Experiment 6
Controlled Circuit Operations

• Objectives:
  – Use multiplexing to provide for control of circuit operations
  – Bridge between simple circuits and more realistic circuits, such as computer circuits
  – Note relationship between hardware and software
The *Control Code* selects the operation that is to be applied to the input data.
Two-Operation Example

Assume a 1-bit control code $T$ and 2 data input signals of 1 bit each: $a$ and $b$

Suppose the desired operations are:

If $T = 0$, output is $\overline{a} \cdot b$

If $T = 1$, output is $a \oplus b$

Truth Table:

<table>
<thead>
<tr>
<th>Row #</th>
<th>$T$</th>
<th>$a$</th>
<th>$b$</th>
<th>$f(x,y)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<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>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Boolean Expression

Minterm representation:

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

Minimized expression:

\[ c = Tab + \overline{a}b \]

Operation-oriented representation:

\[ c = \overline{T}(\overline{a}b) + T(a \oplus b) \]

\[ = [t_0(\overline{a}b)] + [t_1(\overline{a}b + ab)] \]

where \( t_0 = \overline{T}, \ t_1 = T \)
Operation-oriented Representation

\[ c = [t_0(\overline{ab})] + [t_1(a \oplus b)] \]

Note that the control signals \( t_0 \) and \( t_1 \) are \textit{mutually exclusive} -- when \( t_0 = 1, t_1 = 0 \) and when \( t_0 = 0, t_1 = 1 \)

so if \( t_0 = 1 \), then \( t_1 = 0 \), and

\[ c = [1 \cdot (\overline{ab})] + [0 \cdot (a \oplus b)] = (\overline{ab}) + 0 = (\overline{ab}) \]

if \( t_0 = 0 \), then \( t_1 = 1 \), and

\[ c = [0 \cdot (\overline{ab})] + [1 \cdot (a \oplus b)] = 0 + (a \oplus b) \]

\[ = (a \oplus b) \]
Two-Bit Control Code

A 2-bit control code $T=T_1T_0$ can be used to choose between 4 different operations $f_0$, $f_1$, $f_2$, and $f_3$.

<table>
<thead>
<tr>
<th>Control Code $T$</th>
<th>Control Signals</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_1$ $T_0$</td>
<td>$t_0$ $t_1$ $t_2$ $t_3$</td>
</tr>
<tr>
<td>0 0</td>
<td>1 0 0 0</td>
</tr>
<tr>
<td>0 1</td>
<td>0 1 0 0</td>
</tr>
<tr>
<td>1 0</td>
<td>0 0 1 0</td>
</tr>
<tr>
<td>1 1</td>
<td>0 0 0 1</td>
</tr>
</tbody>
</table>

Note: $t_0 = \overline{T_1}\overline{T_0}$, $t_1 = \overline{T_1}T_0$, $t_2 = T_1\overline{T_0}$, $t_3 = T_1T_0$
Two-Bit Control Code

Then the output can be written as

\[ c = t_0 \cdot f_0 + t_1 \cdot f_1 + t_2 \cdot f_2 + t_3 \cdot f_3 \]

so

\[ T = 00 \quad \Rightarrow \quad c = f_0 \]

\[ T = 01 \quad \Rightarrow \quad c = f_1 \]

\[ T = 10 \quad \Rightarrow \quad c = f_2 \]

\[ T = 11 \quad \Rightarrow \quad c = f_3 \]

The same process can be used to select between any four inputs -- for example to "address" 4 different memory locations \( m_0, m_1, m_2, \) and \( m_3 \)
Multiplexers

The *control signal generator* and the *function selector network* taken together form a *multiplexer*. A 4-to-1 multiplexer is shown below:

The two-bit control code, \( A \) and \( B \), selects one of the four inputs, \( I_0, I_1, I_2, I_3 \), to be passed to the output. The output is given by:

\[
Z = \sum_{k=0}^{2^n - 1} t_k I_k
\]

where \( t_k = \text{minterm}_k(A, B) \)

and \( n = \text{number of control code bits} \)
LogicWorks Multiplexers

4-to-1 Multiplexer w/o Enable

4-to-1 Multiplexer w/ Enable

Two-bit Control Code

Data Inputs

Output

Enable (active low)

Data Inputs

Output
Primitive “Computer” Circuit

An “Instruction”

An “Operand”

Accumulator (A)
Operation Code (OC)
Operand Address (OA1) (OA0)

Memory Locations (m0) (m1) (m2) (m3)
Assembly Language

An assembly language is a set of mnemonics that are relatively easy to remember, such as ADD, DIV, MUL, OR, JMP, etc., but that have a one-to-one correspondence with the binary Control Codes (Instructions) for a particular processor.

“Programs” are easily written in assembly language and then translated into machine code.

An “assembler” computer program is usually available to do the tedious translation from assembly language to the required binary machine code.
Assembly Language Instruction Set for the “Computer” Circuit

<table>
<thead>
<tr>
<th>operation mnemonic</th>
<th>usage</th>
<th>meaning</th>
<th>binary code</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>AND $m$</td>
<td>AND the contents of memory location $m$ with the contents of the accumulator $A$</td>
<td>0</td>
</tr>
<tr>
<td>OR</td>
<td>OR $m$</td>
<td>OR the contents of memory location $m$ with the contents of the accumulator $A$</td>
<td>1</td>
</tr>
</tbody>
</table>

Only two *operations* are possible for this simple circuit.

There is no mechanism for storing the circuit output for further use by the circuit, so the circuit cannot execute a sequence of “instructions” -- it cannot be *programmed*. 
### Additional Instruction to “Fudge” a Data Storage Operation

<table>
<thead>
<tr>
<th>operation mnemonic</th>
<th>usage</th>
<th>meaning</th>
<th>binary code</th>
</tr>
</thead>
<tbody>
<tr>
<td>REP</td>
<td>REP</td>
<td>The operator should observe the circuit output and set the accumulator ((A)) to the value of the circuit output.</td>
<td>N/A</td>
</tr>
</tbody>
</table>

With this additional “instruction” one can write sequence of instructions (a *program*) that can be executed by the circuit (with the help of the user)
Assembly Language Program for the “Computer” Circuit

A “program” to AND the contents of memory locations $m_1$ and $m_3$

; the user should initialize program by setting
; $A = 0, m_0, m_1, m_2$, and $m_3$ to desired values
OR $m_1$ ; since $A = 0$, circuit output is the value in memory
; location $m_1$
REP ; the user must intervene to set $A$ to
; value of circuit output
AND $m_3$ ; since $A$ now contains the same data as $m_1$,
; the circuit output is $m_1 \cdot m_3$

The “program” and the switch settings necessary to “execute” the program

<table>
<thead>
<tr>
<th>Assembly language</th>
<th>Binary instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>OR $m_1$</td>
<td>101</td>
</tr>
<tr>
<td>REP</td>
<td>(none; performed by the user)</td>
</tr>
<tr>
<td>AND $m_3$</td>
<td>011</td>
</tr>
</tbody>
</table>