ASM Chart Basics:
Example: In power distribution (supplying electricity to households and businesses),
there is always the possibility of a fault. Faults are short-circuits to ground caused
by downed wires, falling tree branches, etc. The algorithm used by power
companies is a state machine as follows:

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHORT (Detects grounding fault)</td>
<td>NORMAL (Normal operation)</td>
</tr>
<tr>
<td></td>
<td>FAULT (Fault detected)</td>
</tr>
<tr>
<td></td>
<td>PWR_OFF (Line power turned off)</td>
</tr>
<tr>
<td></td>
<td>FAILURE (Fault failed to clear)</td>
</tr>
</tbody>
</table>

ASM CHART FOR FAULT CLEARING
ON A DISTRIBUTION LINE
Explanation of ASM Chart:
- Rectangles represent states (state boxes)
  - The state name is **outside** the box above the upper left (or right) corner
  - State bits can also be displayed here
  - Each active clock edge causes a state transition/change
- Diamonds represent decisions (decision diamonds)
  - Each diamond has at least one *test input*
  - Each diamond at least two conditional branches
  - The present state of the input determines the active conditional branch
- Outputs
  - *Unconditional* output names are placed in state boxes
    (These synchronous outputs, like Moore Machine output.s)
  - *Conditional* output names are placed in ovals on the appropriate branch
    (These asynchronous outputs, like Mealy Machine outputs.)
  - Outputs are placed on an ASM chart **only** when they are **True**
    - All other outputs are assumed to be False

Note: You do not leave a state until you enter another one.
Diamonds and conditional outputs are “contained” in a state.

Example: Design a sequence detector that searches for a series of binary inputs to satisfy the pattern 01[0*]1, where [0*] is any number of consecutive zeroes. The output (Z) should become true every time the sequence is found.
Is it better to use conditional or unconditional outputs?
- Often, it is the designer’s choice.
- Remember that they work slightly differently:
  - Unconditional outputs are synchronous
    - Only change with the state at an active clock edge
  - Conditional outputs are asynchronous
    - If inputs change, they can change within a clock cycle
    - Extraneous changes in this manner are called *spurious pulses*
    - Can cause problems in asynchronous circuits
    - Clocked circuits avoid problems with spurious pulses
- A conditional output on all branches is the same as an unconditional output
  - Moore Machines can always be represented as Mealy Machines
  - Mealy Machines cannot always be represented by Moore Machines

**Designing Controllers:**
- Typical digital design applications require the design of controllers
  - Examples: Cars, Traffic Lights, Jet Engines, Power Plants, Printers, etc.
Example: Design a washing machine controller.

Description: The controller begins when it receives the \texttt{STRT} (start) signal from the user. It fills the washer with either cold or hot water (selected by the user) depending on the \texttt{SHOT} input, where \texttt{SHOT} is True when hot water is desired. The washer agitates until a timer indicates that the cycle is finished. The controller then drains the soapy water and fills the machine with cold water for the rinse cycle. The washer agitates again until the timer indicates that the cycle is finished. The controller drains the rinse water and finally enters the spin cycle, spinning the clothes dry until the timer indicates the end of the cycle. If the off button is ever pushed, the washer holds the current state until the start button is pressed again.

\begin{center}
\begin{tikzpicture}
  \node [state] (controller) {CONTROLLER (ASM)};
  \node [state, right of=controller, xshift=2cm] (WashingMachine) {WASHING MACHINE & TIMER};
  \node [state, above of=controller, yshift=-1cm] (FullEmpTo) {FULL, EMP, TO};
  \draw [->] (controller) -- (WashingMachine);
  \draw [->] (FullEmpTo) -- (controller);
  \draw [->] (controller) -- (WashingMachine) node [midway, above] {START};
  \draw [->] (controller) -- (WashingMachine) node [midway, above] {SHOT};
\end{tikzpicture}
\end{center}

WASHING MACHINE CONTROLLER

Step 1: List the inputs and outputs:

\begin{itemize}
  \item **Inputs**
    \begin{itemize}
      \item \texttt{STRT} (Start from user)
      \item \texttt{SHOT} (Select hot from user)
      \item \texttt{FULL} (Tub full indicator)
      \item \texttt{EMP} (Tub empty indicator)
      \item \texttt{TO} (Timer out indicator)
    \end{itemize}
  \item **Outputs**
    \begin{itemize}
      \item \texttt{OHOT} (Turn on hot water valve)
      \item \texttt{PUMP} (Turn on pump)
      \item \texttt{FILL} (True: Water in, False: Water out)
      \item \texttt{AG} (Agitate)
      \item \texttt{STIME} (Clear and start timer)
      \item \texttt{PTIME} (Pause the timer)
      \item \texttt{SPIN} (Spin dry)
      \item \texttt{STRTOFF} (Go to idle state)
    \end{itemize}
\end{itemize}

Note: 5 inputs could require $2^5 = 32$ arrows going out of each bubble in a state diagram. This is why we use an ASM chart.
Step 2: Draw ASM chart:

WASHING MACHINE CONTROLLER ASM

Note: The STIME output must be synchronized (by adding a D flip-flop) so that its effects on the timer do not take place until a clock edge triggers the next state. If we used an asynchronous output for STIME, the controller could not transition out of the RINSE 2 state because when STIME fired the TO decision diamond would immediately become false. This would cause the washing machine to continue agitating indefinitely.
Step 3: Implementation:
- Assign states: Idle = 000, Agitate = 001, Drain = 010,
  Rinse1 = 011, Rinse2 = 100, SpinIt = 101
- Make Next State Truth Table (NSTT)
- At this point, we could use K-maps and solve the combinational logic.
- Instead, we will implement using a 256 x 8 ROM
  - Store the NSTT in the ROM
  - Wire up inputs, outputs, and state register.
    - Problem: Not enough outputs on the ROM
    - Solution: Find outputs that are functions of other outputs
      or choose outputs to directly implement outside the ROM
Constructing an ASM Chart from a Timing Diagram

Construct an ASM chart that could yield the following timing diagram.

Note: There are 2 inputs, so there are $2^2 = 4$ possible conditional branches for each state. Assume the simplest decision diamond that yields the correct result.

(The result should be the same ASM chart as shown in the next section.)
Producing a Timing Diagram from an ASM Chart

Complete the timing diagram, given the ASM chart below.

(Figure 7.2 modified, Fundamentals of Computer Engineering: Logic Design and Microprocessors by Lam, O’Malley, and Arroyo)

Note: IN.BIT only effects the transition from state A.

BUF.FULL only effects the transition from state C.

(The result should be the same as the timing diagram shown in the previous section.)