

Combinational Logic Circuits

CIT 595
Spring 2008

Computer Components

- Computer components are made from both combinational and sequential logic circuits
- We will apply the knowledge of Boolean Algebra to realize these circuits
- First we will look at Combinational Logic Circuit

CIT 595

2

Combinational Logic Circuits

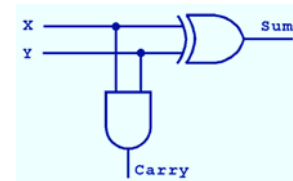
- Always gives the same output for a given set of inputs
- Do not store any information (memoryless)
- Examples: adder, decoder, multiplexer (mux), shifter
 - These are combined to form larger units such as ALU

CIT 595

3

1 Bit Addition Unit (Half Adder)

Inputs		Outputs	
X	Y	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



This circuit is known as half adder

- Does half the job – does not account for carry-in input

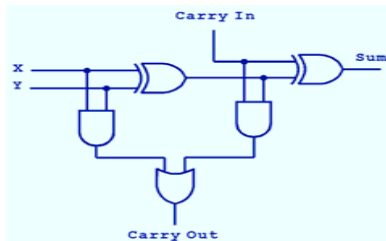
CIT 595


4

1 Bit Addition Unit (Full Adder)

Inputs			Outputs	
X	Y	Carry In	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

$$\begin{aligned} \text{Sum} &= \bar{x}\bar{y}C_{in} + \bar{x}y\bar{C}_{in} + x\bar{y}\bar{C}_{in} + xyC_{in} \\ &= \bar{C}_{in}(\bar{x}y + x\bar{y}) + C_{in}(\bar{x}\bar{y} + xy) \\ &= \bar{C}_{in}(x \oplus y) + C_{in}(\overline{x \oplus y}) \\ &= (x \oplus y) \oplus C_{in} \end{aligned}$$



NOTE:  $A \oplus B$

A	B	Out
0	0	1
0	1	0
1	0	0
1	1	1

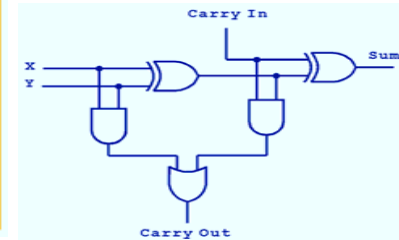
CIT 595

$$\bar{x}\bar{y} + xy = \overline{x \oplus y}$$

5

1 Bit Addition Unit (Full Adder) contd..

Inputs			Outputs	
X	Y	Carry In	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

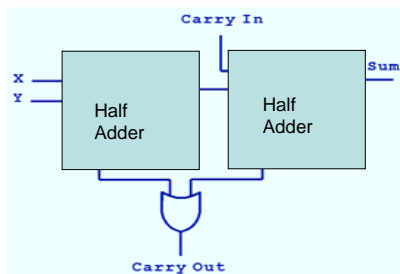


$$\begin{aligned} \text{Carry Out} &= \bar{x}\bar{y}C_{in} + xyC_{in} + \bar{x}yC_{in} + xy\bar{C}_{in} \\ &= C_{in}(\bar{x}\bar{y} + \bar{x}y) + xy(C_{in} + \bar{C}_{in}) \\ &= C_{in}(x \oplus y) + xy \end{aligned}$$

CIT 595

6

1 Bit Full Adder



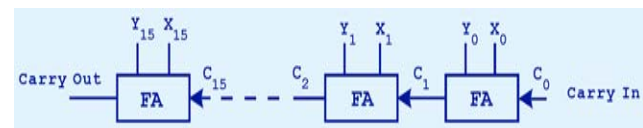
Two half adders make a full adder

CIT 595

7

N-bit Adder

- Just as we combined half adders to make a full adder, full adders can be connected in series
- The carry bit "ripples" from one full adder to the next; hence, this configuration is called a *ripple-carry adder*



16-bit Ripple Carry Adder

C_0 is assumed to be 0

CIT 595

8

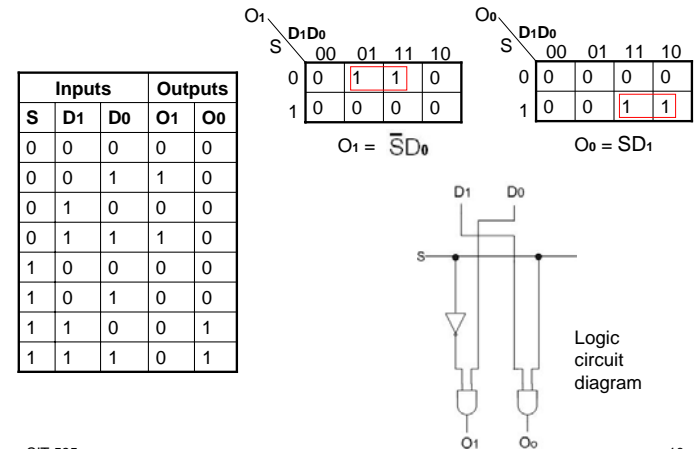
Bit Shifter

- Lets see the design of an unsigned 2-bit shifter
- To determine 1-bit shift to left or right
- Assume that this decided by control variable/signal input called S
 - If S = 0, then we do 1 bit left shift
 - Else, we do 1 bit right shift
- Lets say the input is 2-bit value D(D₁, D₀) where D₁ is the most significant bit(MSB)
- Lets call the output of the shift be O(O₁, O₀)

CIT 595

9

Creating Logic for 2-bit Shifter

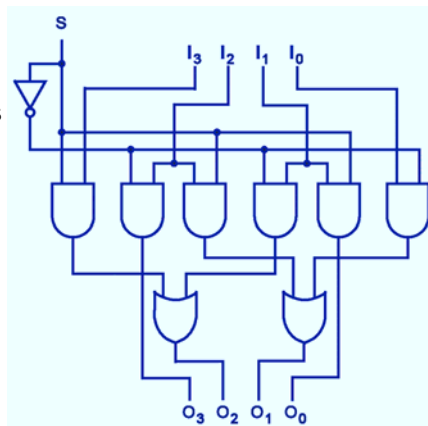


CIT 595

10

4-Bit Bit Shifter

- Similarly, we can make 4-bit shifter that moves the bits of a nibble (half of a byte) one position to the left or right
- How many rows will the truth table contain?

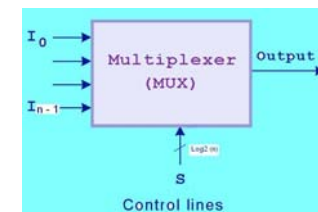


CIT 595

11

Multiplexer

- A multiplexer sets its single output to the same value as one of its many inputs
- Output is determined by the value of the multiplexer's control lines (a.k.a selector)
- To be able to select among n inputs, $\log_2 n$ control lines are needed



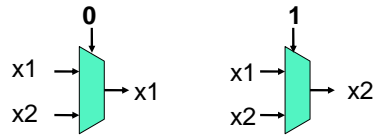
This is a block diagram for a multiplexer

CIT 595

12

2-to-1 MUX

- Selects between two inputs



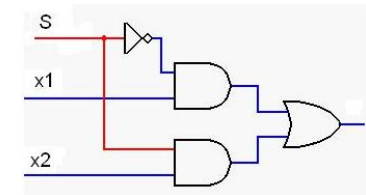
- What is the logic behind selection?

CIT 595

13

2 to 1 MUX

s	x1	x2	F(s,x1,x2)
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1



$$s'x1 + sx2$$

CIT 595

14

Subtraction

- The adder logic circuit seen before does only addition
- Recall that $X - Y = X + (-Y)$
 - We find 1's complement of Y and add 1 to get negative value of Y i.e. $-Y$
 - Then we add X and $-Y$

$$\begin{array}{r}
 01101000 \text{ (104)} \\
 - 00010000 \text{ (16)} \\
 \hline
 01101000 \text{ (104)} \\
 + 11110000 \text{ (-16)} \\
 \hline
 01011000 \text{ (88)}
 \end{array}
 \qquad
 \begin{array}{r}
 11110110 \text{ (-10)} \\
 - 11110111 \text{ (-9)} \\
 \hline
 11110110 \text{ (-10)} \\
 + 00001001 \text{ (9)} \\
 \hline
 11111111 \text{ (-1)}
 \end{array}$$

CIT 595

15

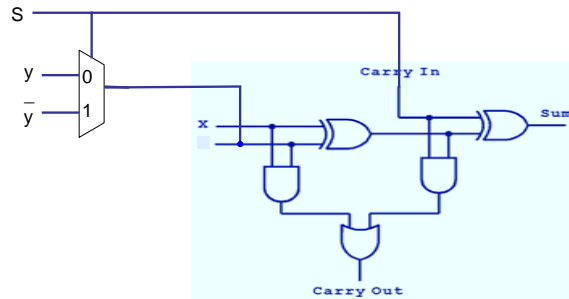
Implementing Subtraction Logic in 1 Bit Adder Unit

- Let Y be the 2nd input
- We need both the Y and Y complement (Y')
- To choose between addition and subtraction we will use a select signal "S" (we will learn later that S is actually generated by the control unit)
 - S = 0, then addition i.e. Y input is chosen
 - S = 1, then subtraction i.e. Y complement is chosen
- If subtraction then we need to add 1 to Y complement (Y') so as to get $-Y$
 - +1 can be achieved by making the first carry C_0 into the adder be 1
 - Hence, $C_0 = S$ (this will allow both operations i.e. add/sub)

CIT 595

16

Modification to the 1 Bit Adder (w/ Subtraction)



CIT 595

17

Detecting Arithmetic Overflow

- Overflow is said to occur if result is too large to fit in the number of bits used in the representation

Carry into	→	1		0
			+	
MSB		01000 (8)		11000 (-8)
			+	
		01001 (9)		10111 (-9)
Carry Out	→	0		1
		10001 (-15)		01111 (+15)

- We have overflow if
 - Signs of both numbers are the same, and Sign of sum is different
 - If Positive number is subtracted from a Negative number, result is positive and vice versa
- Another test (easy for hardware)
 - Carry into MSB does not equal carry out

CIT 595

18

Detecting Overflow

- Circuit outputs 1 when the Carry into MSB (MCin) does not equal carry out (Cout)

- If you observe carefully, the output is equivalent to XOR gate

- Thus to detect overflow we XOR the values of Cout and MCin

- In general, for n-bit adder

$$\text{Overflow} = C_n \oplus C_{n-1}$$

\downarrow
Cout

\downarrow
Cin into MSB

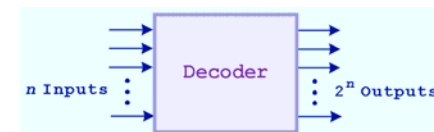
Cout	MCin	Overflow
0	0	0
0	1	1
1	0	1
1	1	0

CIT 595

19

Decoder

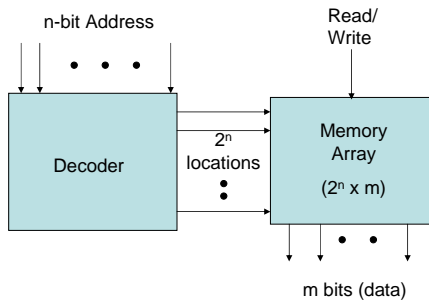
- Decoders are circuits used to decode encoded information
- A binary decoder converts binary information from n-bit input code to a maximum of 2^n unique outputs
 - Decoder logic uses n-bit input value to choose exactly one of the 2^n outputs (only a particular output is active)
 - Example: Memory Address Decoding



CIT 595

20

Address Decoder Example



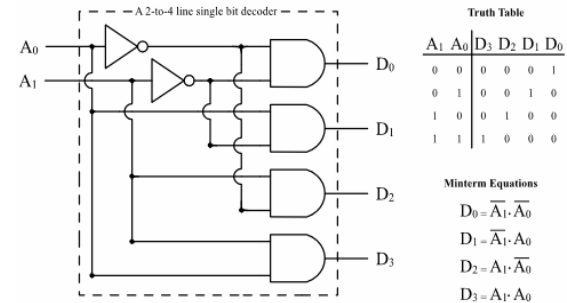
Decoder will select only 1 memory location (row) based on address

CIT 595

21

2-to-4 Decoder Logic

- A binary number with 2 bits as its input
- Selects exactly 1 of 4 outputs
 - At any time, only 1 output line is "ON" or "1" and all others are "OFF" or "0" (referred to as one-hot encoded)



1 – active/asserted/chosen

0 – not active/deasserted/not chosen

CIT 595

22

Encoder

- Opposite of decoder
- Given information is transformed into more compact form
- Example:
 - In Interrupt Driven I/O - need to determine higher priority among devices who interrupted at the same time
 - **Priority encoder** circuit determines which interrupt request should be serviced by the processor
 - > In priority encoder each input has a priority level associated with it
 - > The encoder outputs indicate active input that has the highest priority

CIT 595

23

Resolving Interrupts

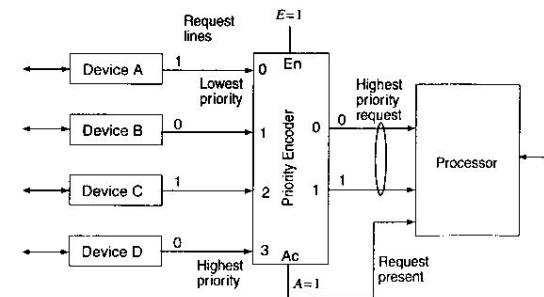


Figure 9.17 Resolving interrupt requests using a priority encoder.

CIT 595

24

Example: 4:2 Priority Encoder

- x3 (Device D) has highest priority

- x2 the next highest...x0 has lowest priority

- y1y0 are outputs determine the which request goes to processor for servicing

Inputs				Outputs	
x3	x2	x1	x0	y1	y0
1	x	x	x	1	1
0	1	x	x	1	0
0	0	1	x	0	1
0	0	0	1	0	0
0	0	0	0	x	x

X – don't care

CIT 595

25

Code Converters

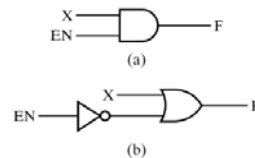
- In general encoders and decoders are known as code converters
- Convert from one type of coded information (encoding) to another output encoding
- Example: BCD to 7 segment display (calculators and digital number displays etc)
 - see "Handout" on code converters
 - BCD - Read sec 2.6.1 on chp2 first – pg 74

CIT 595

26

Enabling/Gating Outputs

- Combinational logic circuits produce an output based on certain inputs
- We may not want to use its output all the time
 - Same inputs are shared amongst different logic units
 - Some how we only want to make one unit active and disable all others
- Hence we want some sort of control to temporarily disable the circuit and only enable it if the control is set
- Basic gates allow us to achieve this:
 - AND gate - Figure a. $EN = 1, F = X$
 - OR gate - Figure b. $\sim EN = 0, F = X$



Who decides value of EN?

Example on the next slide

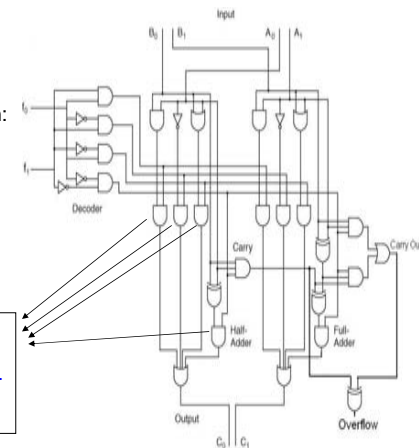
CIT 595

27

2-Bit ALU

- f0 and f1 control lines (generated by control unit)
- The value of control lines determine which operation:
 - 00 – A + B (Addition)
 - 01 – NOT A
 - 10 – A OR B
 - 11 – A AND B
- Similarly a N-Bit ALU can be made

All sub units form their operation, but final output is chosen only if enabled (EN = 1). Here EN is decided by the decoder logic.



CIT 595

28

