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

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

1 Bit Addition Unit (Half Adder)

This circuit is known as half adder
- Does half the job – does not account for carry-in input
1 Bit Addition Unit (Full Adder)

\[ \text{Sum} = \overline{x}y\text{Cin} + xy\overline{\text{Cin}} + \overline{x}y\overline{\text{Cin}} + xy\text{Cin} + \overline{\text{Cin}}(x\overline{y} + xy) + \text{Cin}(\overline{x}y + xy) + \text{Cin}(x \oplus y) + \text{Cin} \]

NOTE: \( A + B + C \downarrow = \overline{A} \oplus B \oplus C \downarrow \)

1 Bit Full Adder

Two half adders make a full adder

1 Bit Addition Unit (Full Adder) contd..

\[ \text{Carry Out} = xy\text{Cin} + xy\overline{\text{Cin}} + \overline{xy}\text{Cin} + xy\overline{\text{Cin}} + \overline{\text{Cin}}(x\overline{y} + xy) + \text{Cin}(\overline{x}y + xy) + \text{Cin}(x \oplus y) + xy \]

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.

Co is assumed to be 0
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.

2-to-1 MUX

- Selects between two inputs.

What is the logic behind selection?

2 to 1 MUX

<table>
<thead>
<tr>
<th>s</th>
<th>x1</th>
<th>x2</th>
<th>F(s,x1,x2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

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 \( -Y \).

\[
\begin{align*}
01101000 \ (104) & \quad 11110110 \ (-10) \\
-00010000 \ (16) & \quad -11110111 \ (-9) \\
01101000 \ (104) & \quad 11110110 \ (-10) \\
+11110000 \ (-16) & \quad +00001001 \ (9) \\
01011000 \ (88) & \quad 11111111 \ (-1)
\end{align*}
\]
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 C0 into the adder be 1
  - Hence, C0 = S (this will allow both operations i.e. add/sub)

Modification to the 1 Bit Adder (w/ Subtraction)

Detecting Arithmetic Overflow

- Overflow is said to occur if result is too large to fit in the number of bits used in the representation
  
<table>
<thead>
<tr>
<th>Carry into MSB</th>
<th>01000 (8)</th>
<th>11000 (-8)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Carry Out</td>
<td>+01001 (9)</td>
<td>+10111 (-9)</td>
</tr>
</tbody>
</table>

- 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

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

<table>
<thead>
<tr>
<th>Cout</th>
<th>MCin</th>
<th>Overflow</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
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)

Creating Logic for 2-bit Shifter

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>D1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

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

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 chose exactly one of the $2^n$ outputs (only a particular output is active)
  - Example: Memory Address Decoding
Address Decoder Example

- n-bit Address
- Read/Write
- Decoder
- Memory Array (2^n x m)
- 2^n locations
- m bits (data)

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

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)

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

Resolving Interrupts

- In priority encoder each input has a priority level associated with it
- The encoder outputs indicate active input that has the highest priority

Figure 9.17 Resolving interrupt requests using a priority encoder.
Example: 4:2 Priority Encoder

- \( x_3 \) has highest priority
- \( x_2 \) the next highest...\( x_0 \) has lowest priority
- \( y_1\)y_0\) are outputs determine the which request goes to processor for servicing

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_3 )</td>
<td>( x_2 )</td>
</tr>
<tr>
<td>1</td>
<td>x</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

\( X \) – don’t care

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)
  - Binary-Coded Decimal (BCD) is an encoding for decimal numbers in which each digit is represented by its own binary sequence.
  - see “Handout” on code converters

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. \( \neg EN = 0, F = X \)

Who decides value of \( EN \)?

Example on the next slide

2-Bit ALU

- \( f_0 \) and \( f_1 \) control lines (generated by control unit)
- The value of control lines determine which operation:
  - 00 – \( A + B \) (Addition)
  - 01 – \( \neg A \)
  - 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.

Example on the next slide
Propagation/Gate Delay

- The length of time starting from when the input to a logic gate becomes stable and valid, to the time that the output of that logic gate is stable and valid.

- The levels of gate for digital can add to more delay
  
  ▶ E.g. in Ripple Carry adder, the carry bit has to propagate through, end to end
  
  ▶ Assume one gate has a propagation delay of \( x \) units
  
  ▶ Carry passes through 2 gate levels per adder
  
  ▶ Total carry propagation time for \( n \)-bit adder = \( 2^n x \) units

Efficient Design E.g. Carry Look Ahead

- Recap
  
  ▶ \( S_i = A_i \oplus B_i \oplus C_i \)
  
  ▶ \( S_i = P_i \oplus C_i \) where \( P_i = A_i \oplus B_i \)
  
  ▶ \( C_{i+1} = C_i (A_i \oplus B_i) + A_i B_i \)
  
  ▶ \( C_{i+1} = C_i P_i + G_i \) where \( G_i = A_i B_i \)

- \( G_i \) is called carry generate and it produces a carry of 1 when both \( A_i \) and \( B_i \) are 1 regardless of the carry in \( C_i \)

- \( P_i \) is called carry propagate and is associated with propagation of \( C_i \) to \( C_{i+1} \)

- The addition will carry whenever there is an input carry, but will not carry if there is no input carry
Carry Look Ahead

- A digit of addition will carry:
  \[ C_{i+1} = C_i + (P_i \cdot C_i) \]
- Note that carry \( C_1, C_2, C_3 \) are calculated at the same time

Programmable Logic Array (PLA)

- A PLA is a prepackaged circuit that can be tailored to suit various needs
- Any truth table can be represented by some AND gates feeding into an OR gate
  - Sum Of Product form
- # of AND gates determines how many truth table rows can have 1’s in them over all the functions implemented

FPGAs

- Field Programmable Gate Arrays
- Used to implement larger circuit
  - > 20K gates
- Contains
  - I/O blocks
  - Logic blocks for implementing required functions
  - Interconnection (contain wires & programmable switches so as allow logic blocks to interconnected in many ways)

FPGA logic block

- Typically has small number of inputs and one output
- Contains Look Up Table (LUT) which contains storage cells used to implement a small logic function
Hardware Descriptive Languages

- Language to describe the circuit's operation, its design and organization, and tests to verify its operation by means of simulation

- A simulation program, designed to implement the underlying semantics of the language statements, coupled with simulating the progress of time provides the hardware designer with the ability to model a piece of hardware before it is created physically

E.g. Verilog code

- 2-input (32-bit) mux

```verilog
module Mux_2_32(mux_out, data_1, data_0, select);
parameter word_size = 32;
output [word_size – 1: 0] mux_out;
input [word_size – 1: 0] data_1, data_0;
Input select;

always@(a or b)
begin
  if(a == 1'b1)
    q = b;
  else
    q = 1'b0;
end
assign mux_out = select ? data_1: data_0;
endmodule
```

- ???

```verilog
module circuit_1 (q, a, b);
output q;
input a, b;
always@(a or b)
begin
  if(a == 1'b1)
    q = b;
  else
    q = 1'b0;
end
endmodule
```