <= r_a + 1;
5    initial #201 $display("%d", r_a);
6    initial #199 $display("%d", r_a);
7  endmodule
```

シミュレーションの出力は次の通りです。

```
0

1
```

Q9. 次のコードのシミュレーションの表示を示してください。

```
1 `define EDGE 200
2  module m_top();
3    reg r_clk = 0; initial #`EDGE r_clk = 1;
4    reg [7:0] r_a = 0;
```

```
5    always@(posedge r_clk) r_a <= r_a + 1;
6    initial #(`EDGE+1) $display("%d", r_a);
7    initial #(`EDGE-1) $display("%d", r_a);
8  endmodule
```

シミュレーションの出力は次の通りです。

```
 0

 1
```

第 4 章

Q1. 次の命令列を実行した後の、x5, x6, x30 の値を示してください。

```
addi x5, x0, 7
addi x6, x0, 8
add x30, x5, x6
```

x5, x6, x30 の値はそれぞれ、7, 8, 15 です。

Q2. 次の命令列を実行した後の、x1, x3, x5, x30 の値を示してください。

```
addi x5, x0, 7
sw x5, 512(x0)
add x3, x5, x5
lw x1, 512(x0)
add x30, x1, x3
```

x1, x3, x5, x30 の値はそれぞれ、7, 14, 7, 21 です。

Q3. 次の命令列を実行した後の、x10 の値を示してください。行頭の数字は、その命令が格納されるアドレスです。

```
32'h0          addi x10, x0, 0
32'h4          addi x3, x0, 0
32'h8          addi x1, x0, 11
32'hc   label: add  x10, x10 , x3
32'h10         addi x3, x3,1
32'h14         bne  x3, x1, label
```

x10 の値は 55 です。

Q4. add x10, x11, x12 を 32 ビットのビット列に変換し、それを 2 進数で示してください。

この命令は 16 進数で表現すると 32'h00c58533 になります。

2 進数では、32'b0000_0000_1100_0101_1000_0101_0011_0011 になります。

Q5. addi x10, x11, -2 を 32 ビットのビット列に変換し、それを 2 進数で示してください。

この命令は 16 進数で表現すると 32'hffe58513 になります。

2 進数では、32'b1111_1111_1110_0101_1000_0101_0001_0011 になります。

Q6. lw x10, 32(x8)を 32 ビットのビット列に変換し、それを 2 進数で示してください。

この命令は 16 進数で表現すると 32'h02042503 になります。

2 進数では、32'b0000_0010_0000_0100_0010_0101_0000_0011 になります。

Q7. sw x9, 24(x7)を 32 ビットのビット列に変換し、それを 2 進数で示してください。

この命令は 16 進数で表現すると 32'h0093ac23 になります。

2 進数では、32'b0000_0000_1001_0011_1010_1100_0010_0011 になります。

Q8. 次の命令列に含まれる beq 命令を 2 進数のビット列で示してください。行頭の数字は、その命令が格納されるアドレスです。

```
32'h0  label: addi x5, x0, 1    # x5 = 1
32'h4         addi x6, x0, 2    # x6 = 2
```

```
32'h8          addi x7, x0, 3    # x7 = 3
32'hc          beq x6, x7 label  # if (x6==x7) goto label
```

この命令は 16 進数で表現すると 32'hfe730ae3 になります。

2 進数では、32'b1111_1110_0111_0011_0000_1010_1110_0011 になります。

第 5 章

Q1. コード 5-38 の命令列を記述する asm.txt を利用してプロセッサ proc5 の動作をシミュレーションして、その表示を示してください。

```
1  `MM[0] ={12'd7,5'd0,3'h0,5'd1,7'h13};          // 00   addi x1,x0,7
2  `MM[1] ={7'd0,5'd1,5'd0,3'h2,5'd0,7'h23};      // 04   sw   x1,0(x0)
3  `MM[2] ={12'd0,5'd0,3'h2,5'd2,7'h3};           // 08   lw   x2,0(x0)
4  `MM[3] ={12'd0,5'd2,3'h0,5'd10,7'h13};         // 0c   addi x10,x2,0
5  `MM[4] =32'h00050f13;                          // 10   HALT
```

シミュレーションの出力は次の通りです。

```
CC01 00000000          0          7          7
CC02 00000004          0          0          0
CC03 00000008          0          0          7
CC04 0000000c          7          0          7
CC05 00000010          7          0          7
```

Q2. コード 5-39 の命令列を記述する asm.txt を利用してプロセッサ proc5 の動作をシミュレーションして、その表示を示してください。

```
1  `MM[0]={12'd3,5'd0,3'h0,5'd1,7'h13};           // 00   addi x1,x0,3
2  `MM[1] ={7'd0,5'd1,5'd1,3'h0,5'd2,7'h33};      // 04   add  x2,x1,x1
3  `MM[2] ={7'd0,5'd1,5'd1,3'h0,5'd3,7'h33};      // 08   add  x3,x1,x1
4  `MM[3] ={7'd0,5'd1,5'd1,3'h0,5'd4,7'h33};      // 0c   add  x4,x1,x1
5  `MM[4] ={7'd0,5'd3,5'd2,3'h0,5'd5,7'h33};      // 10   add  x5,x2,x3
6  `MM[5] ={7'd0,5'd4,5'd5,3'h0,5'd5,7'h33};      // 14   add  x5,x5,x4
7  `MM[6]={12'd0,5'd5,3'h0,5'd10,7'h13};          // 18   addi x10,x5,0
8  `MM[7]=32'h00050f13;                           // 1c   HALT
```

シミュレーションの出力は次の通りです。

```
CC01 00000000          0          3          3
CC02 00000004          3          3          6
CC03 00000008          3          3          6
CC04 0000000c          3          3          6
CC05 00000010          6          6         12
CC06 00000014         12          6         18
CC07 00000018         18          0         18
```

```
CC08 0000001c          18          0          18
```

第 6 章

Q1. add, addi, lw, sw, bne 命令に加えて、and 命令も実行できるように、コード 6-9 の m_proc6 と m_alu の記述を修正してください。次の命令列をシミュレーションして、その表示を示してください。

```
1  `MM[0] ={12'd7, 5'd0,3'h0,5'd1,7'h13};       // 00   addi x1,x0,7
2  `MM[1] ={12'd11,5'd0,3'h0,5'd2,7'h13};       // 04   addi x2,x0,11
3  `MM[2] ={7'd0,5'd2,5'd1,3'b111,5'd10,7'h33}; // 08   and  x10, x1, x2
4  `MM[3] =32'h00050f13;                        // 0c   HALT
```

コード 6-9 の m_proc6 の m_alu の入力ポートを変更します。

```
19    m_alu m8 (P1_ir, w_r1, w_s2, w_alu, w_tkn);
```

また、m_alu の記述を次のように修正します。

```
1  module m_alu(w_ir, w_in1, w_in2, w_out, w_tkn);
2    input  wire [31:0] w_ir, w_in1, w_in2;
3    output wire [31:0] w_out;
4    output wire w_tkn;
5    assign w_out = (w_ir[14:12]==3'b111) ? w_in1 & w_in2 : w_in1 + w_in2;
6    assign w_tkn = w_in1 != w_in2;
7  endmodule
```

シミュレーションの出力は次の通りです。

```
CC1 00000000 00000000          0          0          0
CC2 00000004 00000000          0          7          7
CC3 00000008 00000004          0         11         11
CC4 0000000c 00000008          7         11          3
CC5 00000010 0000000c          3          0          3
```

Q2. add, addi, lw, sw, bne 命令に加えて、and, andi 命令も実行できるように、コード 6-9 の m_proc6 と m_alu のコードを修正してください。次の命令列をシミュレーションして、その表示を示してください。

```
1  `MM[0] ={12'd7, 5'd0,3'h0,5'd1,7'h13};       // 00   addi x1,x0,7
2  `MM[1] ={12'd11,5'd0,3'h0,5'd2,7'h13};       // 04   addi x2,x0,11
```

```
3  `MM[2] ={7'd0,5'd2,5'd1,3'b111,5'd9,7'h33};  // 08   and  x9, x1, x2
2  `MM[3] ={12'd5,5'd9,3'b111,5'd10,7'h13};     // 0c   andi x10,x9,5
4  `MM[4] =32'h00050f13;                        // 10   HALT
```

コードの修正は Q1 と同じで、次の通りです。

コード 6-9 の m_proc6 の m_alu の入力ポートを変更します。

```
19    m_alu m8 (P1_ir, w_r1, w_s2, w_alu, w_tkn);
```

また、m_alu の記述を次のように修正します。

```
1  module m_alu(w_ir, w_in1, w_in2, w_out, w_tkn);
2    input  wire [31:0] w_ir, w_in1, w_in2;
3    output wire [31:0] w_out;
4    output wire w_tkn;
5    assign w_out = (w_ir[14:12]==3'b111) ? w_in1 & w_in2 : w_in1 + w_in2;
6    assign w_tkn = w_in1 != w_in2;
7  endmodule
```

シミュレーションの出力は次の通りです。

```
CC1 00000000 00000000          0          0          0
CC2 00000004 00000000          0          7          7
CC3 00000008 00000004          0         11         11
CC4 0000000c 00000008          7         11          3
CC5 00000010 0000000c          3          5          1
CC6 00000014 00000010          1          0          1
```

Q3. add, addi, lw, sw, bne 命令に加えて、and 命令も実行できるように、コード 6-20 の m_proc8 と m_alu のコードを修正してください。次の命令列をシミュレーションして、その表示を示してください。

```
1  `MM[0] ={12'd7, 5'd0,3'h0,5'd1,7'h13};       // 00   addi x1,x0,7
2  `MM[1] ={12'd11,5'd0,3'h0,5'd2,7'h13};       // 04   addi x2,x0,11
3  `MM[2] ={7'd0,5'd2,5'd1,3'b111,5'd10,7'h33}; // 08   and  x10, x1, x2
4  `MM[3] =32'h00050f13;                        // 0c   HALT
```

コードの修正は Q3 と同じで、次の通りです。

コード 6-20 の m_proc8 の 31 行目の m_alu のインスタンスを生成する部分を次の記述に変更します。

```
31  reg [31:0] P2_ir=0;
32  always @(posedge w_clk) P2_ir <= P1_ir;
33  m_alu m8 (P2_ir, w_in1, w_in2, w_alu, w_tkn);
```

また、m_alu の記述を次のように修正します。

```
1  module m_alu(w_ir, w_in1, w_in2, w_out, w_tkn);
2    input  wire [31:0] w_ir, w_in1, w_in2;
3    output wire [31:0] w_out;
4    output wire w_tkn;
5    assign w_out = (w_ir[14:12]==3'b111) ? w_in1 & w_in2 : w_in1 + w_in2;
6    assign w_tkn = w_in1 != w_in2;
7  endmodule
```

シミュレーションの出力は次の通りです。

```
CC1 00000000 00000000 00000000 00000000          0          0          0
CC2 00000004 00000000 00000000 00000000          0          0          0
CC3 00000008 00000004 00000000 00000000          0          7          7
CC4 0000000c 00000008 00000004 00000000          0         11         11
CC5 00000010 0000000c 00000008 00000004          7         11          3
CC6 00000014 00000010 0000000c 00000008          3          5          1
CC7 00000018 00000014 00000010 0000000c          1          0          1
CC8 0000001c 00000018 00000014 00000010          0          0          0
```

Q4. add, addi, lw, sw, bne 命令に加えて、and, andi 命令も実行できるように、コード 6-20 の m_proc8 と m_alu のコードを修正してください。次の命令列をシミュレーションして、その表示を示してください。

```
1  `MM[0] ={12'd7, 5'd0,3'h0,5'd1,7'h13};       // 00   addi x1,x0,7
2  `MM[1] ={12'd11,5'd0,3'h0,5'd2,7'h13};       // 04   addi x2,x0,11
3  `MM[2] ={7'd0,5'd2,5'd1,3'b111,5'd9,7'h33}; // 08   and  x9, x1, x2
4  `MM[3] ={12'd5,5'd9,3'b111,5'd10,7'h13};     // 0c   andi x10,x9,5
5  `MM[4] =32'h00050f13;                        // 10   HALT
```

コード 6-20 の m_proc8 の 31 行目の m_alu のインスタンスを生成する部分を次の記述に変更します。

```
31  reg [31:0] P2_ir=0;
32  always @(posedge w_clk) P2_ir <= P1_ir;
33  m_alu m8 (P2_ir, w_in1, w_in2, w_alu, w_tkn);
```

また、m_alu の記述を次のように修正します。

```
1  module m_alu(w_ir, w_in1, w_in2, w_out, w_tkn);
2    input  wire [31:0] w_ir, w_in1, w_in2;
3    output wire [31:0] w_out;
4    output wire w_tkn;
5    assign w_out = (w_ir[14:12]==3'b111) ? w_in1 & w_in2 : w_in1 + w_in2;
6    assign w_tkn = w_in1 != w_in2;
7  endmodule
```

シミュレーションの出力は次の通りです。

```
CC1 00000000 00000000 00000000 00000000          0          0          0
CC2 00000004 00000000 00000000 00000000          0          0          0
CC3 00000008 00000004 00000000 00000000          0          7          7
CC4 0000000c 00000008 00000004 00000000          0         11         11
CC5 00000010 0000000c 00000008 00000004          7         11          3
CC6 00000014 00000010 0000000c 00000008          3          0          3
CC7 00000018 00000014 00000010 0000000c          0          0          0
```

第 7 章

Q1. コード 7-11 の記述を修正して、1024 エントリの PHT を持つ bimodal 分岐予測のコードを記述してください。

次の記述になります。

```
 1  module m_bimodal(w_clk, w_radr, w_pred, w_wadr, w_we, w_tkn);
 2    input  wire w_clk, w_we;
 3    input  wire [9:0] w_wadr, w_radr;
 4    input  wire w_tkn;
 5    output wire w_pred;
 6    reg [1:0] mem [0:1023];
 7    wire [1:0] w_data = mem[w_radr];
 8    assign w_pred = w_data[1];
 9    wire [1:0] w_cnt  = mem[w_wadr];
10    always @(posedge w_clk) if (w_we)
11      mem[w_wadr] <= (w_cnt < 3 &  w_tkn) ? w_cnt + 1 :
12                     (w_cnt > 0 & !w_tkn) ? w_cnt - 1 : w_cnt;
13    integer i; initial for (i=0; i<1023; i=i+1) mem[i] = 1;
14  endmodule
```

Q2. コード 7-13 の記述を修正して、1024 エントリの PHT を持つ gshare 分岐予測のコードを記述してください。

次の記述になります。

```
 1  module m_gshare(w_clk, w_radr, w_pred, w_wadr, w_we, w_tkn);
 2    input  wire w_clk, w_we;
 3    input  wire [9:0] w_wadr, w_radr;
 4    input  wire w_tkn;
 5    output wire w_pred;
 6    reg [1:0] mem [0:1023];
 7    reg [9:0] r_bhr = 0;
 8    always @(posedge w_clk) if (w_we) r_bhr <= {w_tkn, r_bhr[9:1]};
 9    wire [1:0] w_data = mem[w_radr ^ r_bhr];
10    assign w_pred = w_data[1];
11    wire [1:0] w_cnt  = mem[w_wadr ^ r_bhr];
12    always @(posedge w_clk) if (w_we)
```

```
13      mem[w_wadr ^ r_bhr] <= (w_cnt < 3 &  w_tkn) ? w_cnt + 1 :
14                             (w_cnt > 0 & !w_tkn) ? w_cnt - 1 : w_cnt;
15    integer i; initial for (i=0; i<1023; i=i+1) mem[i] = 1;
16  endmodule
```

Q3. bimodal 分岐予測と gshare 分岐予測で用いる 2 ビットの飽和型カウンタを 3 ビットの飽和型カウンタに変更することで予測の精度が向上することがあります。3 ビットの飽和型カウンタは、その値が 7 より小さくて分岐結果が成立の時に値を 1 だけ増やし、値が 0 より大きくて分岐結果が不成立の時に値を 1 だけ減らします。カウンタの値が 3 より大きい時に成立と予測します。コード 7-13 の記述を修正して、32 エントリの PHT を持ち、3 ビットの飽和型カウンタを採用する gshare 分岐予測を記述してください。

次の記述になります。

```
 1  module m_gshare(w_clk, w_radr, w_pred, w_wadr, w_we, w_tkn);
 2    input  wire w_clk, w_we;
 3    input  wire [4:0] w_wadr, w_radr;
 4    input  wire w_tkn;
 5    output wire w_pred;
 6    reg [2:0] mem [0:31];
 7    reg [4:0] r_bhr = 0;
 8    always @(posedge w_clk) if (w_we) r_bhr <= {w_tkn, r_bhr[4:1]};
 9    wire [2:0] w_data = mem[w_radr ^ r_bhr];
10    assign w_pred = w_data[2];
11    wire [2:0] w_cnt  = mem[w_wadr ^ r_bhr];
12    always @(posedge w_clk) if (w_we)
13      mem[w_wadr ^ r_bhr] <= (w_cnt < 7 &  w_tkn) ? w_cnt + 1 :
14                             (w_cnt > 0 & !w_tkn) ? w_cnt - 1 : w_cnt;
15    integer i; initial for (i=0; i<32; i=i+1) mem[i] = 3;
16  endmodule
```

第 8 章

Q1. コード 8-10 の記述を修正して、8 バイトのブロックを採用する 32 エントリのダイレクトマップ方式のキャッシュのコードを記述してください。

次の記述になります。

```
1  module m_cache4(w_clk, w_adr, w_hit, w_dout, w_wadr, w_we, w_wd);
2    input  wire w_clk, w_we;
3    input  wire [31:0] w_adr;
4    input  wire [4:0] w_wadr;
5    input  wire [151:0] w_wd;
6    output wire w_hit;
7    output wire [31:0] w_dout;
8    wire w_v;
9    wire [22:0] w_tag;
10   wire [31:0] w_d1, w_d2, w_d3, w_d4;
11   reg [151:0] mem [0:31];  // 1 + 23 + 128 = 152
12   always @(posedge w_clk) if (w_we) mem[w_wadr] <= w_wd;
13   assign {w_v, w_tag, w_d1, w_d2, w_d3, w_d4} = mem[w_adr[8:4]];
14   assign w_hit = w_v & (w_tag==w_adr[31:9]);
15   assign w_dout = (w_adr[3:2]==3) ? w_d1 :
16                   (w_adr[3:2]==2) ? w_d2 :
17                   (w_adr[3:2]==1) ? w_d3 : w_d4;
18   integer i; initial for (i=0; i<32; i=i+1) mem[i] = 0;
19 endmodule
```

Q2. コード 8-10 の記述を修正して、16 バイトのブロックを採用する 32 エントリのダイレクトマップ方式のキャッシュのコードを記述してください。

次の記述になります。

```
1  module m_cache5(w_clk, w_adr, w_hit, w_dout, w_wadr, w_we, w_wd);
2    input  wire w_clk, w_we;
3    input  wire [31:0] w_adr;
4    input  wire [4:0] w_wadr;
5    input  wire [278:0] w_wd;
6    output wire w_hit;
7    output wire [31:0] w_dout;
```

```
 8    wire w_v;
 9    wire [21:0] w_tag;
10    wire [31:0] w_d1, w_d2, w_d3, w_d4, w_d5, w_d6, w_d7_w_d8;
11    reg [278:0] mem [0:31];  // 1 + 22 + 256 = 279
12    always @(posedge w_clk) if (w_we) mem[w_wadr] <= w_wd;
13    assign {w_v, w_tag, w_d1, w_d2, w_d3,
14            w_d4, w_d5, w_d6, w_d7, w_d8} = mem[w_adr[9:5]];
15    assign w_hit = w_v & (w_tag==w_adr[31:10]);
16    assign w_dout = (w_adr[4:2]==7) ? w_d1 :
17                    (w_adr[4:2]==6) ? w_d2 :
18                    (w_adr[4:2]==5) ? w_d3 :
19                    (w_adr[4:2]==4) ? w_d4 :
20                    (w_adr[4:2]==3) ? w_d5 :
21                    (w_adr[4:2]==2) ? w_d6 :
22                    (w_adr[4:2]==1) ? w_d7 : w_d8;
23    integer i; initial for (i=0; i<32; i=i+1) mem[i] = 0;
24  endmodule
```

Q3. コード 8-12 の記述を修正して、8 バイトのブロックを採用する 32 エントリの 2 ウェイのセットアソシアティブ方式のキャッシュのコードを記述してください。

次の記述になります。

```
 1  module m_cache6(w_clk, w_adr, w_hit, w_dout, w_wadr, w_we, w_wd);
 2    input  wire w_clk, w_we;
 3    input  wire [31:0] w_adr;
 4    input  wire [4:0] w_wadr;
 5    input  wire [88:0] w_wd;
 6    output wire w_hit;
 7    output wire [31:0] w_dout;
 8    wire w_hit1, w_hit2, w_we1, w_we2;
 9    wire [31:0] w_dout1, w_dout2;
10    m_cache2 way1 (w_clk, w_adr, w_hit1, w_dout1, w_wadr, w_we1, w_wd);
11    m_cache2 way2 (w_clk, w_adr, w_hit2, w_dout2, w_wadr, w_we2, w_wd);
12    assign w_hit = w_hit1 | w_hit2;
13    assign w_dout = (w_hit1) ? w_dout1 : w_dout2;
14    reg [0:0] lru [0:31];
15    always @(posedge w_clk) if (w_hit) lru[w_adr[7:3]] <= w_hit1;
16    assign w_we1 = (w_we & lru[w_wadr]==0);
17    assign w_we2 = (w_we & lru[w_wadr]==1);
18    integer i; initial for (i=0; i<32; i=i+1) lru[i] = 0;
19  endmodule
```

Q4. コード 8-12 の記述を修正して、4 バイトのブロックを採用する 32 エントリの 3 ウェイのセットアソシアティブ方式のキャッシュのコードを記述してください。置き換えアルゴリズムは LRU を利用してください。

例えば、次の記述になります。

```
 1  module m_cache7(w_clk, w_adr, w_hit, w_dout, w_wadr, w_we, w_wd);
 2    input  wire w_clk, w_we;
 3    input  wire [31:0] w_adr;
 4    input  wire [4:0] w_wadr;
 5    input  wire [57:0] w_wd;
 6    output wire w_hit;
 7    output wire [31:0] w_dout;
 8    wire w_hit1, w_hit2, w_hit3, w_we1, w_we2, w_we3;
 9    wire [31:0] w_dout1, w_dout2, w_dout3;
10    m_cache1 way1 (w_clk, w_adr, w_hit1, w_dout1, w_wadr, w_we1, w_wd);
11    m_cache1 way2 (w_clk, w_adr, w_hit2, w_dout2, w_wadr, w_we2, w_wd);
12    m_cache1 way3 (w_clk, w_adr, w_hit3, w_dout3, w_wadr, w_we3, w_wd);
13    assign w_hit = w_hit1 | w_hit2 | w_hit3;
14    assign w_dout = (w_hit1) ? w_dout1 : (w_hit2) ? w_dout2 : w_dout3;
15    reg [1:0] lru1 [0:31];  // least recently used
16    reg [1:0] lru2 [0:31];  // 2nd least recently used
17    reg [1:0] lru3 [0:31];  // most recently used
18    wire [1:0] w_hitway = (w_hit1) ? 1 : (w_hit2) ? 2 : 3;
19    always @(posedge w_clk) if (w_hit) begin
20      if (lru1[w_adr[6:2]]==w_hitway) begin  // rotation
21         lru1[w_adr[6:2]] <= lru2[w_adr[6:2]];
22         lru2[w_adr[6:2]] <= lru3[w_adr[6:2]];
23         lru3[w_adr[6:2]] <= lru1[w_adr[6:2]];
24      end
25      else if (lru2[w_adr[6:2]]==w_hitway) begin  // swap
26         lru2[w_adr[6:2]] <= lru3[w_adr[6:2]];
27         lru3[w_adr[6:2]] <= lru2[w_adr[6:2]];
28      end
29    end
30    assign w_we1 = (w_we & lru1[w_wadr]==1);
31    assign w_we2 = (w_we & lru1[w_wadr]==2);
32    assign w_we3 = (w_we & lru1[w_wadr]==3);
33    integer i; initial for (i=0; i<32; i=i+1) begin
34      lru1[i] = 3; lru2[i] = 2; lru3[i] = 1;
35    end
36  endmodule
```