Control Logic Implementation and Microprogramming
Objectives

• Be familiar with the process of turning an RTL specification of instruction execution into a state diagram for control logic

• Understand microprogramming and the design of a microprogrammed control unit for the LC-3bα
The LC-3bα Datapath
LC-3bα State Diagram

IF1
- MAR <- PC;
- PC <- PC + 2;
- Ready

IF2
- MDR <- MEM[MAR];
- Ready

IF3
- IR <- MDR;

ADD
- Rd <- Ra ADD Rb;
- genCC;

AND
- Rd <- Ra AND Rb;
- genCC;

NOT
- Rd <- NOT Ra;
- genCC;

Decode

CalcAdr
- MAR <- Ra + ADJ6(IR[5:0]);

ST1
- MDR <- Rd;
- Ready

ST2
- MEM[MAR] <- MDR;
- Ready

MEM[MAR] <- MDR;

LD1
- MDR <- MEM[MAR];
- Ready

LD2
- Rd <- MDR;
- genCC;

BR1
- CND true
- PC <- PC + ADJ9(IR[8:0]);

BR2
- CND false
- IF1

9/6/06 ECE 411, Lecture 4
Assigning State Numbers to States

• Issue 1: How many state bits do you use?
  – More state bits typically means simpler logic to generate the next state.
  – More state bits means more logic to hold the state, often more total logic.
    • One common choice: \( \# \text{ state bits} = \log_2(\# \text{ states}) \)
    • Another option: \( \# \text{ state bits} = \# \text{ states} \) (one-hot encoding)

• Issue 2: Mapping of state to state numbers
  – Somewhat arbitrary, in that any mapping can give correct implementation
  – However, choice of mapping has a large impact on how complex the next state logic is
    • Some software programs to do this, but often a task that gets done by hand
State Assignments for the LC-3bα

**IF1 1100**
- \( \text{MAR} \leftarrow \text{PC} \)
- \( \text{PC} \leftarrow \text{PC} + 2 \)

**IF2 1100**
- \( \text{MDR} \leftarrow \text{MEM}[[\text{MAR}]] \)
- Ready

**IF3 1101**
- \( \text{IR} \leftarrow \text{MDR} \)

**ADD 0001**
- \( \text{Rd} \leftarrow \text{Ra} \text{ ADD Rb}; \text{genCC} \)
- IF1

**AND 0101**
- \( \text{Rd} \leftarrow \text{Ra} \text{ AND Rb}; \text{genCC} \)
- IF1

**NOT 1001**
- \( \text{Rd} \leftarrow \text{NOT Ra}; \text{genCC} \)
- IF1

**Decode 0100**
- Ready

**BR1 0000**
- CND true

**BR2 1111**
- CND false

**CalcAdr 0011**
- MAR \( \leftarrow \text{Ra} + \text{ADJ6}(\text{IR}[5:0]) \)

**CalcAdr 0010**
- MAR \( \leftarrow \text{Ra} + \text{ADJ6}(\text{IR}[5:0]) \)

**ST1 1011**
- MDR \( \leftarrow \text{Rd} \)

**ST2 1010**
- MEM[MAR] \( \leftarrow \text{MDR} \)
- Ready

**LD1 0110**
- MAR \( \leftarrow \text{MEM}[\text{MAR}] \)
- Ready

**LD2 0111**
- \( \text{Rd} \leftarrow \text{MDR} \)
- genCC;
- IF1
Implementing State Machines

• Two general approaches
  – Hardwired
  – Microprogrammed

• In a hardwired state machine, we build a circuit that implements the state machine directly

• In a microprogrammed state machine, we use a memory to hold the state transition tables.
Hardwired Control Units

- Current State
- Combinational Logic
- Inputs
- State Register
- Next State
- Control Signals

Diagram: Flowchart of Hardwired Control Units.
Computing the Next State

Much less structured process than we’ve seen so far

Basically, construct a state table that lists, for each state, the next state as a function of the inputs (this is where the state encoding starts to matter.)

Then, use ECE 290 techniques to minimize the logic to implement.

<table>
<thead>
<tr>
<th>Ex:</th>
<th>State Name</th>
<th>State Value</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td>BR1</td>
<td>0000</td>
<td>111[cnd]</td>
<td></td>
</tr>
<tr>
<td>DECODE</td>
<td>0100</td>
<td>IR[15:12]</td>
<td></td>
</tr>
</tbody>
</table>
Computing Outputs

One Approach:

• Construct a set of logic equations for each output based on the states that produce them

Ex:

LoadMAR = (curr_state == 1110) OR
          (curr_state == 0010) OR
          (curr_state == 0011);

• Implement logic for this table
  – Often use decoder to convert binary encoding of states into one-hot signal
Microprogramming

- **Observation:** Much of the difficulty of designing/implementing hardwired control units comes from the need to optimize the next state generation logic.

- **Idea:** If we could use a memory to hold the state transition diagram for the control unit, we wouldn’t have to bother optimizing any logic.
  - Use current state as “address” into memory.
  - Contents of each memory location are the next state for each state.

- Control units that use this technique are called *microprogrammed* control units.
Using Memory to Implement Logic

\[ f(x, y, z) = xyz + xz \]

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Z</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

8x1 bit ROM/RAM (contents = 01110000)

ECE 411, Lecture 4
Some Vocabulary

• **Control Store**: The RAM or ROM that holds the truth table for a microprogrammed control system

• **Microprogramming**: The process of realizing a combinational circuit in a control unit based on ROM/RAM

• **Microprogram/Microcode**: The contents of the control store in a microprogrammed control system

• **Microinstruction**: The contents of a single location in the control store
A Microprogrammed Control Unit, Take 1

Control Store

Next uAddr

Microinstruction Register

Control Signals

Clock
A Microprogrammed Control Unit

- Control Store
- Sequencer Logic
- Microinstruction Register
- Control Signals

Inputs

Clock

Next uAddr
Programming a Microprogrammed Control Unit

- Can translate RTL description/state diagram directly
- Each microinstruction executed in exactly one clock cycle
  - Any wait loops need to be implemented using multiple states/microinstructions
Why Would Anyone Want to do This?

- Simple design
  - Encode truth table directly in ROM/RAM, no need to design/optimize/minimize logic to compute next state and control signals
  - Mapping of states to state numbers less critical
  - May take less chip area than hardwired logic, depending on complexity of next state and control signal equations
    - Very likely to be smaller when building system out of TTL logic, for example

- Bug fixes
  - Bugs in the control logic can usually be fixed by changing the contents of the RAM/ROM
    - Particularly useful on older computers, where the control logic was implemented out of many discrete parts

- Tie-In to ISA Design
  - ISAs with very complex instructions often benefit more
Why Isn’t Microprogramming Perfect?

• Speed
  – Reading a large control store is often slower than performing the equivalent computation in logic
  – Centralized microprogrammed control doesn’t work so well when you start doing multiple things in parallel
    • State space explodes

• Microprogramming has fallen out of favor for workstation/PC processors for a number of reasons
  – Entire chip fabbed at one time – makes the ability to do bug fixes by changing control store less useful
  – Speed paramount, size of control logic less critical
    • Transistors are cheap
A Microprogrammed Control Unit for the LC-3bα

• Three questions:
  – What does sequencer logic look like?
  – What is the format of a microinstruction?
  – How many bits do we need in the control store?
Sequencer Logic

• LC-3bα control logic has fifteen states
  – Four states have two possible next states
  – One state (decode) has six possible next states

• Brute force approach:
  – Allow each microinstruction to specify six possible next states
    and multiplex between them using control inputs
    • Many wasted bits in control store

• Space/Complexity Compromise:
  – Each microinstruction specifies two possible targets, and
    special-case the decode state
LC-3bα Sequencer Logic (Microsequencer)

Microinstruction

Control Signals | Target 1 | Target 2

UseReady | UseCnd | Decode

(UseReady \& Ready) | (UseCnd \& Cnd)

IR[15:12]

Decode

Next uAddr
Microinstruction Format and Control Store Size

- FSM has 15 states -> 4 bits to hold next state, 16 addresses in control store
- LC-3bα has 16 control signals
- Therefore, 27 (16 + 4 + 4 + 3) bits/microinstruction, 432 bits in the control store
Optimizations

• General multi-way branching in control store
  1. Assign all targets of a multi-way branch to addresses in the control store that differ only in their LSBs
  2. When microinstructions have multiple targets, use the inputs to compute the LSBs of the microinstruction, while the target field holds the more significant bits

• Control Signal Encoding
  – Many control units have lots of control signals
    • Require many bits of control store if each signal has a bit in the microinstruction
  – Identify groups of control signals that are never high at the same time
  – Use a group of bits in the control word to select which control signal in the group is active on each cycle
    • Decoder converts encoded back into one-hot
Next Time

• Introduction to Computer Architecture: Measuring Performance and Physical Memories

• **Announcement**: There will be no lecture Wednesday, September 13
  – I’m going to steal one of the cancelled lectures because of evening exams
Backup Slides
Instruction Fetch State Machine

RTL

MAR <- PC; PC <- PC + 2;
MDR <- MEM[MAR];
IR <- MDR;
Instruction Fetch State Machine 2

RTL

MAR <- PC; PC <- PC + 2;
MDR <- MEM[MAR];
IR <- MDR;

State Machine

IF1
IF2
IF3
Decide

MAR <- PC;
PC <- PC + 2;
MDR <- MEM[MAR];
IR <- MDR;

ADD, AND, NOT, LD/ST, BR
Arithmetic Instructions

- **ADD**
  \[ \text{Rd} \leftarrow \text{Ra ADD Rb}; \text{genCC} \]

- **AND**
  \[ \text{Rd} \leftarrow \text{Ra AND Rb}; \text{genCC} \]

- **NOT**
  \[ \text{Rd} \leftarrow \text{NOT Rb}; \text{genCC} \]
Branch

RTL

If ((n AND N) OR (z and Z) OR (p AND P))
PC <- PC + ADJ9(IR[8:0]);
LD/ST

RTL

• LD
MAR <- Ra + ADJ6(IR[5:0]);
MDR <- MEM[MAR];
Rd <- MDR; genCC;

• ST
MAR <- Ra + ADJ6(IR[5:0]);
MDR <- Rd;
MEM[MAR] <- MDR;