Announcements

• HW6 Due Today
• HW7 will be made available at some point today
What we are going to cover today

• Expand our machine to cover other instructions (Id)
• Expand next state logic
• Walk through datapath for a simple program
Adding LD

• We want to add LD to the diagram.
• What should we modify first?
  • How about REG! REG needs to be INST[8:4] for LD
Quick Auxiliary Review

- Auxiliary registers have three ports: din, dout, and we
  - When we is low (0) dout ignores din, and continues to be set to what din was before we went low.
  - When we is high (1) dout = din

<table>
<thead>
<tr>
<th>We</th>
<th>0</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Din</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Dout</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Modifying REG

```
INST
  ^   4
 /    /
we d   dout
  16    1,7
      8
REG
```
Modifying REG

• We have two different inputs and we need to choose
• What is our way of choosing between signals?
• MUX
Modifying REG
Modifying Microarchitecture Basics

- Different instructions modify how we parse or pass values
- "Choose" values using a MUX and a control signal
- Control signals are added to our state control machine
Complex State Machine

- State isn't as simple as adding and looping around
- State machine needs to move and choose between values
- Create a table for moving between states based on opcode
Complex State Machine

Fetch

Decode

Memory

Writeback

INST+PO[PC]

REG1, INST[7:4]

REG1, INST[8:4]

ADD=RF[26], RF[27]

VAL=INST[11:0], INST[3:0]

VAL=REG[ADDR]

RF(REG)=VAL

PC=PC+1
### Complex State Machine

- Assign a number to each state

<table>
<thead>
<tr>
<th>State Description</th>
<th>State Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>INST = PM[PC]</td>
<td>0</td>
</tr>
<tr>
<td>REG = 1, INST[7:4]</td>
<td>1</td>
</tr>
<tr>
<td>REG = INST[8:4]</td>
<td>2</td>
</tr>
<tr>
<td>ADDR =X</td>
<td>3</td>
</tr>
<tr>
<td>VAL = INST[11:8],INST[3:0]</td>
<td>4</td>
</tr>
<tr>
<td>VAL = RAM[ADDR]</td>
<td>5</td>
</tr>
<tr>
<td>RF[REG] = VAL</td>
<td>6</td>
</tr>
<tr>
<td>PC = PC + 1</td>
<td>7</td>
</tr>
</tbody>
</table>
### Complex State Machine

- A new table for choosing state based on opcode

<table>
<thead>
<tr>
<th>State Description</th>
<th>State Number</th>
<th>Current State</th>
<th>Opcode</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td>INST = PM[PC]</td>
<td>0</td>
<td>0</td>
<td>LDI</td>
<td>1</td>
</tr>
<tr>
<td>REG = 1, INST[7:4]</td>
<td>1</td>
<td>0</td>
<td>LD</td>
<td>2</td>
</tr>
<tr>
<td>REG = INST[8:4]</td>
<td>2</td>
<td>1</td>
<td>LDI</td>
<td>4</td>
</tr>
<tr>
<td>ADDR = X</td>
<td>3</td>
<td>2</td>
<td>LD</td>
<td>3</td>
</tr>
<tr>
<td>VAL = INST[11:8],INST[3:0]</td>
<td>4</td>
<td>3</td>
<td>LD</td>
<td>5</td>
</tr>
<tr>
<td>VAL = RAM[ADDR]</td>
<td>5</td>
<td>4</td>
<td>LDI</td>
<td>6</td>
</tr>
<tr>
<td>RF[REG] = VAL</td>
<td>6</td>
<td>5</td>
<td>LD</td>
<td>6</td>
</tr>
<tr>
<td>PC = PC + 1</td>
<td>7</td>
<td>6</td>
<td>LD, LDI</td>
<td>7</td>
</tr>
</tbody>
</table>
Complex State Machine

State

Next State Logic

state

Next_state

we/clk

opcode
Adding LD (ADDR and RAM)

• What pins should we have for an ADDR register and RAM?
  • ADDR register will be an auxiliary register, so din, dout, and we
  • RAM will have...
    • Addr for the address of memory
    • We for writing/storing
    • Din for the data input
    • Dout for the data coming from the RAM
Adding LD (ADDR and RAM)
Adding LD (Modifying VAL)

• Works the same way as REG
• Two inputs, select between the two
• Add a MUX to select, and a control signal from the state control machine to the MUX
Adding LD (Modifying VAL)
Adding LD (State Machine)
## Complex State Machine

<table>
<thead>
<tr>
<th>Condition</th>
<th>PC_we</th>
<th>INST_we</th>
<th>REG_we</th>
<th>REG_sel</th>
<th>VAL_we</th>
<th>VAL_sel</th>
<th>RF_we</th>
<th>ADDR_we</th>
<th>RAM_we</th>
</tr>
</thead>
<tbody>
<tr>
<td>INST=PM[PC]</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>REG=1,INST[7:4]</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>REG=INST[8:4]</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ADDR=X</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>VAL=INST[11:8],INST[3:0]</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>VAL=RAM[ADDR]</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>RF[REG]=VAL</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PC=PC+1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Why are there blanks on some of the control signals in some states?

- Well, we don’t care about some of those, whether they are set one way or another during a particular state.
- What makes not caring possible?
  - Auxiliary Registers
Walkthrough

- Let’s walk through a simple program

  LDI R16, 1
  LD R10, X
Walkthrough: LDI R16, 16
0b0000_0000_0000_0001

State: 0
Walkthrough: LDI R16, 16
0b0000_0000_0000_0000

State: 1 REG = 1, INST[7:4]
Walkthrough: LDI R16, 16

0b0000_0000_0000_0000_0001

State: 4 VAL = INST[11,8], INST[3:0]
Walkthrough: LDI R16, 16
0b0000_0000_0000_0001

State: 6 RF[REG] = VAL
Walkthrough: LDI R16, 16
0b0000_0000_0000_0001

State: 7 PC = PC + 1
Walkthrough: LD R10, X
0b1001_0000_1010_0001
Walkthrough: LD R10, X
0b1001_0000_1010_0001

State: 2 REG = INST[8:4]
Walkthrough: LD R10, X
0b1001_0000_1010_0001

State: 3 ADDR = X
Walkthrough: LD R10, X
0b1001_0000_1010_0001

State: 5 VAL = RAM[ADDR]
Walkthrough: LD R10, X
0b1001_0000_1010_0001

State: 6 RF[REG] = VAL

STATE MACHINE CONTROLLER
Walkthrough: LD R10, X
0b1001_0000_1010_0001

State:
PC = PC + 1

STATE MACHINE CONTROLLER
Adding the rest

• Other instructions can be added
  • Add states and modify the state machine to transition properly
  • Add MUX’s and registers to save different values, add more register sources, and select between different options
  • Add control signals from the state machine to control MUX select switches and write enables.
Adding the rest
Summary

• Adding instructions requires adding needed auxiliary registers and components to facilitate doing the instruction
• We also need to add MUX’s to select between different possibilities for things like destination registers
• New control signals and new states need to be added to our state machine control logic