Implement the following logic:

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>d/l</th>
<th>s</th>
<th>fucnt</th>
<th>op</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>11</td>
<td>10</td>
<td>xxx10</td>
<td>11</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>11</td>
<td>10</td>
<td>xxx11</td>
<td>11</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>10</td>
<td>11</td>
<td>xxx</td>
<td>11</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>01</td>
<td>11</td>
<td>xxx</td>
<td>11</td>
</tr>
</tbody>
</table>
\textbf{HW2 is in locker.}

\begin{itemize}
  \item \texttt{case ( (OP, funct))}
  \begin{itemize}
    \item \texttt{\{2'b01, 5'b x\}: begin \ Has priority \}
      \begin{itemize}
        \item \texttt{Sel = 2'b111;}
        \item \texttt{B = 1'b0;}
        \item \texttt{end}
      \end{itemize}
    \item \texttt{\{2'b11, 5'b xxxxx11\}: begin}
      \begin{itemize}
        \item \texttt{Sel = 2'b01;}
        \item \texttt{B = 1'b1;}
        \item \texttt{end}
      \end{itemize}
    \item \texttt{\{2'b11, 5'b xxxxx01\}: begin}
      \begin{itemize}
        \item \texttt{Sel = 2'b10;}
        \item \texttt{B = 1'b1;}
        \item \texttt{end}
      \end{itemize}
    \item \texttt{default: begin}
      \begin{itemize}
        \item \texttt{Sel = 2'b xxx;}
        \item \texttt{B = 1'b x;}
        \item \texttt{end}
      \end{itemize}
    \end{itemize}
  \end{itemize}
\end{itemize}
Other Procedural Block Examples

Loops can be used (with care) to specify logic:

```verilog
integer reg [7:0] A;
integer i, N, 0;

always@ (A)
begin
    if (A[i]) OddParity = ~OddParity;
    OddParity = 1'b0;
    for (i=0; i<=N; i=i+1)
        if (A[i]) OddParity = ~OddParity;
end

Is the following code fragment synthesizable?

No

H is a dynamic variable - Synthesis won't know its value
```
Instead:

```java
0 -> integer i;
parameter N=7;

New synthesizable.
- Synopsis known loop size.
```
A continuous assignment logic style must be used in synthesis for bit-slice logic.

Continuous Assignment Logic

Example:

- `foo` = `select` & `bar`.
- `write [3:0] bar`.
- `write [7:0] mush`.
- `trt [3:0] bus`.
- `mush = enable`.

Diagram:

```
  enable
   |
  --
  mush

  select
  --
  foo

  bus
```

Additional notes:

- ECE 320 Class Notes
- NC State University
Alternative

(feedback is not needed)
Reset is used in the reset if synchronous block. It may cause a false level at reset, unless reset logic is used. Right after reset, level to reduce impact of noise.

Synchronize reset at the block. BUT it is a good idea to don't need clock to reset registers with synchronous reset are smaller than registers with asynchronous reset.

Note:

Some flip-flop types:

Registers
Example: 7-state FSM, states 0...7:
- Usually gives fastest, next state, logic
- One bit per state
- One-hot encoding
- Minimizes switching activity in state vector register
- State bit changes by only one bit between sequential states
- Gray encoding
- Does not necessarily optimize, next state, logic, size
- Minimum number of bits and states in sequence
- Minimum sequential encoding
- Minimum number of bits
- Minimum encoding
- State vector encoding

FSM Types
DRAM

address
  row
  column

data

address row column

RCAS
RAS

"clock"

(reduces # pins
  - row and column address share some pins)