What 314 Is About

System Architecture

Processor Design

Logic Design

Assembly Language

Machine Instructions

```
jal _getnext
ori $a0,$0,0
lw $t0,8($v0)
lw $t0,12($t0)
beq $t0,0,0x401834
li $t1,4
beq $t0,$t1,0x4018a0
```
Information is encoded with bits: 0’s and 1’s.
(we’ve already seen 2’s complement numbers)

These are encoded using voltages...
+ well understood
+ easy to generate, detect
- affected by environment

But why 1’s and 0’s only?
Digital Representation

Example: representing a B&W picture:
- Black = 0 V
- White = 1 V
- 80% grey = 0.8 V
- ...

Represent by scanning picture in fixed order.

Let’s try doing some computation with the voltages...
Digital Representation

Flip image:

Flip back and forth...

What really happens...

Have to build system to tolerate some error (noise).
Logic Levels

- Store just one bit on a wire...
- Gain reliability

Different conventions are possible
Combinational Devices

A *combinational device:*  
- Output is a function of inputs only ("memoryless")  
- Takes input to valid, stable outputs

Combinational devices *restore* marginally valid signals!
Example

- **Input**: logic 0 if $< V_{il}$, logic 1 if $> V_{ih}$
- **Output**: logic 0 if $< V_{ol}$, logic 1 if $> V_{oh}$

Avoid these regions!
Digital View

- If input is 0, output is 1
- If input is 1, output is 0

Normally written in a table, like this:

<table>
<thead>
<tr>
<th>In</th>
<th>Out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Called a "truth-table".
Implementation: Switching Networks

Lots of ways to build switches...

- relays
- vacuum tubes
- transistors
- ...

P–transistor

N–transistor

Connect a and b if \( g = 0 \).

Connect a and b if \( g = 1 \).
Switching Networks: Inverter

- **Function**: NOT
- **Called an inverter**
- **Symbol**:

```
<table>
<thead>
<tr>
<th>in</th>
<th>out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
```
Switching Networks: NAND

- Function: NAND
- Symbol:
Switching Networks: NOR

- **Function:** NOR
- **Symbol:**

```
\[
\begin{array}{ccc}
  a & b & \text{out} \\
  0 & 0 & 1 \\
  1 & 0 & 0 \\
  0 & 1 & 0 \\
  1 & 1 & 0 \\
\end{array}
\]
```
Building Functions From Gates

- **AND:**

  ![AND gate diagram]

  =

  ![OR gate diagram]

- **OR:**

  ![OR gate diagram]

  =

  ![AND gate diagram]

Can specify function by describing gates, truth table, or logic equations.
Logic Equations

AND:

\[ out = a \cdot b \]
\[ out = ab \]
\[ out = a \land b \]

OR:

\[ out = a + b \]
\[ out = a \lor b \]

NOT:

\[ out = \overline{\text{in}} \]
\[ out = \overline{\text{in}} \]
Logic Equations

Fun with identities:

\[ a + \overline{a} = 1 \]
\[ a + 0 = a \]
\[ a + 1 = 1 \]
\[ a\overline{a} = 0 \]
\[ a \cdot 0 = 0 \]
\[ a \cdot 1 = a \]

\[ a(b + c) = ab + ac \]
\[ (a + b) = \overline{a} \cdot \overline{b} \]
\[ (a \cdot b) = \overline{a} + \overline{b} \]
\[ a + \overline{ab} = a + b \]

Check by writing truth tables, or by manipulating logic equations.
Let’s Build An Adder

Write down function:
- Two 1-bit inputs, $a$ and $b$
- Two 1-bit outputs, $\text{sum}$ and $\text{carry}$

Truth-table:

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>carry</th>
<th>sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Let’s Build An Adder

Sum output:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th>Logic term</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>( \bar{a} \cdot \bar{b} )</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>( a \cdot \bar{b} )</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>( \bar{a} \cdot b )</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>( a \cdot b )</td>
</tr>
</tbody>
</table>

Logic equation: \( a \cdot \bar{b} + \bar{a} \cdot b \)

Circuit:
Let's Build An Adder

Carry output:

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>carry</th>
<th>Logic term</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>( \overline{a} \cdot \overline{b} )</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>( a \cdot \overline{b} )</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>( \overline{a} \cdot b )</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>( a \cdot b )</td>
</tr>
</tbody>
</table>

Logic equation: \( a \cdot b \)

Circuit:
Let's Build An Adder

Final Circuit:

Numbers indicate the number of sequential steps from input to output (worst-case).