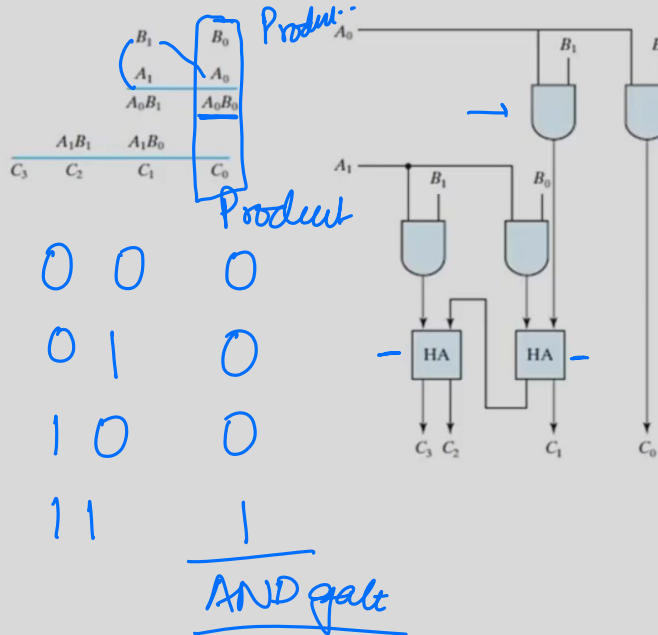


Binary Multiplier



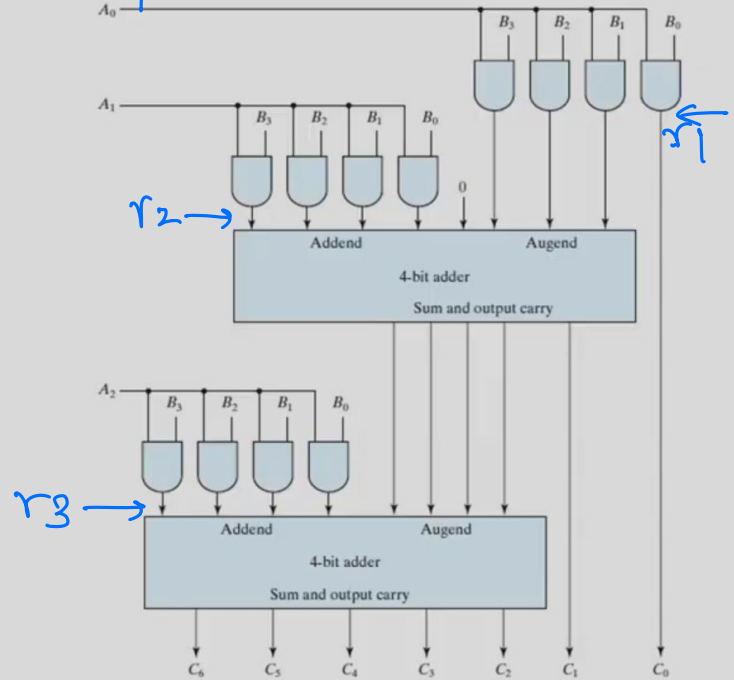
- A bit of the multiplier is ANDed with each bit of the multiplicand in as many levels as there are bits in the multiplier.
- The binary output in each level of AND gates is added with the partial product of the previous level to form a new partial product.
- The last level produces the product.
- For J multiplier bits and K multiplicand bits, we need $J * K$ AND gates and $(J - 1) * K$ bit adders to produce a product of $(J + K)$ bits.

Binary Multiplier

A_2 A_1 A_0

B_3 B_2 B_1 B_0

Carry look ahead-



```
module product(a,b,res);  
  input [3:0] b;  
  input [2:0] a;  
  output [6:0] res;
```

```
  wire [4:0] r1;  
  wire [3:0] r2,r3;  
  wire [4:0] i1;
```

```
  assign r1[0]=b[0]&a[0];  
  assign r1[1]=b[1]&a[0];  
  assign r1[2]=b[2]&a[0];  
  assign r1[3]=b[3]&a[0];
```

```
  assign r2[0]=b[0]&a[1];  
  assign r2[1]=b[1]&a[1];  
  assign r2[2]=b[2]&a[1];  
  assign r2[3]=b[3]&a[1];
```

```
  assign r3[0]=b[0]&a[2];  
  assign r3[1]=b[1]&a[2];  
  assign r3[2]=b[2]&a[2];  
  assign r3[3]=b[3]&a[2];
```

```
  assign r1[4]=1'b0;
```

```
  assign res[0]=r1[0];
```

```
  Carry ← CLA_Adder a1(r1[4:1], r2, 1'b0, i1[3:0], i1[4]);
```

```
  assign res[1]=i1[0];
```

```
  CLA_Adder a2(i1[4:1], r3, 1'b0, res[5:2], res[6]);
```

```
endmodule
```

hard wired logic

```

module testbench ( );
// Inputs
reg [2:0] a;
reg [3:0] b;
// Outputs
wire [6:0] out;
// Instantiate the Design
Under Test (DUT)
product dut (
.a(a),
.b(b),
.res(out)
);

// Clock and reset signals
reg clk;
reg rst;

// Initial values
initial begin
    clk = 0;
    rst = 1;
    #20;
    rst = 0;
end

```

```

// Clock generator
always begin
    #10 clk = ~clk;
end

// Test sequence
reg [3:0] i;
always @(posedge clk, posedge rst) begin
    if (rst) begin
        i = 0;
    end else begin
        // Apply all possible val values (0-9)
        a <= $random%8;
        b <= $random%16;
        #20;
        i = i + 1;
        if (i == 10) $finish;
    end
    $monitor("%d: a = %b, b = %b, res = %b", $time, a, b, out);
end
endmodule

```

sequence

```

10: a = xxx, b = xxxx, res = xxxxxxxx
30: a = 100, b = 0001, res = 0000100
50: a = 001, b = 0011, res = 0000011
70: a = 101, b = 1101, res = 1000001
90: a = 101, b = 0010, res = 0001010
110: a = 001, b = 1101, res = 0001101
130: a = 110, b = 1101, res = 1001110
150: a = 101, b = 1100, res = 0111100
170: a = 001, b = 0110, res = 0000110
190: a = 101, b = 1010, res = 0110010
210: a = 101, b = 0111, res = 0100011

```

Loop (i -- till 0)

Sub-task →



Magnitude Comparator Logic

$$A = A_3A_2A_1A_0$$

$$B = B_3B_2B_1B_0$$

why not use
XOR
↓
XNOR.

$$x_i = A_i'B_i' + A_iB_i$$

for x_i .

$$A = B$$

$$x_3x_2x_1x_0$$

$$A > B$$

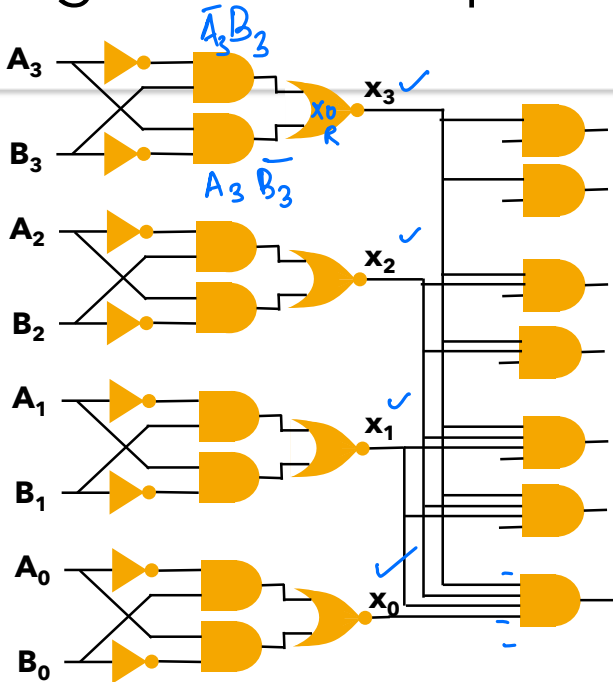
$$A_3B_3' + x_3A_2B_2' + x_3x_2A_1B_1' + x_3x_2x_1A_0B_0'$$

$$A < B$$

$$A_3'B_3 + x_3A_2'B_2 + x_3x_2A_1'B_1 + x_3x_2x_1A_0'B_0$$

Similarity in expressions for the 3 comparisons

Magnitude Comparator Implementation



A > B

$$A_3B_3' + x_3A_2B_2' + x_3x_2A_1B_1' + x_3x_2x_1A_0B_0'$$



A < B

$$A_3'B_3 + x_3A_2'B_2 + x_3x_2A_1'B_1 + x_3x_2x_1A_0'B_0$$

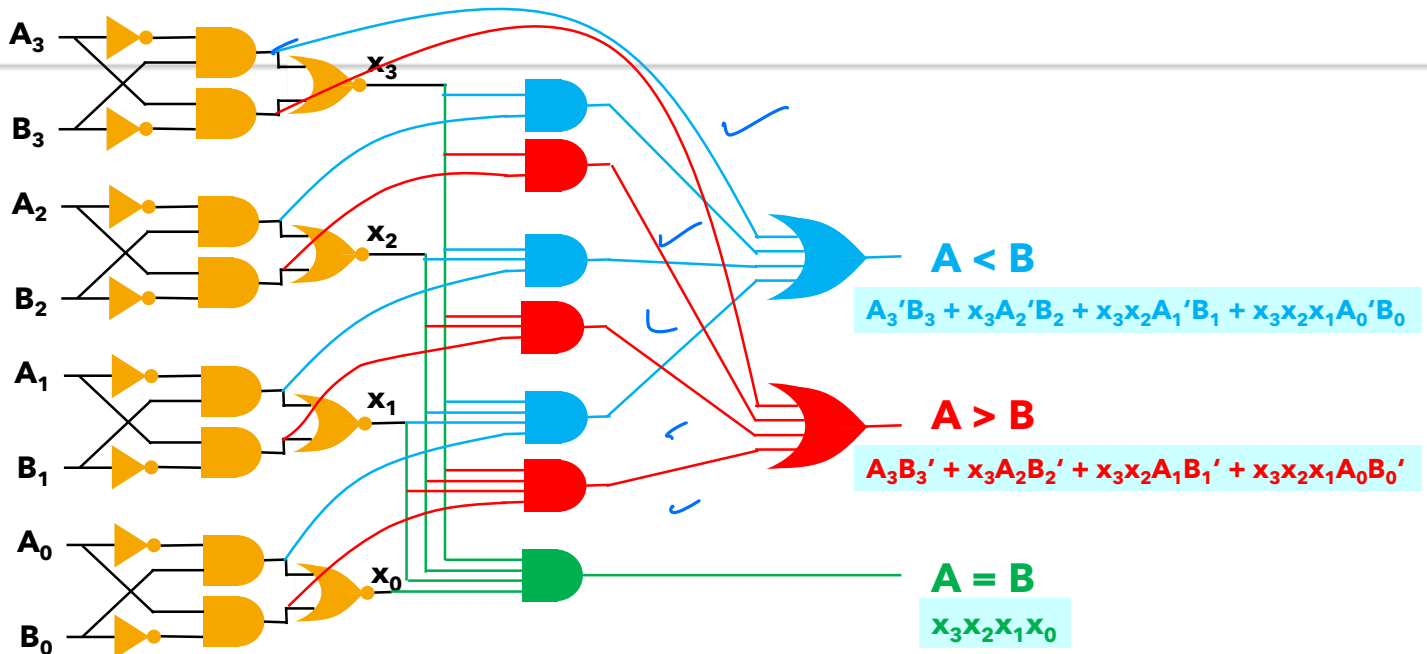
A = B

$$x_3x_2x_1x_0$$

gn $\pm/\text{ue}$

Subtraction logic $\rightarrow$ Out put.

Magnitude Comparator Implementation



Decoder

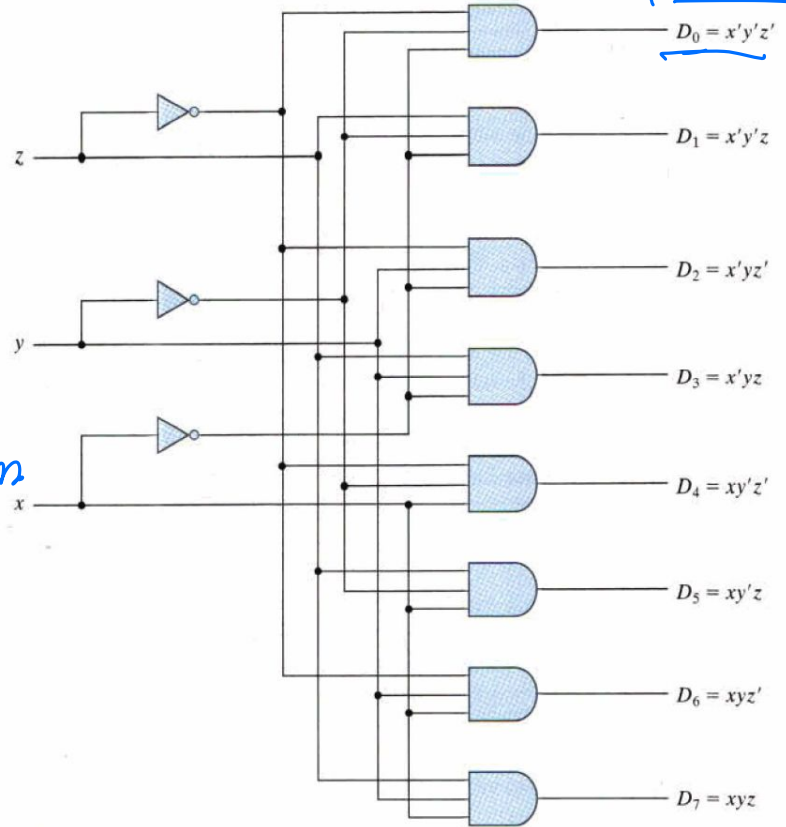
program

↳ CPU.



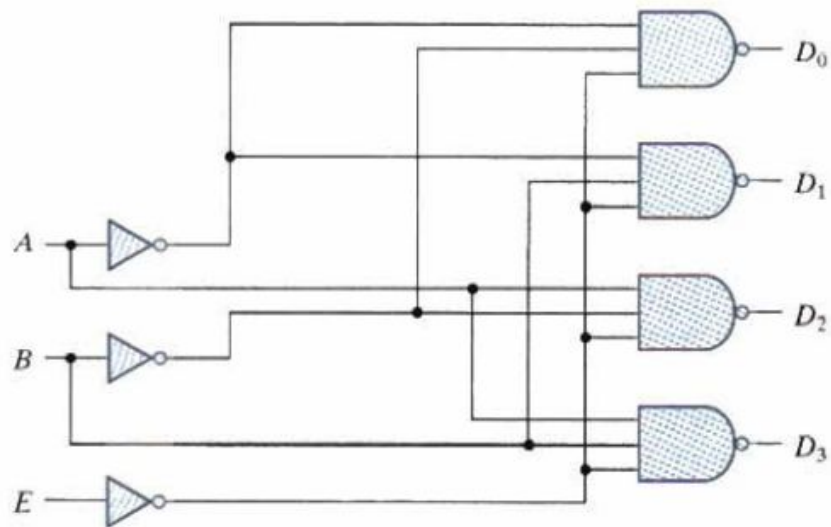
32 bit address

2^{32} memory locations



Truth Table of a Three-to-Eight-Line Decoder

Inputs			Outputs							
<i>x</i>	<i>y</i>	<i>z</i>	<i>D</i> ₀	<i>D</i> ₁	<i>D</i> ₂	<i>D</i> ₃	<i>D</i> ₄	<i>D</i> ₅	<i>D</i> ₆	<i>D</i> ₇
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1



(a) Logic diagram

E	A	B	D_0	D_1	D_2	D_3
1	X	X	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0

(b) Truth table

FIGURE 4.19
Two-to-four-line decoder with enable input

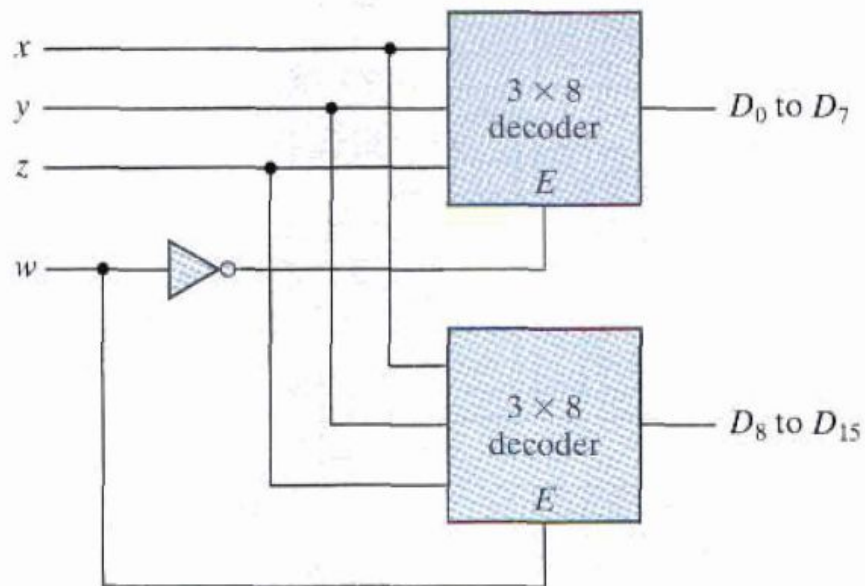


FIGURE 4.20

4 x 16 decoder constructed with two 3 x 8 decoders

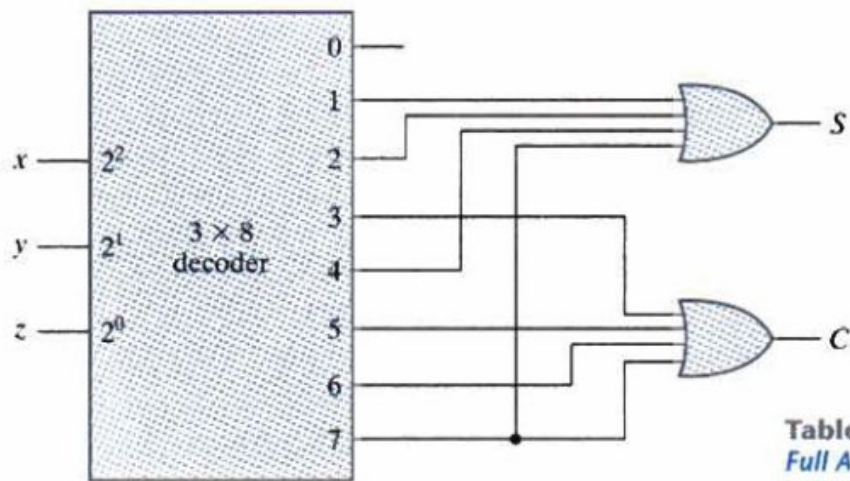


FIGURE 4.21
Implementation of a full adder with a decoder

$$S(x, y, z) = \Sigma(1, 2, 4, 7)$$

$$C(x, y, z) = \Sigma(3, 5, 6, 7)$$

Table 4.4
Full Adder

x	y	z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1