2. 조합 논리 회로 설계

2.1 4비트 비교기 설계

- 비교기 : 두 개의 입력을 서로 비교하여 그 결과를 알려주는 회로.

1) 일치와 반일치

입력신호		출력신호	
A	B	X(일치)	Y(반일치)
0	0	1	0
0	1	0	1
1	0	0	1
1	1	1	0

논리식 : $X = \overline{A}\,\overline{B} + AB = \overline{A \oplus B}$
$Y = \overline{A}\,B + A\,\overline{B} = A \oplus B$

```
module COMPARE_1(A, B, X, Y);
    input A, B ;
    output X, Y;
    assign X = (A~^B);
    assign Y = A^B;
endmodule
```

2) 1비트 비교기

입력신호		출력신호		
A	B	X	Y	Z
0	0	0	0	1
0	1	0	1	0
1	0	1	0	0
1	1	0	0	1

논리식

$X = A\overline{B},$
$Y = \overline{A}B,$
$Z = \overline{A}\,\overline{B} + AB$

```
module COMPARE_1b(A, B, X, Y, Z);
    input A, B ;
    output X, Y, Z;
    assign X = A & !B;
    assign Y = !A & B;
    assign Z = (!A & !B)|(A & B);
endmodule
```

3) 4비트 비교기

- 두 수의 MSB를 먼저 비교하여 크거나 작다면 다음의 비교는 필요 없음. 만약 같다면, 다음 bit를 비교.

예)	MSB				LSB
A =	1	1	0	0	1
B =	0	1	1	1	0

※ 1비트 n단 비교기 진리표

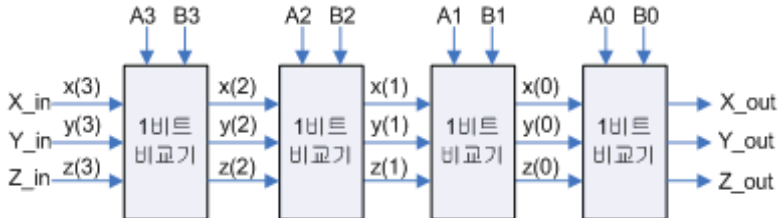
입력신호					출력신호		
X_in(>)	Y_in(<)	Z_in(=)	A	B	X_out(>)	Y_out(<)	Z_out(=)
0	0	1	0	0	0	0	1
0	0	1	0	1	0	1	0
0	0	1	1	0	1	0	0
0	0	1	1	1	0	0	1
0	1	0	X	X	0	1	0
1	0	0	X	X	1	0	0

논리식

$$X_{out} = X_{in} + Z_{in}A\overline{B}$$
$$Y_{out} = Y_{in} + Z_{in}\overline{A}B$$
$$Z_{out} = Z_{in}\overline{A}\,\overline{B} + Z_{in}AB$$
$$= Z_{in}(\overline{A}\,\overline{B} + AB)$$
$$= Z_{in}\overline{(A\overline{B} + \overline{A}B)}$$

```
module COMPARE_1bn(An, Bn, Xin, Yin, Zin, Xout, Yout, Zout);
    input An, Bn, Xin, Yin, Zin ;
    output Xout, Yout, Zout;
    assign Xout = Xin |( Zin & (An & ~Bn));
    assign Yout = Yin |(Zin & (~An & Bn));
    assign Zout = Zin &(An ~^ Bn);
endmodule
```

※ 4비트 비교기

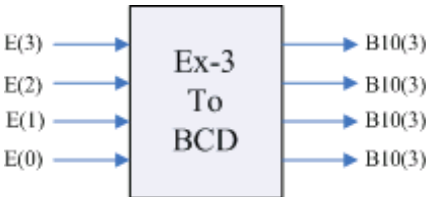



```
module COMPARE_4b(A, B, Xin, Yin, Zin, Xout, Yout, Zout);
    input [3:0] A, B;
    input Xin, Yin, Zin ;
    output Xout, Yout, Zout;

    COMPARE_1bn U0 (A[3], B[3],  Xin, Yin, Zin, X3, Y3, Z3);
    COMPARE_1bn U1 (A[2], B[2],  X3, Y3, Z3, X2, Y2, Z2);
    COMPARE_1bn U2 (A[1], B[1],  X2, Y2, Z2, X1, Y1, Z1);
    COMPARE_1bn U3 (A[0], B[0],  X1, Y1, Z1, Xout, Yout, Zout);
endmodule
```

2.2 코드 변환 회로 설계

- BCD(binary coded decimal) 코드 : 10진 부호로 0~9까지의 2진 부호.
- 3초과 부호(Excess-3 code) : BCD 코드에 0011을 더한 부호.
- Gray code : 0~15를 2진 부호로 변환할 때, 1비트씩 변하도록 한 부호.

1) (Ex-3 코드)-(BCD 코드) 변환기 설계



논리식

$$B10(3) = E(3)E(2) + E(3)E(1)E(0)$$
$$B10(2) = \overline{E(2)}\,\overline{E(1)} + \overline{E(2)}\,\overline{E(0)} + E(2)E(1)E(0)$$
$$B10(1) = \overline{E(1)}E(0) + E(1)\overline{E(0)}$$
$$B10(0) = \overline{E(0)}$$

Ex-3 코드				BCD 코드			
E(3)	E(2)	E(1)	E(0)	B10(3)	B10(2)	B10(1)	B10(0)
0	0	1	1	0	0	0	0
0	1	0	0	0	0	0	1
0	1	0	1	0	0	1	0
0	1	1	0	0	0	1	1
0	1	1	1	0	1	0	0
1	0	0	0	0	1	0	1
1	0	0	1	0	1	1	0
1	0	1	0	0	1	1	1
1	0	1	1	1	0	0	0
1	1	0	0	1	0	0	1

don't care : d(E(3),E(2),E(1),E(0)) = $\sum m(0,1,2,13,14,15)$

```
module EX3_BCD(E, B10);
  input [3:0] E;
  output [3:0] B10;
  assign B10[3]=(E[3] & E[2]) | (E[3] & E[1] & E[0]);
  assign B10[2]=(~E[2]& ~E[1]) | (~E[2] & ~E[0]) | (E[2] & E[1] & E[0]);
  assign B10[1]=(~E[1] & E[0]) | (E[1] & ~E[0]);
  assign B10[0]=~E[0];
endmodule
```

```
module EX3_BCD_1(E, B10);
  input [3:0] E;
  output [3:0] B10;
  reg [3:0] B10;
 always @(E)
  begin
   if (E==4'h3) B10=4'h0;
   else if (E==4'h4) B10=4'h1;
   else if (E==4'h5) B10=4'h2;
   else if (E==4'h6) B10=4'h3;
   else if (E==4'h7) B10=4'h4;
   else if (E==4'h8) B10=4'h5;
   else if (E==4'h9) B10=4'h6;
   else if (E==4'ha) B10=4'h7;
   else if (E==4'hb) B10=4'h8;
   else if (E==4'hc) B10=4'h9;
  end
endmodule
```

```
module EX3_BCD_2(E, B10);
  input [3:0] E;
  output [3:0] B10;
  reg [3:0] B10;
 always @(E)
    case(E)
     3 : B10 = 4'h0;
     4 : B10 = 4'h1;
     5 : B10 = 4'h2;
     6 : B10 = 4'h3;
     7 : B10 = 4'h4;
     8 : B10 = 4'h5;
     9 : B10 = 4'h6;
     10 : B10 = 4'h7;
     11 : B10 = 4'h8;
     12 : B10 = 4'h9;
    endcase
endmodule
```

```
module EX3_BCD_3(E, B10);
  input [3:0] E;
  output [3:0] B10;
  assign B10=(E==4'h3)? 4'h0:
             (E==4'h4)? 4'h1:
             (E==4'h5)? 4'h2:
             (E==4'h6)? 4'h3:
             (E==4'h7)? 4'h4:
             (E==4'h8)? 4'h5:
             (E==4'h9)? 4'h6:
             (E==4'hA)? 4'h7:
             (E==4'hB)? 4'h8:
             (E==4'hC)? 4'h9:
                        4'hF;
  endmodule
```

```
module EX3_BCD_4(E, B10);
  input [3:0] E;
  output [3:0] B10;
  assign B10 = F_EX3_BCD (E);
function [3:0] F_EX3_BCD;
  input [3:0] A;
  begin
    case(A)
      4'h3 : F_EX3_BCD = 4'h0;
      4'h4 : F_EX3_BCD = 4'h1;
      4'h5 : F_EX3_BCD = 4'h2;
      4'h6 : F_EX3_BCD = 4'h3;
      4'h7 : F_EX3_BCD = 4'h4;
      4'h8 : F_EX3_BCD = 4'h5;
      4'h9 : F_EX3_BCD = 4'h6;
      4'hA : F_EX3_BCD = 4'h7;
      4'hB : F_EX3_BCD = 4'h8;
      4'hC : F_EX3_BCD = 4'h9;
    endcase
  end
  endfunction
endmodule
```

```
module EX3_BCD_4(E, B10);
  input [3:0] E;
  output [3:0] B10;
  always @(E)
  T_EX3_BCD(E, B10);
task T_EX3_BCD;
  input [3:0] E;
  output [3:0] B10;
  begin
    case(E)
    4'h3 : F_EX3_BCD = 4'h0;
    4'h4 : F_EX3_BCD = 4'h1;
    4'h5 : F_EX3_BCD = 4'h2;
    4'h6 : F_EX3_BCD = 4'h3;
    4'h7 : F_EX3_BCD = 4'h4;
    4'h8 : F_EX3_BCD = 4'h5;
    4'h9 : F_EX3_BCD = 4'h6;
    4'hA : F_EX3_BCD = 4'h7;
    4'hB : F_EX3_BCD = 4'h8;
    4'hC : F_EX3_BCD = 4'h9;
    endcase
  end
endtask
endmodule
```

2) (BCD 코드)-(Ex-3 코드) 변환기 설계

BCD 코드				Ex-3 코드			
D	C	B	A	W	X	Y	Z
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0

$$W = D + CB + CA$$
$$= D + C(B + A)$$

$$X = \overline{C}B + \overline{C}A + C\overline{B}\,\overline{A}$$
$$= \overline{C}(B + A) + C\overline{(B + A)}$$
$$= C \oplus (B + A)$$

$$Y = \overline{B}\,\overline{A} + BA = \overline{B \oplus A}$$

$$Z = \overline{A}$$

don't care : d(D, C, B, A) = $\sum m(10,11,12,13,14,15)$

```
module BCD_EX3(D, C, B, A, W, X, Y, Z);
  input D, C, B, A;
  output W, X, Y, Z;
  assign W=D|(C&(B|A));
  assign X=C^(B|A);
  assign Y=B~^A;
  assign Z=~A;
endmodule
```

```
module BCD_EX3_2(B10,E);
   input [3:0] B10;
   output [3:0] E;
   assign E=B10+4'h3;
endmodule
```

```
module BCD_EX3_1(D, C, B, A, W, X, Y, Z);
   input D,C,B,A;
   output W,X,Y,Z;
   reg [3:0] E;
 always @(D|C|B|A)
  case({D,C,B,A})
    0 : E = 4'h3;
    1 : E = 4'h4;
    2 : E = 4'h5;
    3 : E = 4'h6;
    4 : E = 4'h7;
    5 : E = 4'h8;
    6 : E = 4'h9;
    7 : E = 4'hA;
    8 : E = 4'hB;
    9 : E = 4'hC;
  endcase
 assign W=E[3],X=E[2], Y=E[1], Z=E[0];
endmodule
```

2.3 패리티 비트 회로 설계 : 통신에서 에러 검출, 홀수(odd)/짝수(even) 패리티

1) 패리티 비트 발생기

- 홀수 패리티 : 입력 데이터에서 '1'의 개수가 홀수가 되도록 함.
- 짝수 패리티 : '1'의 개수가 짝수가 되도록 함.

입력 데이터			패리티 비트	
A	B	C	홀수	짝수
0	0	0	1	0
0	0	1	0	1
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	1	0
1	1	1	0	1

홀수 패리티 발생기의 논리식

$$PB\_O = \overline{A}\,\overline{B}\,\overline{C} + AB\overline{C} + \overline{A}BC + A\overline{B}C$$
$$= \overline{A \oplus B \oplus C}$$

짝수 패리티 발생기의 논리식

$$PB\_E = \overline{A}\,\overline{B}C + \overline{A}B\overline{C} + A\overline{B}\,\overline{C} + ABC$$
$$= A \oplus B \oplus C$$

```
module parity_even(A,B,C,PB_E, PB_O);
   input A, B, C;
   output PB_E, PB_O ;
   assign PB_E = A^B^C;
   assign PB_O = ~(A^B^C);
endmodule
```

```
module parity_even_1(A,B,C,PB_E, PB_O);
   input A, B, C;
   output PB_E, PB_O ;
   reg PB_E, PB_O;
   always@(A|B|C)
    case ({A,B,C})
     0 : begin PB_O = 1'b1; PB_E = 1'b0; end
     1 : begin PB_O = 1'b0; PB_E = 1'b1; end
     2 : begin PB_O = 1'b0; PB_E = 1'b1; end
     3 : begin PB_O = 1'b1; PB_E = 1'b0; end
     4 : begin PB_O = 1'b0; PB_E = 1'b1; end
     5 : begin PB_O = 1'b1; PB_E = 1'b0; end
     6 : begin PB_O = 1'b1; PB_E = 1'b0; end
     7 : begin PB_O = 1'b0; PB_E = 1'b1; end
    endcase
endmodule
```

2) 패리티 체커

- 짝수 패리티 체커 : 입력 data와 패리티 비트를 함께 수신하여 '1'의 개수가 홀수이면 에러가 발생, 짝수이면 에러 없이 수신한 것임.
- 홀수 패리티 체커 : '1'의 개수가 짝수이면 에러가 발생, 홀수이면 OK.

입력 data (수신 data)			패리티 체커 (에러 판정)	
A	B	P (패리티 bit)	홀수 (PC_O)	짝수 (PC_E)
0	0	0	1	0
0	0	1	0	1
0	0	0	0	1
0	0	1	1	0

논리식

$$PC\_O = \overline{A \oplus B \oplus C}$$

$$PC\_E = A \oplus B \oplus C$$

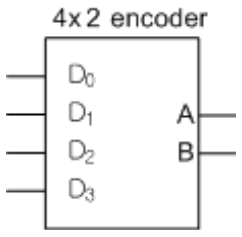
```
module parity_chek(A,B,P,PC_E, PC_O);
    input A, B, P;
    output PC_E, PC_O ;

    assign PC_E = A^B^P;
    assign PC_O = ~(A^B^P);

 endmodule
```

```
module parity_chek_1(A,B,P,PC_E, PC_O);
    input A, B, P;
    output PC_E, PC_O ;
    reg PC_E, PC_O;
    always @(A|B|P)
     case ({A,B,P})
       0 : begin PC_O = 1'b1; PC_E = 1'b0; end
       1 : begin PC_O = 1'b0; PC_E = 1'b1; end
       2 : begin PC_O = 1'b0; PC_E = 1'b1; end
       3 : begin PC_O = 1'b1; PC_E = 1'b0; end
       4 : begin PC_O = 1'b0; PC_E = 1'b1; end
       5 : begin PC_O = 1'b1; PC_E = 1'b0; end
       6 : begin PC_O = 1'b1; PC_E = 1'b0; end
       7 : begin PC_O = 1'b0; PC_E = 1'b1; end
     endcase
endmodule
```

2.4 인코더 설계

- 인코더 : 입력을 특정의 부호로 변환(부호화기), 최대 2^n 입력-> n개 출력

1) 4X2 인코더



입력				출력	
D_0	D_1	D_2	D_3	Y1	Y0
1	0	0	0	0	0
0	1	0	0	0	1
0	0	1	0	1	0
0	0	0	1	1	1

논리식

$$Y1 = D2 + D3$$

$$Y0 = D1 + D3$$

```
module encoder_42(D, Y);
    input [3:0] D;
    output [1:0] Y ;
    assign Y[0]= (D[1] | D[3]);
    assign Y[1]= (D[2] | D[3]);
endmodule
```

```
module encoder_42_1(D, Y);
   input [3:0] D;
   output [1:0] Y;
   assign Y=D[3] ? 2'b11 :
             D[2] ? 2'b10 :
             D[1] ? 2'b01 :
             D[0] ? 2'b00 : 2'bxx;  // xx-> zz
endmodule
```

```
module encoder_42_2(D, Y);
    input [3:0] D;
    output [1:0] Y;
    reg [1:0] Y;
  always @(D)
   case(D)
     8 : Y = 2'h3;
     4 : Y = 2'h2;
     2 : Y = 2'h1;
     1 : Y = 2'h0;
   endcase
 endmodule
```

```
 module encoder_42_3(D, Y);
    input [3:0] D;
    output [1:0] Y;
    reg [1:0] Y;
  always @(D)
   case(D)
     4'b1000 : Y = 2'h3;
     4'b0100 : Y = 2'h2;
     4'b0010 : Y = 2'h1;
     4'b0001 : Y = 2'h0;
   endcase
 endmodule
```

```
 module encoder_42_4(D, Y);
    input [3:0] D;
    output [1:0] Y;
    reg [1:0] Y;
  always @(D)
  begin
  if(D[3])       Y=2'h3;
  else if(D[2]) Y=2'h2;
  else if(D[1]) Y=2'h1;
  else if(D[0]) Y=2'h0;
  end
 endmodule
```

2) 인에이블이 있는 4X2 인코더

입력					출력	
E	D0	D1	D2	D3	Y1	Y0
0	X	X	X	X	0	0
1	1	0	0	0	0	0
1	0	1	0	0	0	1
1	0	0	1	0	1	0
1	0	0	0	1	1	1

논리식

$$Y1 = E(D2 + D3)$$

$$Y0 = E(D1 + D3)$$

```
module encoder_42_E(E, D, Y);
   input [3:0] D;
   input E;
   output [1:0] Y ;

   assign Y[0]= E&(D[1] | D[3]);
   assign Y[1]= E&(D[2] | D[3]);

 endmodule
```

```
module encoder_42_e_1(E, D, Y);
    input [3:0] D;
    input E;
    output [1:0] Y;
    reg [1:0] Y;
  always @(D)
  begin
  if(E&D[3])       Y=2'h3;
  else if(E&D[2]) Y=2'h2;
  else if(E&D[1]) Y=2'h1;
  else if(E&D[0]) Y=2'h0;
  end
endmodule
```

3) 우선순위가 있는 4X2 인코더

입력				출력	
D0	D1	D2	D3	Y1	Y0
1	0	0	0	0	0
X	1	0	0	0	1
X	X	1	0	1	0
X	X	X	1	1	1

```
module encoder_42_Pri(D, Y, Pri);
    input [3:0] D;
    output Pri;
    output [1:0] Y;
    reg [1:0] Y;
    reg Pri;
 always @(D)
 begin
  Pri =1;
  casex(D)
  4'b1xxx : Y=3;
  4'b01xx : Y=2;
  4'b001x : Y=1;
  4'b0001 : Y=0;
  default :  begin Pri=0; Y=2'bxx; end
  endcase
 end
 endmodule
```

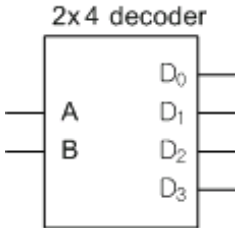
```
module encoder_42_Pri_1(D, Y, Pri);
    input [3:0] D;
    output Pri;
    output [1:0] Y;
    reg [1:0] Y;
    reg Pri;
    integer k;

 always @(D)
 begin
  Y=2'bx;
  Pri=0;
 for(k=0;k<4;k=k+1)
  if(D[k])
    begin Y=k; Pri=1; end
 end
 endmodule
```

4) 8X3 인코더 : 스스로

2.5 디코더 설계

- 디코더 : code화된 신호를 해석하는 회로, n개 입력 -> 최대 2^n 출력.

1) 2X4 디코더



입력		출력			
A	B	D_0	D_1	D_2	D_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

논리식

$D0 = \overline{B}\,\overline{A}$,
$D1 = \overline{B}A$,
$D2 = B\overline{A}$,
$D3 = BA$

```
module decoder_24_2(A,B,D);
    input  A, B;
    output [3:0] D;
    reg [3:0] D;
    always @(B|A)
    begin
      if({B,A}==2'h3) D=4'h8;
      else if({B,A}==2'h2) D=4'h4;
      else if({B,A}==2'h1) D=4'h2;
      else if({B,A}==2'h0) D=4'h1;
    end
 endmodule
```

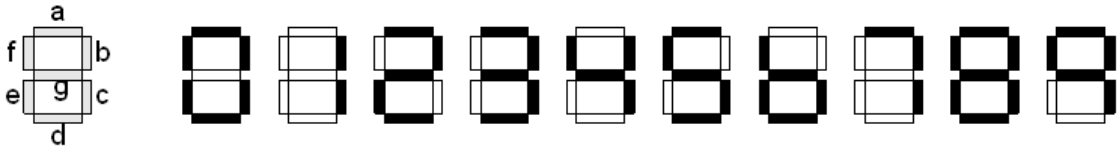
```
module decoder_24_3(A,B,D);
    input  A, B;
    output [3:0] D;
    reg [3:0] D;
    always @(B|A)
        case({B,A})
        3 :  D=4'h8;
        2 :  D=4'h4;
        1 :  D=4'h2;
        0 :  D=4'h1;
     endcase
 endmodule
```

```
module decoder_24(A,B,D);
    input  A, B;
    output [3:0] D;
    assign D[0]= ~B & ~A;
    assign D[1]= ~B & A ;
    assign D[2]= B & ~A ;
    assign D[3]= B & A ;
endmodule
```

```
module decoder_24_1(A,B,D);
    input  A, B;
    output [3:0] D;
    assign D[0]= ({B,A}==2'h0);
    assign D[1]= ({B,A}==2'h1);
    assign D[2]= ({B,A}==2'h2);
    assign D[3]= ({B,A}==2'h3);
endmodule
```

※ 3X8 디코더/BCD-10진 디코더 설계 : 스스로

2) BCD-7세그먼트 디코더



입력				출력						
D	C	B	A	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1

논리식

$a = D + B + \overline{C \oplus A}$

$b = \overline{C \cdot (B \oplus A)}$

$c = C + \overline{B} + A$

$d = D + \overline{C}\,\overline{A} + \overline{C}B + B\overline{A} + C\overline{B}A$

$e = \overline{C}\,\overline{A} + B\overline{A}$

$f = D + \overline{B}\,\overline{A} + C\overline{B} + C\overline{A}$

$g = D + (C \oplus B) + C\overline{A}$

don't care : d(D, C, B, A) = $\sum m(10,11,12,13,14,15)$

```
module decoder_7_SEG(A, B, C, D, sa, sb, sc, sd, se, sf, sg);
    input  D,C,B,A;
    output sa,sb,sc,sd,se,sf,sg;
    assign sa = D | B |(C ~^ A);
    assign sb= ~(C &( B^A ));
    assign sc= C | ~B | A;
    assign sd= D|(~C & ~A)|(~C & B)|(B & ~A)|(C & ~B & A);
    assign se= (~C & ~A)|(B & ~A);
    assign sf= D|(~B & ~A)|(C & ~B)|(C & ~A);
    assign sg= D|( C^B )|(C & ~A);
endmodule
```

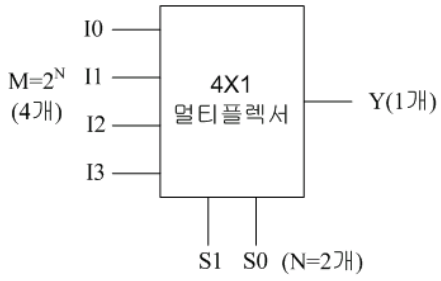
```
module decoder_7_SEG_1(A, B, C, D, sa, sb, sc, sd, se, sf, sg);
    input   D,C,B,A;
    output sa,sb,sc,sd,se,sf,sg;
    reg [6:0] SO;
    always @(D|C|B|A)
      casex({D,C,B,A})
       0 :  SO=7'b111_1110;
       1 :  SO=7'b011_0000;
       2 :  SO=7'b110_1101;
       3 :  SO=7'b111_1001;
       4 :  SO=7'b011_0011;
       5 :  SO=7'b101_1011;
       6 :  SO=7'b101_1111;
       7 :  SO=7'b111_0000;
       8 :  SO=7'b111_1111;
       9 :  SO=7'b111_1011;
     endcase
    assign sa=SO[6], sb=SO[5], sc=SO[4], sd=SO[3], se=SO[2], sf=SO[1], sg=SO[0];
endmodule
```

※ (2진 코드)-(7세그먼트) 디코더 : 스스로
(0000~1111) => (0~9, A, b, C, d, E, F)

2.6 멀티플렉서와 디멀티플렉서 설계 (다중화기, 데이터 선택기)

- 멀티플렉서 : 여러 개의 입력 신호 중 하나를 선택하여 출력에 전달.

1) 4X1 MUX



입력신호				선택 신호		출력 신호
I0	I1	I2	I3	S1	S0	Y
X	X	X	X	0	0	I0
X	X	X	X	0	1	I1
X	X	X	X	1	0	I2
X	X	X	X	1	1	I3

여기서 X는 '0' 또는 '1'

논리식 : $Y = \overline{S1}\,\overline{S0}\,I0 + \overline{S1}\,S0\,I1 + S1\,\overline{S0}\,I2 + S1\,S0\,I3$

```
module mux_41_1(I,S, Y);
    input [3:0] I;
    input [1:0] S;
    output Y;
    assign Y = S[1]? (S[0] ? I[3] : I[2]) : (S[0] ? I[1] : I[0]);
 endmodule
```

```
module mux_41(I, S, Y);
    input [3:0] I;
    input [1:0] S;
    output Y;
    assign Y =(~S[1]& ~S[0] & I[0]) |(~S[1]& S[0] & I[1]) |
              (S[1]& ~S[0] & I[2]) | (S[1]& S[0] & I[3]);
 endmodule
```

```
module mux_41_2(I, S, Y);
    input [3:0] I;
    input [1:0] S;
    output Y;
    reg Y;
    always@(I or S)
     if(S[1]==0)
       if(S[0]==0) Y=I[0]; else Y=I[1];
     else
       if(S[0]==0) Y=I[2]; else Y=I[3];
endmodule
```

```
module mux_41_4(I, S, Y);
    input [3:0] I;
    input [1:0] S;
    output Y;
    reg Y;
    always @(S or I)
    case(S)
     0 : Y=I[0];
     1 : Y=I[1];
     2 : Y=I[2];
     3 : Y=I[3];
    endcase
endmodule
```

```
module mux_41_3(I, S, Y);
    input [3:0] I;
    input [1:0] S;
    output Y;
     assign Y=(S==0)? I[0]:
              (S==1)? I[1]:
              (S==2)? I[2]:
              (S==3)? I[3]:
                      1'bx;
endmodule
```

```
module mux_41_5(I, S, Y);
    input [3:0] I;
    input [1:0] S;
    output Y;
    wire  [1:0] M;
    mux_21 U0 (I[1:0],S[0],M[0]);
    mux_21 U1 (I[3:2],S[0],M[1]);
    mux_21 U2 (M[1:0],S[1],Y);
endmodule
```

※ 8X1 멀티플렉서 : 스스로

2) 1X4 DEMUX

선택신호		입력신호	출력신호			
S1	S0	I	D0	D1	D2	D3
0	0	X	I	0	0	0
0	1	X	0	I	0	0
1	0	X	0	0	I	0
1	1	X	0	0	0	I

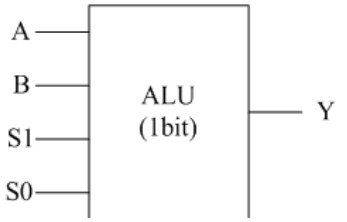
```
module DEMUX_14(I,S, Y);
    input  I;
    input [1:0] S;
    output [3:0] Y;
    assign Y[0]= I&(~S[1] & ~S[0]);
    assign Y[1]= I&(~S[1] & S[0]);
    assign Y[2]= I&(S[1] & ~S[0]);
    assign Y[3]= I&(S[1] & S[0]);
endmodule
```

```
module demux_14_1(I, S, Y);
    input  I;
    input [1:0] S;
    output[3:0] Y;
    reg [3:0] Y;
    always @(I or S)
    case(S)
     3 : Y[3]=I;
     2 : Y[2]=I;
     1 : Y[1]=I;
     0 : Y[0]=I;
    endcase
endmodule
```

2.7 ALU 설계

- 산술논리 연산장치(arithmetic logic unit)는 CPU의 핵심 요소로서 설계하는 ALU는 모두 조합회로로 구성됨.

1) 4가지 기능을 가진 ALU 설계(1비트)



S1	S0	논리식	기능
0	0	Y = A nand B	NAND
0	1	Y = A and B	AND
1	0	Y = A or B	OR
1	1	Y = A xor B	XOR

```
module ALU_4(A, B, S, Y);
    input A,B;
    input [1:0] S;
    output Y;
    reg Y;

    always @(A or B or S)
    case(S)
     0 : Y= ~(A & B);
     1 : Y= A & B;
     2 : Y= A | B;
     3 : Y= A^B;
    endcase
endmodule
```

2) 8가지 기능을 가진 ALU 설계(4비트) - 동영상 숙제