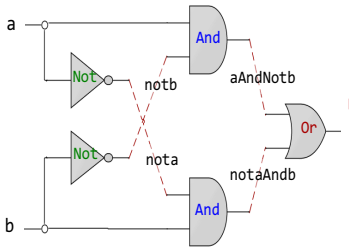
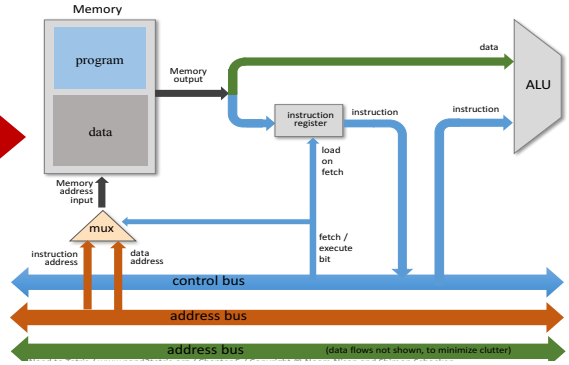
Digital Logic Design




```
CHIP Xor {
  IN a, b;
  OUT out;
  PARTS:
  Not(in=a, out=nota);
  Not(in=b, out=notb);
  And(a=nota, b=b, out=w1);
  And(a=a, b=notb, out=w2);
  Or(a=w1, b=w2, out=out);
}
```




```
@R1
D=M
@temp
M=D
```

```
0000000000000001
1111110000010000
0000000000010000
1110001100001000
```

Lecture # 11-12

Decoder, Encoder, Mux and DeMux

```
#include<stdio.h>
#include<stdlib.h>
int main(){
  printf("Learning is fun with Arif\n");
  exit(0);
}
```

```
global main
SECTION .data
   msg: db "Learning is fun with Arif", 0Ah, 0h
   len_msg: equ $ - msg
SECTION .text
   main:
      mov rax,1
      mov rdi,1
      mov rsi,msg
      mov rdx,len_msg
      syscall
      mov rax,60
      mov rdi,0
      syscall
```

```
0:  b8 01 00 00 00
5:  bf 01 00 00 00
a:  48 be 00 00 00 00 00
11: 00 00 00
14: ba 1b 00 00 00
19: 0f 05
1b: b8 3c 00 00 00
20: bf 00 00 00 00
25: 0f 05
```

Slides of first half of the course are adapted from:
https://www.nand2tetris.org
Download s/w tools required for first half of the course from the following link:
https://drive.google.com/file/d/0B9c0BdDJz6XpZUh3X2dPR1o0MUE/view

Instructor: Muhammad Arif Butt, Ph.D.

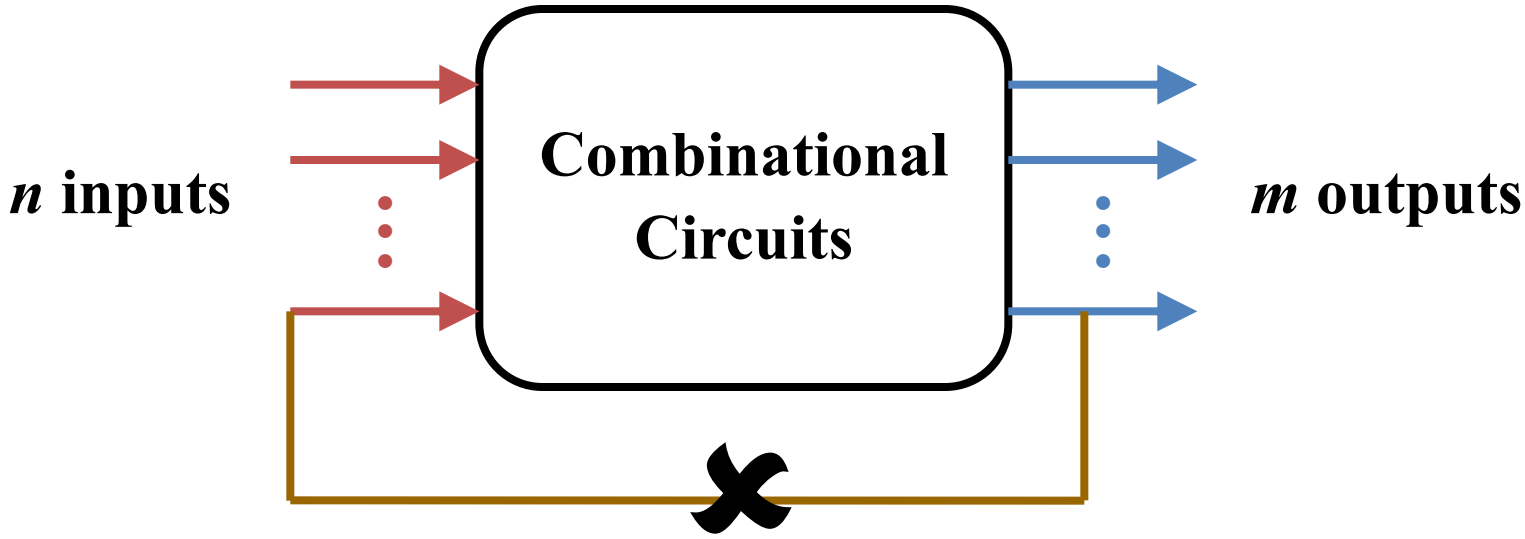
Today's Agenda

- Writing HDL for Combinational Circuits like
  - Decoder
  - Encoder
  - Multiplexer
  - De-Multiplexer
- Demo on above chips on H/W Simulator

Combinational Circuits

- Output is function of input only

  i.e. no feedback

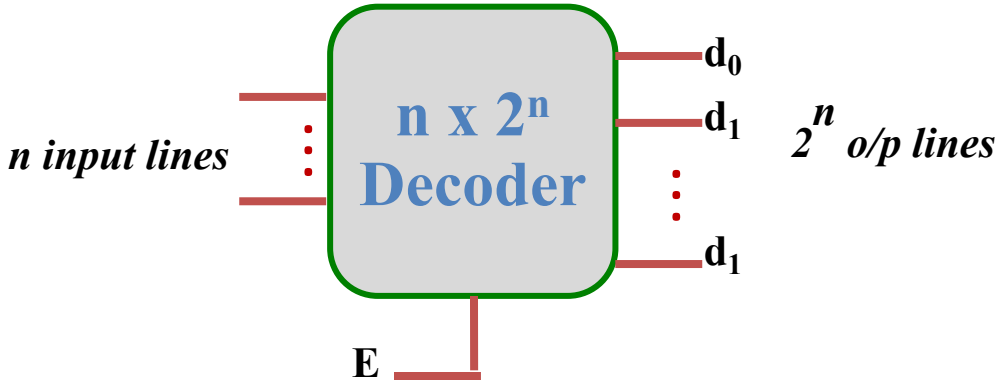


When input changes, output may change (after a delay)

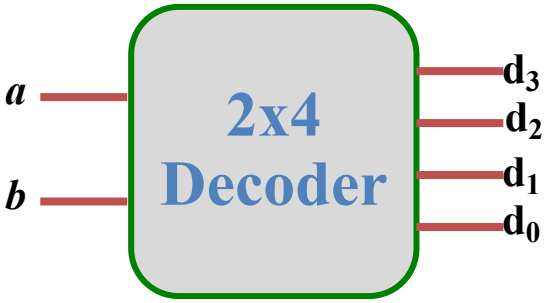
Decoder

Decoder

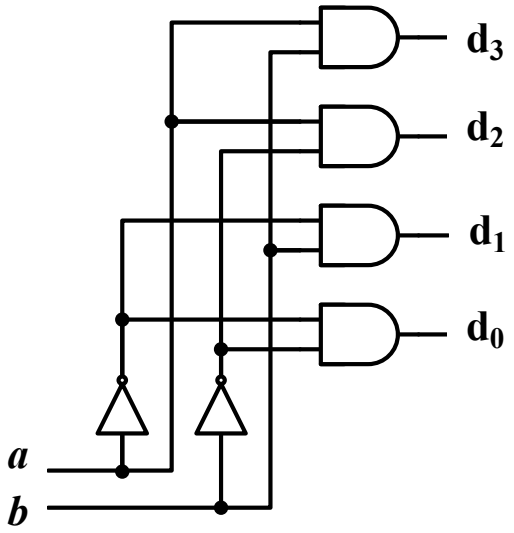
- A decoder is a combinational circuit that converts binary information from **n** input lines to a maximum of $\mathbf{2^n}$ unique output lines
- If a binary decoder receives **n** inputs it activates one and only one of its $\mathbf{2^n}$ outputs based on that input with all other outputs deactivated
- The o/p of a decoder are min-terms of **n** i/p variables
- A decoder can also have enable input. If E=0, then all o/p of decoder are zero. If E= 1, then only one o/p of decoder is one.
- If the n-bit coded information at the input has unused combinations, the decoder may have less than $\mathbf{2^n}$ output lines, e.g., a BCD-to-7 segment decoder is actually a 4x16 decoder, in which the output lines from d_{10} to d_{15} are not used



2x4 Decoder Implementation



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



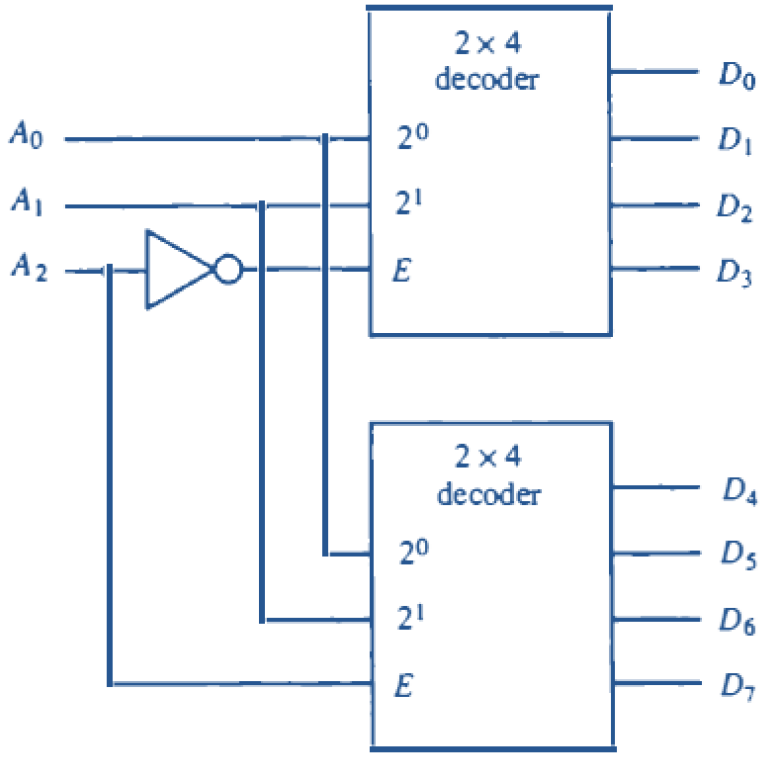
$d_0 = a'.b'$
$d_1 = a'.b$
$d_2 = a.b'$
$d_3 = a.b$

Decoder.hdl

```
CHIP Decoder {
 IN a, b;
 OUT d0, d1, d2, d3;
 PARTS:
    Not(in=a, out=Nota);
    Not(in=b, out=Notb);
    And(a=Nota, b=Notb, out=d0);
    And(a=Nota, b=b, out=d1);
    And(a=a, b=Notb, out=d2);
    And(a=a, b=b, out=d3);
}
```

Constructing Higher Order Decoder

- **Example:** Draw a 3x8 Decoder using two 2x4 Decoder with enable input



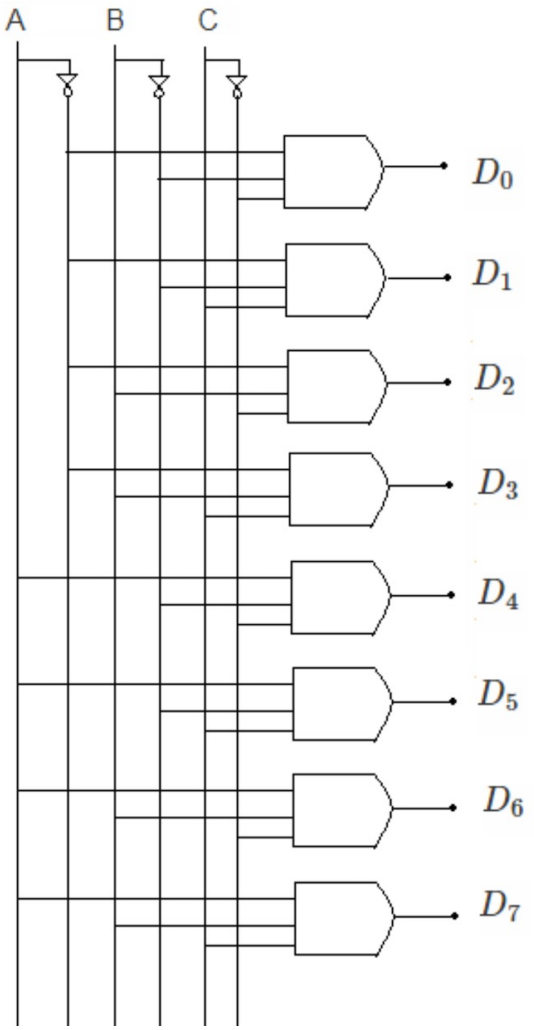
- **Example:** Draw a 4x16 Decoder using two 3x8 Decoder with enable input
- **Example:** Draw a 5x32 Decoder using four 3x8 Decoder and a 2x4 Decoder

Applications of Decoder (Code Conversion)

A 3x8 Decoder can be used as Binary to Octal decoder. You give it a binary number at its input in 3 bits, and it decodes that information by making the corresponding output bit as 1

A	B	C	D0	D1	D2	D3	D4	D5	D6	D7
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

$$D_0 = \bar{A}\bar{B}\bar{C}, \quad D_1 = \bar{A}\bar{B}C, \quad D_2 = \bar{A}B\bar{C},$$
$$D_3 = \bar{A}BC, \quad D_4 = A\bar{B}\bar{C}, \quad D_5 = A\bar{B}C,$$
$$D_6 = AB\bar{C}, \quad D_7 = ABC$$



Applications of Decoder (Logic Circuit Implementation)

Since the output of a decoder are min-terms, so we can implement a logic circuit using appropriate size decoder

- **Example:** Implement following Boolean function using decoder

$$F(a, b, c, d) = \Sigma (1, 2, 5, 6, 7, 10, 12)$$

- **Example:** Implement Full Adder using Decoder

$$Sum(a, b, c) = \Sigma (0, 1, 4, 7)$$
$$Carry(a, b, c) = \Sigma (3, 5, 6, 7)$$

- **Example:** Implement Full Subtractor using Decoder

$$Diff(a, b, c) = \Sigma (1, 2, 4, 7)$$
$$Borrow(a, b, c) = \Sigma (1, 2, 3, 7)$$

Applications of Decoder (Address Decoding)

- A Decoder can be used to decode a memory address.

- For example, suppose you have a memory of 32 words and each word is of 16 bits. The address bus will be of 5 bits and the data bus will be of 16 bits. A 5x32 Decoder can be used to decode the address and enable one of the eight memory locations

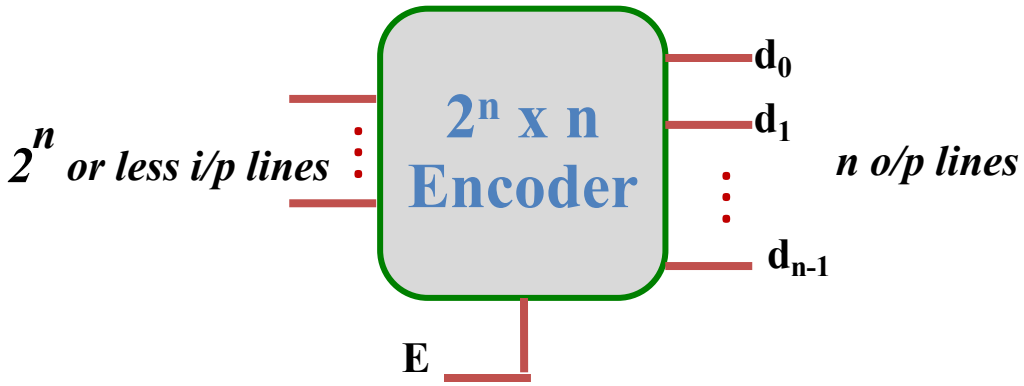
- If you have a memory of 4MB with each memory location of 32 bits. What size of decoder is required?

Demo Combinational Circuits

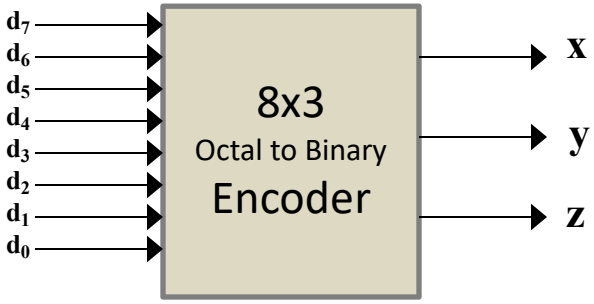
Encoder

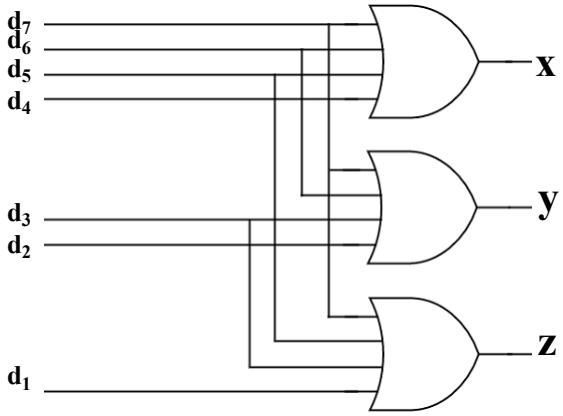
Encoder

- An Encoder is a combinational circuit that converts information in the form of 2^n input lines into **n** or less output lines
- It is the opposite of Decoder
- In case of encoder, it is assumed that at any given time, only one of the input is high. And depending on which input is high we get the specific code at the output
- Normally used to encode various symbols and alphabetic characters to a coded format. For example: Octal to Binary encoder takes 8 input lines and generates 3 output lines. Similarly, an ASCII encoder will have 128 input lines and 7 output lines



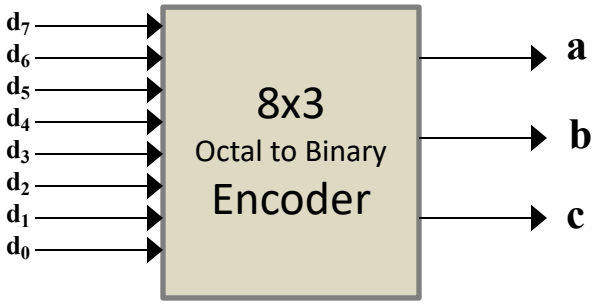
Octal to Binary Encoder

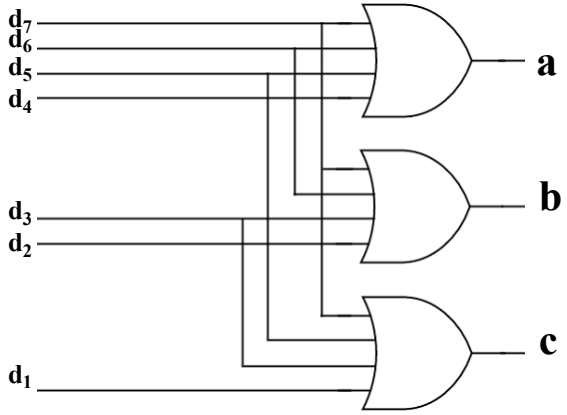




d_0	d_1	d_2	d_3	d_4	d_5	d_6	d_7	x	y	z
0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

Octal to Binary Encoder Implementation





$$a = d_4 + d_5 + d_6 + d_7$$
$$b = d_2 + d_3 + d_6 + d_7$$
$$c = d_1 + d_3 + d_5 + d_7$$

Encoder.hdl

```
CHIP Encoder {
 IN d0,d1,d2,d3,d4,d5,d6,d7;
 OUT a, b, c;
 PARTS:
   Or(a=d4, b=d5, out=w1);
   Or(a=d6, b=d7, out=w2);
   Or(a=w1, b=w2, out=a);

   Or(a=d2, b=d3, out=w3);
   Or(a=d6, b=d7, out=w4);
   Or(a=w3, b=w4, out=b);

   Or(a=d1, b=d3, out=w5);
   Or(a=d5, b=d7, out=w6);
   Or(a=w5, b=w6, out=c);
}
```

- **Decimal to BCD Encoder**
- **ASCII Encoder**

Limitations of Encoder

- **Limitation 1:** Output 0 may mean that all inputs are zero or may mean that d_0 is one

d_0	d_1	d_2	d_3	a	b
0	0	0	0	**0**	**0**
1	0	0	0	**0**	**0**
0	1	0	0	**0**	**1**
0	0	1	0	**1**	**0**
0	0	0	1	**1**	**1**

- To solve this issue, we can have an additional output 'v' of the encoder, which will be zero when all the inputs are zero.

d_0	d_1	d_2	d_3	a	b	v
0	0	0	0	**0**	**0**	**0**
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**	**1**

Limitations of Encoder

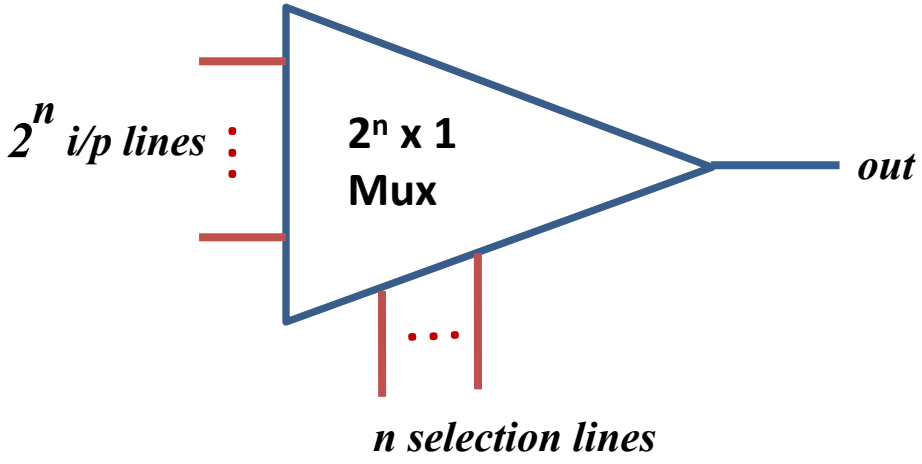
- **Limitation 2:** If more than one input lines of an encoder are high, the output of the encoder will be invalid.
- For example, in a 4x2 Encoder if both d1 and d3 are one, both a and b outputs will be one.
- Solution is Priority encoder
  - A high priority encoder will consider the higher order input, when more than one inputs are one.
  - A low priority encoder will consider the lower order input, when more than one inputs are one.

Demo Combinational Circuits

Multiplexer

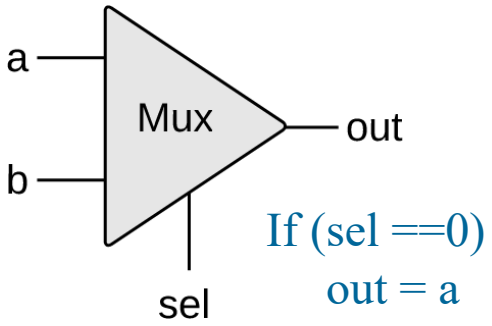
Multiplexer

- Multiplexing is a technique of putting multiple signals on a single data path
- A Multiplexer (data selector) is a combinational circuit that selects binary information from one of many (2^n) input lines and transmit it to a single output line. The selection of a particular input line is controlled by a set of n selection lines
- Depending on the select input, only one of the input lines is available at output



2x1 Mux

- A 2-way multiplexor (2x1 Mux) enables selecting, and outputting, one of two possible inputs as shown above



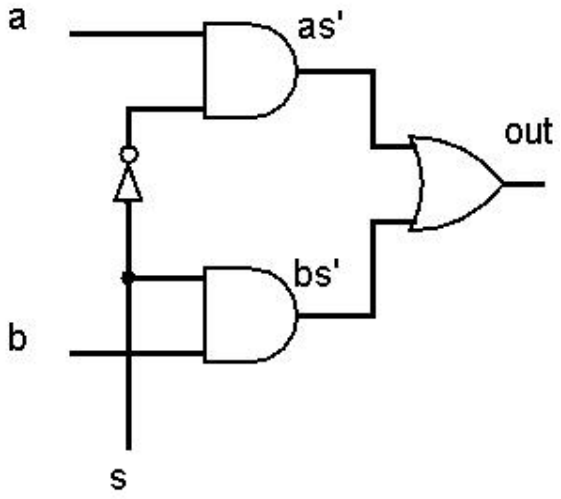
If (sel ==0)
    out = a
else
    out = b

a	b	sel	out
0	0	0	**0**
0	1	0	**0**
1	0	0	**1**
1	1	0	**1**
0	0	1	**0**
0	1	1	**1**
1	0	1	**0**
1	1	1	**1**

sel out

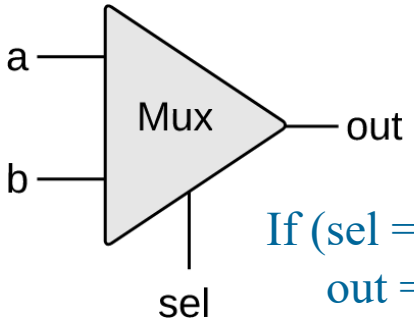
sel	out
0	**a**
1	**b**

abbreviated truth table



$$out = as' + bs$$

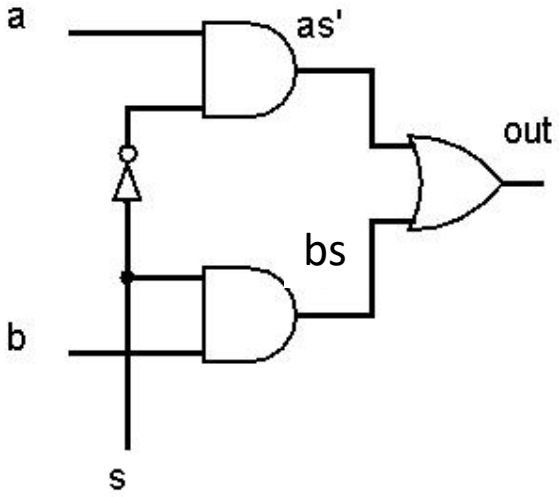
2x1 Mux Implementation



If (sel ==0)
    out = a
else
    out = b

sel	out
0	a
1	b

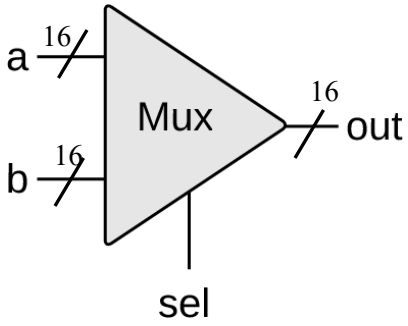
$out = as' + bs$



Mux.hdl

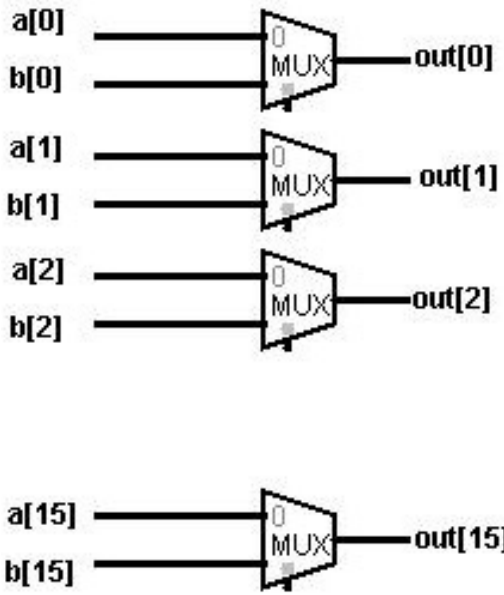
```
CHIP Mux {
 IN a, b, sel;
 OUT out;
 PARTS:
    Not(in=sel, out=Notsel);
    And(a=a, b=Notsel, out=aAndNotsel);
    And(a=b, b=sel, out=bAndsel);
    Or(a=aAndNotsel, b=bAndsel, out=out);
}
```

Mux16

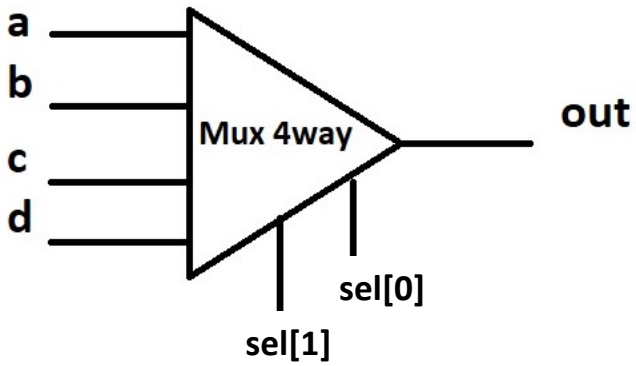


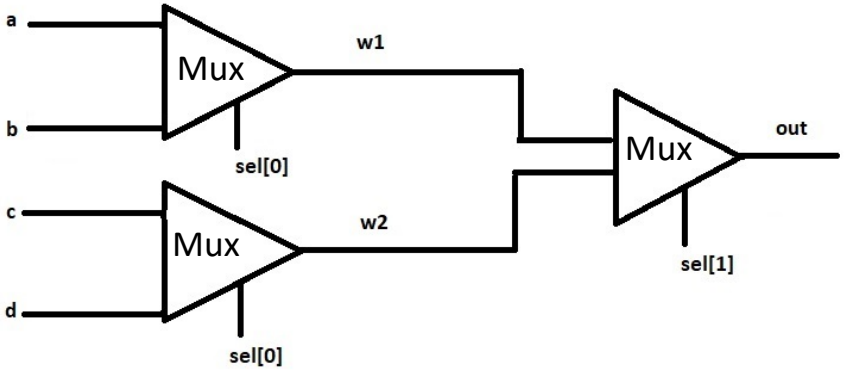
Mux16.hdl

```
/*
 * 16-bit multiplexor:
 * for i = 0..15 out[i] = a[i] if sel == 0
 *                        b[i] if sel == 1
 */
CHIP Mux16 {
    IN a[16], b[16], sel;
    OUT out[16];
    PARTS:
    Mux(a=a[0], b=b[0], sel=sel, out=out[0]);
    Mux(a=a[1], b=b[1], sel=sel, out=out[1]);
    Mux(a=a[2], b=b[2], sel=sel, out=out[2]);
          …
    Mux(a=a[15], b=b[15], sel=sel, out=out[15]);
}
```



Mux4way (using three 2x1 Mux)





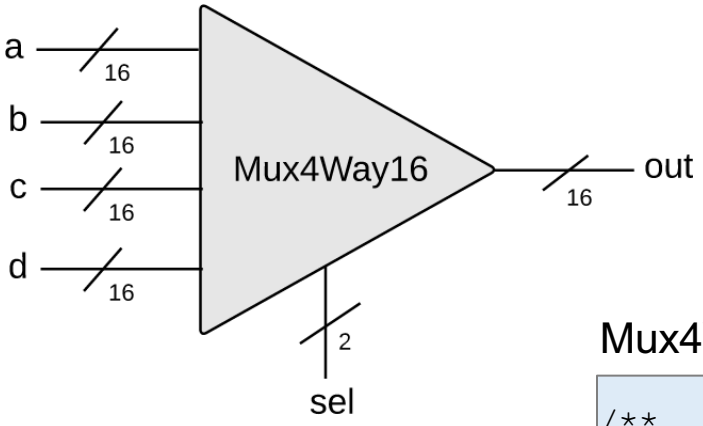
Sel[1]	Sel[0]	out
0	0	a
0	1	b
1	0	c
1	1	d

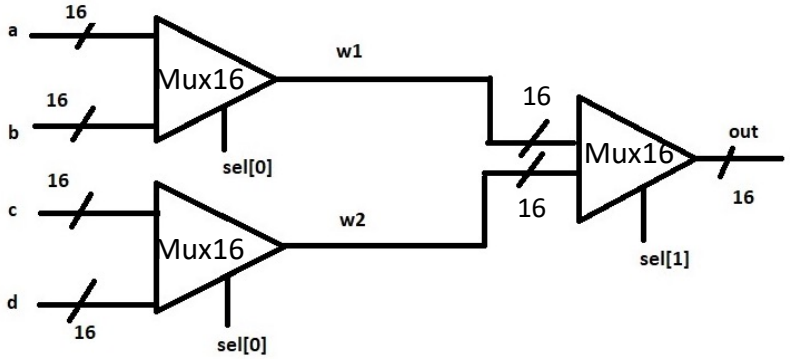
Mux4way.hdl

```
CHIP Mux4way{
  IN a,b,c,d, sel[2];
  OUT out;

  PARTS:
  Mux(a=a, b=b, sel=sel[0], out=w1);
  Mux(a=c, b=d, sel=sel[0], out=w2);
  Mux(a=w1, b=w2, sel=sel[1], out=out);
}
```

Mux4way16 (using three Mux16)





Sel[1]	Sel[0]	out
0	0	a
0	1	b
1	0	c
1	1	d

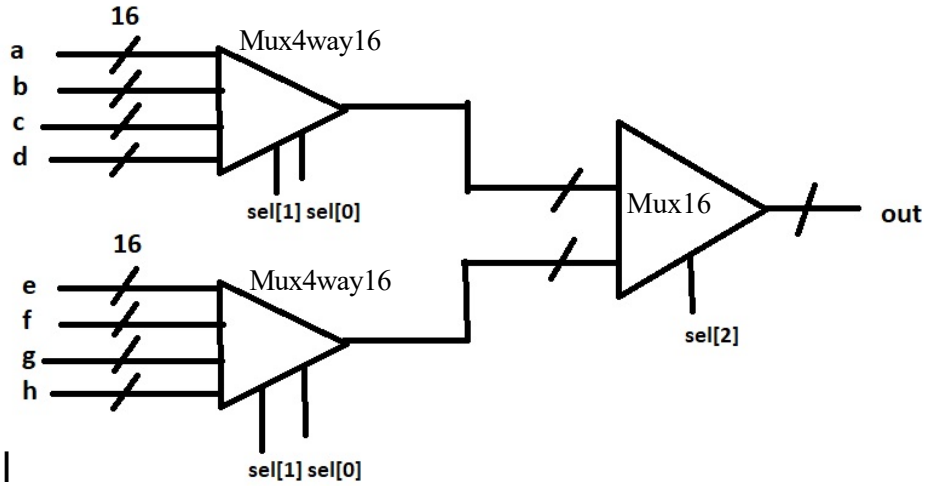
Mux4Way16.hdl

```
/**
 * 4-way 16-bit multiplexor:
 * out = a if sel == 00
 *       b if sel == 01
 *       c if sel == 10
 *       d if sel == 11
 */
CHIP Mux4Way16 {
    IN a[16], b[16], c[16], d[16], sel[2];
    OUT out[16];

    PARTS:

    Mux16(a=a, b=b, sel=sel[0], out=w1);
    Mux16(a=c, b=d, sel=sel[0], out=w2);
    Mux16(a=w1, b=w2, sel=sel[1], out=out);
}
```

Mux8way16 (using two Mux4way16 and a Mux16)




```
/*  8-way 16-bit mux
 *   out = a if sel == 000
 *   out = b if sel == 001
        .....
 *   out = b if sel == 001
*/
```

Mux8Way16.hdl

```
CHIP Mux8Way16 {
    IN a[16], b[16], c[16], d[16], e[16], f[16], g[16], h[16],
       sel[3];
    OUT out[16];
    PARTS:
    Mux4Way16(a=a, b=b, c=c, d=d, sel=sel[0..1], out=Mux4abcd);
    Mux4Way16(a=e, b=f, c=g, d=h, sel=sel[0..1], out=Mux4efgh);
    Mux16(a=Mux4abcd, b=Mux4efgh, sel=sel[2], out=out);
}
```

Implementing Boolean Function using Mux

- **Example:** Implement following Boolean function using multiplexer

$$F(a, b, c) = \Sigma (1, 2, 4, 6)$$

- **Example:** Implement Full Adder using multiplexer

$$Sum(a, b, c) = \Sigma (0, 1, 4, 7)$$

$$Carry(a, b, c) = \Sigma (3, 5, 6, 7)$$

- **Example:** Implement Full Subtractor using multiplexer

$$Diff(a, b, c) = \Sigma (1, 2, 4, 7)$$

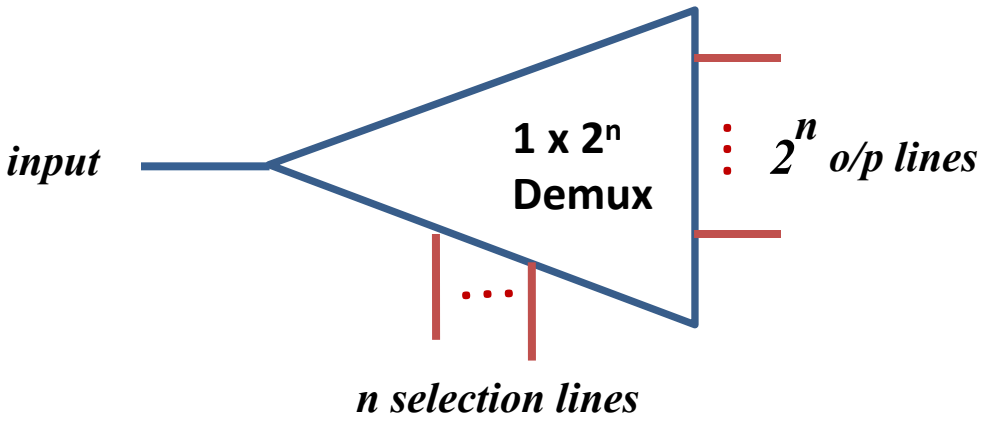
$$Borrow(a, b, c) = \Sigma (1, 2, 3, 7)$$

Multi-Bit Gates with Buses: Demo

Demux

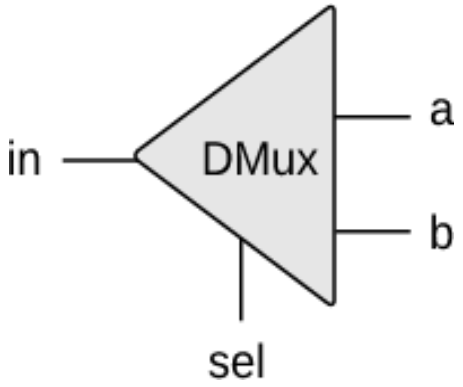
De-Multiplexer

- A De-Multiplexer (data distributer) is a combinational circuit that receives information on a single line and transmit this information on one of 2^n possible output lines.
- The selection of a particular output line is controlled by a set of **n** selection lines
- Depending on the select input, the only i/p is placed on an appropriate o/p line



1x2 Demux

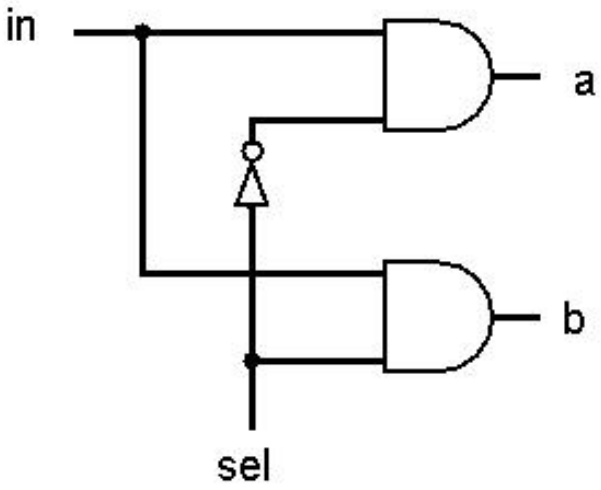
- A 1x2 demultiplexer distributes the single input value into one of two possible destinations as shown above




```
If (sel==0)
   {a,b} = {in,0}
else
   {a,b}={0,in}
```

in	sel	a	b
0	0	0	0
0	1	0	0
1	0	1	0
1	1	0	1

a = in.sel'
b = in.sel

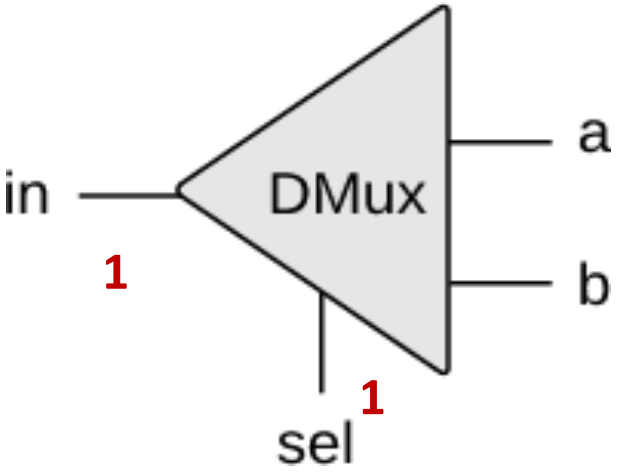


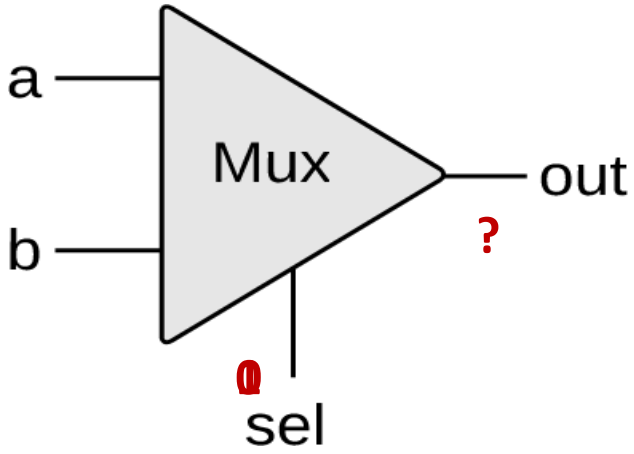
DMux.hdl

```
CHIP DMux{
   IN in, sel;
   OUT a, b;

   PARTS:
      Not(in=sel, out=Notsel);
      And(a=in, b=Notsel, out=a);
      And(a=in, b=sel, out=b);
}
```

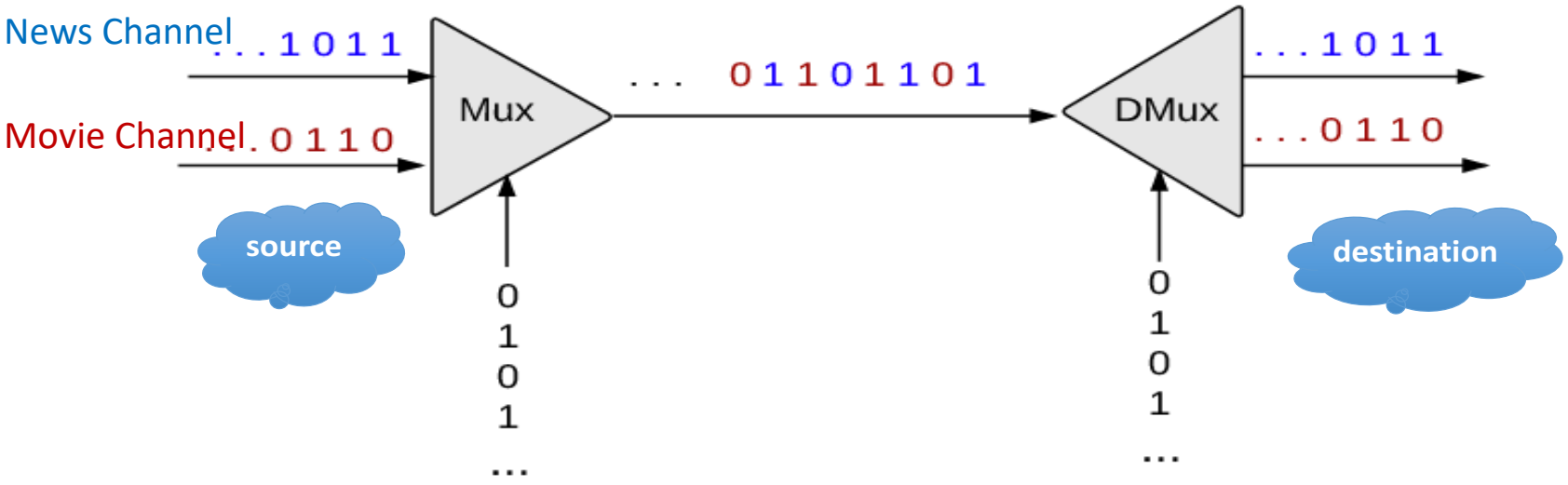
Question





- Consider the above scenario, in which we have a 1x2 DMux and a 2x1 Mux
- The two outputs of Dmux are fed into the two inputs of the Mux
- Suppose the **in** of the Dmux is 1, and its **sel** input is also 1
- What is there on the **out** of Mux, if the **sel** of the Mux is set to 0?
- What is there on the **out** of Mux, if the **sel** of the Mux is set to 1?

Example: Multiplexing/Demultiplexing In Communication Network



- A common use of multiplexing / de-multiplexing logic is shown above that enables transmitting multiple channels/messages on a single shared communication line
- Suppose we have two channels coming in the 2x1 Mux (source). We want to transmit both these channels via a single communication line
- The **sel** bit of the Mux is connected to an oscillator that produces a repetitive train of alternating 0 and 1 signals, so transmitting one bit from each channel in each oscillation (Time Division Multiplexing)
- On the destination side, one can use the **sel** input of the Dmux to select the channel to watch ☺

Demo Combinational Circuits

Things To Do

- Perform interactive and script based testing of the chips designed in today's session on the h/w simulator. You can download the .hdl, .tst and .cmp files of above chips from the course bitbucket repository:

  https://github.com/arifpucit/COAL_VLecs

Coming to office hours does NOT mean you are academically weak!