21 October

- Only 12 to go!
- Been to the Fair?
- Assignment 9 due 28th instead of 26th
- Today Control

Synchronous Systems

On the leading edge of the clock, the input of a latch is transferred to the output and held.
We must be sure the combinational logic has settled before the next leading clock edge.

Asynchronous Systems

No clock!
The data carries a "valid" signal along with it
System goes at greatest possible speed.
Only "computes" when necessary.

Everything we look at will be synchronous

Fetching Sequential Instructions

How about branch?

Datapath for R-type Instructions

Fun with MUXes

Remember the MUX?

This will route 1 of 4 different 1 bit values to the output.
The select signal determines which of the inputs is connected to the output.

Our Register File has 3 ports

This is one reason we have only a small number of registers. What’s another reason?

1 Write Port

2 Read Ports

Instr Bits 25-21

Instr Bits 20-15

Instr Bits 15-11

Write Reg.

Write Data

Really Lots of Connections!

We can get 1 bit out with a MUX

Put the input here.

Select 0 1 2

Input

Output

For example, input AND has INPUT7 wired HIGH and all the others wired LOW.

Inside there is a 32 way MUX per bit

For each bit in the 32 bit register

Lots of connections!

And this is just one port!

Inside there is a 32 way MUX per bit

For each bit in the 32 bit register

Lots of connections!

And this is just one port!

Implementing Logical Functions

Suppose we want to map M input bits to N output bits.

For example, we need to take the OPCODE field from the instruction and determine what OPERATION to send to the ALU.

OPCODE bits from instruction

Map to ALU op

We can get 1 bit out with a MUX

Or use a ROM

Put the input here.

Select 0 1 2

Input

Output

For example, input AND has INPUT7 wired HIGH and all the others wired LOW.

Or use a ROM

Read-Only Memory

M-bit Address

N-bit Result
Or use a PLA

Programmable Logic Array

M-bit Input → AND Array → Product Terms → OR Array → N-bit Output

Think of the SUM of PRODUCTS form.
The AND Array generates the products of various input bits
The OR Array combines the products into various outputs

Finite State Machines

- A set of STATES
- A set of INPUTS
- A set of OUTPUTS
- A function to map the STATE and the INPUT into the next STATE and an OUTPUT

Remember “Shoots and Ladders”?

Traffic Light Controller

Implementing a FSM

Recognizing Numbers

Recognize the regular expression for floating point numbers

[ [ -+] [0-9]* ]; [0-9]* ]; [e [ -+] [0-9]* ]?

Examples:
+123.456e23
-.456
1.5e-10
-123

FSM Diagram
### FSM Table

<table>
<thead>
<tr>
<th>IN : STATE</th>
<th>NEW STATE</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>.</code></td>
<td>start → start</td>
</tr>
<tr>
<td><code>0</code></td>
<td><code>1</code> ...</td>
</tr>
<tr>
<td><code>.</code></td>
<td><code>1</code> ...</td>
</tr>
<tr>
<td><code>.</code></td>
<td>start → frac</td>
</tr>
<tr>
<td><code>0</code></td>
<td><code>1</code> ...</td>
</tr>
<tr>
<td><code>.</code></td>
<td>sign → frac</td>
</tr>
<tr>
<td><code>0</code></td>
<td><code>1</code> ...</td>
</tr>
<tr>
<td><code>.</code></td>
<td>whole → done</td>
</tr>
<tr>
<td><code>.</code></td>
<td>whole → exp</td>
</tr>
</tbody>
</table>

- `.` : frac → exp
- `0` | `1` ... | `9` : frac → frac
- `.` : frac → done
- `0` | `1` ... | `9` : exp → exp
- `.` : exp → done

STATE ASSIGNMENTS
- start = 0 = 000
- sign = 1 = 001
- whole = 2 = 010
- frac = 3 = 011
- exp = 4 = 100
- done = 5 = 101
- error = 6 = 110

### FSM Implementation

```
char in
```

Our PLA has:
- 11 inputs
- 5 outputs

### FSM Take Home

- With **JUST** a register and some logic, we can implement complicated sequential functions like recognizing a FP number.

- This is useful in its own right for compilers, input routines, etc.

- The reason we’re looking at it here is to see how designers implement the complicated sequences of events required to implement instructions

- Think of the OP-CODE as playing the role of the input character in the recognizer. The character AND the state determine the next state (and action).