Redo Pbms 4 & 5 of HW 3
Redo Pbms 4 & 5 of HW 3

```python
# Homework 3, Problem 5

def decoderActiveHighE(E, A1, A0):
    Y = [0] * 4
    if E==1:
        index = A1*2 + A0
        Y[index] = 1
    return Y

def decoderActiveLowE_Outputs(E, A1, A0):
    Y = [1] * 4
    if E==0:
        index = A1*2 + A0
        Y[index] = 0
    return Y

def NOT(a):
    return 1 - a

def main():
    for a in [0, 1]:
        for b in [0, 1]:
            Y = decoderActiveLowE_Outputs(NOT(b), a, 0)
            f = Y[0] or Y[1]
            print(a, b, f)

main()
```
Using a Decoder with an FSM
Using a Decoder with an FSM

4 States ==> 2 Flip-flops: Q1, Q0

<table>
<thead>
<tr>
<th>Tn</th>
<th>Tn+1</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>S1</td>
</tr>
<tr>
<td>S1</td>
<td>S2</td>
</tr>
<tr>
<td>S2</td>
<td>S3</td>
</tr>
<tr>
<td>S3</td>
<td>S0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tn</th>
<th>Tn+1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1Q0</td>
<td>Q1Q0</td>
</tr>
<tr>
<td>00</td>
<td>01</td>
</tr>
<tr>
<td>01</td>
<td>11</td>
</tr>
<tr>
<td>11</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>00</td>
</tr>
</tbody>
</table>

\[ D1 = Q0 \]
\[ D0 = Q1' \]

\[ G = Q1' \cdot Q0' \]
\[ Y = Q1'.Q0 \]
\[ R = Q1 \]
"Don't Care" Conditions in Truth Tables and Karnaugh Maps

<table>
<thead>
<tr>
<th>Tn</th>
<th>Q1Q0</th>
<th>Tn+1</th>
<th>Q1Q0</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>01</td>
<td>01</td>
<td>10</td>
</tr>
<tr>
<td>01</td>
<td>10</td>
<td>10</td>
<td>00</td>
</tr>
<tr>
<td>10</td>
<td>00</td>
<td>11</td>
<td></td>
</tr>
</tbody>
</table>
"Don't Care" Conditions in Truth Tables and Karnaugh Maps

<table>
<thead>
<tr>
<th></th>
<th>ab</th>
<th>cd</th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Outline for This Week

- Moore vs Mealy FSM
- Unwanted States
- JK Flipflops
- ROM Sequencers
Two Types of Controllable FSMs: Moore & Mealy
Input → Combina-national

Flip-Flops → Clock

Combina-national

Moore Machine

Output
Flip-Flops

Combinational

Clock

Input

Output

Mealy Machine
How to Deal with Unwanted States?
Option 1
Option 2
Moore Machine With Input Command

Input -> Combinational

Flip-Flops

Clock

Combinational

Output
Exercise

How many states? What about Cmd?

An Example
• If the FSM is controlled by \( n \) inputs, then there must be \( 2^n \) arrows leaving each state of the state diagram.

• We *never* alter the clock signal in a Moore machine.

• The inputs to a Moore machine affect the state of the FSM *only during the transitions* of the clock.
Create a sequencer controlled by 1 switch. When the switch outputs 0, the sequencer actives 3 lights in this sequence: *Green, Yellow, Red, Green*, etc.

When the switch is 1, the sequencer goes *Yellow*, all off, *Yellow*, all off, etc…
# Initialize the flip-flops and the input switch
Q1 = 0
Q0 = 0
S = 0

def NOT(a):
    return 1 - a

# simulate 20 ticks
for step in range(20):

    # wait for the next clock tick (the user presses Enter)
    # get the value of the switch S from the user. User
    # must enter 0 or 1.
    while True:
        S = input(" > ").strip()
        if S != "0" and S != "1":
            print("Invalid input signal! Please reenter")
        else:
            S = int(S)
            break

    # combine switch value with current outputs (present)
    # to compute what the future outputs will be.
    D1 = NOT(S) | (Q0 ^ Q1)
    D0 = NOT(Q1 & Q0)

    # show the present outputs of the flip-flops
    print("S Q1Q0 =%d %d%d" % (S, Q1, Q0))

    # Tick! Whatever is on the D input of flip-flop
    # becomes the output.
    Q1 = D1
    Q0 = D0

http://www.science.smith.edu/dftwiki/index.php/CSC270_Python_simulator_for_Controllable_Sequencer
The JK Flipflop
Characteristic Table

<table>
<thead>
<tr>
<th>J K</th>
<th>Qt+1</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Qt</td>
</tr>
<tr>
<td>01</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>Qt'</td>
</tr>
</tbody>
</table>

\[
\text{Characteristic Table}
\]
Characteristic Table

<table>
<thead>
<tr>
<th>J K</th>
<th>Qt+1</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Qt</td>
</tr>
<tr>
<td>01</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>Qt'</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Qn Qn+1</th>
<th>Jn Kn</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td></td>
</tr>
<tr>
<td>0 1</td>
<td></td>
</tr>
<tr>
<td>1 0</td>
<td></td>
</tr>
<tr>
<td>1 1</td>
<td></td>
</tr>
</tbody>
</table>
Implement this FSM with JKS

Exercise:
Redo this with JKs

Implement this FSM with JKS
Where else do we find Similar FSMs and State Diagrams?
Instruction Cycle (with Interrupts) - State Diagram

Instruction fetch

Operand fetch

Operand store

Operand address calculation

Operand address calculation

Operand address calculation

Instruction operation decoding

Multiple operands

Multiple results

Instruction complete, fetch next instruction

Return for string or vector data

No interrupt

Data Operation

Interrupt check

Interrupt

http://images.slideplayer.com/18/6064420/slides/slide_3.jpg
Ready for Lab 6!

http://www.science.smith.edu/dftwiki/index.php/CSC270_Python_simulator_for_Controllable_Sequencer