Dillon Dragomir

Philip Wilsey

EECE 3026 - Computer Architecture and Organization

4 February 2018

**Project 2: Digital Systems Review, Sequential Circuits**

**Objective**

Design a sequential circuit to emulate part of the instruction interpretation cycle.

**References:**

Computer Organization and Design, 4th edition

**Programs:**

Logisim 2.7.0

Finite State Machine Designer

**Components**:

Main: AND: 2 OR: 3 NOT: 5

D-Latch: AND: 8 OR: 0 NOT: 10

3-8 Decoder: AND: 8 OR: 0 NOT: 3

**Inputs**:

indirect, program\_check, C1, clock, reset

**Outputs**:

fetch, decode, indirect, execute, trap, S1, S2

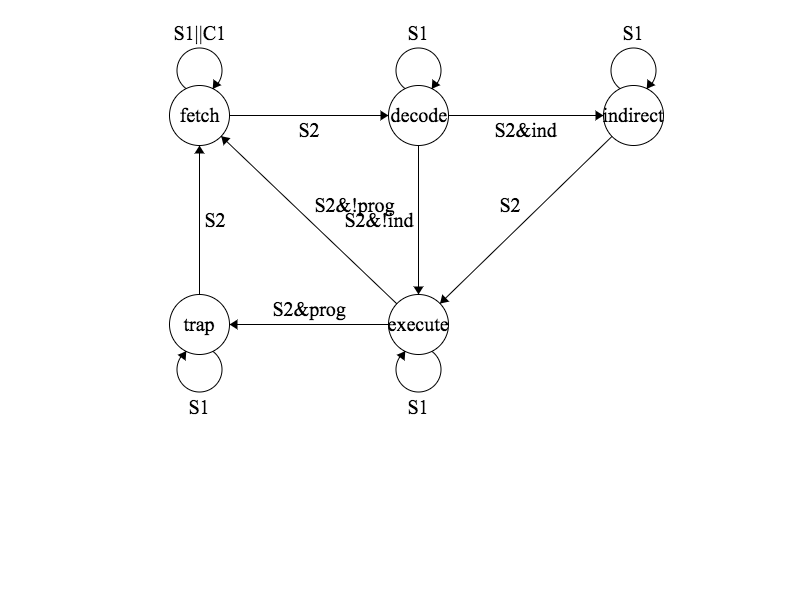
**Abstract**:

The main purpose of this project was to create a state machine that goes through the instruction interpretation cycle, which could be used in a control unit. The design procedure was as follows: a state machine diagram of the desired states was created and then each state was assigned a 3-bit binary representation. A table was then created with the previous state, future state and the inputs that cause variations to this. These tables were then converted to karnaugh maps, which were then used to create next state algebraic expressions. These expressions were then simplified using boolean algebra. Next, the state of the machine was stored in three D-Latches and S1/S2 was stored as a single bit in a final D-Latch. Finally, this three-bit representation of the circuit is decoded to the given state using a 3-8 Decoder.

**Procedure:**

**State Diagram**

The given state machine was designed using Finite State Machine Designer to match the given specifications in the design document. The state machine in figure 1 shows the state transitions based on the given inputs for the instruction interpretation cycle. The || symbol was used to represent “OR”, the & symbol represents “AND”, and ! symbol to represent “NOT”.



**Figure 1: Instruction Cycle State Machine**

Next, each state was mapped to a binary representation of the given state that will be stored as 3-bits of data in D-Latches.

|  |  |
| --- | --- |
| **State** | **Binary Representation** |
| fetch | 000 |
| decode | 001 |
| execute | 010 |
| indirect | 011 |
| trap | 100 |
| **S** | **Binary Representation** |
| S1 | 0 |
| S2 | 1 |

**Figure 2: Mapping Template**

**State Table**

The table displayed in figure 3 shows the previous and future states of the state machine and the inputs that affect the future state. The “x” notation means that the future state should not be dependent on that input.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Previous State** | | | **Inputs** | | | **Future State** | | |
| **I0** | **I1** | **I2** | **Indirect** | **Program** | **C1** | **I0** | **I1** | **I2** |
| 0 | 0 | 0 | x | x | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | x | x | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | x | x | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | x | x | 0 | 1 | 1 |
| 0 | 1 | 0 | x | 0 | x | 0 | 0 | 0 |
| 0 | 1 | 0 | x | 1 | x | 1 | 0 | 0 |
| 0 | 1 | 1 | x | x | x | 0 | 1 | 0 |
| 1 | 0 | 0 | x | x | x | 0 | 0 | 0 |

**Figure 3: State Transitions on S2**

**Karnaugh Maps**

The data from figure 3 was then inputted into karnaugh maps to help create the next state equations. Since some states are not utilized, they can be ignored and make simplification easier. Some future states in figure 4 cannot be determined from the previous state alone, and thus are represented as 0/1. The future state is dependent on an input and looking back at figure 3 can provide insight into the input the given bit is affected by.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **I0** | | **I2** | | **I1** | | **I2** | | **I2** | | **I2** | |
| **0** | **1** | **0** | **1** | **0** | **1** |
| **I0/**  **I1** | **00** | 0 | 0 | **I0/**  **I1** | **00** | 0 | 1 | **I0/**  **I1** | **00** | 0/1 | 0/1 |
| **01** | 0/1 | 0 | **01** | 0 | 1 | **01** | 0 | 0 |
| **10** | 0 | x | **10** | 0 | x | **10** | 0 | x |
| **11** | x | x | **11** | x | x | **11** | x | x |

**Figure 4: Karnaugh Maps**

**Algebraic Expressions**

Since I0(future) only occurs in one state when the Program bit is enabled the given expression can be derived.

I0(future) = (I0’ \* I1 \* I2’ \* Program)

I1(future) can be set equal to I2(previous) because there is a direct mapping between these two variables.

I1(future) = I2

Finally, I2(future) is “1” in two instances and equations can be derived as the following.

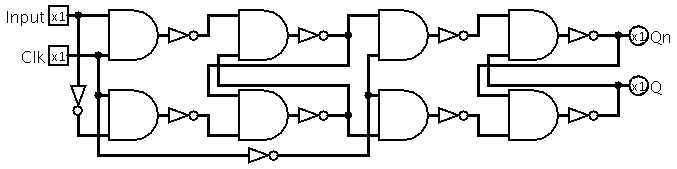
I2(future) = (I0’ \* I1’ \* I2’ \* C1’) + (I0’ \* I1’ \* I2 \* Indirect)

The equation is further simplified to limit the number of NOT gates. This is derived using DeMorgan’s Law.

I2(future) = (I0 + I1 + I2 + C1)’ + (I0’ \* I1’ \* I2 \* Indirect)

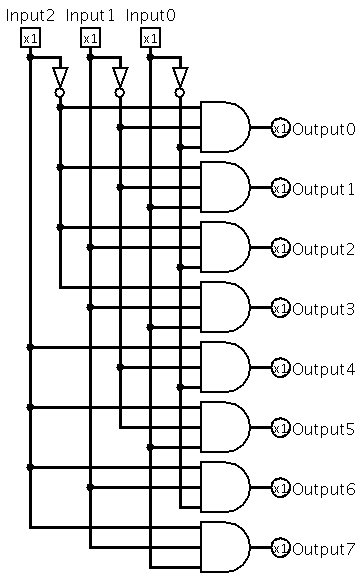
**Logisim Sub-circuits**

This circuit depicted in figure 5 is a master slave D-Latch and requires two clock cycles to update the value it stores. This is gets rid of the risk of oscillation in the stored bit. This is also the reason why S1 and S2 is needed and state transitions only happen ever two clock signals.



**Figure 5: Master-Slave D-Latch**

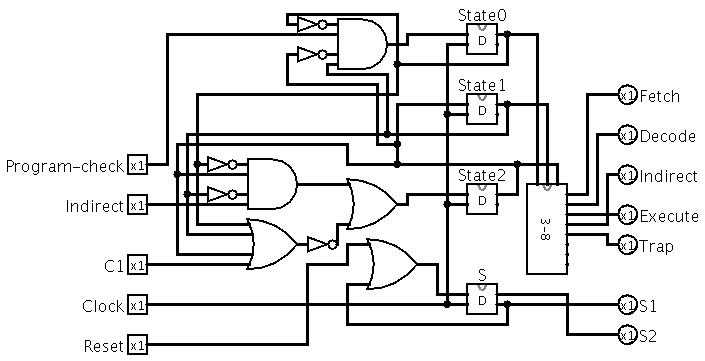
The circuit depicted in figure 6 is a 3-8 Decoder and is used to map the 3-bit binary representation of the state into an output for each state the machine could be in. For this specific state machine only outputs 0-4 are used.



**Figure 6: 3-8 Decoder**

**Logisim Main Circuit**

The final circuit depicted in figure 7 is where the main logic described in the future state equations is implemented. The previous state and input bits are used to set the future state of the D-Latch and two clock cycles later the future state is set. S1 and S2 is stored as a single bit and oscillates every clock cycle. The previous states 3-bits is connected to a 3-8 decoder and then the outputs of the decoder are mapped to the state they represent.



**Figure 7: Instruction Cycle Main Circuit**

**Initializing Circuit**

To initialize the circuit, set reset to “1”, send one clock cycle through the circuit and set reset back to “0”. Finally, set C1 to “1” and send two clock signals through the circuit and the state machine will be in the fetch state. The circuit will now behave as desired.

**Testing Circuit**

This circuit was tested to meet all specifications laid out in the design document. When the indirect bit is enabled the circuit will loop through the states on S2 as follows: fetch, decode, indirect and execute. When the program\_check bit is enabled the circuit will loop through the states on S2 as follows: fetch, decode, execute and trap. When both indirect and program check bits are enabled the circuit will loop through all states on S2. Finally, when C1 is enabled, when the circuit reaches the fetch state, it will stay in the fetch state.