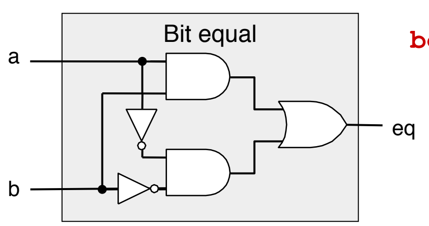
Logic Design III

CSci 2021: Machine Architecture and Organization

With Slides from: Stephen McCamand, Ahmad Almulhem, and Hai Zhou

Bit Equality

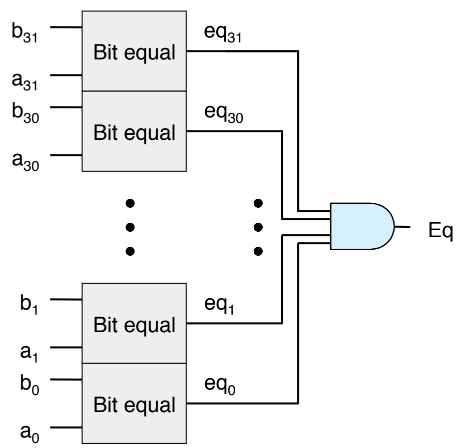



```
bool eq = (a&&b)||(!a&&!b)
```

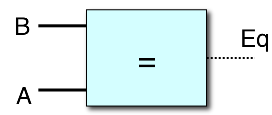
Generate 1 if a and b are equal

a	b	eq
0	0	1
0	1	0
1	0	0
1	1	1

Word Equality



Word-Level Representation

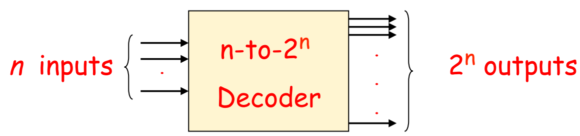



```
bool Eq = (A == B)
```

32-bit word size

Decoder

Decoder

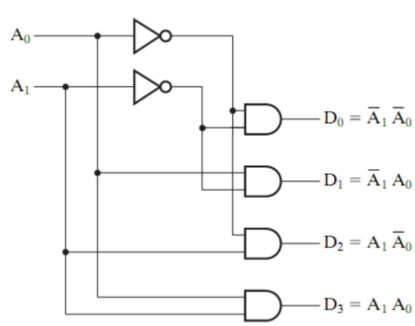


- Information is represented by binary codes
- **Decoding** - the conversion of an *n*-bit input code to an *m*-bit output code with $n <= m <= 2^n$ such that each valid code word produces a unique output code
- Circuits that perform decoding are called **decoders**

2-to-4 Decoder

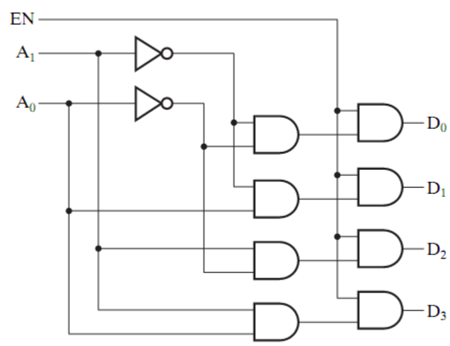
- A 2-to-4 Decoder
  - 2 inputs (A_1, A_0)
  - 2^2 = 4 outputs (D_3, D_2, D_1, D_0)
  - **Truth Table**

A_1	A_0	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

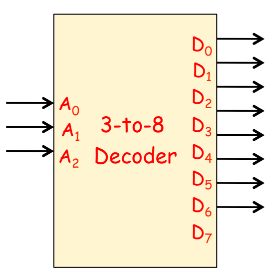


2-to-4 Decoder with Enable

EN	A_1	A_0	D_0	D_1	D_2	D_3
0	X	X	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

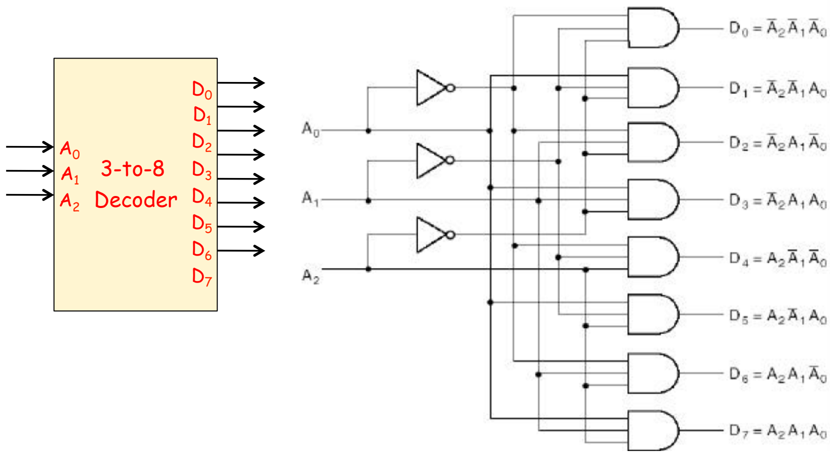


3-to-8 Decoder



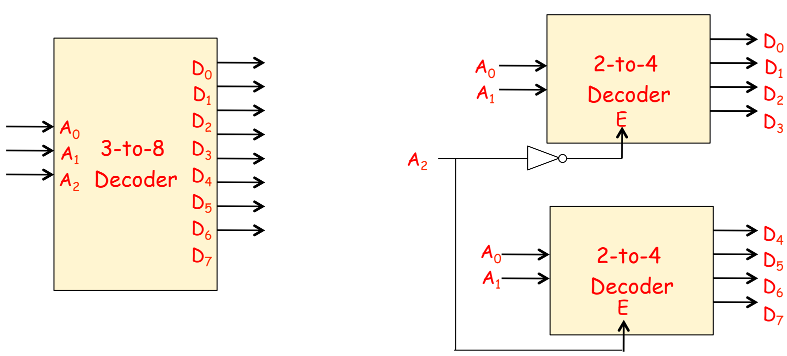
A_2	A_1	A_0	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

3-to-8 Decoder



KFUPM

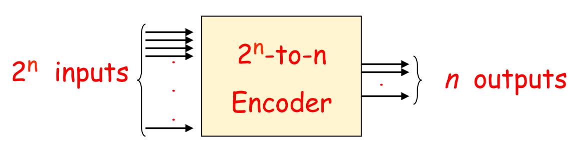
3-to-8 Decoder (using 2 2-to-4 decoders)



KFUPM

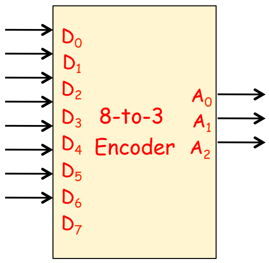
Encoder

Encoder



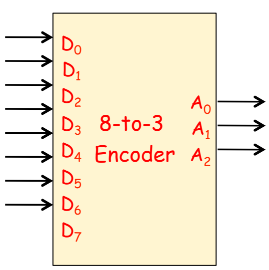
- **Encoding** - the opposite of decoding - the conversion of an *m*-bit input code to a *n*-bit output code with $n \le m \le 2^n$ such that each valid code word produces a unique output code
- Circuits that perform encoding are called **encoders**
- An encoder has 2^n (or fewer) input lines and *n* output lines which generate the binary code corresponding to the input values
- Typically, an encoder converts a code containing exactly one bit that is 1 to a binary code corresponding to the position in which the 1 appears.

8-to-3 Encoder



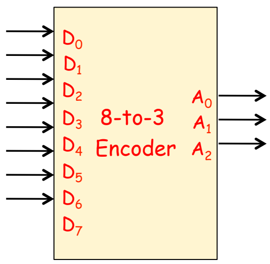
- **Description:**
- 2^3 = 8 inputs, 3 outputs
- one input =1, others = 0's
- Each input generate unique binary code

8-to-3 Encoder (truth table)



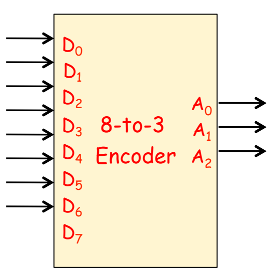
inputs								outputs		
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	A_2	A_1	A_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

8-to-3 Encoder



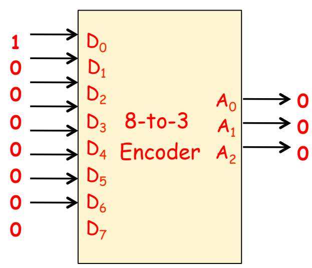
- **Description:**
- 2^3 = 8 inputs, 3 outputs
- one input =1, others = 0's
- Each input generate unique binary code

8-to-3 Encoder (truth table)



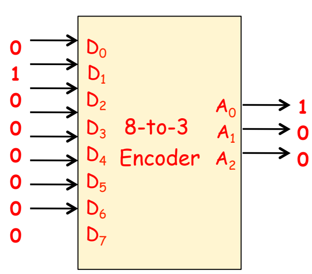
inputs								outputs		
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	A_2	A_1	A_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

8-to-3 Encoder (truth table)



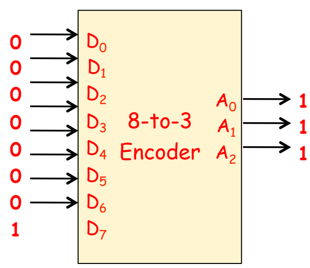
inputs								outputs		
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	A_2	A_1	A_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

8-to-3 Encoder (truth table)



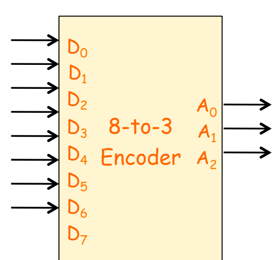
inputs								outputs		
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	A_2	A_1	A_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

8-to-3 Encoder (truth table)



inputs								outputs		
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	A_2	A_1	A_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

8-to-3 Encoder (equations)



inputs								outputs		
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	A_2	A_1	A_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

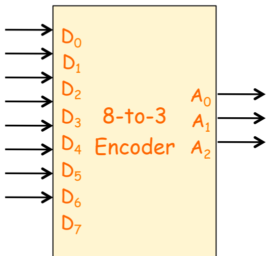
Output equations:

$A_0 = D_1 + D_3 + D_5 + D_7$

$A_1 = ?$

$A_2 = ?$

8-to-3 Encoder (equations)



inputs								outputs		
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	A_2	A_1	A_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

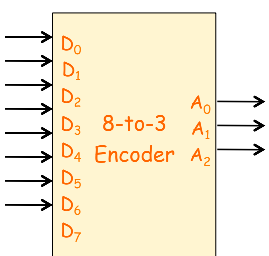
Output equations:

$A_0 = D_1 + D_3 + D_5 + D_7$

$A_1 = D_2 + D_3 + D_6 + D_7$

$A_2 = ?$

8-to-3 Encoder (equations)



inputs								outputs		
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	A_2	A_1	A_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

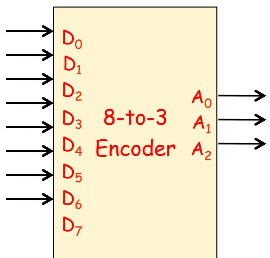
Output equations:

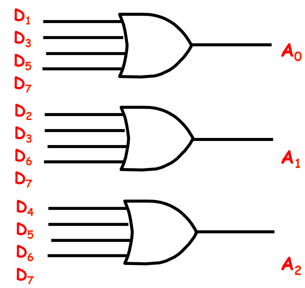
$A_0 = D_1 + D_3 + D_5 + D_7$

$A_1 = D_2 + D_3 + D_6 + D_7$

$A_2 = D_4 + D_5 + D_6 + D_7$

8-to-3 Encoder (circuit)





Output equations:

$A_0 = D_1 + D_3 + D_5 + D_7$
$A_1 = D_2 + D_3 + D_6 + D_7$
$A_2 = D_4 + D_5 + D_6 + D_7$

8-to-3 Encoder (limitations)

Two Limitations:

1. Two or more inputs = 1
   - Example: D_3 = D[illegible]= 1
   - $A_2A_1A_0$ = 111
2. All inputs = 0
   - Same as D_0 =1

Output equations:

$A_0 = D_1 + D_3 + D_5 + D_7$
$A_1 = D_2 + D_3 + D_6 + D_7$
$A_2 = D_4 + D_5 + D_6 + D_7$

inputs								outputs		
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	A_2	A_1	A_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

Priority Encoder

- Address the previous two limitations

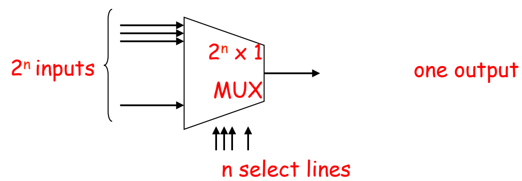
1. Two or more inputs = 1
   - Consider the bit with highest priority
2. All inputs = 0
   - Add another output v to indicate this combination

Take Home Assignment

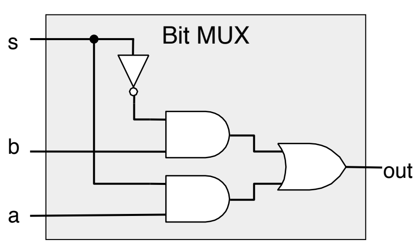
Multiplexer

Multiplexers

- Is a combinational circuit
- Has a single output
- Directs one of 2^n input to the output
- Input to output direction is done based on a set of n select bits



Bit-Level Multiplexor

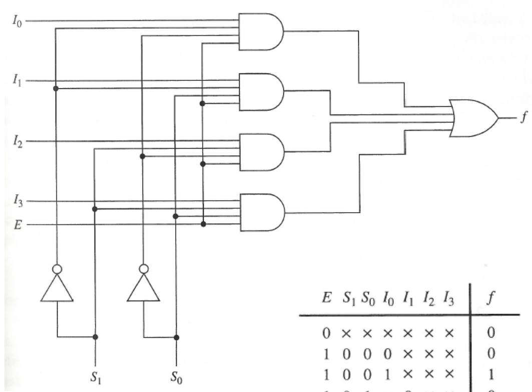


- Control signal s
- Data signals a and b
- Output a when s=1, b when s=0

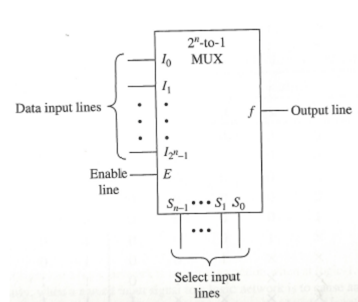
```
bool out = (s&&a)||(!s&&b)
```

a	b	s	out
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Realization of 4-to-1 line multiplexer with enable



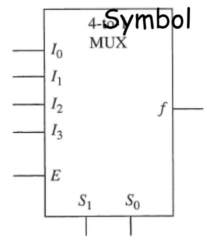
Logic Diagram



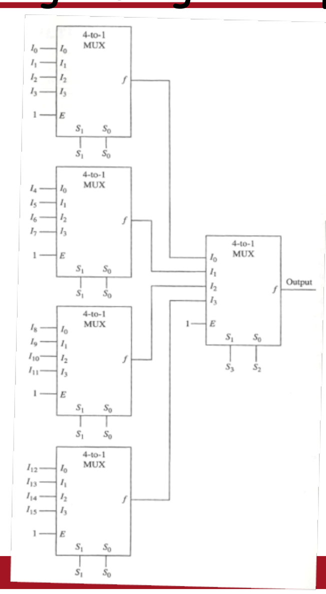
E	S_1	S_0	I_0	I_1	I_2	I_3	f
0	×	×	×	×	×	×	0
1	0	0	0	×	×	×	0
1	0	0	1	×	×	×	1
1	0	1	×	0	×	×	0
1	0	1	×	1	×	×	1
1	1	0	×	×	0	×	0
1	1	0	×	×	1	×	1
1	1	1	×	×	×	0	0
1	1	1	×	×	×	1	1

Truth Table

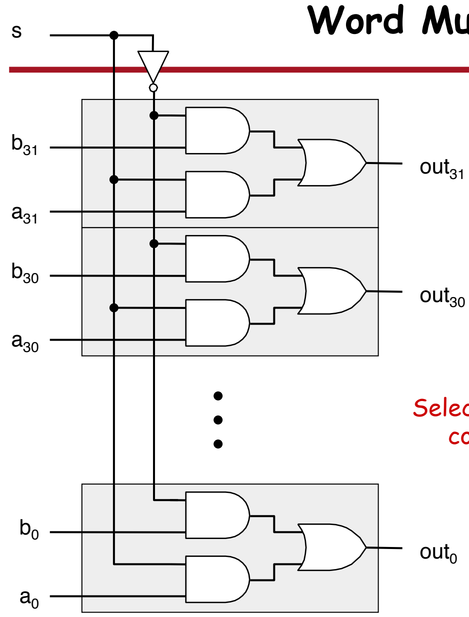
Symbol



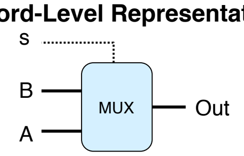
Building a Large Multiplexer



Word Multiplexor



Word-Level Representation

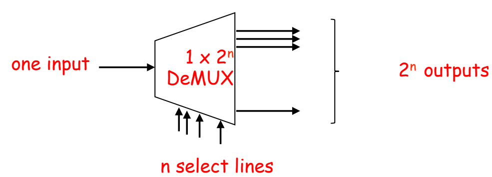


Select input word A or B depending on control signal s

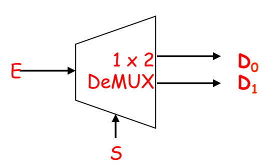
DeMultiplexer

DeMultiplexer

- Performs the inverse operation of a MUX
- It has one input and 2^n outputs
- The input is passed to one of the outputs based on the n select line



1x2 DeMUX

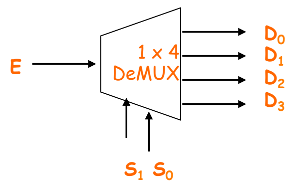


The circuit has an input E, the outputs are given by:

D_0 = E, if S=0 $\qquad$ $D_0 = S' E$

D_1 = E, if S=1 $\qquad$ $D_1 = S E$

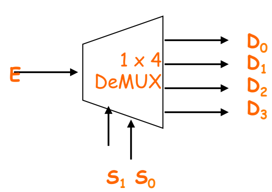
1x4 DeMUX

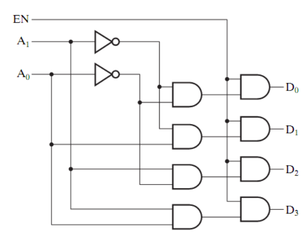


The circuit has an input E, the outputs are given by:

D_0 = E, if S_0S_1=00	$D_0 = S_1'S_0'\ E$
D_1 = E, if S_0S_1=01	$D_1 = S_1'S_0\ E$
D_2 = E, if S_0S_1=10	$D_2 = S_1S_0'\ E$
D_3 = E, if S_0S_1=11	$D_3 = S_1S_0\ E$

DeMUX vs Decoder





- A 1x4 DeMUX is equivalent to a 2x4 Decoder with an Enable
  - Think of S_1S_0 a the decoder's input
  - Think of E as the decoder's enable
- In general, a DeMux is equivalent to a Decoder with an Enable

EN /E	A_1 S_1	A_0 S_0	D_0	D_1	D_2	D_3
0	X	X	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

Adder

Adder

Carry In →	0	0	0	0	1	1	1	1
Operand A →	0	0	1	1	0	0	1	1
Operand B →	+ 0	+ 1	+ 0	+ 1	+ 0	+ 1	+ 0	+ 1
	0 0	0 1	0 1	1 0	0 1	1 0	1 0	1 1

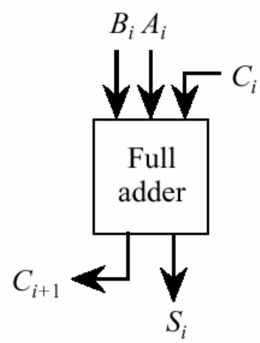
Carry Out ↑ ↑ Sum

Example:

Carry		1	0	0	0
Operand A		0	1	0	0
Operand B	+	0	1	1	0
Sum		1	0	1	0

Full Adder

A_i	B_i	C_i	S_i	C_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



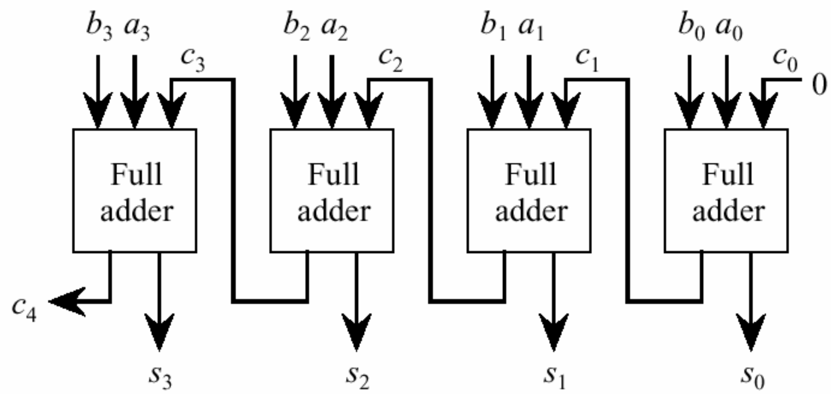
Carry-In	A	B	Sum	Carry-out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Sum = A'BC' + AB'C' + A'B'C + ABC

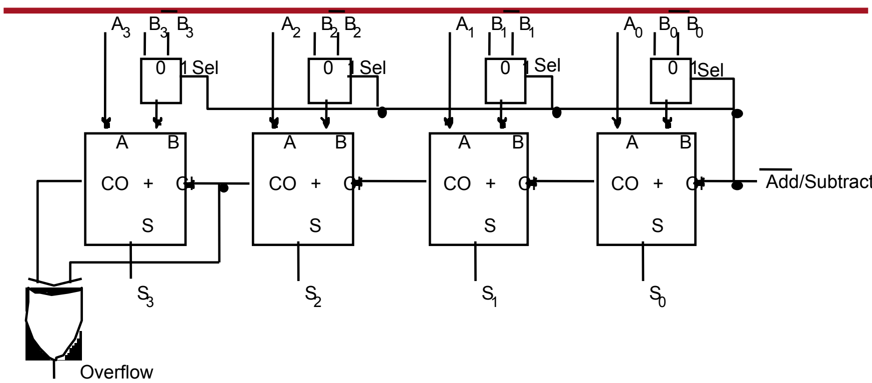
Carry-out = ABC' + A'BC + AB'C + ABC

Four-Bit Ripple-Carry Adder

- **Four full adders connected in a ripple-carry chain form a four-bit ripple-carry adder.**



Adder/Subtractor



$$A - B = A + (-B) = A + \overline{B} + 1$$

Carry Lookahead Logic

Carry Generate Gi = Ai Bi *must generate carry when A = B = 1*

Carry Propagate Pi = Ai xor Bi *carry in will equal carry out here*

Sum and Carry can be reexpressed in terms of generate/propagate:

Si = Ai xor Bi xor Ci = Pi xor Ci

Ci+1 = Ai Bi + Ai Ci + Bi Ci

= Ai Bi + Ci (Ai + Bi)

= Ai Bi + Ci (Ai xor Bi)

= Gi + Ci Pi

Carry Lookahead Logic

Reexpress the carry logic as follows:

C1 = G0 + P0 C0

C2 = G1 + P1 C1 = G1 + P1 G0 + P1 P0 C0

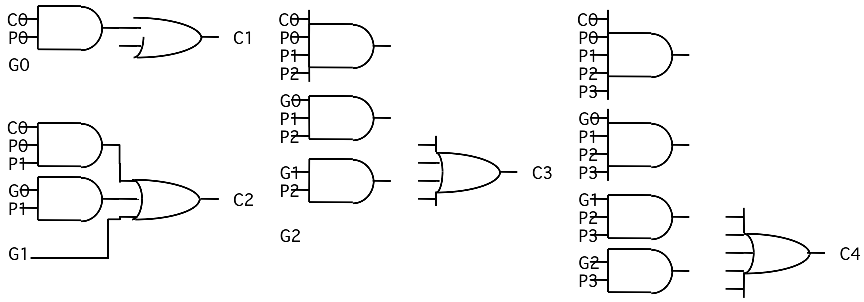
C3 = G2 + P2 C2 = G2 + P2 G1 + P2 P1 G0 + P2 P1 P0 C0

C4 = G3 + P3 C3 = G3 + P3 G2 + P3 P2 G1 + P3 P2 P1 G0 + P3 P2 P1 P0 C0

Each of the carry equations can be implemented in a two-level logic network

Variables are the adder inputs and carry in to stage 0!

Carry Lookahead Implementation



Multiplier

Theory of Multiplication

Basic Concept

```
multiplicand        1101   (13)
multiplier       *  1011   (11)
                 ----------
                     1101
                    1101
Partial products   0000
                  1101
                 ----------
                 10001111    (143)
```

product of 2 4-bit numbers is an 8-bit number

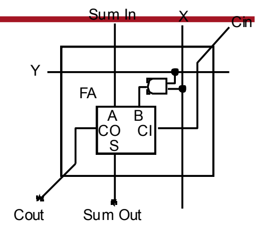
Combinational Multiplier

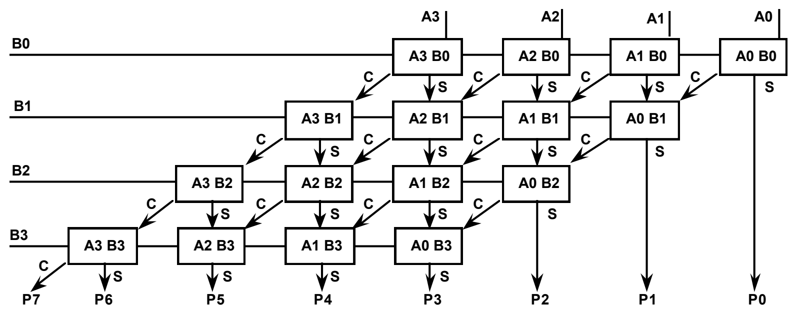
Partial Product Accumulation

				A3	A2	A1	A0
				B3	B2	B1	B0
				A2 B0	A2 B0	A1 B0	A0 B0
			A3 B1	A2 B1	A1 B1	A0 B1	
		A3 B2	A2 B2	A1 B2	A0 B2		
	A3 B3	A2 B3	A1 B3	A0 B3			
S7	S6	S5	S4	S3	S2	S1	S0

Combinational Multiplier

Building block: full adder + and





4 x 4 array of building blocks